МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет1 Информационных технологий 1

Кафедра1 Информационные системы и технологии 1

Специальность 1 1–40 05 01–03 Информационные системы и технологии 1

Специализация1 Издательско-полиграфический комплекс 1

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к дипломному проекту на тему:**

Дипломник 1 1

(Ф.И.О.)

Руководитель проекта 1 доц., к.т.н. С. И. Акунович 1

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой 1 проф., д.т.н. Урбанович П.П. 1

(учен. степень, звание, подпись, Ф.И.О.)

Консультанты: 1 доц., к.т.н. С. И. Акунович 1

(учен. степень, звание, должность, подпись, Ф.И.О.)

1

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер 1 доц., к.т.н. Жиляк Н.А. 1

(учен. степень, звание, должность, подпись, Ф.И.О.)

Дипломный проект защищен с оценкой 1 1

Председатель ГЭК

(учен. степень, звание, подпись, Ф.И.О.)

Минск 20151

# Реферат

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

4

*ДП 00.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Пров.*

*.*

*Акунович С.И.*

*Консульт.*

*Акунович С.И.*

*Н. контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Реферат*

*Лит*.

*Листов*

1

*БГТУ 84419015, 2015*

Пояснительная записка содержит \_\_ страниц, \_\_ рисунка, \_\_ таблиц, \_\_ литературных источников, \_ приложений.

*У*

4

# Содержание

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

5

*ДП 00.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консульт.*

*Акунович С.И.*

*Н. контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Содержание*

*Лит.*

*Листов*

2

*БГТУ 84419015, 2015*

*БГТУ 64419815, 2013*

[Реферат 4](#_Toc419933835)

[Содержание 5](#_Toc419933836)

[Введение 7](#_Toc419933837)

[1 Аналитический обзор литературы 10](#_Toc419933838)

[1.1 Программируемый логический контроллер для систем автоматизации 12](#_Toc419933839)

[1.1.1 Стандартные языки программирования ПЛК 15](#_Toc419933840)

[1.2 Краткие сведения о системе ISaGRAF 19](#_Toc419933841)

[1.2.1 Основные свойства системы 19](#_Toc419933842)

[1.2.2 Спектр возможностей 20](#_Toc419933843)

[1.2.3 Средства технологического программирования 20](#_Toc419933844)

[1.2.4 Основные языки описания алгоритмов логического управления 22](#_Toc419933845)

[1.3. Язык функциональных блоковых диаграмм FBD 25](#_Toc419933846)

[1.3.1 Описание языка FBD 25](#_Toc419933847)

[1.3.2 Программирование на FBD в среде ISaGRAF 6.1 27](#_Toc419933848)

[1.5 Патентный поиск 30](#_Toc419933849)

[1.6 Вывод 30](#_Toc419933850)

[2 Математические модели, положенные в основу проекта, и теоретические исследования 32](#_Toc419933851)

[2.1 Описание поставленной задачи 32](#_Toc419933852)

[2.2 Обоснование актуальности исследуемой задачи 32](#_Toc419933853)

[2.3 Требования к программному средству 32](#_Toc419933854)

[2.4 Алгоритм решения задачи 33](#_Toc419933855)

[2.5 Структура программного кода 33](#_Toc419933856)

[2.6 Вывод 34](#_Toc419933857)

[3 Обоснование выбора средств разработки 35](#_Toc419933858)

[3.1 Язык программирования DELPHI 35](#_Toc419933859)

[3.2 Среда разработки Embarcadero RAD Studio XE5 36](#_Toc419933860)

[3.3 Вывод 38](#_Toc419933861)

[4 Проектирование и реализация программного средства 39](#_Toc419933862)

[4.1 Исходные данные 39](#_Toc419933863)

[4.2 Назначение разработанного приложения 39](#_Toc419933864)

[4.3 Интерфейс программы 40](#_Toc419933865)

[4.4 Принцип работы программы 41](#_Toc419933866)

[4.4.1 Описание структуры файла Prog1.isaxml 43](#_Toc419933867)

[4.4.2 Детальное описание алгоритма работы 46](#_Toc419933868)

[4.4.3 Реализация программы 47](#_Toc419933869)

[4.5 Вывод 57](#_Toc419933870)

[5 Руководство пользователя программы Logical\_Formulas\_Converter 58](#_Toc419933871)

[5.1 Исходные файлы 58](#_Toc419933872)

[5.2 Главное окно программы 58](#_Toc419933873)

[5.3 Преобразование формул в функционый вид 60](#_Toc419933874)

[5.4 Преобразование в функциональные блочные диаграммы 61](#_Toc419933875)

[6 Экономический раздел 65](#_Toc419933876)

[6.1 Общая характеристика программного средства 65](#_Toc419933877)

[6.2 Исходные данные для расчета 66](#_Toc419933878)

[6.3 Методика обоснования цены 66](#_Toc419933879)

[6.3.1 Определение объема программного средства 66](#_Toc419933880)

[6.3.2 Расчет трудоемкости выполняемой работы 68](#_Toc419933881)

[6.3.3 Расчет основной заработной платы 68](#_Toc419933882)

[6.3.4 Расчет дополнительной заработной платы 69](#_Toc419933883)

[6.3.5 Расчет отчислений на социальные цели 69](#_Toc419933884)

[6.3.6 Расчет расходов на материалы 69](#_Toc419933885)

[6.3.7 Расчет расходов на оплату машинного времени 70](#_Toc419933886)

[6.3.8 Расчет прочих прямых затрат 70](#_Toc419933887)

[6.3.9 Расчет общепроизводственных и общехозяйственных расходов 70](#_Toc419933888)

[6.3.10 Расчет суммы расходов на разработку программного средства 71](#_Toc419933889)

[6.3.11 Расчет расходов на сопровождение и адаптацию 71](#_Toc419933890)

[6.3.12 Расчет общей суммы расходов 71](#_Toc419933891)

[6.3.13 Определение цены, оценка эффективности 71](#_Toc419933892)

[6.4 Выводы 72](#_Toc419933893)

[7 Мероприятия по охране труда и безопасности жизнедеятельности 74](#_Toc419933894)

[7.1 Особенности труда пользователя ПЭВМ 74](#_Toc419933895)

[7.2 Обеспечение санитарно-гигиенических условий при использовании ПЭВМ 76](#_Toc419933896)

[7.2.1 Освещение 76](#_Toc419933897)

[7.2.2 Параметры микроклимата 77](#_Toc419933898)

[7.2.3. Шум и вибрация 78](#_Toc419933899)

[7.2.4. Электромагнитное и ионизирующее излучения 79](#_Toc419933900)

[7.2.5 Эргономические требования к рабочему месту 80](#_Toc419933901)

[7.2.6 Режим труда 83](#_Toc419933902)

[7.3 Эргономика пользовательских интерфейсов 84](#_Toc419933903)

[7.3.1 Основные принципы проектирования интерфейсов 84](#_Toc419933904)

[7.7 Вывод 88](#_Toc419933905)

[Заключение 89](#_Toc419933906)

[Список использованных источников 90](#_Toc419933907)

[ПРИЛОЖЕНИЕ А. Структурная схема БУ БКУ 92](#_Toc419933908)

[ПРИЛОЖЕНИЕ Б. Преобразование блок-схемы алгоритма 93](#_Toc419933909)

[ПРИЛОЖЕНИЕ В. Алгоритм программы Gerakl to SFC Converter 94](#_Toc419933910)

[ПРИЛОЖЕНИЕ Г. Исходные блок-схемы алгоритмов 95](#_Toc419933911)

[ПРИЛОЖЕНИЕ Д. Визуальная программа на языке SFC 96](#_Toc419933912)

[ПРИЛОЖЕНИЕ Е 97](#_Toc419933913)

# Введение

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

*7*

*ДП 00.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консульт.*

*Акунович С.И.*

*Н. контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Введение*

*Лит.*

*Листов*

3

*БГТУ 84419015, 2015*

Автоматизация различных видов производства является важным направлением научно-технического развития общества. Автоматизация ведет к повышению производительности труда, освобождению человека из производственного процесса, к повышению качества продукции и к более полному удовлетворению потребностей общества.

Важнейшей проблемой автоматизации производственных процессов в различных отраслях промышленности и машиностроении является построение систем логического управления (СЛУ) [1].

Логическое управление заключается в подаче в требуемой очередности управляющих переключающих воздействий на исполнительные устройства технологического объекта. Управляющие воздействия формируются на основе данных о состоянии объекта (сигналов с датчиков), а также сигналов от цифровой информационной модели объекта, в которой отображается число деталей, их размещение в накопителях, число циклов работы оборудования, параметры настройки и т. д [1].

Таким образом, логическое управление обеспечивает согласованную работу механизмов и агрегатов.

Аппаратной основой СЛУ является логический контроллер, который может быть реализован двумя способами:

* с помощью «жесткой» логики, то есть логического устройства, преобразующего входные сигналы в выходные;
* на базе программируемого логического контроллера (ПЛК), то есть специализированного компьютера, используемого непосредственно в контуре управления[1].

СЛУ находят широкое применение не только на производстве, но и в сфере транспорта, в атомной энергетике, сетях связи, информационно-справочных системах, ракетно-космической отрасли, компьютерах, бытовой аппаратуре и т.д.

Логическое управление заключается в подаче в требуемой очередности управляющих переключающих воздействий на исполнительные устройства технологического объекта, обеспечивающих заданное функционирование объекта. Управляющие воздействия формируются на основе данных о состоянии объекта (сигналов с датчиков), а также сигнал от цифровой информационной модели объекта, в которой отображается число деталей, их размещение в накопителях, число циклов работы оборудования, параметры настройки и т. д [1].

*У*

*7*

Важное значение имеет проработка логики управления, позволяющая получить схему СЛУ технологическим процессом[1]. Лишь после четкого описания алгоритма, его проверки и отладки можно перейти к конструированию технической системы, реализующей этот алгоритм.

Ключевой задачей в решении проблемы логической отработки СЛУ является моделирование и проверка процессов ее функционирования на стадии проектирования, до начала программной или аппаратной реализации. Если в проекте СЛУ допущены и не выявлены ошибки, дальнейшая реализация может оказаться затруднительной.

В настоящее время разработан и используется мощный арсенал средств для логической отработки исключительно широкого разнообразия СЛУ методами компьютерного моделирования. Выбор наиболее адекватных из этих средств, их адаптация и доработка применительно к конкретному классу СЛУ являются сложной практической задачей.

Так как в дипломном проекте методы и средства моделирования алгоритмов управления излагаются на примере системы логического управления навигационного спутника, то кратко рассмотрим логику его функционирования. В процессе функционирования спутник последовательно находится в следующих режимах работы [2]: выведение на орбиту, ожидание после выведения на орбиту, начальная ориентация, дежурное функционирование, ввод штатного функционирования, штатное функционирование.

После выведения спутников на заданную орбиту по команде, выдаваемой разгонным блоком непосредственно после остановки двигателей, включаются электропитание некоммутируемых шин питания и дежурные схемы спутников. Через 15 с после этого система управления разгонным блоком выдает команду на отделение спутников. Данный процесс контролируется с помощью телеметрической системы разгонного блока [2].

Перевод спутника в режим начальной ориентации происходит в зоне радиовидимости наземных средств в следующей последовательности:

* раскрытие солнечных батарей и штанги магнитометра;
* успокоение спутника;
* начальная ориентация спутника на Солнце [2].

В режиме успокоения осуществляется торможение вращательного движения спутника с помощью двигательной установки с одновременным разворотом панелей солнечных батарей для выставки крыльев активной поверхностью в одну сторону и установкой их в плоскости, совпадающей с продольной осью спутника[2].

В режиме начальной солнечной ориентации осуществляется разворот спутника вокруг продольной оси и разворот вокруг поперечной оси до совмещения нормали солнечной батареи с направлением на Солнце [3].

Режим начальной ориентации на Землю проводится в зоне радиовидимости. При этом обеспечивается торможение вращения спутника и его разворот вокруг направления, ориентированного на Солнце до попадания Земли в поле зрения прибора ориентации на Землю. После этого включается режим одновременного слежения за Солнцем и Землей и спутник переводится в режим дежурного функционирования [3].

Вывод из целевого использования спутника осуществляется при возникновении аварийных ситуаций, приводящих к кратковременному нарушению целевого функционирования спутника, а также после возникновения необратимых отказов, исключающих возможность целевого применения спутника [3].

Аварийная ситуация на борту спутника регистрируется телеметрической аппаратурой и запоминается с последующей передачей по командной радиолинии сигнала «вызов НКУ» при вхождении спутника в зону радиовидимости наземных средств [3].

При возникновении аварийных ситуаций на спутнике необратимого типа принимается решение о прекращении его обслуживания.

Целью данного дипломного проекта является создание программного средства, предназначенного для преобразования логических формул в визуальную программу на языке FBD [4] в среде разработки ISaGRAF 6.1 [5].

# 1 Аналитический обзор литературы

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

*10*

*ДП 01.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консультант*

*Акунович С.И.*

*Н. Контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Аналитический обзор языков описания алгоритмов логического управления*

*Лит.*

*Листов*

22

*БГТУ 84419015, 2015*

При создании ответственных технологических объектов одной из важнейших составляющих процесса их проектирования является комплексная автоматизация. Различная природа физических процессов, протекающих в объектах, сложный характер взаимодействия между ними и управляющими системами обуславливает трудности алгоритмизации и программирования задач управления. Вопросы унификации аппаратного обеспечения в настоящее время решаются достаточно успешно на основе микропроцессорных и микроконтроллерных комплектов, программируемых логических контроллеров и промышленных компьютеров [6].

Однако при формировании общих подходов к созданию алгоритмического и программного обеспечения возникают трудности, связанные с необходимостью достижения их наглядности, структурированности, наблюдаемости и управляемости. Эти трудности усугубляются тем, что при создании различных систем управления обычно используются и различные технологии алгоритмизации и программирования [6].

Спектр таких технологий широк: от технологий на основе алгоритмических языков высокого уровня для промышленных компьютеров до технологий на основе специализированных языков для программируемых логических контроллеров. Очевидно, что алгоритмизация и программирование систем управления техническими средствами, должны основываться на единой методологии, позволяющей строить, читать, проверять, верифицировать алгоритмы и программы. До настоящего времени алгоритмы логического (основанные на истинности и ложности) управления задаются Проектантом (Технологом) в словесной форме, и поэтому они могут быть противоречивыми и неполными. Это вносит дополнительные трудности в процесс алгоритмизации и программирования управляющих устройств, так как каждый участник разработки понимает в силу неоднозначности и словесного описания алгоритмы логического управления по-своему. По этой причине даже самый «точный» словесный алгоритм может удовлетворять каждого из участников разработки в отдельности, но, в конечном счете, являться некорректным [6].

Ситуация усугубляется при создании систем управления, ядро которых представляет собой систему взаимосвязанных управляющих устройств. При этом с одной стороны, совокупность таких устройств (вследствие большой размерности) становится плохо обозримой, наблюдаемой и управляемой, а с другой – плохо понимаемой Заказчиком и Проектантом (Технологом). Это во многом определяется тем, что обычно автоматизация каждого комплекса технических средств выполняется самостоятельным коллективом Разработчиков

*У*

10

и Программистов, которые используют специфическую технологию алгоритмизации и программирования. В настоящее время в практике проектирования систем управления, наиболее часто употребляемыми языками спецификаций алгоритмов логического управления являются язык функциональных схем и язык блок-схем алгоритмов, называемых обычно схемами алгоритмов. Указанные языки удобны, как правило, только для одних участников разработки, но крайне неудобны для других участников. Так, например, наиболее часто при согласовании алгоритмов управления судовыми техническими средствами применяется язык функциональных схем, который достаточно хорошо понятен только Разработчику, так как такой язык для последовательных алгоритмов, реализуемых схемами с обратными связями, описывает их структуру (статику), но не отражает в явном виде их поведение (динамику) [6].

Чтобы алгоритм выполнил свое предназначение, его необходимо строить по определенным правилам. Поэтому нужно говорить не о свойствах алгоритма, а о правилах построения алгоритма, или о требованиях, предъявляемых к алгоритму.

Первое правило – при построении алгоритма, прежде всего, необходимо задать множество объектов, с которыми будет работать алгоритм. Формализованное (закодированное) представление этих объектов носит название данных. Алгоритм приступает к работе с некоторым набором данных, которые называются входными, и в результате своей работы выдает данные, которые называются выходными. Таким образом, алгоритм преобразует входные данные в выходные. Это правило позволяет сразу отделить алгоритмы от «методов» и «способов». Пока мы не имеем формализованных входных данных, мы не можем построить алгоритм [7].

Второе правило – для работы алгоритма требуется память. В памяти размешаются входные данные, с которыми алгоритм начинает работать, промежуточные данные и выходные данные, которые являются результатом работы алгоритма. Память является дискретной, т.е. состоящей из отдельных ячеек. Поименованная ячейка памяти носит название переменной. В теории алгоритмов размеры памяти не ограничиваются, т.е. считается, что мы можем предоставить алгоритму любой необходимый для работы объем памяти. В школьной «теории алгоритмов» эти два правила не рассматриваются [7].

Третье правило – дискретность. Алгоритм строится из отдельных шагов (действий, операций, команд). Множество шагов, из которых составлен алгоритм, конечно [7].

Четвертое правило – детерменированность. После каждого шага необходимо указывать, какой шаг выполняется следующим, либо давать команду остановки [7].

 Пятое правило – сходимость (результативность). Алгоритм должен завершать работу после конечного числа шагов. При этом необходимо указать, что считать результатом работы алгоритма [7].

Можно выделить два уровня языков – языки алгоритмизации или спецификации (языки общения) и языки программирования (языки реализации). Языки этих классов могут, как совпадать (при наличии транслятора с языка алгоритмизации), так и различаться между собой. Так, например, при аппаратной реализации систем в базисе релейно-контактных схем в качестве языка алгоритмизации применялись функциональные схемы, а в качестве языка реализации – собственно релейно-контактные схемы. Однако плохая «читаемость», как функциональных схем, так и релейно-контактных схем, привела к необходимости использования промежуточного языка – функционально-принципиальных схем, которые в релейно-контактном виде отражают лишь алгоритм управления и не содержат другой информации, характерной для принципиальных схем (например, обозначений разъемов, гасящих сопротивлений, устройств контроля и т.д.). Однако и эти схемы, весьма удобные для представления комбинационных схем (автоматов без памяти), трудно читаются для автоматов с памятью, так как они обычно реализуют, но не отображают в своей структуре динамику переходов и смену состояний синтезируемого автомата [6].

Отметим, что, например, при программной реализации на базе аппаратуры «Selma-2» фирмы «ABB Stromberg» (Финляндия) в качестве, как языка алгоритмизации, так и языка программирования, применяются функциональные схемы. Для программируемых логических контроллеров «Autolog» фирмы «FF – Automation» (Финляндия) в качестве языка программирования используется язык программирования ALPro, в то время как язык алгоритмизации не определен. Последний не определен также и для многих других типов программируемых логических контроллеров, таких как, например, контроллеры «Melsec» фирмы «Mitsubishi Electric» (Япония), языками программирования, которых являются язык инструкций и язык лестничных схем. Последний из языков является языком релейно-контактных схем, который дополнен большим числом вычислительных операций [6].

В настоящее время в качестве языков алгоритмизации в системах логического управления наиболее часто используются функциональные схемы и схемы алгоритмов. При этом в качестве языков программирования в зависимости от типа управляющего вычислительного устройства применяются три разновидности языков: алгоритмические языки высокого уровня (например, СИ, «Паскаль», ПЛ/М, «Форт»), алгоритмические языки низкого уровня (языки инструкций, ассемблеры) и специализированные языки (функциональные и лестничные схемы) [6].

При логическом управлении могут использоваться модификации традиционных (классических) путей формализации, как процедур управления, так и описаний объектов управления, что отличает этот класс задач, например, от ситуационного управления.

Для ответственных технологических объектов системы логического и ситуационного управления могут применяться совместно.

## 1.1 Программируемый логический контроллер для систем автоматизации

С появлением мощных и дешевых микроконтроллеров в 1972 г. рынок ПЛК начал расти экспоненциально и за период с 1978 по 1990 год увеличился с 80 млн. долл. до 1 млрд. долл. и к 2002 г. составил 1,4 млрд. долл. В настоящее время мировой рынок ПЛК продолжает расти, хотя и гораздо меньшими темпами. Однако где события изменяются очень быстро в связи с возрождением экономики, появлением сильных отечественных производителей и системных интеграторов, а также огромными инвестициями международных корпораций [8].

ПЛК используются практически во всех сферах человеческой деятельности для автоматизации технологических процессов, в системах противоаварийной защиты и сигнализации, в станках с ЧПУ, для управления дорожным движением, в системах жизнеобеспечения зданий, для сбора и архивирования данных, в системах охраны, в медицинском оборудовании, для управления роботами, в системах связи, при постановке физического эксперимента, для управления космическими кораблями, для автоматизации испытаний продукции и т. д. Тем не менее, до сих пор остается много отраслей экономики, куда контроллерная автоматизация только начинает проникать [8].

Контроллеры используются не только как автономные средства локального управления технологическими установками, но и в составе широкомасштабных систем автоматизированного управления целыми предприятиями [8].

В настоящее время на рынке преобладают контроллеры иностранных фирм: Siemens, Mitsubishi, ABB, Schneider Electric, GE Fanuc, однако с течением времени увеличивается доля рынка, занятая отечественной продукцией Российских фирм (НИЛ АП, Текон, Фаствел, ДЭП, Овен, Элемер, Эмикон и др.), что соответствует общемировой тенденции, когда в большинстве стран отечественные фирмы занимают большую долю рынка, чем иностранные [8].

Широкому распространению ПЛК в большой степени способствует рост компьютерной грамотности населения, спецкурсы в ВУЗах, множество курсов повышения квалификации, проводимых ведущими системными интеграторами [8].

Жесткие ограничения на стоимость и огромное разнообразие целей автоматизации привели к невозможности создания универсального ПЛК, как это случилось с офисными компьютерами. Область автоматизации выдвигает множество задач, в соответствии с которыми развивается и рынок, содержащий сотни непохожих друг на друга контроллеров, различающихся десятками параметров. Каждый производитель выпускает несколько типов ПЛК разной мощности и стоимости, чтобы увеличить прибыль за счет сегментирования рынка [8].

Выбор оптимального для конкретной задачи контроллера основывается обычно на соответствии функциональных характеристик контроллера решаемой задаче при условии минимальной его стоимости. Учитываются также другие важные характеристики (температурный диапазон, надежность, бренд изготовителя, наличие разрешений Ростехнадзора, сертификатов и т. п.) [8].

Несмотря на огромное разнообразие контроллеров, в их развитии заметны следующие общие тенденции:

* уменьшение габаритов;
* расширение функциональных возможностей;
* увеличение количества поддерживаемых интерфейсов и сетей;
* использование идеологии «открытых систем»;
* использование языков программирования стандарта МЭК 61131-3;
* снижение цены [8].

Еще одной тенденцией является появление в контроллерах признаков компьютера (наличие мыши, клавиатуры, монитора, ОС Windows, возможности подключения жесткого диска), а в компьютерах - признаков контроллера (расширенный температурный диапазон, электронный диск, защита от пыли и влаги, крепление на DIN-рейку, наличие сторожевого таймера, увеличенное количество коммуникационных портов, использование ОС жесткого реального времени, функции самотестирования и диагностики, контроль целостности прикладной программы). Появились компьютеры в конструктивах для жестких условий эксплуатации. Аппаратные различия между компьютером и контроллером постепенно исчезают. Основными отличительными признаками контроллера остаются его назначение и наличие технологического языка программирования [8].

Для классификации огромного разнообразия существующих в настоящее время контроллеров рассмотрим их существенные различия.

Основным показателем ПЛК является количество каналов ввода-вывода. По этому признаку ПЛК делятся на следующие группы:

* нано-ПЛК (менее 16 каналов);
* микро-ПЛК (более 16, до 100 каналов);
* средние (более 100, до 500 каналов);
* большие (более 500 каналов) [8].

По расположению модулей ввода-вывода ПЛК бывают:

* моноблочными, в них устройство ввода-вывода не может быть удалено из контроллера или заменено на другое. Конструктивно контроллер представляет собой единое целое с устройствами ввода-вывода (например, одноплатный контроллер). Моноблочный контроллер может иметь, например, 16 каналов дискретного ввода и 8 каналов релейного вывода;
* модульные, состоящие из общей корзины (шасси), в которой располагаются модуль центрального процессора и сменные модули ввода-вывода. Состав модулей выбирается пользователем в зависимости от решаемой задачи. Типовое количество слотов для сменных модулей - от 8 до 32;
* распределенные (с удаленными модулями ввода-вывода), в которых модули ввода-вывода выполнены в отдельных корпусах, соединяются с модулем контроллера по сети (обычно на основе интерфейса RS-485) и могут быть расположены на расстоянии до 1,2 км от процессорного модуля.

Часто перечисленные конструктивные типы контроллеров комбинируются, например, моноблочный контроллер может иметь несколько съемных плат; моноблочный и модульный контроллеры могут быть дополнены удаленными модулями ввода-вывода, чтобы увеличить общее количество каналов.

Многие контроллеры имеют набор сменных процессорных плат разной производительности. Это позволяет расширить круг потенциальных пользователей системы без изменения ее конструктива [8].

По конструктивному исполнению и способу крепления контроллеры делятся на:

* панельные (для монтажа на панель или дверцу шкафа);
* для монтажа на DIN-рейку внутри шкафа;
* для крепления на стене;
* стоечные (для монтажа в стойке);
* бескорпусные для применения в специализированных конструктивах производителей оборудования [8].

По области применения контроллеры делятся на следующие типы:

* универсальные общепромышленные;
* для управления роботами;
* для управления позиционированием и перемещением;
* коммуникационные;
* ПИД-контроллеры;
* специализированные [8].

По способу программирования контроллеры бывают:

* программируемые с лицевой панели контроллера;
* программируемые переносным программатором;
* программируемые с помощью дисплея, мыши и клавиатуры;
* программируемые с помощью персонального компьютера.

Контроллеры могут программироваться на следующих языках:

* на классических алгоритмических языках (C, С#, Visual Basic);
* на языках МЭК 61131-3 [8]..

Контроллеры могут содержать в своем составе модули ввода-вывода или не содержать их. Примерами контроллеров без модулей ввода-вывода являются коммуникационные контроллеры, которые выполняют функцию межсетевого шлюза, или контроллеры, получающие данные от контроллеров нижнего уровня иерархии АСУ ТП[8].

### 1.1.1 Стандартные языки программирования ПЛК

Главная задача ПЛК – это выполнение прикладной программы управления технологическим процессом. Очевидно, что незапрограммированный контроллер – это всего лишь пустая железяка, не приносящая никакой пользы человечеству [9].

Современные средства разработки чрезвычайно функциональны и предлагают разработчику множество возможностей:

* разнообразные программные библиотеки, функциональные блоки, готовые процедуры и шаблоны. Использование предподготовленных компонентов сильно ускоряет процесс разработки программного обеспечения для ПЛК;
* инструменты для отладки, тестирования и симуляции прикладной программы. Последние позволяют выполнять программу ПЛК на персональном компьютере без загрузки в реальный контроллер;
* инструменты для автоматизированного документирования разработанной программы в соответствие с принятыми стандартами [9].

Существует международный стандарт IEC 61131, разработанный Международной Электротехнической Комиссией (МЭК, IEC) и состоящий из восьми частей. Наиболее интересной является третья часть, IEC 61131-3, описывающая языки программирования ПЛК. Первоначальной целью стандарта IEC 61131-3 была унификация языков программирования ПЛК и предоставление разработчикам ряда аппаратно-независимых языков, что, по замыслу создателей стандарта, обеспечило бы простую переносимость программ между различными аппаратными платформами и снимало бы необходимость изучения новых языков и средств программирования при переходе разработчика на новый ПЛК [9].

К сожалению, цели в полном объеме достигнуты не были. Каждый производитель ПЛК сопровождает свой продукт собственной средой программирования, которая, как правило, не совместима с другими, да и о кроссплатформенности программного кода можно забыть. Тем не менее, в части описания языков программирования стандарт IEC 61131 остается чрезвычайно актуальным и является ориентиром для большинства разработчиков ПЛК [9].

Языки, используемые, для программирования промышленных контроллеров:

1. Язык LD (LAD, Ladder) является графическим языком разработки, программа на котором представляет собой аналог релейной схемы. По идеи авторов стандарта, такая форма представления программы облегчит переход инженеров из области релейной автоматики на ПЛК.

К недостаткам данного языка можно отнести то, что по мере увеличения количества «реле» в схеме она становится сложнее для интерпретации, анализа и откладки. Еще один недостаток языка LD заключается в следующем: язык, построенный по аналогии с релейными схемами, может быть эффективно использован только для описания процессов, имеющих дискретный (двоичный) характер; для обработки «непрерывных» процессов (с множеством аналоговых переменных) такой подход теряет смысл;

1. Язык FBD (Functional Block Diagram, Диаграмма Функциональных Блоков) является языком графического программирования, так же, как и LD, использующий аналогию с электрической (электронной) схемой. Программа на языке FBD представляет собой совокупность функциональных блоков (functional flocks, FBs), входа и выхода которых соединены линиями связи (connections). Эти связи, соединяющие выхода одних блоков с входами других, являются по сути дела переменными программы и служат для пересылки данных между блоками. Каждый блок представляет собой математическую операцию (сложение, умножение, триггер, логическое «или» и т.д.) и может иметь, в общем случае, произвольное количество входов и выходов. Начальные значения переменных задаются с помощью специальных блоков – входов или констант, выходные цепи могут быть связаны либо с физическими выходами контроллера, либо с глобальными переменными программы.

Практика показывает, что FBD является наиболее распространенным языком стандарта IEC. Графическая форма представления алгоритма, простота в использовании, повторное использование функциональных диаграмм и библиотеки функциональных блоков делают язык FBD незаменимым при разработке программного обеспечения ПЛК. Вместе с тем, нельзя не заметить и некоторые недостатки FBD. Хотя FBD обеспечивает легкое представление функций обработки как «непрерывных» сигналов, в частности, функций регулирования, так и логических функций, в нем неудобным и неочевидным образом реализуются те участки программы, которые было бы удобно представить в виде конечного автомата;

1. Язык последовательных функциональных схем SFC (Sequential Function Chart), использующийся совместно с другими языками (обычно с ST и IL), является графическим языком, в котором программа описывается в виде схематической последовательности шагов, объединенных переходами. Язык SFC построен по принципу, близкому к концепции конечного автомата, что делает его одним из самых мощных языков программирования стандарта IEC 61131-3.

Наиболее простым и естественным образом на языке SFC описываются технологические процессы, состоящие из последовательно выполняемых шагов, с возможностью описания нескольких параллельно выполняющихся процессов, для чего в языке имеются специальные символы разветвления и слияния потоков (дивергенции и конвергенции, в терминах стандарта IEC 61131-3).

Шаги последовательности располагаются вертикально сверху вниз. На каждом шаге выполняется определенный перечень действий (операций). При этом для описания самой операции используются другие языки программирования, такие как IL или ST.

Действия (операции) в шагах имеют специальные классификаторы, определяющие способ их выполнения внутри шага: циклическое выполнение, однократное выполнение, однократное выполнение при входе в шаг и т.д. В сумме таких классификаторов насчитывается девять, причем среди них есть, например, классификаторы так называемых сохраняемых и отложенных действий, заставляющие действие выполняться даже после выхода программы из шага.

После того, как шаг выполнен, управление передается следующему за ним шагу. Переход между шагами может быть условным и безусловным. Условный переход требует выполнение определенного логического условия для передачи управления на следующий шаг; пока это условие не выполнено программа будет оставаться внутри текущего шага, даже если все операции внутри шага уже выполнены. Безусловный переход происходит всегда после полного выполнения всех операций на данном шаге. С помощью переходов можно осуществлять разделение и слияние ветвей последовательности, организовать параллельную обработку нескольких ветвей или заставить одну выполненную ветвь ждать завершения другой.

Как и любому другому языку, SFC свойственны некоторые недостатки. Хотя SFC может быть использован для моделирования конечных автоматов, его программная модель не совсем удобна для этого. Это связано с тем, что текущее состояние программы определяется не переменной состояния, а набором флагов активности каждого шага, в связи с чем при недостаточном контроле со стороны программиста могут оказаться одновременно активными несколько шагов, не находящихся в параллельных потоках.

Еще одно неудобство языка связано с тем, что шаги графически располагаются сверху вниз, и переход, идущий в обратном направлении, изображается в неявной форме, в виде стрелки с номером состояния, в которое осуществляется переход;

1. Язык ST (Structured Text, Структурированный Текст) представляет собой язык высокого уровня, имеющий черты языков Pascal и Basic. Данный язык имеет те же недостатки, что и IL, однако они выражены в меньшей степени.

С помощью ST можно легко реализовывать арифметические и логические операции (в том числе, побитовые), безусловные и условные переходы, циклические вычисления; возможно использование как библиотечных, так и пользовательских функций. Язык также интерпретирует более 16 типов данных.

Язык ST может быть освоен технологом за короткий срок, однако текстовая форма представления программ служит сдерживающим фактором при разработке сложных систем, так как не дает наглядного представления ни о структуре программы, ни о происходящих в ней процессах;

1. Язык IL (Instruction List, Список Команд) представляет собой ассемблероподобный язык, достаточно несложный по замыслу авторов стандарта, для его практического применения в задачах промышленной автоматизации пользователем, не имеющим, с одной стороны, профессиональной подготовки в области программирования, с другой стороны, являющимся специалистом в той или иной области производства. Однако, как показывает практика, такой подход себя не оправдывает.

Ввиду своей ненаглядности, IL практически не используется для программирования комплексных алгоритмов автоматизированного управления, но часто применяется для кодирования отдельных функциональных блоков, из которых впоследствии складываются схемы FBD или CFC. При этом IL позволяет достичь высокой оптимальности кода: программные блоки, написанные на IL, имеют высокую скорость исполнения и наименее требовательны к ресурсам контроллера.

Язык IL имеет все недостатки, которые присущи другим низкоуровневым языкам программирования: сложность и высокую трудоемкость программирования, трудность модификации написанных на нем программ, малую степень «видимого» соответствия исходного текста программы и решаемой задачи.

Многие производители инструментальных средств, опирающиеся на стандарт IEC, не ограничиваются поддержкой рассмотренных выше пяти языков стандарта. Можно выделить, как минимум, еще один язык визуального программирования, который довольно популярен среди разработчиков;

1. Язык CFC (Continuous Flow Chart) – еще один высокоуровневый язык визуального программирования. По сути, CFC – это дальнейшее развития языка FBD. Этот язык был специально создан для проектирования систем управления непрерывными технологическими процессами.

Проектирование сводится к выбору из библиотек готовых функциональных блоков, их позиционированию на экране, установке соединений между их входами и выходами, а также настройке параметров выбранных блоков. В отличие от FBD, функциональные блоки языка CFC выполняют не только простые математические операции, а ориентированы на управление целыми технологическими единицами. Так в типовой библиотеке CFC блоков находятся комплексные функциональные блоки, реализующие управление клапанами, моторами, насосами; блоки, генерирующие аварийные сигнализации; блоки PID-регулирования и т.д. Вместе с тем доступны и стандартные блоки FBD. Унаследовав от FBD саму концепцию программирования, язык CFC в наибольшей степени ориентирован на сам технологический процесс, позволяя разработчику абстрагироваться от сложного математического аппарата.

CFC прост в освоении, и при этом позволяет разрабатывать сложнейшие алгоритмы автоматизированного управления без каких-либо специфических знаний других языков программирования [9].

## 1.2 Краткие сведения о системе ISaGRAF

### 1.2.1 Основные свойства системы

Контроллеры являются семейством программно- и системносовместимых приборов, обеспечивающих:

* переносимость технологических программ между контроллерами семейства;
* возможность работы в одной контроллерной сети и взаимодействия по единым сетевым протоколам;
* связь с верхним уровнем с помощью единого ОРС-сервера [10].

Эти возможности обеспечиваются использованием в контроллерах семейства популярной системы программирования ISaGRAF, соответствующей международному стандарту открытых систем IEC 1131-3, что обеспечивает совместимость контроллеров семейства с контроллерами других фирм, использующих ISaGRAF [10].

ISaGRAF представлен в виде двух частей: набора средств разработки ISaGRAF Workbench и исполняемого на контроллере ядра-интерпретатора ISaGRAF Target. Набор средств разработки исполняется на компьютере проектировщика, например, компьютере типа IBM PC, и состоит из редактора, отладчика и препроцессора, который подготавливает описанный проектировщиком алгоритм к формату, «понятному» ядру-интерпретатору. Этот набор имеет современный пользовательский интерфейс, позволяет тестировать алгоритм в режиме эмуляции и получать листинг алгоритма на языках его описания. Ядро-интерпретатор ISaGRAF Target размещается в контроллере в качестве его резидентного программного обеспечения [10].

После создания пользовательская программа загружается в контроллер для исполнения. Ядро-интерпретатор, как следует уже из его названия, транслирует пользовательский алгоритм во время исполнения. Это позволяет сконцентрировать машинозависимый код и таким образом снизить накладные расходы при переходе на другой контроллер [10].

В ISaGRAF представлены все шесть языков стандарта IEC 1131-3: четыре графических языка (SFC, FC, FBD, LD) и два текстовых (ST и IL). Имеется возможность расширения набора языковых средств за счет языка «C», т.е. пользователь может создать произвольную функцию на языке «C» и скомпоновать ее с ядром-интерпретатором. CJ International гарантирует обеспечение переносимости ядра-интерпретатора на любую платформу, имеющую «C»-компилятор, т.е. ядро-интерпретатор написан на «C» [10].

* + 1. **Спектр возможностей**

В ISaGRAF заложена методология структурного программирования, которая дает возможность пользователю описать автоматизируемый процесс в наиболее легкой и понятной форме. Интерфейс с пользователем системы ISaGRAF соответствует международному стандарту GUI (Graphical User Interface), включающему многооконный режим работы, работу с мышью и т.п [11].

ISaGRAF позволяет широко смешивать программы и процедуры, написанные на разных языках, а также вставлять кодовые последовательности из одного языка в коды, написанные на другом языке [11].

Также ISaGRAF обладает следующими достоинствами:

* наличие мощного отладчика, позволяющего во время работы прикладной задачи просматривать состояние программного кода, переменных, программ и многое другое;
* поддержка основных функций протокола MODBUS (RTU, SLAVE);
* наличие дополнительных интерактивных редакторов для описания переменных, определений и конфигураций ввода/вывода;
* встроенные средства контроля над внесением изменений в программный код ISaGRAF-приложения и печати отчетов по разработанному проекту с большой степенью детализации, включая печать таблиц перекрестных ссылок для программ и отдельных переменных;
* открытость системы для доступа извне к внутренним структурам данных прикладной ISaGRAF-задачи, возможностью разработки драйверов на модули ввода/вывода самим пользователем и портации ISaGRAF-ядра под любую аппаратно-программную платформу (зависит от заказанной версии системы);
* полное документирование системы разработки и языков программирования (в том числе и на русском языке) [11].
  + 1. **Средства технологического программирования**

Средства технологического программирования ЦП используют систему разработки ISaGRAF Workbench в соответствии со стандартом МЭК 1131-3, расширенную новыми алгоритмами, в том числе из библиотеки контроллера Р-130, а также средствами конфигурирования пульта технолога-оператора. Система содержит шесть типов технологических языков, грамматика которых многократно описана в литературе:

* язык последовательных функциональных схем SFC (происхождение – язык Grafcet фирмы Telemechanique-Groupe Schneider);
* язык потоковых диаграмм FC (происхождение – логические блок-схемы алгоритмов);
* язык функциональных блоков FBD, расширенный библиотекой алгоритмов Р-130 (происхождение – функциональные схемы электронный устройств);
* язык релейных диаграмм LD (происхождение – различные варианты языка релейно-контактных схем фирм Allen-Bradley, AEG Schneider Automation, GE-Fanuc, Siemens);
* язык структурированного текста ST (происхождение – язык Grafcet фирмы Telemechanique - Groupe Schneider);
* язык инструкций IL (происхождение – язык STEP 5 фирмы Siemens) [10].

Привычный для многих потребителей язык функциональных блоков FBD в системе ISaGRAF построен на библиотеке элементарных арифметических, логических и динамических функций и не содержит сложных алгоритмов, как, например, в контроллере Р-130. Некоторые сложные алгоритмы можно «набрать» из элементарных функций средствами самого ISaGRAF, расширив таким образом стандартную библиотеку, но это требует дополнительных усилий, снижает скорость выполнения и значительно увеличивает размеры технологической программы. Кроме этого, в системе ISaGRAF не предусмотрены процедуры сохранения значений внутренних переменных функциональных блоков FBD в энергонезависимой памяти для безударного рестарта контроллера и отсутствуют процедуры обратного счета для безударного перехода из ручного режима в автоматический [10].

С целью устранения этих недостатков и адаптации языка FBD к российским традициям применения контроллеров его библиотека расширена следующими классами алгоритмов, в том числе функционально соответствующим алгоритмам библиотеки контроллера Р-130:

* алгоритмы регулирования с поддержкой процедуры обратного счета (аналоговый и импульсный регуляторы, задание, задание локальное, ручное управление);
* алгоритмы динамических преобразований (интегрирование, дифференцирование, динамическая балансировка, фильтрация и т.п.);
* алгоритмы статических преобразований (суммирование, масштабирование, кусочно - линейная функция, корень квадратный, скользящее среднее и т.п.);
* алгоритмы аналого-дискретных преобразований (ШИМ - модулятор, запрет изменения, запрет знака, слежение, запоминание и т.п.);
* логические операции (мажорирование, триггеры, регистры, выделение фронта и т.п.); е) алгоритмы дискретного управления (таймер, счетчик, мультивибратор, одновибратор и т.п.);
* алгоритмы архивации данных, сетевого обмена;
* прочие алгоритмы [10].

В первом классе в алгоритмы регулирования введена поддержка процедуры обратного счета специально для организации простых и каскадных регуляторов с возможностью ручного управления и безударного перехода в автоматический режим, поскольку ISaGRAF не позволяет реализовать процедуру обратного счета стандартными средствами [10].

Следует иметь в виду, что указанные алгоритмы являются лишь функциональными, а не грамматическими, аналогами алгоритмов Р-130. Реализация этих алгоритмов имеет следующие особенности, обусловленные системой ISaGRAF:

* отсутствие модификаторов, алгоритмы имеют фиксированное число входов и выходов;
* отсутствие масштаба времени, все временные параметры (постоянные времени, длительности и т.п.) задаются в секундах вещественным числом;
* алгоблоки и их входы-входы не имеют порядковых номеров, поэтому вход (выход) алгоблока, к которому необходимо обеспечить доступ со стороны другого языка ISaGRAF или верхнего уровня, идентифицируется переменной ISaGRAFа, а не номером алгоблока и входа (выхода) [10].

### Основные языки описания алгоритмов логического управления

В системах логического управления для описания алгоритмов логического управления используются булевы функции и системы булевых функций [6], задаваемые в форме таблиц истинности для полностью определенных функций и таблиц решений для не полностью определенных функций. При этом таблицы истинности, описывающие автоматы с памятью, носят название кодированных таблиц переходов или кодированных таблиц переходов и выходов.

Применение таблиц истинности ограничивается задачами небольшой размерности, а использование таблиц решений – в основном автоматами без памяти – комбинационными схемами. Табличное задание автоматов с памятью ненаглядно.

Аналитической формой представления булевых функций являются булевы формулы и системы булевых формул, которые позволяют описывать как автоматы с памятью, так и автоматы без памяти большой размерности [6]. Системы булевых формул могут быть изоморфно реализованы лестничными или функциональными схемами.

Иногда используются также и другие аналитические формы представления булевых функций, например, пороговые, спектральные или арифметические. Основным ограничением на применение систем булевых формул для автоматов с памятью является их низкая наглядность.

К достоинствам функциональных схем [6] при их использовании в качестве языка алгоритмизации относятся традиционность и однозначность описания, в том числе и параллельных процессов, а к недостаткам:

* отсутствие указания значений выходных и внутренних переменных в схеме;
* применение в большинстве случаев двоичных внутренних переменных, запоминаемых в триггерах, реализуемых виртуально, которые позволяют работать с многозначными переменными;
* трудоемкость их чтения с целью получения исчерпывающего представления о реализованном в них последовательных процессов;
* проблема выбора тестов для их полной проверки;
* сложность гарантированного внесения изменений.

Необходимо отметить, что чтение функциональных схем представляет собой вычисления по их отдельным цепям с целью определения значений выходных переменных при различных наборах входных переменных. В этой ситуации при наличии даже сравнительно небольшого числа входов по функциональной схеме весьма трудно определить какие воздействия влияют на тот или иной переход в ней. В схемах этого класса достаточно сложно составить целостное представление о поведении даже сравнительно небольшого фрагмента схемы, содержащего триггера и обратные связи. Так, например, при трех взаимосвязанных триггерах в схеме, непосредственно по ней (без вычислений) весьма трудно определить, какое число состояний эта схема реализует, так как с помощью указанного числа триггеров может быть закодировано от трех до восьми состояний.

При этом необходимо отметить, что использование в качестве тестов соотношений «вход-выход», обеспечивающих полноту проверки для схем без памяти, не решает проблему определения всех функциональных возможностей для схем с памятью, так как в этом случае необходимо проверять также и правильность порядка изменений выходных переменных. Однако, именно соотношения «вход-выход» и применяются при создании методик проверки на функционирование большинства систем логического управления. Это, как отмечалось выше, не обеспечивает качественной их проверки, так как такие соотношения не позволяют анализировать все имеющиеся в схеме переходы между состояниями. Более того, число состояний и переходов в схемах обычно не известно, так как они часто строятся эвристически без использования понятия «состояние».

Функциональные схемы при их применении в качестве языка программирования обладают всеми достоинствами декларативных языков функционального программирования, основным из которых является функциональность (прозрачность по ссылкам). При этом каждое выражение определяет единственную величину, а все ссылки на нее эквивалентны самой этой величине, и тот факт, что на выражение можно ссылаться из другой части программы, никак не влияет на величину рассматриваемого выражения. Это свойство определяет различие между математическими функциями и функциями, которые можно написать на процедурных языках программирования таких, например, как Паскаль, позволяющих функциям ссылаться на глобальные данные и применять «разрушающее» присваивание. Такое присваивание может привести к побочным эффектам, например, к изменению значения функции при повторном ее вызове даже без изменения значений аргументов. Это приводит к тому, что функцию трудно использовать, так как для того, чтобы определить, какая величина получится при ее вычислении, необходимо рассмотреть текущую величину глобальных данных, что, в свою очередь, требует изучения предыстории вычислений для определения того, что порождает эту величину в каждый «момент времени».

При определенных условиях (переобозначениях выходных переменных) в системах булевых формул, по которым функциональные схемы могут строиться, даже для автоматов с памятью удается обеспечить и другое достоинство декларативных языков – независимость результатов от порядка вычисления формул.

Достоинство временных диаграммы и циклограммм таких форм представления алгоритмов состоит в изображении динамики процессов, а их недостаток – в практической невозможности отражения всех допустимых значений выходных (а тем, более внутренних) переменных при всех возможных изменениях значений входных переменных даже для задач сравнительно небольшой размерности [6]. Поэтому на практике такие диаграммы строятся в основном для описания «основного» режима, а алгоритм в целом отражается лишь в программе, которая по указанной причине строится по таким диаграммам во многом неформально.

К достоинствам схем алгоритмов [6] при их использовании в качестве языка алгоритмизации относится возможность отражения в явном виде последовательностей изменений значений входных переменных и реакций на эти изменения, представляемых в виде значений выходных переменных, в том числе и вычисляемых параллельно. При наличии двоичных значений переменных, записываемых в явном виде в операторных вершинах схем алгоритмов, резко упрощается понимание последних по сравнению с функциональными схемами.

К недостаткам схем алгоритмов относятся:

* отсутствие требований к тому, что должна отражать схема: алгоритм управления; алгоритм реализации алгоритма управления; алгоритм, учитывающий свойства управляющих конструкций используемого языка программирования; алгоритм выполнения программы;
* отсутствие требований к их организации (за исключением структурирования), обеспечивающих простоту «чтения»;
* необходимость в общем случае их многократных преобразований для учета свойств управляющих конструкций языка программирования (например, линеаризация и структурирование);
* наличие внутренних (промежуточных) переменных, отсутствующих в «словесном алгоритме» логического управления, которые резко затрудняют чтение схем алгоритмов другими, отличными от Разработчика, Специалистами и, в особенности, Заказчиком;
* наличие в общем случае большого числа внутренних обычно битовых переменных, каждую из которых приходится не только устанавливать, но и принудительно сбрасывать, и которые характеризуют лишь отдельные компоненты состояний автомата, а его состояния в целом обычно не известны;
* использование битовых внутренних переменных является естественным при аппаратной реализации алгоритмов, но при их реализации с помощью языков программирования, позволяющих обрабатывать многозначные переменные, применение битовых внутренних переменных нецелесообразно;
* наличие флагов и умолчаний значений внутренних и выходных переменных в операторных вершинах, которые затрудняют чтение схем алгоритмов ввиду необходимости помнить предысторию, особенно в тех случаях, когда значения переменных в этих вершинах изменяются в зависимости от путей, по которым можно «попасть» в рассматриваемую вершину;
* проверка в условных вершинах значений обычно только одиночных двоичных переменных, что приводит к громоздкости схем алгоритмов;
* связь операторных вершин через условные вершины, затрудняющая внесение изменений, так как модификация условий перехода между двумя операторными вершинами влияет на условия переходов в другие вершины.

При применении схем алгоритмов, в большинстве случаев переход от алгоритмизации к программированию для сложных задач логического управления представляет большую проблему. Это объясняется тем, что обычно процесс алгоритмизации почти никогда не завершается тем, чем положено – созданием алгоритма в математическом смысле, который, по определению, должен однозначно выполняться любым Вычислителем. Обычно этот процесс заканчивается лишь некоторой «картинкой», называемой алгоритмом, которую в той или иной степени приходится додумывать при программировании (например, структурировать схему алгоритма или вводить безусловные переходы в неструктурированную программу). В этой ситуации либо Разработчик должен сам программировать, либо Программист должен знать особенности технологического процесса, либо они вместе должны при испытаниях устранять неминуемые ошибки традиционного проектирования программ.

Рассмотренный язык используется фирмой «Опто» (США) для программирования ПЛК «Mistic».

Логические схемы алгоритмов являются строчной формой записи линеаризованных схем алгоритмов [6]. Они образованы буквами, которые соответствуют условным, безусловным и операторным вершинам линеаризованных схем алгоритмов, и пронумерованными стрелками, указывающими переходы, которые осуществляются при невыполнении условий.

Логические схемы алгоритмов обеспечивают компактность описания, но ненаглядны и весьма трудно строятся и читаются.

## 1.3. Язык функциональных блоковых диаграмм FBD

### 1.3.1 Описание языка FBD

FBD (Function Block Diagram) — это графический язык программирования. Диаграмма FBD очень напоминает принципиальную схему электронного устройства на микросхемах. «Проводники» в FBD могут проводить сигналы (передавать переменные) любого типа (логический» аналоговый, время и т. д.). Иногда говорят, что в релейных схемах соединительные проводники передают энергию. Проводники FBD тоже передают энергию, но в более широком смысле. Очевидно, что шины питания и контакты здесь уже не эффективны. Шины питания на FBD диаграмме не показываются. Выходы блоков могут быть поданы на входы других блоков либо непосредственно на выходы ПЛК. Сами блоки, представленные на схеме как «черные ящики», могут выполнять любые функции. FBD-схемы очень четко отражают взаимосвязь входов и выходов диаграммы. Если алгоритм изначально хорошо описывается с позиции сигналов, то его FBD-представление всегда получается нагляднее, чем в текстовых языках.

Блок FBD описывает функцию между входными переменными и выходными переменными. Функция описывается как множество элементарных функциональных блоков. Входные и выходные переменные связываются в блоки при помощи линий связи. Выход функционального блока может быть также связан с входом другого функционального блока.

Условные графические обозначения компонентов FBD-программ приведены на рисунке 1.1.

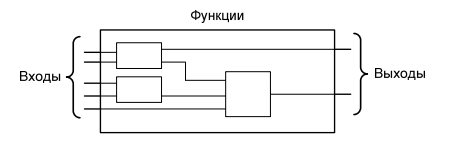


Рисунок 1.1 – Условное обозначение в FBD – программах.

Все функции FBD программы построены из стандартных элементарных функциональных блоков. Каждый функциональный блок имеет фиксированное количество входных точек связи и фиксированное количество выходных точек связи. Функциональный блок представляется одиночным прямоугольником. Входы соединяются с левым краем. Выходы соединяются с правым краем. Элементарный функциональный блок реализует одну функцию между входами и выходами. Имя функции, реализуемой блоком, пишется на символе прямоугольника, как показано на рисунке 1.2. Каждый вход или выход блока имеют определенный тип.

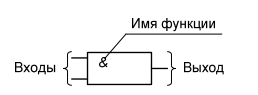


Рисунок 1.2 – Структура блока в FBD – программах.

Входные переменные FBD программы должны быть связаны с точками входа функционального блока. Тип каждой переменной должен быть тем же, что и тип соответствующего входа. Входом FBD блока может быть константа, любая внутренняя, входная или выходная переменная.

Выходные переменные FBD программы должны быть связаны с точками выхода функционального блока. Тип каждой переменной должен быть тем же, что и тип соответствующего выхода. Выходом FBD блока может быть внутренняя или выходная переменная, или имя программы (только для подпрограмм). Когда выходом является имя редактируемой подпрограммы, оно представляет присвоение возвращаемого значения подпрограммы (возвращаемого в вызывающую программу). Входные и выходные переменные, входы и выходы функциональных блоков соединены линиями связи. Линии могут быть использованы для соединения двух логических точек диаграммы:

* Входной переменной и входа функционального блока;
* Выхода функционального блока и входа другого блока;
* Выхода функционального блока и выходной переменной.

Связи ориентированы, это означает, что данные передаются с левого конца к правому. Левый и правый концы связи должны быть одного типа.

### 1.3.2 Программирование на FBD в среде ISaGRAF 6.1

Для создания программы на языке FBD в среде ISaGRAF необходимо выполнить следующие действия:

Создать новый проект «Simulator», как показано на рисунке 1.3.

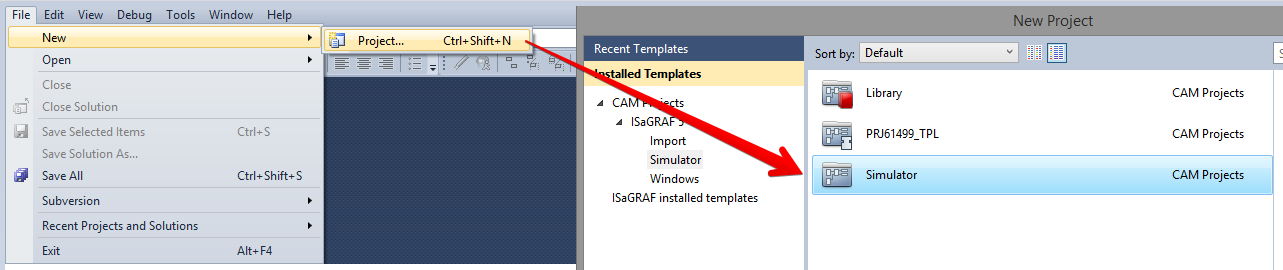


Рисунок 1.3 – Создание проекта.

В созданном проекте следует перейти в обозреватель решения, нажать правой кнопкой мыши по объекту Resource1.Programs. Далее следует выбрать Add, FBD, как показано на рисунке 1.4.

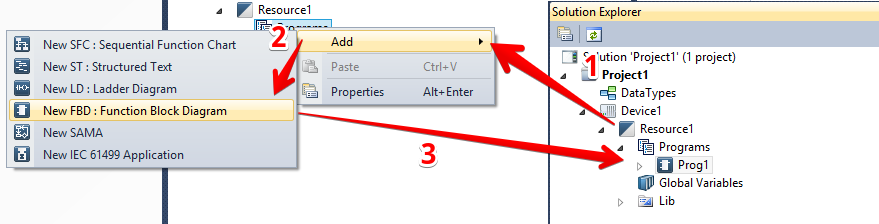


Рисунок 1.4 – Добавление программы FBD.

Для программирования на языке FBD используется панель инструментов. Следует перетащить нужный компонент из панели на рабочую область, как показано на рисунке 1.5.

Для создания блока логической переменной следует перетащить объект «Variable» из панели инструментов на рабочую область. Далее появится окно выбора переменной, как показано на рисунке 1.6.

Все переменные хранятся в словарях. Существует 2 типа словарей: локальный и глобальный. Локальный словарь содержит переменные, доступные только текущей программе, глобальный – доступный всем программам ресурса.

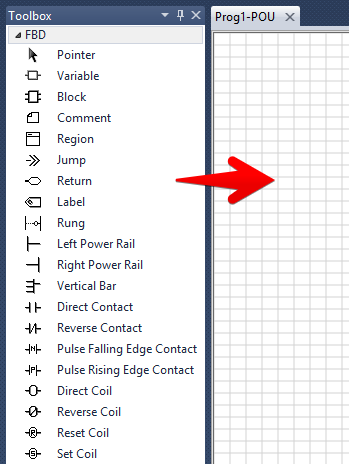


Рисунок 1.5 – Панель инструментов и рабочая область.

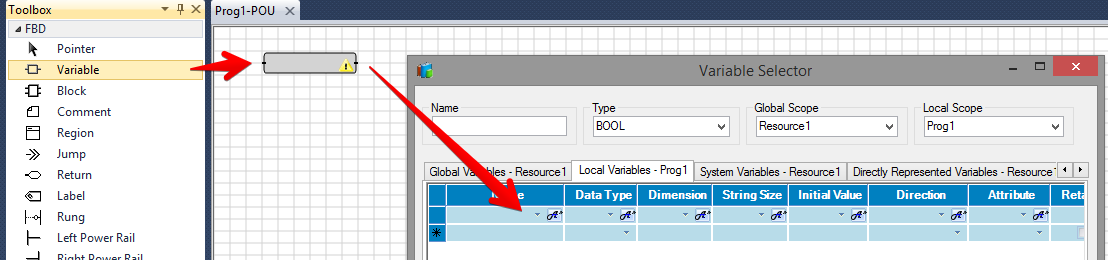


Рисунок 1.6 – Добавление переменной.

Следует отметить, что ISaGRAF не поддерживает русские символы в названиях переменных. Расположение локального и глобального словарей в обозревателе решений приведено на рисунке 1.7.

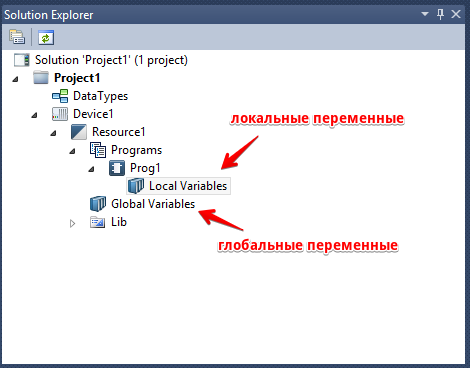


Рисунок 1.7 – Словари.

Для редактирования словаря следует два раза нажать по нему левой клавишей мыши. Внешний вид словаря при его редактировании приведен на рисунке 1.8.

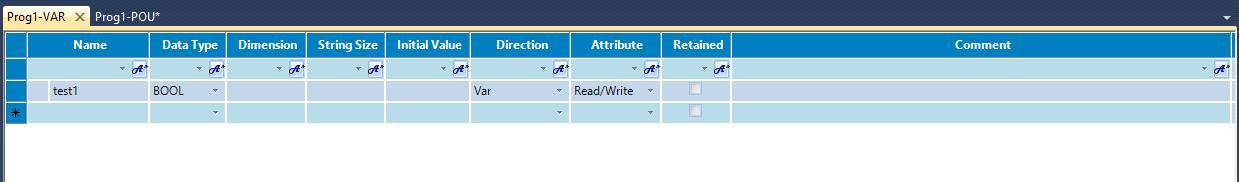


Рисунок 1.8 – Редактирование словаря.

Для создания блока следует перетащить объект «Block» из панели инструментов на рабочую область. Далее появится окно выбора типа блока, как показано на рисунке 1.9.

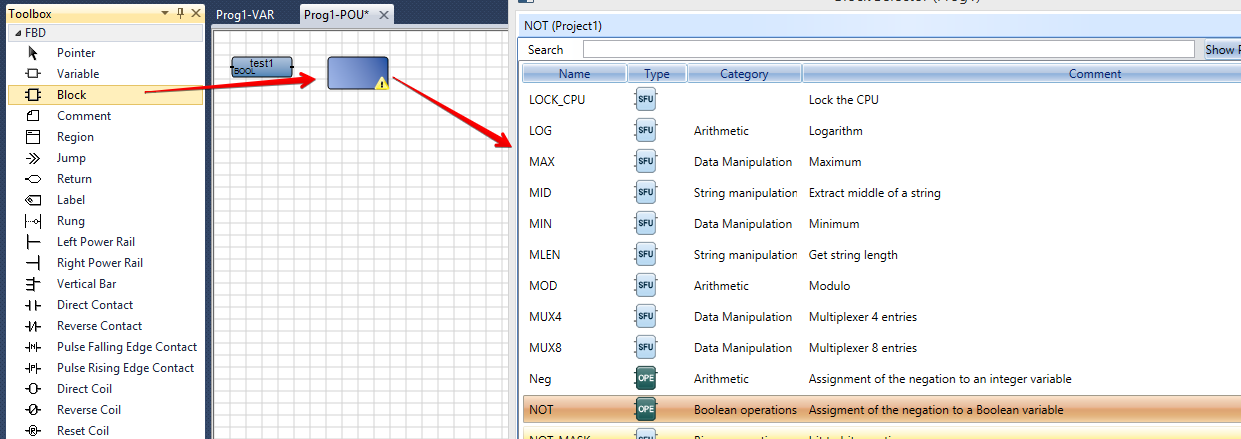


Рисунок 1.9 – Добавление переменной.

Далее следует соеденить переменные и блоки связями. Для этого требуется с зажатой левой клавишей мыши вести от выхода переменной или блока ко входу другой переменной или блока. Для того чтобы программа успешно скомпилировалась, все входы и выходы блока должны быть соединены с переменными или другими блоками. Готовая функциональная блок-схема показана на рисунке 1.10.

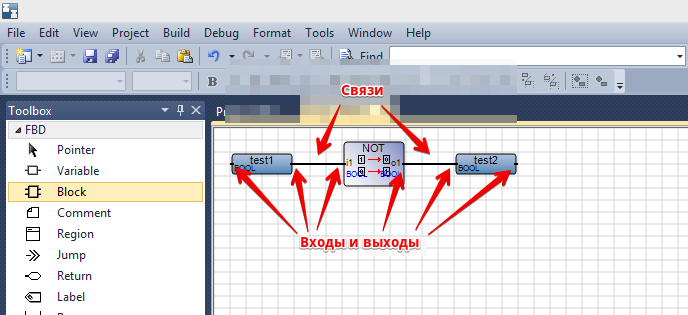


Рисунок 1.10 – Готовая функциональная блок-схема.

## 1.5 Патентный поиск

В ходе проведения патентного поиска было найдено несколько патентов на смежную с разрабатываемым программным средством тему. При выполнении патентного поиска была изучена научно-техническая и патентная информация:

− описание изобретений к патентам;

− заявки на изобретения;

− рефераты иностранных изобретений.

Результаты поиска сведены в таблицу 1.1.

Таблица 1.1 – Результаты патентного поиска

|  |  |  |  |
| --- | --- | --- | --- |
| Страна | Номер патента | Авторы | Название патента |
| США | 7350199 | Yuichi Ito, Paul L. Bleisch | Converting XML code to binary format |
| США | 8453126 | Pavel Ganelin | System and method for converting base SAS runtime macro language scripts to JAVA target language |
| США | 2001086462 | Chern Vincent, Dinh Dung John | Method of converting html/xml to hdml/wml in real-time for display on mobile devices |

Общим с рассмотренными программными средствами является только тип их назначения – преобразование из одной программы в другую (на другом языке), однако они не реализуют преобразование логических формул в функциональные блочные диаграммы ISaGRAF.

## 1.6 Вывод

В настоящее время в практике проектирования систем, наиболее часто употребляемыми языками спецификаций алгоритмов логического управления являются язык функциональных схем и язык блок-схем алгоритмов. Таким образом, в данном разделе было подробно описано назначение систем логического управления. Также рассказно об использовании и языках программирования программируемых логических контроллев, ведь широкое применение средств автоматизации производственных процессов, напрямую влияет на сокращение издержек и повышение качества продукции. Далее следует раздел о системе ISaGRAF, которая является инструментом разработки прикладных программ для программируемых логических контроллеров. Рассмотрены сновные языки описания алгоритмов логического управления и более подробно язык программирования FBD, который является главным с точки зрения цели данного дипломного проекта. Также был проведен патентный поиск по теме разрабатываемого программного средства.

# 2 Математические модели, положенные в основу проекта, и теоретические исследования

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

*32*

*ДП 02.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консультант*

*Акунович С.И.*

*Н. Контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Математические модели, положенные в основу проекта, и теоретические исследования вание выбора средств разработки*

*Лит.*

*Листов*

3

*БГТУ 84419015, 2015*

## 2.1 Описание поставленной задачи

Целью данного дипломного проекта является создание программного средства, предназначенного для преобразования логических формул, представленных в текстовом виде, в визуальную программу на языке FBD в среде разработки ISaGRAF 6.1.

## 2.2 Обоснование актуальности исследуемой задачи

Современные технологические машины и процессы содержат многочисленные датчики, сигналы от которых поступают на вход системы (блока) управления, которая должна анализировать текущее состояние и вырабатывать адекватные воздействия на исполнительные механизмы машины. Например, блок логического управления космического аппарата, используемый в качестве сложного управляющего устройства, описывается в виде алгоритмов управления (около 500) и логических функций (около 5000), при этом общее количество входных и выходных контактов блока превышает 1600. Если в таком описании допущены ошибки, дальнейшая его реализация может оказаться затруднительной.

Перед разработкой средств программной реализации этих алгоритмов в микропроцессорном контроллере очень важна их предварительная проверка на ПК.

В дипломном проекте для такой проверки выполнено преобразование логических формул блока управления спутника на язык технологического программирования FBD (стандарт IEC 61131-3) в среде разработки ISaGRAF, а также их моделирование (симуляция).

**2.3 Требования к программному средству**

Для обеспечения работы программного обеспечения необходимы программно-технические средства. Рабочая станция - персональный компьютер следующей конфигурации:

* процессор с частотой не меньше 166 МГц (минимально);
* оперативная память - 64 Мбайт (минимально);

*У*

32

* жесткий диск - 100 Мбайт (минимально для размещения программы и базы данных);
* монитор цветной (минимально с разрешением 800х600 точек);
* клавиатура, устройство управления "мышь".

Система работает в среде операционной системы - MS Windows XP, 7.

## 2.4 Алгоритм решения задачи

Для решения поставленной цели, логические формулы преобразутся в формат isaxml (вариант XML), обеспечивающий хранение структуры блоковых диаграмм на языке FBD, а также для хранения словаря проекта.

Логические формулы представлены как обычный текстовый файл в формате TXT.

Выходным файлом является соответствующий файл в формате isaxml.

Блок-схема алгоритма программы представлена на рисунках 2.1.

Открыть исходный файл

Открыть файл словаря isaxml

Создать словарь

Закрыть файл словаря isaxml

Закрыть исходный файл

Преобразовать в FBD

Рис 2.1 – Блок-схема алгоритма программы.

## 2.5 Структура программного кода

В таблице 2.1 приведены некоторые основные функции программы и краткое описание выполняемых ими действий.

Таблица 2.1 – Описание функций программы

|  |  |
| --- | --- |
| Название функции | Выполняемые действия |
| buildTree | Представляет логическую формулу в виде структуры дерева. |
| drawTree | Преобразует логические формулы в функциональные выражения. |
| findVariable | Проходит по логической формуле и заносит переменные в словарь, если эти переменные еще не представлены в словаре. |
| drawFormula | Преобразует логическую формулу в специальный язык описания блок-схем FBD. |
| functionWriteFile | Преобразует файл isaxml, в котором содержится структура программы FBD, записывает преобразованные логические формулы для последующего чтения средой ISaGRAF. |

## 2.6 Вывод

В данном разделе были представлены и описаны математические модели, положенные в основу. Рассказано об актуальности исследуемой задачи и ее решения. Ведь очень важна предварительная проверка на ПК перед разработкой средств программной реализации алгоритмов управления. В дипломном проекте для такой проверки выполнено преобразование алгоритмов управления (логических формул) на язык технологического программирования FBD (стандарт IEC 61131-3) в среде разработки ISaGRAF, а также их моделирование (симуляция).

# 3 Обоснование выбора средств разработки

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

*35*

*ДП 03.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консультант*

*Акунович С.И.*

*Н. Контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Обоснование выбора средств разработки*

*Лит.*

*Листов*

4

*БГТУ 84419015, 2015*

В качестве средств разработки программного средства по теме данного дипломного проекта были выбраны язык программирования DELPHI и среда разработки Embarcadero RAD Studio XE5. В следующих подразделах содержится обоснование выбора данных средств.

## 3.1 Язык программирования DELPHI

Delphi (Дельфи) [10] – среда программирования, использующая язык Object Pascal, разработанный фирмой Borland и изначально реализованный в её пакете Borland Delphi, от которого и получил в 2003 году своё нынешнее название. По сути, является наследником языка Pascal с объектно-ориентированными расширениями.

Наиболее популярное использование - разработка настольных приложений и баз данных предприятия, а как инструмент язык программирования Делфи может использоваться и для большинства типов развивающихся проектов. Это был один из первых языков, который стал известным как инструмент RAD (быстрая разработка приложений), когда вышел в 1995 году. Delphi 2, вышедший год спустя, поддерживал 32-разрядную среду Windows, и а только несколько лет спустя вышли C, C + + Builder. В 2001 году стала доступна версия под ОС Linux известная как Kylix (классическая греческая урна). Каждый год выходило по одной новой версии, в 2002 году продукт стал известен как Delphi 7 Studio, язык стал официально называться язык программирования Delphi, а не Object Pascal, а также была добавлена поддержка Linux (через Kylix) и. NET (через Предварительный компилятор). Замечательные особенности языка Delphi включают:

* прозрачная обработка объектов через ссылки или указатели;
* свойства как часть языка, вкупе с функциями Get и Set, которые являются прозрачной инкапсуляцией доступа к членам полям;
* свойства индекса и свойствами по умолчанию, которые обеспечивают доступ к коллекции удобным и прозрачным способом;
* делегаты или по-другому методы указателей безопасного типа, которые используются для приведения в действие события вызванных компонентами;
* делегирование реализации интерфейса в поле или свойство класса;
* простота внедрения обработчики Windows сообщение, отметь метод класса с числом / имя окна сообщений для обработки;
* большинство функций, перечисленных выше, были введены в Delphi первой и адаптированы на других языках позже.

*У*

35

Главный архитектором Delphi, и его предшественника Turbo Pascal, был Андерс Хейлсбергом, пока он не перешел в Microsoft в 1996 году посвятил свое дело основам программирования. Продукт Delphi распространяется в различных комплектах, каждый из которых предлагает большую функциональность по сравнению с другими:

* персональный;
* профессиональный;
* предприятие;
* архитектор.

Веские причины для использования Delphi:

* очень информативные и полезные сообществу новости;
* может компилировать в один исполняемый, упрощая распределение и сокращение вопросов с разными DLL;
* VCL и сторонние компоненты, как правило, доступны с полным исходным кодом;
* мощный и быстрый оптимизирующий компилятор;
* из одного исходного кода получаются отличные машинные коды для разных ОС;
* Поддержка новейших технологий и стандартов.

## 3.2 Среда разработки Embarcadero RAD Studio XE5

Программное обеспечение **Embarcadero RAD Studio XE5 [11]** – это комплексная среда разработки нативных приложений для рабочих групп, которая помогает создавать ПО для мобильных платформ Android и iOS, быстро предоставлять их магазинам приложений и предприятиям.

Команды разработки ПО все чаще вынуждены одновременно и в сжатые сроки создавать приложения для различных устройств. Однако программирование с использованием различных исходных кодов и API для нескольких платформ, а также управление ими стоит дорого и отнимает много времени. Поэтому Embarcadero RAD Studio XE5 содержит одну и ту же базу исходного кода, не требуя от разработчиков жертвовать качеством приложений, возможностями подключения или производительностью. Благодаря нативной поддержке Android и iOS производители программного обеспечения могут выходить на крупнейшие мобильные рынки в мире.

RAD Studio XE5 на сегодняшний день является единственным решением на рынке, предоставляющим действительно нативный исполняемый код для четырех самых популярных операционных систем: Android, iOS, Windows и Mac OS X. При этом используется одна и та же база исходного кода, исключающая необходимость жертвовать качеством приложений, возможностями подключения или производительностью.

RAD Studio XE5 позволяет создавать по-настоящему нативные приложения для персональных компьютеров, смартфонов и планшетов, быстро загружать готовые программы в магазины приложений и предоставлять ПО предприятиям. Разработанные при помощи комплекса RAD Studio XE5 приложения запускаются непосредственно на оборудовании без дополнительных скриптовых и интерпретативных слоев, что дает разработчикам возможность обеспечивать оптимальное юзабилити.

Преимущества версии Embarcadero RAD Studio XE5:

* Нативная поддержка Android и iOS для разработки приложений под смартфоны и планшеты. Платформа также предоставляет инструменты доступа к распространенным сенсорам и службам мобильных устройств, средства использования данных устройства, облаков и предприятия;
* Одна база исходного кода. Команды разработчиков могут прототипировать и создавать нативные приложения на C++ или Delphi для множества устройств одновременно. В результате не нужно выстраивать множество проектов, расписаний, бюджетов с отдельными группами разработчиков, инструментами, языками и библиотеками для каждой платформы;
* Платформа разработки ПО для множества устройств. Поддержка REST-клиента упрощает подключение к любому сервису на базе REST, в том числе социальным сетям, платежным системам, хранилищам, MBaaS и др. RAD Studio XE5 – оптимальная облачная или локальная платформа для поставки корпоративных данных и служб в создаваемые приложения.

Теперь разработчики могут легко обеспечивать соответствие бизнес-приложений концепции BYOD (Bring Your Own Device) и гарантировать, что ПО не только запускается на разном оборудовании, но и поддерживает многочисленные backend-сервисы – внутри корпоративной сети и в облаке.

Редакции Embarcadero RAD Studio XE5:

1. Professional – версия для независимых поставщиков ПО, разрабатывающих нативные приложения для компьютеров, планшетных ПК и смартфонов со встроенной или локальной базой данных. Разработчики могут управлять одной кодовой базой, одной командой и одним расписанием без ущерба для производительности. Полнофункциональные нативные приложения обеспечивают более полный контроль, повышенную безопасность и максимальное удобство для пользователей;
2. Enterprise – версия для независимых поставщиков ПО и предприятий. Создание приложений для компьютеров, планшетных ПК и смартфонов с поддержкой клиент-серверных и многозвенных сред, подключением к корпоративным базам данных и облачным службам на основе SQL Server, DB2, Oracle, Sybase, InterBase, Azure, Amazon и многоуровневой технологии DataSnap;
3. Ultimate – версия для предприятий, которые занимаются созданием нативных приложений для компьютеров, планшетных ПК и смартфонов, активно работающих с корпоративными системами баз данных. Кроме того, RAD Studio Ultimate содержит средства для разработки кода SQL, управления изменениями в базах данных, а также профилирования и настройки кода SQL с помощью DB PowerStudio;
4. Architect – версия для предприятий, создающих приложения для компьютеров, планшетных ПК и смартфонов, интегрированные с корпоративными системами баз данных. В редакцию также включено средство визуального моделирования данных ER/Studio Developer Edition, позволяющее разработчикам наиболее эффективно использовать корпоративные структуры и базы данных.

## 3.3 Вывод

Таким образом, в данном разделе было выполнено обоснование выбора средств разработки программного средства по теме дипломного проекта.

# 4 Проектирование и реализация программного средства

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

*39*

*ДП 04.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консультант*

*Акунович С.И.*

*Н. Контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Разработка программного средства*

*Лит.*

*Листов*

19

*БГТУ 84419015, 2015*

Целью данного раздела является проектирование и описание схем построения архитектуры представленного в дипломном проекте приложения для преобразования логических формул в графические функциональные блочные диаграммы на языке программирования FBD, представленные в формате проекта в среде разработки ISaGRAF 6.1. Так же описан процесс реализации представленного программного продукта по выбранному и спроектированному архитектурному решению, и методу разработки.

## 4.1 Исходные данные

Первичные данные (логические формулы) на разработку блока логического управления представлены в текстовом виде в TXT-файле, который изображен на рисунке 4.1.

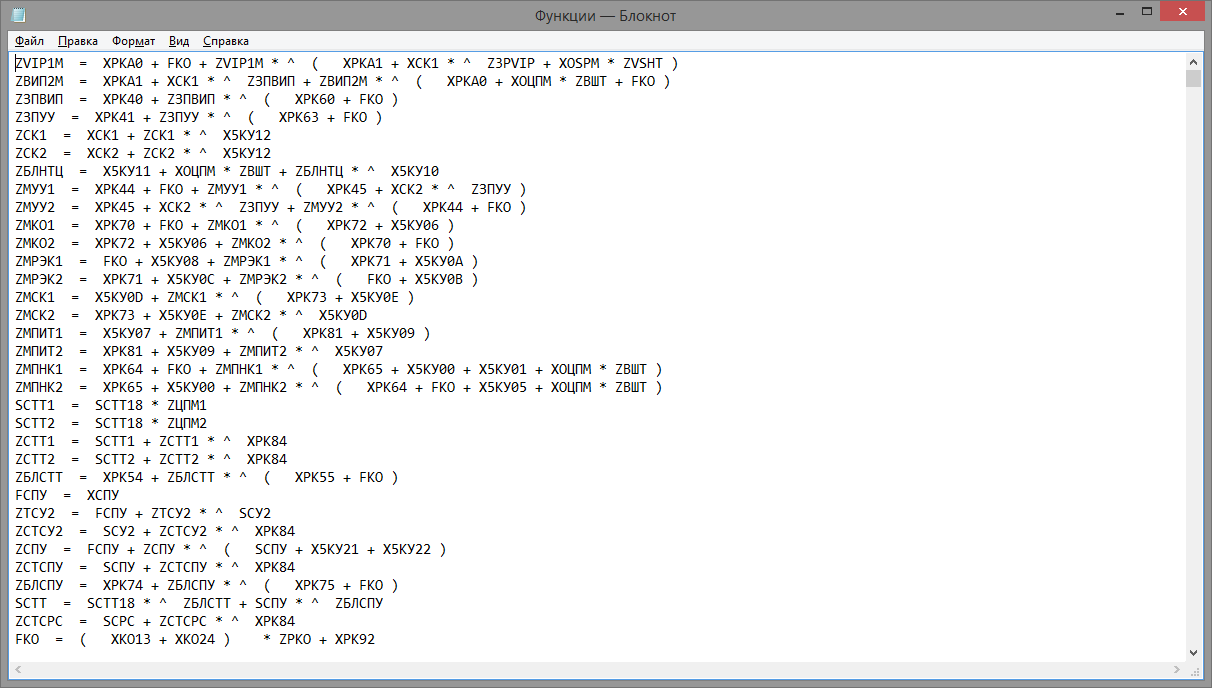


Рисунок 4.1 – Исходный файл TXT.

## 4.2 Назначение разработанного приложения

Программа-конвертер Logical\_Formulas\_Converter представляет собой обычное Windows приложение и осуществляет преобразование логических формул, записанных в текстовом файле, на язык FBD в файл с расширением. isaxml, а также формирование при этом словаря проекта в том же файле проекта.

Использование функциональных блочных диаграмм дает следующие преимущества:

* графическое разделение процессов на основные логические части;
* более быстрое повторрное выполнение отдельных частей логики;
* более простое представление информации на экране;

*У*

39

* меньше времени на разработку и отладку программы;
* более быстрая и простая диагностика;
* непосредственный доступ к точке логики, где произошла машинная ошибка;
* простота обновления и модернизации.

Создание функциональных блочных диаграмм алгоритмов, записанных в текстовом виде, занимает достаточно много времени, особенно если рисовать очень сложные функциональные блочные диаграммы. Для ускорения и автоматизации этого процесса и предназначена программа Logical\_Formulas\_Converter.

При создании в среде разработки ISaGRAF 6.1 программы на языке FBD, она сохраняется в директорию проекта в файл с расширением. isaxml. Этот файл хранит в себе структуру функциональных блочных диаграмм и именно с него считывается диаграмма, при открытии ее в среде ISaGRAF 6.1.

## 4.3 Интерфейс программы

Окно программы Logical\_Formulas\_Converter состоит из двух вкладок. Первая вкладка представлена на рисунке 4.4, вторая – на рисунке 4.5.

Первая вкладка состоит:

1. текстовое поле для ввода пути к файлу, содержащему логические формулы, а также кнопка «Выбор», позволяющая сделать это с помощью диалогового окна;
2. текстовое поле для ввода пути к файлу, куда будет сохранен результат преобразования логических формул в функциональные формулы, а также кнопка «Выбор», позволяющая сделать это с помощью диалогового окна;
3. кнопка «Преобразовать».

Кнопка «Преобразовать» запускает алгоритм преобразования логических формул в функциональные выражения.



Рисунок 4.4 – Первая вкладка главного окна программы.

Вторая вкладка состоит:

1. текстовое поле для ввода пути к файлу Prog.isaxml, а также кнопка «Выбор», позволяющая сделать это с помощью диалогового окна;
2. кнопка «Преобразовать».

Кнопка «Преобразовать» запускает алгоритм преобразования логических формул на язык функциональных блочных диаграмм FBD.



Рисунок 4.5 – Первая вкладка главного окна программы.

## 4.4 Принцип работы программы

Работа программы заключается в преобразовании алгоритмов, записанных в текстовом файле в виде логических формул, в функциональные блочные диаграммы на языке FBD, которые распознает среда ISaGRAF 6.1. Преобразование происходит в два этапа: преобразование логических формул в функциональные выражения; преобразование в функциональные блочные диаграммы. При запуске преобразования первого этапа программа выполняет следующие действия:

* считываение данных с исходного текстового файла, содержащего в себе логические формулы, записанные в текстовом виде. Пример исходного файла приведен на рисунке 4.6;
* считанные данные обрабатываются построчно (по одной формуле). В итоге каждая формула представляется в виде дерева;
* анализ каждого полученного дерева, преобразование его в функциональное выражение и запись результата в файл;
* уведомление об окончании преобразования.

Результат выполнения первого этапа показан на рисунке 4.7.

При запуске преобразования второго этапа программа выполняет следующие действия:



Рисунок 4.6 – Пример исходного файла.

* анализ каждого дерева, полученного на предыдущем этапе. При анализе все переменные заносятся в словарь, также в отдельные словари записываются функции и связи между ними;
* считывание файла isaxml и его преобразование. При преобразовании происходит запись данных из словарей переменных, функций и связей.



Рисунок 4.7 – Результат выполнения первого этапа.

* уведомление об окончании преобразования.

Результат выполнения второго этапа показан на рисунке 4.8.



Рисунок 4.8 – Результат выполнения второго этапа.

### 4.4.1 Описание структуры файла Prog1.isaxml

Файл Prog1.isaxml хранится в папке проекта среды ISaGRAF 6.1 и предназначен для хранения структуры функциональных блочных диаграмм на языке FBD, а также для хранения словаря проекта (рисунок 4.9). В этом файле используется специальный язык для описания функциональных блочных диаграмм и словаря переменных для представления в ISaGRAF 6.1.



Рисунок 4.9 – Пример файла Prog1.isaxml.

Вид функциональных блочных диаграмм, структуру которых хранит файл, приведенный на рисунке 4.9, показан на рисунке 4.10.

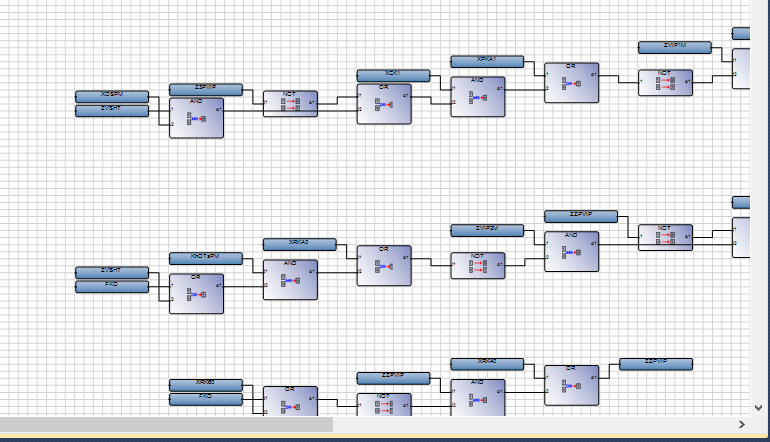


Рисунок 4.10 – Функциональные блочные диаграммы на языке FBD в ISaGRAF 6.1.

Составляющие файла Prog1.isaxml отображены на рисунке 4.11.

Рисунок 4.11 – Структура файла Prog1.isaxml.

Первая строчка файла Prog1.isaxml указывает, что документ является XML документом и использует кодировку UTF-8. В данном случае тег имеет следующий вид:

<?xml version=«1.0» encoding=«utf-8»?>

Т.к. описание всех возможных действий содержится в XML-файле, это позволяет удобно редактировать имеющиеся компоненты и добавлять новые.

В следующей строке находится тег Pou с определенными в нем через атрибуты параметрами проекта:

<Pou FileVersion=«200.00.000» Name=«ProgOR» ElementGuid=«88b1fb05-09ce-41ff-a77e-c63ccd5fae2c» Comment=«« SingleTextLine=«« Language=«FBD» CustomFbd=«0» Localization=«Prj» GroupName=«(Определяемый пользователем)» GenDebugInfo=«false» UseResPassword=«true» GenerateSymbMon=«true» Is1499=«false»>

Где FileVersion указывает на версии файла, Name - имени проекта, Comment -комментарии, Language - язык программы проекта, GenDebugInfo - генерация отладочной информации и различные другие атрибуты. Данный тег содержит в себе все остальные теги, включая структуру блок-схемы алгоритма.

Тег <LocalVars> представляет собой словарь переменных проекта. Каждая переменная описывается рядом параметров через атрибуты в теге <Variable />: имя переменной (Name), тип данных (DataType), начальное значение (InitialValue), комментарий (Comment), адрес (Address), направление (Kind), алиас (Alias), права доступа (AccessRights), размер строки (StringSize) и другие атрибуты, которые можно задать в редакторе переменных среды ISaGRAF 6.1.

Далее идет тег <PouBody> и в нем секция CDATA, содержащая в себе уже только символьные данные, которые не смогут быть обработаны средой ISaGRAF 6.1 как разметка XML-документа:

<PouBody><![CDATA[

В этой секции и размещается структура функциональных блочных дигаграмм на языке FBD со всеми ее элементами, которая отображается при открытии проекта в ISaGRAF 6.1.

Рассмотрим теперь основные конструкции элементов файла Prog1.isaxml.

Конструкция BOX описывает блок функций: название функции (AND, OR, NOT), его расположение на координатной плоскости проекта, а также идентификатор блока. Пример такой конструкции отображен на рисунке 4.12.



Рисунок 4.12 – Конструкция BOX файла Prog1.isaxml.

Конструкция VAR описывает блок переменной: имя переменной и расположение на координатной плоскости проекта, а также иденификатор блока. Пример такой конструкции отображен на рисунке 4.13.



Рисунок 4.13 – Конструкция VAR файла Prog1.isaxml.

Конструкция ARC описывает связи между блоками: идентификаторы блоков и их выходов, которые соединиет связь, а также расположение связи на координатной плоскости проекта. Пример описания данной конструкции отображен на рисунке 4.14.



Рисунок 4.14 – Конструкция ARC файла Prog1.isaxml.

### 4.4.2 Детальное описание алгоритма работы

Для начала пользователь должен с помощью кнопки «Выбор», расположенной напротив текстового поля «Входной файл с логическими формулами» на первой вкладке, открыть файл с логическими формулами. Далее пользователь должен с помощью кнопки «Выбор», расположенной напротив текстового поля «Входной файл с функциональными формулами», выбрать файл, в который будет записан результат преобразования в функциональные формулы.

При нажатии на кнопку «Преобразовать», выполняется считывание файла с исходными данными в коллекцию TStringList, где каждый элемент коллекции представляет собой одну логическю формулу. Далее каждая логическая формула анализируется и представляется в виде дерева. За представление логической формулы в виде дерева отвечает процедура buildTree. Данная процедура принимает на входе формулу, элемент дерева, который является результатом выполнения данной формулы, позиции скобок в формуле, а также флаг, указывающий, должен ли быть поставлен блок NOT перед присвоением результата. Процедура работает следующим образом: формула, которая имеется на входе, разбивается на 2 части (правую и левую), выясняется, какая функция должна быть выполнена («+» или «\*») между левой и правой частями, чтобы они дали результат, который имеется на входе; эта функция записывается как значение внутреннего узла дерева, которое пришло на входе, если данная функция стоит вместе с отрицанием («^»), то оно помечается как отдельный флаг данного узла; правая и левая части являются дочерними узла данного родительского узла, если они предствляют из себя одно значение переменной; если же какая либо из дочерних листов представляет собой формулу, то для нее рекурсивно вызывается процедура buildTree.

Далее вызывается процедура drawTree, которая анализирует получившиеся деревья на предыдущем этапе и преобразует их в функциональные выражения, которое после записывается в текстовый файл. Работа процедуры в общем виде выглядит так: на входе подается узел дерева, если у данного узла нет дочерних, то в строку функционального выражения записывается значение этого узла, которое вляется именем переменной и на этом работа процедры заканчивается. В другом случае, анализируется, какая функция («\*» или «+») соответствует данному узлу и записывается в строку в виде вызова функции и вызывается данная процедура для дочернего узла. В итоге получается функциональное выражение, записанное в текстовом виде. Данное выражение записывается в текстовый файл.

Далее, для преобразования на язык FBD пользователю следует перейти на вторую вкладку и с помощью кнопки «Выбор», указать расположение файла ProgN.isaxml проекта ISaGRAF, созданного заранее, и нажать на кнопку «Преобразовать».

При нажатии на кнопку, для кажого дерева, которое было получено на предыдущих этапах с помощью процедуры buildTree, вызывается процедура findVariable, которая заносит в словарь переменные, дубликаты игнорируются.

Далее будет вызвана процедура drawFormula для каждого дерева. Данная процедура, так же как и drawTree рекурсивно проходит по узлам дерева и преобразует их в соответствующие конструкции FBD (VAR, BOX, ARC), которые хранятся в разных коллекциях.

В конце вызывается процедура functionWriteFile, которая преобразует isaxml-файл, выбранный пользователь, записывая в него словарь переменных и все конструкции.

После выполнения программы, ISaGRAF, который должен быть в это время запущен, уведомит пользователя, что файл ProgN.isaxml изменился, и предложит обновить его в среде разработки.

### 4.4.3 Реализация программы

Рассмотрим основные процедуры и текст исходного кода программы.

Процедура Button2Click производит вызов диалогового окна, с помощью которого пользователь указывает расположение файла с исхоными данными:

procedure TForm1.Button2Click(Sender: TObject);

begin

DataModule2.OpenDialog1.Title := 'Выберите входной файл с логическими формулами';

DataModule2.OpenDialog1.Filter := 'Text Files(\*.txt)|\*.txt';

DataModule2.OpenDialog1.FilterIndex := 0;

DataModule2.OpenDialog1.FileName := '';

if (DataModule2.OpenDialog1.Execute()) then

Form1.Edit1.Text := DataModule2.OpenDialog1.FileName;

end;

Процедура Button3Click производит вызов диалогового окна, с помощью которого пользователь указывает расположение файла, в который будет записаны логические функции, преобразованные в функциональные выражения:

procedure TForm1.Button3Click(Sender: TObject);

begin

DataModule2.SaveDialog1.Title := 'Выберите выходной файл для функциональных формул';

DataModule2.SaveDialog1.Filter := 'Text Files(\*.txt)|\*.txt';

DataModule2.SaveDialog1.FilterIndex := 0;

DataModule2.SaveDialog1.FileName := '';

if (DataModule2.SaveDialog1.Execute()) then

Form1.Edit2.Text := DataModule2.SaveDialog1.FileName;

end;

Процедура Button4Click запускает процесс преобразования из логических формул в функциональные выражения, построчно выполняя транслитерацию с русского алфавита на английский, удаление всех пробелов, преобразование формулы в структуру в виде дерева, преобразование структуры в виде дерева в функциональное выражение и запись результатов в файл:

procedure TForm1.Button4Click(Sender: TObject);

var

\_message, \_message2 : TStringList;

i: Integer;

output: String;

\_tree: Unit1.Tree;

begin

\_message := TStringList.Create;

\_message2 := TStringList.Create;

\_message.LoadFromFile(edit1.Text, TEncoding.Default);

for i := 0 to (\_message.Count - 1) do

begin

\_message[i] := TranslitRus2Lat(\_message[i]);

\_message[i] := StringReplace(\_message[i], ' ', '', [rfReplaceAll]);

output := Copy(\_message[i], 1, IndexOfAny(\_message[i], '=') - 1);

\_message[i] := Copy(\_message[i], IndexOfAny(\_message[i], '=') + 1, MAXINT);

\_tree := Tree.Create;

\_tree.root := Node.Create;

\_tree.value := output;

view := '';

buildTree(\_message[i], \_tree.root, brackets(\_message[i]), false);

drawTree(\_tree.root);

\_message2.Add(\_tree.value + ' =' + view);

end;

\_message2.SaveToFile(edit2.Text);

ShowMessage('Выполнено');

end;

Удаление всех пробелов выполняется с помощью стандартной функции StringReplace. Транслитерация букв русского алфавита на английский выполняется с помощью функции TranslitRus2Lat:

function TranslitRus2Lat(const Str: string): string;

const

RArrayL = 'абвгдеёжзийклмнопрстуфхцчшщьыъэюя';

RArrayU = 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯ';

colChar = 33;

arr: array[1..2, 1..ColChar] of string =

(('a', 'b', 'v', 'g', 'd', 'e', 'yo', 'zh', 'z', 'i', 'y',

'k', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'f',

'kh', 'ts', 'ch', 'sh', 'shch', '''', 'y', '''', 'e', 'yu', 'ya'),

('A', 'B', 'V', 'G', 'D', 'E', 'Yo', 'Zh', 'Z', 'I', 'Y',

'K', 'L', 'M', 'N', 'O', 'P', 'R', 'S', 'T', 'U', 'F',

'Kh', 'Ts', 'Ch', 'Sh', 'Shch', '''', 'Y', '''', 'E', 'Yu', 'Ya'));

var

i: Integer;

LenS: Integer;

p: integer;

d: byte;

begin

result := '';

LenS := length(str);

for i := 1 to lenS do

begin

d := 1;

p := pos(str[i], RArrayL);

if p = 0 then

begin

p := pos(str[i], RArrayU);

d := 2

end;

if p <> 0 then

result := result + arr[d, p]

else

result := result + str[i]; //если не русская буква, то берем исходную

end;

end;

Класс, описывающий дерево, имеет следующий вид:

type

Tree = class

public

root: Node;

value: String;

published

constructor Create;

end;

Его конструктор:

constructor Tree.Create;

begin

inherited;

self.root := nil;

self.value := '';

end;

Класс, описывающий узел дерева, имеет следующий вид:

type

Node = class

public

symbol: String;

\_not: bool;

left, right: Node;

id: Integer;

published

constructor Create;

end;

Его конструктор:

constructor Node.Create;

begin

inherited;

self.symbol := '';

self.\_not := false;

self.left := nil;

self.right := nil;

end;

За преобразование логической формулы в стрктуру в виде дерева отвечает процедура buildTree:

procedure buildTree(str: String; \_node: Node; list: TStringList; \_not: bool);

var

index1, index2, last, index: Integer;

left, right, value, tmp: String;

leftNot, rightNot: Bool;

leftNode, rightNode: Unit1.Node;

begin

left := ''; right := ''; value := '';

leftNot := false; rightNot := false;

index1 := IndexOfAny(str, '+\*');

index2 := IndexOfAny(str, '(');

if (index1 >= 1) and (index2 >= 1) and (index2 < index1) then // если первые скобки в левой части

begin

if (index2 - 1 >= 1) and (str[index2 - 1] = '^') then leftNot := true;

tmp := IntToStr(index2);

last := StrToInt(list.Values[tmp]);

left := Copy(str, index2 + 1, (last - index2) - 1);

right := Copy(str, last + 2, MaxInt);

if ((last + 1) >= 0) and ((last + 1) < Length(str)) then value := str[last + 1];

index := last + 2; // удаляем вторые скобки в правой части при их налич

if (index >= 0) and (index < Length(str)) and (str[index] = '^') then

begin

index := last + 3;

rightNot := true;

end;

if (list.IndexOfName(IntToStr(index)) >= 0) and (list.IndexOfName(IntToStr(index)) < (list.Count)) then

begin

last := StrToInt(list.Values[IntToStr(index)]);

if (last >= 0) and (last = Length(str) - 1) then

begin

if (rightNot = false) then right := Copy(right, 1, Length(right) - 2)

else right := Copy(right, 2, Length(right) - 3);

end;

end

else if (IndexOfAny(right, '+\*') = 0) then // удаляем ^ перед вторым числом при его наличии

begin

if(index >= 0) and (index < Length(str)) then

begin

index := last + 2;

if str[index] = '^' then

begin

rightNot := true;

right := Copy(right, 2, length(right) - 1);

end;

end;

end

else rightNot := false;

end

else if IndexOfAny(str, '+\*') = 0 then // если это простое присвоение

begin

if (str[1] <> '^') then value := Copy(str, 1, MaxInt)

else

begin

value := Copy(str, 2, MaxInt); \_not := true;

end;

end

else // если первое число в левой части

begin

if (str[1] <> '^') then left := Copy(str, 1, index1 - 1)

else if (index1 - 1 >= 0) and (index1 -1 < Length(str)) then

begin

left := Copy(str, 2, index1 - 2); leftNot := true;

end;

right := Copy(str, index1 + 1, MaxInt);

if (index1 >= 0) and (index1 < Length(str)) then value := str[index1];

index := index1 + 1; // удаляем вторые скобки в правой части при их наличии

if (index >= 0) and (index < Length(str)) and (str[index] = '^') then

begin

index := index1 + 2; rightNot := true;

end;

if (list.IndexOfName(IntToStr(index)) >= 0) and (list.IndexOfName(IntToStr(index)) < (list.Count)) then

begin

last := StrToInt(list.Values[IntToStr(index)]);

if (last >= 0) and (last = Length(str)) then

begin

if (rightNot = false) then right := Copy(right, 2, Length(right) - 2)

else right := Copy(right, 3, Length(right) - 3);

end;

end

else if IndexOfAny(right, '+\*') = 0 then // удаляем ^ перед вторым числом при его наличии

begin

if (index >= 0) and (index < Length(str)) then

begin

index := index1 + 1;

if (str[index] = '^') then

begin

rightNot := true; right := Copy(right, 2, Length(right) - 1);

end;

end;

end

else rightNot := false;

end;

\_node.\_not := \_not;

\_node.symbol := value;

if (IndexOfAny(left, '+\*(') = 0) and (left <> '') then

begin

leftNode := Node.Create;

leftNode.symbol := left;

leftNode.left := nil;

leftNode.right := nil;

leftNode.\_not := leftNot;

leftNot := false;

\_node.left := leftNode;

end;

if (IndexOfAny(right, '+\*(') = 0) and (right <> '') then

begin

rightNode := Node.Create;

rightNode.symbol := right;

rightNode.left := nil;

rightNode.right := nil;

rightNode.\_not := rightNot;

rightNot := false;

\_node.right := rightNode;

end;

if IndexOfAny(left, '+\*(') <> 0 then

begin

//\_node.\_not := leftNot;

\_node.left := Node.Create;

buildTree(left, \_node.left, brackets(left), {false}leftNot);

end;

if IndexOfAny(right, '+\*(') <> 0 then

begin

\_node.right := Node.Create;

buildTree(right, \_node.right, brackets(right), rightNot);

end;

end;

На вход данной процедуры передается расположение всех скобок в логической формуле, которое находится с помощью функции brackets:

function brackets(str: String): TStringList;

var

input, output, list: TStringList;

i, j, max: Integer;

begin

// индексы всех открывающихся скобок / закрывающихся скобок

input := TStringList.Create;

output := TStringList.Create;

// сопоставление каждой открывающейся скобки и закрывающейся скобки

list := TStringList.Create;

for i := 0 to Length(str) do begin

if str[i] = '(' then

begin

input.Append(IntToStr(i));

end;

if str[i] = ')' then

begin

output.Append(IntToStr(i));

end;

end;

for i := 0 to output.Count - 1 do

begin

max := 0;

for j := 0 to input.Count - 1 do

begin

if (StrToInt(input.Strings[j]) < StrToInt(output.Strings[i])) and

(StrToInt(input.Strings[j]) > max) then

max := StrToInt(input[j]);

end;

input.Delete(input.IndexOf(IntToStr(max)));

list.Add(Format('%s=%s', [IntToStr(max), output[i]]))

end;

brackets := list;

end;

За преобразование дерева в функциональное выражение отвечает процедура drawTree:

procedure drawTree(\_node: Node);

begin

if (\_node.left = nil) and (\_node.right = nil) then

begin

if (\_node.\_not = true) then view := view + ' НЕ (' + \_node.symbol + ')'

else view := view + \_node.symbol;

end;

if (\_node.left <> nil) then

begin

if (\_node.left.left = nil) and (\_node.left.right = nil) then

if \_node.\_not = true then view := view + ' НЕ ('; // 2 вариант

if (\_node.symbol = '\*') then view := view + ' И ('

else if \_node.symbol = '+' then view := view + ' ИЛИ (';

if (\_node.left.left <> nil) or (\_node.left.right <> nil) then

if (\_node.\_not = true) then view := view + ' НЕ (('; // 1 вариант

drawTree(\_node.left);

end;

if (\_node.right <> nil) then

begin

if (\_node.left.left <> nil) or (\_node.left.right <> nil) then

if (\_node.\_not = true) then view := view + ')'; // 1 вариант

if (\_node.symbol = '\*') then view := view + ';'

else if (\_node.symbol = '+') then view := view + ';';

drawTree(\_node.right);

if (\_node.symbol = '\*') then view := view + ')'

else if (\_node.symbol = '+') then view := view + ')';

if (\_node.left.left = nil) and (\_node.left.right = nil) then

if (\_node.\_not = true) then view := view + ')'; // 2 вариант

end;

end;

Функциональные выражения записываются в файл с помощью функции SaveToFile, которая является методом стандартной коллекции TStringList.

Процедура Button5Click производит вызов диалогового окна, с помощью которого пользователь указывает расположение файла программы isaxml для записи преобразования на язык FBD:

procedure TForm1.Button5Click(Sender: TObject);

begin

DataModule2.SaveDialog1.Title := 'Выберите выходной файл на языке FBD';

DataModule2.SaveDialog1.Filter := 'IsaXml Files(\*.isaxml)|\*.isaxml';

DataModule2.SaveDialog1.FilterIndex := 0;

DataModule2.SaveDialog1.FileName := '';

if (DataModule2.SaveDialog1.Execute()) then

Form1.Edit3.Text := DataModule2.SaveDialog1.FileName;

end;

Процедура Button1Click запускает процесс преобразования логических формул на язык функциональных блоковых диаграмм FBD:

procedure TForm1.Button1Click(Sender: TObject);

var

\_tree: Tree;

\_message, output: String;

f: TextFile;

begin

AssignFile(f, edit1.Text);

Reset(f);

while not Eof(f) do

begin

readln(f, \_message);

\_message := StringReplace(\_message, ' ', '', [rfReplaceAll]);

output := Copy(\_message, 1, IndexOfAny(\_message, '=') - 1);

\_message := Copy(\_message, IndexOfAny(\_message, '=') + 1, MAXINT);

\_tree := Tree.Create;

\_tree.root := Node.Create;

\_tree.value := output;

view := '';

buildTree(\_message, \_tree.root, brackets(\_message), false);

// запись в файл схемы блоков (foreach tree = для каждой строки формул)

functionWriteFilePrepare(\_tree);

end;

functionWriteFile();

ShowMessage('Выполнено');

end;

Внутри данной процедуры каждая логическая формула обрабатывается построчно. Сначала удаляются все пробелы с помощью стандартной процедуры StringReplace. Затем формула преобразуется в структуру в виде дерева с помощью процедры buildTree, которая была рассмотрена ранее. Далее вызывается процедура functionWriteFilePrepare, которая преобразует каждый узел дерева в конструкцию для записи в isaxml файл:

procedure functionWriteFilePrepare(\_tree: Tree);

var

k: Integer;

begin

id := id + 1;

setIDs(\_tree.root);

findVariable(\_tree.root);

vars.Add(\_tree.value);

if (\_tree.root.\_not = false) then for k := 0 to (count\_box + count\_vars + 1) do buff.Add('0')

else for k := 0 to (count\_box + count\_vars + 2) do buff.Add('0');

width := ((count\_box + 3 - width\_temp) \* 40);

width\_temp := count\_box;

height := height + 20;

variables.Add('@VAR:' + IntToStr(\_tree.root.id - 1) + ',P=(' + IntToStr(width) + ',' + IntToStr(height) + '),S=(32,4),C=(1,1),X=' + \_tree.value);

arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_tree.root.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_tree.root.id - 1) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

width := width - 32;

drawFormula(\_tree.root);

end;

Сначала в данной процедуре происходит присвоение уникального идентификатора каждому узлу. Это присвоение происходит с помощью процедуры setIDs:

procedure setIDs(\_node: Node);

begin

\_node.id := id;

if \_node.\_not = true then begin \_node.id := id + 1; id := id + 2; end;

if \_node.\_not = false then id := id + 1;

if \_node.left <> nil then setIDs(\_node.left);

if \_node.right <> nil then setIDs(\_node.right);

end;

Затем, с помощью процедуры findVariable, проиходит добавление каждой переменной в словарь переменных для isaxml:

procedure findVariable(\_node: Node);

begin

if (\_node.left = nil) and (\_node.right = nil) then

begin

count\_vars := count\_vars + 1;

vars.Add(\_node.symbol);

end;

if (\_node.left <> nil) or (\_node.right <> nil) then

begin

count\_box := count\_box + 1;

if (\_node.\_not = true) then count\_box := count\_box + 1;

if (\_node.left <> nil) then findVariable(\_node.left);

if (\_node.right <> nil) then findVariable(\_node.right);

end;

end;

Далее, с помощью процедуры drawFormula, каждый узел дерева преобразуется в формат записи конструкций isaxml:

procedure drawFormula(\_node: Node);

begin

if (\_node.left = nil) and (\_node.right = nil) then

begin

if (\_node.\_not = true) then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id - 1) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 4) + '),S=(24,8),C=(1,1),X=NOT' + #13 + 'T=0');

arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id - 1) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

width := width - 40;

end;

variables.Add('@VAR:' + IntToStr(\_node.id) + ',P=(' + IntToStr(width) + ',' + IntToStr(height) + '),S=(32,4),C=(1,1),X=' + \_node.symbol);

height := height + 4;

end;

if (\_node.left <> nil) or (\_node.right <> nil) then

begin

if (\_node.\_not = true) then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id - 1) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 4) + '),S=(24,8),C=(1,1),X=NOT' + #13 + 'T=0');

arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id - 1) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

width := width - 40;

end;

if (\_node.symbol = '\*') then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 2) + '),S=(24,12),C=(2,1),X=AND' + #13 + 'T=0');

if \_node.left.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

if \_node.right.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

end

else if (\_node.symbol = '+') then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 2) + '),S=(24,12),C=(2,1),X=OR' + #13 + 'T=0');

if \_node.left.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

if \_node.right.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

end;

width := width - 40;

if (\_node.left <> nil) then drawFormula(\_node.left);

if (\_node.right <> nil) then drawFormula(\_node.right);

end;

end;

На последнем этапе выполняется процедура functionWriteFile, которая преобразует файл программы FBD, заполняя словарь переменных и записывая конструкции, преобразованные ранее в нужный формат:

procedure functionWriteFile();

var

str, str\_out: TStringList;

f: TextFile;

j: Integer;

filename1: String;

begin

str := TStringList.Create;

str\_out := TStringList.Create;

filename1 := Unit1.Form1.Edit3.Text;

AssignFile(f, filename1); // Prog1.isaxml для записи

str.LoadFromFile(filename1, TEncoding.Default); // Prog1.isaxml для чтения

str\_out.Add('<?xml version="1.0" encoding="utf-8"?>'); // <?xml version="1.0" encoding="utf-8"?>

str\_out.Add(str[1]); // <Pou FileVersion= ........>

str\_out.Add(str[2]); // <Program />

str\_out.Add(' <LocalVars>'); // <LocalVars />

for j := 0 to (vars.Count - 1) do

begin

if (str\_out.IndexOf(' <Variable Name="' + vars[j] + '" DataType="BOOL" InitialValue="" Comment="" Address="" Kind="Var" Alias="" AccessRights="ReadWrite" StringSize="0" />') = -1) then

str\_out.Add(' <Variable Name="' + vars[j] + '" DataType="BOOL" InitialValue="" Comment="" Address="" Kind="Var" Alias="" AccessRights="ReadWrite" StringSize="0" />');

end;

str\_out.Add(' </LocalVars>');

str\_out.Add(' <PouBody><![CDATA[PROGRAM Prog1');

str\_out.Add('#info= FBD');

str\_out.Add('@@NBID=' + IntToStr(count\_box + count\_vars + 1));

for j := 0 to (boxs.Count - 1) do

begin

str\_out.Add(boxs[j]);

end;

for j := 0 to (variables.Count - 1) do

begin

str\_out.Add(variables[j]);

end;

for j := 0 to (arcs.Count - 1) do

begin

str\_out.Add(arcs[j]);

end;

str\_out.Add('#end\_info');

str\_out.Add('#info= ID\_MAX');

str\_out.Add('NextId=' + IntToStr(count\_box + count\_vars + 1));

str\_out.Add('#end\_info');

str\_out.Add('END\_PROGRAM]]></PouBody>');

str\_out.Add('</Pou>');

for j := 0 to (str\_out.Count - 1) do

str\_out[j] := TranslitRus2Lat(str\_out[j]);

str\_out.SaveToFile (filename1, TEncoding.Default);

end;

## 4.5 Вывод

В данном разделе были рассмотрены исходные данные на разработку программного средства, проанализированы файлы, предназначенные для преобразования логических формул в программу на языке FBD, описан интерфейс, алгоритм работы и текст исходного кода разработанной программы Logical\_Formulas\_Converter.

# 5 Руководство пользователя программы Logical\_Formulas\_Converter

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

*58*

*ДП 05.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консульт.*

*Акунович С.И.*

*Н. контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Руководство пользователя*

*Лит.*

*Листов*

7

*БГТУ 84419015, 2015*

Программа Logical\_Formulas\_Converter имеет простой интерфейс. Для того, чтобы начать работу с программой преобразования, следует создать необходимые для этого файлы.

## 5.1 Исходные файлы

Исходный файл с логическими формулами создается как обычный текстовый файл с расширением txt. На рисунке 5.1 приведен пример исходного файла.



Рисунок 5.1 – Исходный файл с алгоритмом на языке Геракл.

Далее следует создать пустой текстовый файл в формате TXT для последующей записи в него функциональных выражений. Теперь для преобразования логических формул в функциональные выражения с помощью программы Logical\_Formulas\_Converter все готово и ее можно запустить.

*У*

*58*

## 5.2 Главное окно программы

Главное окно программы отображено на рисунке 5.2.



Рисунок 5.2 – Главное окно программы Logical\_Formulas\_Converter.

Далее нужно открыть в исходный файл с логическими формулами. Для этого надо нажать на соответствующую кнопку «Выбор» и в появившемся диалоговом окне выбрать нужный текстовый файл (рисунок 5.3).

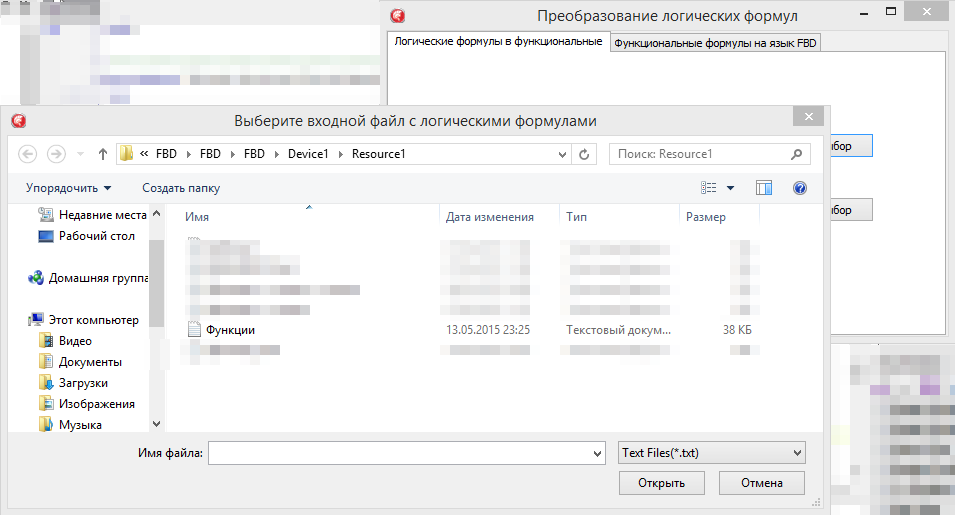


Рисунок 5.3 – Выбор файла с логическими формулами

После открытия исходного файла следует указать путь к файлу, куда будут записаны функциональные выражения. Для этого надо нажать на соответствующую кнопку «Выбор» и в появившемся диалоговом окне выбрать нужный текстовый файл (рисунок 5.4).

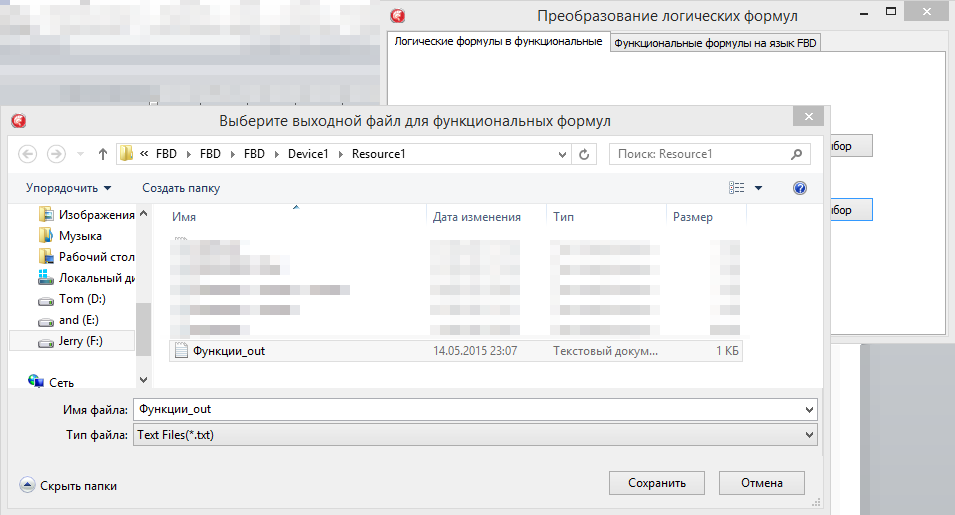


Рисунок 5.4 – Выбор файла для записи функциональных выражений.

## 5.3 Преобразование формул в функционый вид

Для запуска процесса преобразования надо нажать на кнопку «Преобразовать». Сразу после его окончания появляется небольшое окно, информирующее пользователя об успешном безошибочном завершении процесса преобразования. (рисунок 5.5).

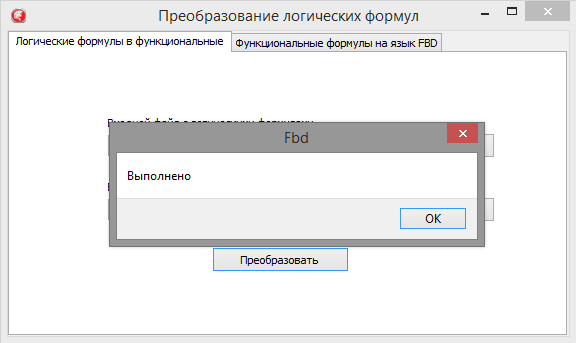


Рисунок 5.5 – Информационное окно программы Logical\_Formulas\_Converter.

## 5.4 Преобразование в функциональные блочные диаграммы

Для преобразования на язык функциональных блочных диаграмм FBD нужно предварительно создать соответствующий проект FBD. Для этого следует проделать сладующие шаги: File 🡪 New 🡪 Project; в появившемся окне выбрать тип проекта «Simulator» и нажать «OK». Далее в обозревателе решений следует выделить элемент «Programs», который расположен по следющему пути: ProjectName 🡪 Device1 🡪 Resouce1 🡪 Programs. Для создания программы FBD, следует правой кнопкой мыши нажать по элементу «Programs» и в секции «New» выбрать «New FBD: Function Block Diagram» - появится программа «Prog1». Результат выполнения данных действий приведен на рисунке 5.6.

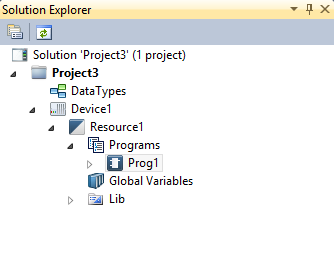


Рисунок 5.6 – Создание программы FBD.

Теперь для преобразования логических формул на язык функциональных блочных диаграмм FBD все готово и его можно запустить. Для этого следует перейти на вторую вкладку программы Logical\_Formulas\_Converter «Функциональные формулы на язык FBD». Вторая вкладка отображена на рисунке 5.7.

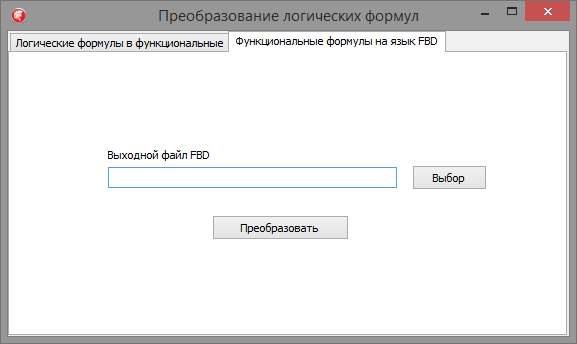


Рисунок 5.7 – Вкладка «Функциональные формулы на язык FBD».

Далее нужно открыть файл Prog1.isaxml для преобразования. Для этого надо нажать на соответствующую кнопку «Выбор» и в появившемся диалоговом окне выбрать нужный файл (рисунок 5.8).

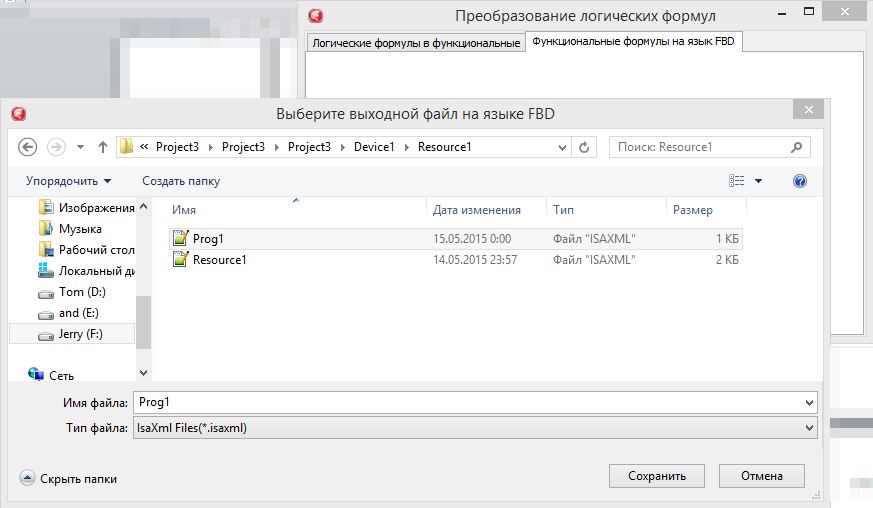


Рисунок 5.8 – Выбор файла Prog1.isaxml для преобразования.

Далее для запуска процесса преобразования надо нажать на кнопку «Преобразовать». Сразу после его окончания появляется небольшое окно, информирующее пользователя об успешном безошибочном завершении процесса преобразования (рисунок 5.9).

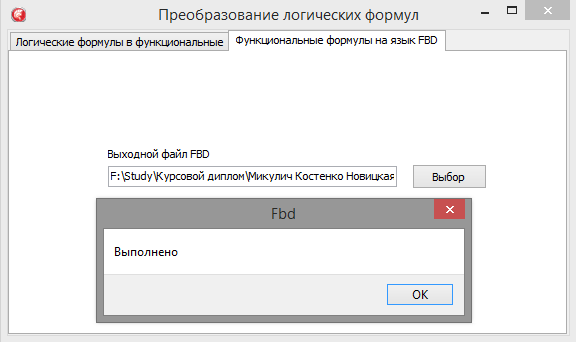


Рисунок 5.9 – Информационное окно программы Logical\_Formulas\_Converter.

Сразу после преобразования, среда ISaGRAF уведомит, что файл Prog1.isaxml был изменен, и предложит обновить содержимое программы (рисунок 5.10).

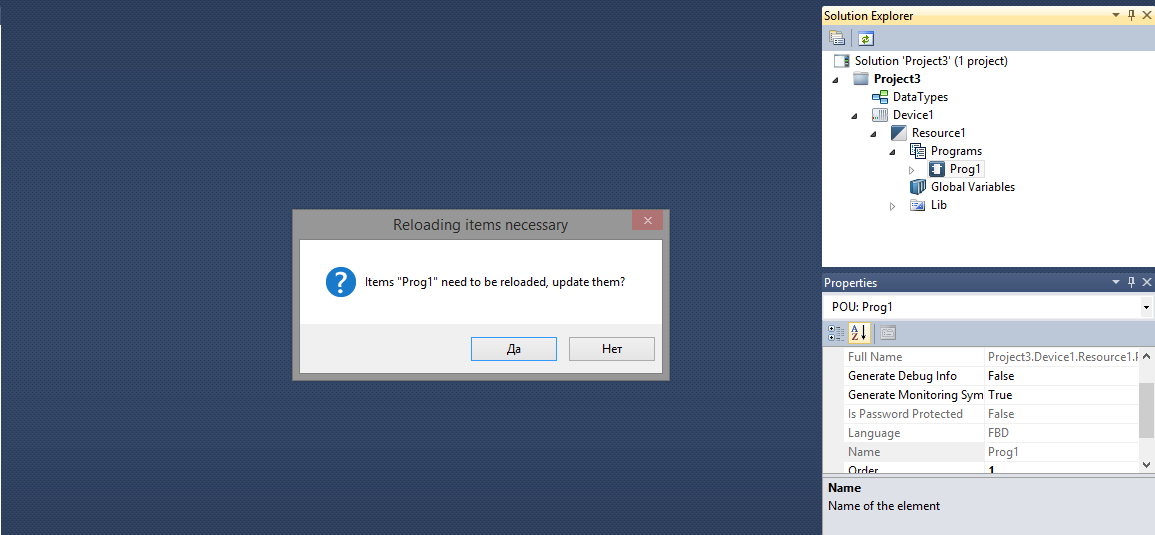


Рисунок 5.10 – Информационное окно ISaGRAF.

Далее следует нажать «Да» и открыть программу «Prog1». Результат отображен на рисунке 5.11. Сравнив запись логических формул с появившейся блок-схемой в программе на языке FBD, можно убедиться в правильности работы программы Logical\_Formulas\_Converter.

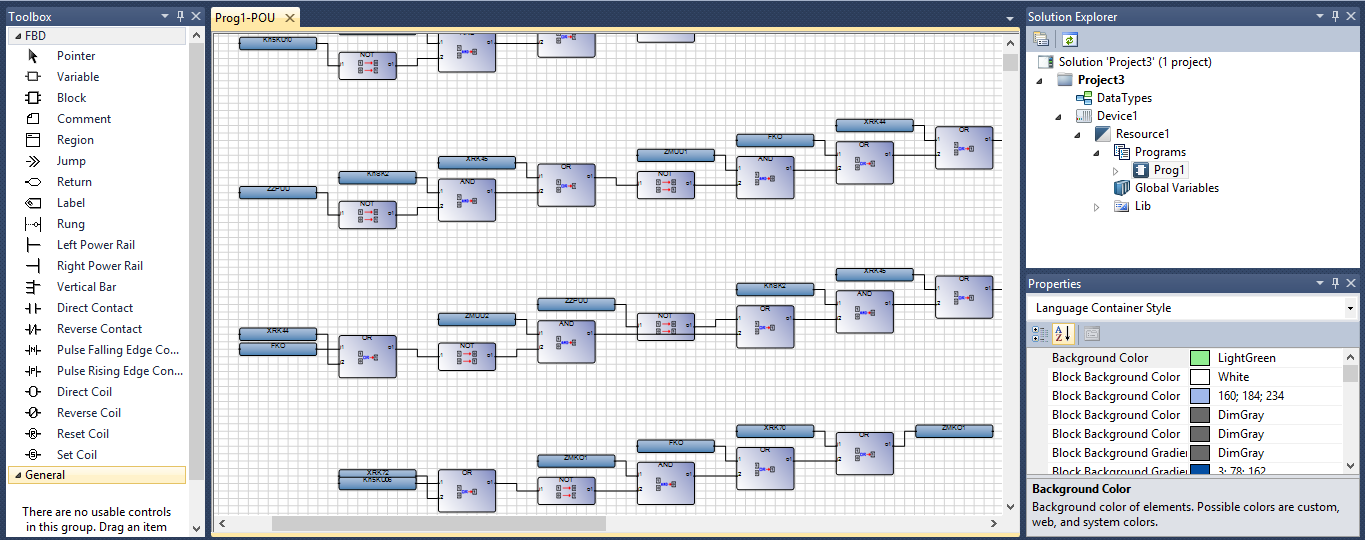


Рисунок 5.11 – Программа Prog1 после преобразования исходных файлов

Для проверки следует также открыть словарь программы, щелкнув дважды по элементу «Локальные переменные» в раскрывающемся списке программы Prog1 (рисунок 5.12). Там создались все переменные соответствующего типа данных, применяемые в исходных логических формулах.

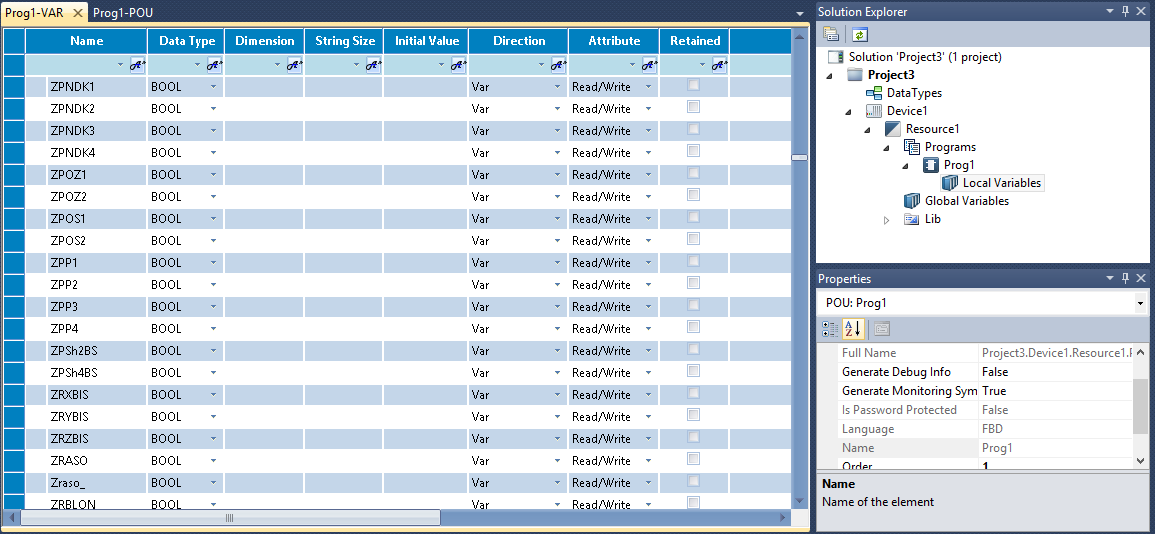


Рисунок 5.12 – Словарь программы после преобразования исходных файлов

В результате выполнения программы Logical\_Formulas\_Converter, текстовые логические формулы были правильно преобразованы в визуальную программу на языке FBD в среде разработки ISaGRAF 6.1. Структура программы правильно построена согласно исходным данным логических формул. Словарь программы содержит все используемые переменные. Учитывая всё это, можно сделать вывод, что цель преобразования была полностью достигнута.

# 6 Экономический раздел

Особенностью современных бизнес процессов в любой отрасли общественной деятельности является автоматизация сбора и обработки информации для принятия управленческих решений. Вместе с тем, автоматизация невозможна без использования программных продуктов. Решения любой информационной задачи связано с применением не только системных программ, но и разнообразных программных средств-приложений.

Разработка проектов программного обеспечения требует затрат разнообразных и, нередко значительных объемов, ресурсов (трудовых, материальных, финансовых). В связи с этим, разработка и реализация каждого проекта должна быть обоснована, как технически, так и экономически.

Проект стоит разрабатывать, если он дает определенные преимущества по сравнению с известными передовыми аналогами или, в крайнем случае, по сравнению с существующей практикой. Поэтому, до того, как приступить к разработке проекта программного обеспечения, специалист должен, используя соответствующие методы, найти наиболее рациональное программное решение, обеспечивающее высокий технический уровень программы и дающее существенную экономию ресурсов, как при разработке проекта в научно-технической организации (у разработчика), так и при его реализации у пользователя (покупателя, заказчика).

Основной целью экономического раздела является экономическое обоснование целесообразности разработки программного средства (ПС), представленного в дипломном проекте. В этом разделе пояснительной записки проводится расчет затрат на всех стадиях разработки, а также анализ экономического эффекта в связи с использованием данного программного средства.

## 6.1 Общая характеристика программного средства

Разрабатываемое программное средство под названием Logical\_Formulas\_Converter представляет собой приложение, предназначенное для преобразования логических формул в функциональные лочные диаграммы на языке программирования FBD, представленные в формате проекта в среде разработки ISaGRAF 6.1.

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

65

*ДП 06.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консульт.*

*Гринчук И.С*

*Н. контр.*

*Жиляк Н.А*

*Утверд.*

*Урбанович П.П.*

*Экономический раздел*

*Лит.*

*Листов*

9

*БГТУ 84419015, 2015*

Программа относится к программным средствам общего назначения первой группы сложности.

Группа новизны разрабатываемого программного средства – А (принципиально новое ПС ВТ, не имеющее доступных аналогов).

*У*

65

## 6.2 Исходные данные для расчета

Исходные данные для расчетов выбраны из установленных нормативов и приведены в таблице 6.1.

Таблица 6.1– Исходные данные для расчетов.

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Значение |
| Коэффициент изменения скорости обработки информации | ед. | Кск | 0,6 |
| Численность разработчиков | чел. | Чр | 1 |
| Тарифная ставка 1-го разряда в организации | руб. | ОТ1 | 2 100 000 |
| Тарифный коэффициент | ед. | Кт | 2,45 |
| Коэффициент естественных потерь рабочего времени | ед. | Кп | 1,2 |
| Коэффициент премирования | ед. | Кпр | 1 |
| Норматив дополнительной заработной платы | % | Нд.з | 10 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нф.с.з.н | 34 |
| Ставка отчислений по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний | % | Нб.г.с | 0,3 |
| Стоимость одного машино-часа | руб. | См.ч | 506,42 |
| Норматив прочих прямых затрат | % | Нп.з | 10 |
| Норматив общепроизводственных и общехозяйственных расходов | % | Нобп, обх | 100 |
| Норматив расходов на сопровождение и адаптацию | % | Нр.с.а | 10 |
| Уровень рентабельности | % | Урент | 33 |
| Ставка НДС | % | Нндс | 20 |

## 6.3 Методика обоснования цены

### 6.3.1 Определение объема программного средства

Объем ПС определяется путем подбора аналогов на основании классификации типов ПС, каталога функций ПС и аналогов ПС в разрезе функций, которые постоянно обновляются и утверждаются в установленном порядке.

На основании информации о функциях разрабатываемого ПС по каталогу функций определяется объем функций. Список функций на разрабатываемое ПС ВТ приведен в таблице 6.2.

Таблица 6.2 – Содержание и объем функций на разрабатываемое ПС ВТ

|  |  |  |
| --- | --- | --- |
| № функции | Содержание функции | Объем, условных машино-команд |
| 101 | Организация ввода информации | 100 |
| 102 | Контроль, предварительная обработка и ввод информации | 520 |
| 103 | Анализ входного языка (синтаксический и семантический) | 630 |
| 104 | Преобразование операторов входного языка в команды другого языка | 1 050 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 220 |
| 111 | Управление вводом/выводом | 2700 |
| 207 | Манипулирование данными | 8400 |
| 301 | Формирование последовательного файла | 340 |
| 305 | Обработка файлов | 750 |
| 306 | Обработка файлов в диалоговом режиме | 2400 |
| 307 | Совместная обработка группы файлов | 4900 |
| 308 | Управление файлами | 5130 |
| 506 | Обработка ошибочных и сбойных ситуаций | 430 |
| 507 | Обеспечение интерфейса между компонентами | 730 |
| 705 | Формирование и вывод на внешние носители | 2650 |
| 707 | Графический вывод результатов | 300 |
|  | ИТОГО | 31250 |

Общий объем программного средства (ПС) рассчитывается по формуле

 (6.1)

где *Vо* – общий объем ПС;

*Vi* – объем *i*-ой функции ПС;

*n* – общее число функций.

На основании имеющейся информации о функциях разрабатываемого ПС определяется его общий объем (в условных машинных командах).

*Vo* = 31 250 (условных машино-команд).

Скорректированный объем функций вычисляется по формуле

*Vo'* = *Vo ·Кск*, (6.2)

где *Vo* – общий объем ПС, условных машино-команд;

*Кск*– коэффициент изменения скорости обработки информации.

*Vo'* = 31 250 ⋅ 0,6 = 18 750 (условных машино-команд).

### 6.3.2 Расчет трудоемкости выполняемой работы

Определение трудоемкости необходимо для дальнейшего расчета суммы основной заработной платы.

Сумма основной заработной платы рассчитывается на основе численности специалистов, соответствующих тарифных ставок и фонда рабочего времени. Причем численность специалистов, календарные сроки разработки программы и фонд рабочего времени определяются по укрупненным нормам времени на разработку, сопровождение и адаптацию программного средства, или экспертным путем. В мелких и средних научно-технических организациях трудоемкость, численность исполнителей и сроки разработки программного средства определяются экспертным путем с использованием данных по базовым моделям.

При определении трудоемкости программного средства учитываются объем программного средства (в тысячах строк условного кода), объем документации (тыс. строк), новизна и сложность программного средства, язык программирования, степень использования типовых (стандартных программ).

На основании общего объема программного средства определяется нормативная трудоемкость (*Тн*) по таблицам. Нормативная трудоемкость устанавливается с учетом сложности программного средства. Выделяется три группы сложности, в которых учтены следующие составляющие программного средства: языковой интерфейса, ввод-вывод, организация данных, режим работы, операционная и техническая среда. Кроме того, устанавливаются дополнительные коэффициенты сложности программного средства.

С учетом дополнительного коэффициента сложности *Ксл* рассчитывается общая трудоемкость программного средства

, (6.3)

где *То* – общая трудоемкость, человеко-дней;

*Тн* – нормативная трудоемкость, человеко-дней;

*Ксл* – дополнительный коэффициент сложности, ед.

*Тн* = 520 человеко-дней – по данным приведенным в методическом пособии приложении 4 таблице 4.1;

*Ксл* = 0,07 – по данным приведенным в методическом пособии приложении 5 таблице 5.2.

*То*= 520 ⋅ (1 + 0,07) = 556,4 (человеко-дней).

### 6.3.3 Расчет основной заработной платы

Месячная тарифная ставка каждого исполнителя определяется путем умножения действующей месячной тарифной ставки 1-го разряда на тарифный коэффициент, соответствующий установленному тарифному разряду

** (6.4)

где *OTj* – тарифная ставка работника за месяц, руб.;

*OTI* – тарифная ставка 1-го разряда за месяц, руб.;

*Кт* – тарифный коэффициент, раз.

*OTj* = 2 100 000 ⋅ 2,45 = 5 145 000 (руб.).

Основная заработная плата исполнителей на конкретное ПС рассчитывается по формуле

 (6.5)

где *TСо.з* – основная заработная плата, руб.;

*TСз.д* – тарифная ставка за день (ОТ разделить на 21.5), руб.;

*То* – общая трудоемкость ПС, человеко-дней;

*Кп* – коэффициент естественных потерь рабочего времени, ед.;

*Кпр* – коэффициент премирования, ед.

*TСо.з* = 239 302 ⋅ 556,4 ⋅ 1,2 ⋅ 1 = 159 777 159 (руб.).

### 6.3.4 Расчет дополнительной заработной платы

Дополнительная заработная плата на конкретное ПС включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по нормативу в процентах к основной заработной плате

, (6.6)

где *TСд.з* – дополнительная заработная плата на конкретное ПС, руб.;

*Нд.з* – норматив дополнительной заработной платы, %.

*TСд.з* = 159 777 159 ⋅ 10 / 100 = 15 977 715 (руб.).

### 6.3.5 Расчет отчислений на социальные цели

Отчисления в Фонд социальной защиты населения (ФСЗН) и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей

 , (6.7)

где *Сф.с.з.н.о* – сумма отчислений в Фонд социальной защиты населения и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний (фонд Белгосстрах), руб.;

*Нф.с.з.н.о* – норматив отчислений в Фонд социальной защиты населения и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний, %.

*Сф.с.з.н.о*  = (159 777 159 + 15 977 715) ⋅ 34 / 100 = 59 756 657 (руб.).

### 6.3.6 Расчет расходов на материалы

Расходы на материалы определяются с учетом действующих нормативов. По данной статье отражаются расходы на бумагу, диски, флэш-карты и другие материалы, необходимые для разработки ПС. Нормы расхода материалов в суммарном выражении определяются в расчете на 100 строк исходного кода. Сумма затрат материалов рассчитывается по формуле

, (6.8)

где *СМ* – сумма расходов на материалы, руб.;

*НМ* – норма расхода материалов в расчете на 100 строк исходного кода ПС, руб.

*СМ* = 380 ⋅ 18 750 / 100 = 71 250 (руб.).

### 6.3.7 Расчет расходов на оплату машинного времени

Расходы включают оплату машинного времени, необходимого для разработки и отладки ПС, которое определяется по нормативам (в машино-часах) на 100 строк исходного кода машинного времени в зависимости от характера решаемых задач и типа ПЭВМ

, (6.9)

где *СМ.В* – сумма расходов на оплату машинного времени, руб.;

*СМ.Ч* – стоимость одного машино-часа, руб.;

*НМ.В* – норматив расхода машинного времени на отладку 100 строк исходного кода, машино-часов.

*СМ.В* = 506,42 ⋅ 18 750 ⋅ 12 / 100= 1 139 445 (руб.).

### 6.3.8 Расчет прочих прямых затрат

Расходы на конкретное ПС включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы, определяются по нормативу в процентах к основной заработной плате

, (6.10)

где *Сп.з*– сумма прочих затрат, руб.;

*Нп.з*  – норматив прочих затрат в целом по организации, %.

*Сп.з* = 159 777 159 ⋅ 10 / 100 = 15 977 715 (руб.).

### 6.3.9 Расчет общепроизводственных и общехозяйственных расходов

Данные затраты, связанные с необходимостью содержания аппарата управления, а также с расходами на общехозяйственные нужды, относятся на конкретное ПС по нормативу в процентном отношении к основной заработной плате исполнителей

, (6.11)

где *Собп,обх* – сумма накладных расходов, руб.;

*Нобп,обх* – норматив накладных расходов в целом по организации, %.

*Собп,обх* = 159 777 159 ⋅ 100 / 100 = 159 777 159 (руб.).

### 6.3.10 Расчет суммы расходов на разработку программного средства

Общая сумма расходов на ПС рассчитывается по формуле

 (6.12)

где *Ср* – сумма расходов на разработку ПС ВТ, руб.;

*Ср*= 159 777 159 + 15 977 715 + 59 756 657 + 71 250 + 1 139 445 + 15 977 715 +

+ 159 777 159 = 412 477 100 (руб.).

### 6.3.11 Расчет расходов на сопровождение и адаптацию

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС, которые определяются по нормативу

, (6.13)

где *Ср.с.а* – сумма расходов на сопровождение и адаптацию ПС ВТ, руб.;

*Нр.с.а* – норматив расходов на сопровождение и адаптацию, %.

*Ср.с.а*= 412 477 100 ⋅ 10 / 100 = 41 247 710 (руб.).

### 6.3.12 Расчет общей суммы расходов

Общая сумма расходов на разработку с затратами на сопровождение и адаптацию ПС определяется по формуле

, (6.14)

где *Сп* – полная себестоимость ПС ВТ, руб.

*Сп*= 412 477 100 + 41 247 710 = 453 724 810 (руб.).

### 6.3.13 Определение цены, оценка эффективности

Отпускная цена определяется на основании цены разработчика, которая формируется на основе показателя рентабельности продукции. Рентабельность и прибыль по создаваемому ПС определяются исходя из результатов анализа рыночных условий, переговоров с заказчиком (потребителем) и согласования с ним отпускной цены, включающей дополнительно налог на добавленную стоимость.

Прибыль от реализации ПС рассчитывается по формуле

, (6.15)

где *Ппс* – прибыль от реализации ПС, руб.;

*Урент* – уровень рентабельности ПС, %;

*Ппс*= 453 724 810 ⋅ 33 / 100 = 149 729 187 (руб.).

Цена разработчика ПС без налогов находится по следующей формуле:

, (6.16)

где *Цр*– прогнозируемая цена разработчика ПС, руб.;

*Цр*  = 453 724 810 + 149 729 187 = 603 453 997 (руб.).

Сумма налога на добавленную стоимость рассчитывается из соотношения:

, (6.17)

где *НДС* – сумма налога на добавленную стоимость, руб.;

*Нндс* – ставка НДС, %.

*НДС* = 603 453 997 ⋅ 20 / 100 = 120 690 799 (руб.).

Планируемая отпускная цена с НДС вычисляется по следующей формуле:

, (6.18)

где *ЦсНДС* – планируемая отпускная цена, руб.

*ЦсНДС* = 603 453 997 + 120 690 799 = 724 144 796 (руб.).

## 6.4 Выводы

Таблица 6.3 содержит результаты расчетов экономических показателей, проведенных в данном разделе.

Таблица 6.3 – Расчет затрат на создание ПС и отпускной цены

|  |  |  |
| --- | --- | --- |
| Наименование показателей | Обозначения | Рассчитанные значения показателей, руб. |
| Основная заработная плата | То.з | 159 777 159 |
| Дополнительная заработная плата | ТСд.з | 15 977 715 |
| Отчисления в фонд социальной защиты населения и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний | Сф.с.з.н.о | 59 756 657 |
| Расходы на материалы | См | 71 250 |
| Расходы на оплату машинного времени | См.в | 1 139 445 |
| Прочие затраты | Сп.з | 15 977 715 |
| Общепроизводственные затраты | Собп, обх | 159 777 159 |
| Производственная себестоимость (расходы на разработку ПС) | Ср | 412 477 100 |
| Расходы на сопровождение и адаптацию | Ср.с.а | 41 247 710 |
| Полная себестоимость ПС | Сп | 453 724 810 |
| Плановая рибыль | Ппс | 149 729 187 |
| Цена без НДС | Цр | 603 453 997 |
| Отпускная цена с НДС | ЦсНДС | 724 144 796 |

Из таблицы 6.3 следует, что большую часть отпускной цены разрабатываемого программного средства составляют накладные расходы, основная заработная плата и расходы на разработку программного средства. При этом дополнительных расходов на спецоборудование не предвидится.

Цена ПС с учетом НДС составляет 724 144 796 руб., при реализации будет получена прибыль около 150 млн. руб., рентабельность продукции – 33%, что позволит обеспечить расширенное воспроизводство ресурсов организации.

В результате применения нового программного средства пользователь может понести значительные капитальные затраты на приобретение и освоение программного средства, на пополнение оборотных средств. Однако, если приобретенное программное средство будет в достаточной степени эффективнее базового, то дополнительные капитальные затраты быстро окупятся. Эффект может быть достигнут за счет сокращения объема программного средства (уменьшения количества машинных команд, количества строк и т. д.), снижение трудоемкости подготовки данных, обработки информации, анализа результатов, уменьшения расходов машинного времени и материалов.

Для определения экономического эффекта от использования нового программного средства у потребителя, необходимо сравнить расходы по всем основным статьям затрат на эксплуатацию нового программного средства (расходы на заработную плату с начислениями, материалы, машинное время) с расходами по соответствующим статьям базового варианта. При этом за базовый вариант следует принимать ручной способ решения задач, т.к. аналогов разрабатываемого программного средства не существует. При сравнении базового и нового вариантов программного средства в качестве экономического эффекта будет выступать общая экономия всех видов ресурсов относительно базового варианта. При этом создание нового программного средства окажется экономически целесообразным лишь в том случае, если все капитальные затраты окупятся за счет получаемой экономии в ближайшие 1 – 2 года.

Экономический эффект применения нового программного средства главным образом заключается в существенной экономии средств, расходующихся на ручное создание функциональных блочных диаграмм в среде разработки ISaGRAF 6.1.

# 7 Мероприятия по охране труда и безопасности жизнедеятельности

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

74

*ДП 07.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консульт.*

*Акунович С.И.*

*Н. Контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Мероприятия по охране труда и безопасности жизнедеятельности*

*Лит.*

*Листов*

15

*БГТУ 84419015, 2015*

## 7.1 Особенности труда пользователя ПЭВМ

Частая и продолжительная работа за компьютером, не обеспеченная определенными организационно-техническими защитными мерами отрицательно сказывается на здоровье и самочувствии пользователей.

Пользователи ПЭВМ сталкиваются с воздействием таких физически опасных и вредных производственных факторов, как повышенный уровень шума, повышенная температура внешней среды, отсутствие или недостаточная освещенность рабочей зоны, электрический ток, статическое электричество и другие.

Работа на компьютере связана с воздействием таких психофизических факторов, как умственное перенапряжение, перенапряжение зрительных и слуховых анализаторов, монотонность труда, эмоциональные перегрузки.

Воздействие неблагоприятных факторов приводит к снижению работоспособности, вызванное развивающимся утомлением. Появление и развитие утомления связано с изменениями, возникающими во время работы в центральной нервной системе, с тормозными процессами в коре головного мозга.

К наиболее распространенным нарушениям обеспечения безопасных условий труда работающих на компьютерах относятся:

* недостаточные площадь и объем производственного помещения;
* несоблюдение требований, предъявляемых к температуре и влажности рабочих помещений;
* низкий уровень освещенности в помещениях и на рабочих поверхностях аппаратуры;
* повышенный уровень низкочастотных магнитных полей от мониторов;
* произвольная расстановка техники и нарушения требований организации рабочих мест;
* несоблюдение требований к режимам труда и отдыха;
* чрезмерная производственная нагрузка работников;
* отсутствие навыков по снижению влияния психоэмоционального напряжения.

При работе на персональном компьютере пользователи могут подвергаться воздействию различных опасных и вредных производственных факторов, основными из которых являются [5]:

* физические: повышенные уровни: электромагнитного, рентгеновского, ультрафиолетового и инфракрасного излучения; статического электричества;

*У*

запыленности воздуха рабочей зоны; повышенное или пониженное содержание аэроионов в воздухе рабочей зоны; повышенный или пониженный уровень освещенности рабочей зоны и др.;

* химические: содержание в воздухе рабочей зоны оксида углерода, озона, аммиака, фенола, формальдегида и полихлорированных фенилов;
* психофизиологические: напряжение зрения, памяти, внимания; длительное статическое напряжение: большой объем информации, обрабатываемой в единицу времени; монотонность труда; нерациональная организация рабочего места; эмоциональные перегрузки.

Основными видами работ на ПЭВМ с использованием видеодисплейных терминалов (ВДТ) являются: считывание информации с экрана с предварительным запросом; ввод информации; творческая работа в режиме диалога с ПЭВМ. Наибольшая нагрузка на орган зрения имеет место при вводе информации в ПЭВМ. Наибольшее общее утомление вызывает работа в режиме диалога. Наибольшее напряжение вызывает выполнение работы при дефиците времени для принятия решения и особенно если это сопряжено с высокой ответственностью за принятые решения (например, при управлении непрерывными технологическими процессами).

Выполнение многих операций при работе на ПЭВМ требует длительного статического напряжения мышц спины, шеи, рук, ног, что приводит к быстрому развитию утомления. Указанные особенности работы часто усугубляются нерациональной высотой рабочей поверхности стола и сидения, отсутствием опорной спинки и подлокотников, неудобными углами сгибания в плечевом и локтевом суставах при выполнении рабочих движений, неправильным углом наклона экрана, отсутствием пространства и подставки для ног.

Неблагоприятное влияние на условия труда работающих с ВДТ оказывают нерациональное естественное и искусственное освещение помещений и рабочих мест, яркие и темные пятна на рабочих поверхностях, засветка экрана посторонним светом, наличие ярких и блестящих предметов.

Важное значение для предупреждения утомления работников имеют также правильный выбор режима работы видеодисплейного терминала, применение защитных фильтров (с обязательным их заземлением), определение оптимальных и допустимых диапазонов визуальных эргономических параметров видеотерминала, использование светозащитных средств.

Использование фильтров-экранов позволяет снизить зрительное утомление и защитить пользователей от электростатического воздействия и частично от воздействия электрической составляющей электромагнитного поля.

Часто при организации рабочих мест для работающих на ПЭВМ не учитывается, что ВДТ генерирует рентгеновское, радиочастотное, видимое ультрафиолетовое излучение, а также имеют место электромагнитные излучения промышленной частоты. Указанные излучения могут оказывать неблагоприятное воздействие и на соседние рабочие места при их нерациональном размещении.

Наличие электростатического поля приводит к уменьшению содержания отрицательных ионов в воздухе помещения и загрязнению экрана в результате притягивания к нему отрицательных ионов и мелких частиц пыли.

Длительная работа компьютера приводит к снижению концентрации кислорода, повышению концентрации озона. Озон является сильным окислителем, и концентрация его выше предельно допустимых величин может привести к неблагоприятным обменным реакциям организма, изменяя активность ряда ферментов, способствует нарушению зрения.

Важным фактором, оказывающим воздействие на состояние здоровья работающих на ПЭВМ, является аэроионный состав воздуха. Его нарушение ухудшает состав крови, работу органа зрения, иммунной системы.

## 7.2 Обеспечение санитарно-гигиенических условий при использовании ПЭВМ

### 7.2.1 Освещение

Правильно спроектированное и выполненное производственное освещение улучшает условия зрительной работы, снижает утомляемость, способствует повышению производительности труда, благотворно влияет на производственную среду, оказывая положительное психологическое воздействие на работающего, повышает безопасность труда и снижает травматизм.

Недостаточность освещения приводит к напряжению зрения, ослабляет внимание, приводит к наступлению преждевременной утомленности. Чрезмерно яркое освещение вызывает ослепление, раздражение и резь в глазах.

Неправильное направление света на рабочем месте может создавать резкие тени, блики, дезориентировать работающего. Все эти причины могут привести к несчастному случаю или профзаболеваниям, поэтому столь важен правильный расчет освещенности.

Согласно требованиям нормативных документов помещения, с ВДТ и ПЭВМ должны иметь естественное и искусственное освещение.

Естественное освещение должно осуществляться через световые проемы, ориентированные преимущественно на север и северо-восток и обеспечивать коэффициент естественной освещенности (КЕО) не ниже 1,5%.

При выполнении работ категории высокой зрительной точности (наименьший размер объекта различения 0,3...0,5 мм) величина коэффициента естественного освещения (КЕО) должна быть не ниже 1,5%, а при зрительной работе средней точности (наименьший размер объекта различения 0,5.. .1,0 мм) КЕО должен быть не ниже 1,0%.

Искусственное освещение должно осуществляться системой общего равномерного освещения. В случаях преимущественной работы с документами в помещениях эксплуатации ВДТ и ПЭВМ допускается применение системы комбинированного освещения. Освещенность на поверхности стола в зоне размещения документов должна быть 300-500 лк. Местное освещение, при этом, не должно создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк.

В качестве источников искусственного освещения обычно используются люминесцентные лампы, которые попарно объединяются в светильники и располагаются над рабочими поверхностями равномерно [9].

Источники света, такие как светильники и окна, которые дают отражение от поверхности экрана, значительно ухудшают точность знаков и влекут за собой помехи физиологического характера, которые могут выразиться в значительном напряжении, особенно при продолжительной работе. Отражение, включая отражения от вторичных источников света, должно быть сведено к минимуму.

Для защиты от избыточной яркости окон могут быть применены шторы и экраны [9].

Окраска помещений и мебели должна способствовать созданию благоприятных условий для зрительного восприятия, хорошего настроения. В зависимости от ориентации окон рекомендуется следующая окраска стен и пола: окна ориентированы на юг: – стены зеленовато-голубого или светло-голубого цвета; пол – зеленый; окна ориентированы на север: – стены светло-оранжевого или оранжево-желтого цвета; пол – красновато-оранжевый; окна ориентированы на восток: – стены желто-зеленого цвета; пол зеленый или красновато-оранжевый; окна ориентированы на запад: – стены желто-зеленого или голубовато-зеленого цвета; пол зеленый или красновато-оранжевый.

Для внутренней отделки помещений должны использоваться диффузно-отражающие материалы с коэффициентом отражения от потолка - 0,7 - 0,8; для стен 0,5 - 0,6; для пола - 0,3 - 0,5.

### 7.2.2 Параметры микроклимата

Параметры микроклимата могут меняться в широких пределах, в то время как необходимым условием жизнедеятельности человека является поддержание постоянства температуры тела благодаря терморегуляции, т.е. способности организма регулировать отдачу тепла в окружающую среду. Принцип нормирования микроклимата – создание оптимальных условий для теплообмена тела человека с окружающей средой.

Вычислительная техника является источником существенных тепловыделений, что может привести к повышению температуры и снижению относительной влажности в помещении. В помещениях, где установлены компьютеры, должны соблюдаться определенные параметры микроклимата. Производственные помещения, в которых установлены компьютеры, должны оборудоваться системами отопления, кондиционирования воздуха или эффективной приточно-вытяжной вентиляцией. Микроклимат должен соответствовать следующим санитарным нормам:

* температура воздуха в теплый период года – не более 23-25 °С, в холодный – 22-24°С;
* относительная влажность воздуха – 40-60%;
* скорость движения воздуха – 0,1 м/с.

Для повышения влажности воздуха в помещениях следует применять увлажнители воздуха, ежедневно заправлять их дистиллированной или кипяченой водой.

Уровни положительных и отрицательных аэроионов в воздухе помещений с ВДТ, ЭВМ и ПЭВМ должны соответствовать нормам [8], приведенным в таблице 7.1.

Таблица 7.1 – Уровни ионизации воздуха помещений при работе на ВДТ, ЭВМ и ПЭВМ

|  |  |  |
| --- | --- | --- |
| Уровни | Число ионов в 1 см воздуха | |
| n+ | n- |
| Минимально необходимые | 400 | 600 |
| Оптимальные | 1500-3000 | 3000-5000 |
| Максимально допустимые | 50000 | 50000 |

Площадь на одно рабочее место с ВДТ и ПЭВМ должна составлять не менее 6,0 м, а объем – не менее 20,0 м (в учебных заведениях не менее 18 м) [6].

В помещениях ежедневно должна проводиться влажная уборка.

Помещения должны быть оснащены аптечкой первой помощи и углекислотными огнетушителями.

### 7.2.3. Шум и вибрация

Шум ухудшает условия труда оказывая вредное действие на организм человека. Работающие в условиях длительного шумового воздействия испытывают раздражительность, головные боли, головокружение, снижение памяти, повышенную утомляемость, понижение аппетита, боли в ушах и т. д. Такие нарушения в работе ряда органов и систем организма человека могут вызвать негативные изменения в эмоциональном состоянии человека вплоть до стрессовых. Под воздействием шума снижается концентрация внимания, нарушаются физиологические функции, появляется усталость в связи с повышенными энергетическими затратами и нервно-психическим напряжением, ухудшается речевая коммутация. Все это снижает работоспособность человека и его производительность, качество и безопасность труда. Длительное воздействие интенсивного шума (выше 80 дБА) на слух человека приводит к его частичной или полной потере [10]. Уровни звука приведены в таблице 7.2.

Таблица 7.2 – Уровни звука, эквивалентные уровни звука и уровни звукового давления в октавных полосах частот

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Категория нормы шума | Уровни звукового давления, дБ в октавных полосах со сред­негеометрическими частотами, Гц | | | | | | | | | Уровни звука, эквивалентные уровни звука, ДБА |
| 31,5 | 63 | 125 | 250 | 500 | 1000 | 2000 | 4000 | 8000 |
| I | 86 | 71 | 61 | 54 | 49 | 45 | 42 | 40 | 38 | 50 |
| II | 93 | 79 | 70 | 63 | 58 | 55 | 52 | 50 | 49 | 60 |
| III | 96 | 83 | 74 | 68 | 63 | 60 | 57 | 55 | 54 | 65 |
| IV | 103 | 91 | 83 | 77 | 73 | 70 | 68 | 66 | 64 | 75 |

При выполнении работ с ВДТ и ПЭВМ в производственных помещениях уровень вибрации не должен превышать допустимых значений согласно СанПиН 9-131 РБ 2000 [8].

Для снижения уровня шума стены и потолок помещений, где установлены компьютеры, могут быть облицованы звукопоглощающими материалами. Дополнительным звукопоглощением служат однотонные занавеси из плотной ткани гармонирующие с окраской стен и подвешенные в складку на расстоянии 15-20 см от ограждения. Ширина занавеси должна быть в два раза больше ширины окна.

Уровень вибрации в помещениях вычислительных центров может быть снижен путем установки оборудования на специальные виброизоляторы. Шумящее оборудование (АЦПУ, принтеры и д.р.), уровни шума которого превышают допустимые, должно находиться вне помещения с ВДТ, ЭВМ и ПЭВМ.

### 7.2.4. Электромагнитное и ионизирующее излучения

Общеизвестно, что компьютерная техника является источником излучений и электромагнитных полей, потенциально опасных для здоровья человека, особенно при неправильном ее использовании. Особо необходимо отметить, что в спектре электромагнитных полей, создаваемых компьютерами, присутствуют низкочастотные электромагнитные колебания от единиц герц до нескольких десятков герц, частоты которых близки к частотам биоритмов человеческого организма. В этом принципиальное отличие ПЭВМ по их потенциальной экологической опасности в сравнении с обычными бытовыми электроприборами и другими излучающими техническими средствами, которые по роду своего использования могут находиться (как и ПЭВМ) в близком контакте с человеком.

При работе ВДТ уровни напряженности, плотности магнитного потока электромагнитного поля, напряженности электростатического поля согласно СанПиН 9-131 РБ 2000 [8] на расстоянии 50 см от экрана, правой, левой, верхней и тыльной поверхности видеомонитора не должны превышать допустимых значений, приведенных в таблице 7.3

Таблица 7.3 – Допустимые значения параметров неионизирующих электромагнитных излучений

|  |  |
| --- | --- |
| Наименование параметра | Допустимые значения |
| Напряженность электромагнитного поля. Электрическая составляющая, не более: |  |
| диапазон частот 5 Гц-2 кГц; | 25,0 В/м |
| диапазон частот 2-400 кГц | 2,5 В/м |
| Плотность магнитного потока, не более: |  |
| диапазон частот 5 Гц-2 кГц; | 25° нТл |
| диапазон частот 2-400 кГц | 25 нТл |
| Напряженность электростатического поля, не более | 15 кВ/м |

Допустимые уровни напряженности (плотности потока мощности) электромагнитных полей, излучаемых клавиатурой, системным блоком, манипулятором «мышь», беспроводными системами передачи информации на расстояния и иными вновь разработанными устройствами в зависимости от основной рабочей частоты изделия, не должны превышать значений, приведенных в таблице 7.4.

Таблица 7.4 – Допустимые уровни электромагнитных полей

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Диапазон частот | 0,3­-300 кГц | 0,3-3,0 МГц | 3,0 - 30,0 МГц | 30,0-300,0 МГц | 0,3-300 ГГц |
| Допустимый уровень | 25 В/м | 15 В/м | 10 В/м | 3 В/м | 10 мкВт/см |

Допустимые уровни напряженности электрического поля тока промышленной частоты 50 Гц, создаваемые монитором, системным блоком, клавиатурой, изделием в целом, не должны превышать 0,5 кВ/м.

Допустимые уровни напряженности электростатического поля, создаваемые монитором, клавиатурой, системным блоком, манипулятором «мышь», изделием в целом, не должны превышать 15,0 кВ/м.

Интенсивность ультрафиолетового излучения от экрана видеомонитора не должна превышать в диапазоне 0,28-0,315 мкм 0,1-10-3 Вт/м2; в диапазоне 0,315-0,4 мкм - 0,1 Вт/м2. Излучение в диапазоне 0,2-0,28 мкм не допускается.

Конструкция ВДТ, ЭВМ и ПЭВМ должна обеспечивать безопасный для пользователя уровень мощности экспозиционной дозы рентгеновского излучения в любой точке пространства на расстоянии 0,05 м от экрана и частей корпуса ВДТ, ЭВМ или ПВЭМ при любых положениях регулировочных устройств. Уровень мощности экспозиционной дозы рентгеновского излучения не должен превышать на расстоянии 0,5 м от экрана и частей корпуса ВДТ не должен превышать 7,74-10-12 А/кг, что соответствует мощности эквивалентной дозы, равной 100 мкР/ч (0,03 мкР/с) [8].

Компьютеры с жидкокристаллическим экраном не имеют источников мощного электромагнитного излучения и не наводят статического электричества. Однако при использовании блока питания возникает некоторое превышение уровня на частоте 50 Гц, поэтому рекомендуется работать больше с использованием аккумулятора.

Во всех случаях для снижения уровня облучения монитор рекомендуется располагать на расстоянии не ближе 50 см от пользователя.

Для снижения воздействия этих видов излучения рекомендуется применять мониторы с пониженным уровнем излучения (MPR-II, TCO-92, TCO-99), устанавливать защитные экраны, а также соблюдать регламентированные режимы труда и отдыха.

### 7.2.5 Эргономические требования к рабочему месту

Проектирование рабочих мест, оборудованных ПЭВМ и ВДТ, относится к числу важных проблем эргономического проектирования в области вычислительной техники.

Эргономическими аспектами проектирования видеотерминальных рабочих мест, в частности, являются: высота рабочей поверхности, размеры пространства для ног, требования к расположению документов на рабочем месте (наличие и размеры подставки для документов, возможность различного размещения документов, расстояние от глаз пользователя до экрана, документа, клавиатуры и т.д.), характеристики рабочего кресла, требования к поверхности рабочего стола, регулируемость элементов рабочего места [3].

Рабочее место и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы. В частности, при организации рабочего места пользователя ПЭВМ должны быть соблюдены следующие основные условия: оптимальное размещение оборудования, входящего в состав рабочего места и достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения.

Рабочие места должны располагаться таким образом, чтобы естественный свет падал сбоку, преимущественно слева.

Расстояние между рабочими столами с мониторами (в направлении тыла поверхности одного монитора и экрана другого) должно быть не менее 2 м, а между боковыми поверхностями мониторов - не менее 1,2 м.

Рекомендуемое расположение рабочих мест с компьютерами показано на рисунке 7.1 [6].



Рисунок 7.1 – Рекомендуемое расположение рабочих мест с компьютерами

Рабочие места при выполнении творческой работы, требующей значительного умственного напряжения или высокой концентрации внимания, следует изолировать одно от другого перегородками высоты 1,5-2 м.

Главными элементами рабочего места пользователя ПЭВМ являются стол и кресло. Основным рабочим положением является положение сидя. Рабочая поза сидя вызывает минимальное утомление.

Зона досягаемости ограничивается дугами, описываемыми максимально вытянутыми руками при движении их в плечевом суставе. Зона легкой досягаемости – часть моторного поля, ограниченная дугами, описываемыми расслабленными руками при движении их в плечевом суставе. Оптимальная зона досягаемости моторного поля - предплечьями при движении в локтевых суставах с опорой.

Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне легкой досягаемости рабочего пространства.

Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья. При этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию. Поверхность сиденья, спинки и других элементов стула (кресла) должна быть полумягкой, с нескользящим, неэлектризующимся и воздухопроницаемым покрытием, обеспечивающим легкую очистку от загрязнений.

Тип рабочего стула (кресла) должен выбираться в зависимости от характера и продолжительности работы с ВДТ, ЭВМ и ПЭВМ с учетом роста пользователя.

Экран видеомонитора должен находиться от глаз пользователя на оптимальном расстоянии 600-700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов.

Во всех случаях для снижения уровня облучения монитор рекомендуется располагать на расстоянии не ближе 50 см от пользователя.

Уровень глаз при вертикальном расположенном экране ВДТ должен приходиться на центр или 2/3 высоты экрана. Линия взора должна быть перпендикулярна центру экрана (рисунок 7.2а). При работе на клавиатуре необходимо соблюдать правильное положение рук оператора (рисунок 7.2б) [6].



Рисунок 7.2 – Правильная позиция оператора (а) и правильное положение рук оператора при работе на клавиатуре (б)

Большое значение также придается правильной рабочей позе пользователя. При неудобной рабочей позе могут появиться боли в мышцах, суставах и сухожилиях.

Клавиатуру следует располагать на поверхности стола на расстоянии не менее чем 300 мм от края, обращенного к пользователю или на специальной, регулируемой по высоте рабочей поверхности, отделенной от основной столешницы.

### 7.2.6 Режим труда

При работе с персональным компьютером очень важную роль играет соблюдение правильного режима труда и отдыха. В противном случае у персонала отмечаются значительное напряжение зрительного аппарата с появлением жалоб на неудовлетворенность работой, головные боли, раздражительность, нарушение сна, усталость и болезненные ощущения в глазах, в пояснице, в области шеи и руках [1].

Режимы труда и отдыха должны организовываться в зависимости от вида и категории трудовой деятельности.

Виды трудовой деятельности разделяются на три группы: группа А - работа по считыванию информации с экрана ВДТ, ЭВМ и ПЭВМ с предварительным запросом; группа Б - работа по вводу информации; группа В - творческая работа в режиме диалога с ЭВМ. При выполнении в течение рабочей смены работ, относящихся к разным видам трудовой деятельности, за основную работу с ВДТ, ЭВМ и ПЭВМ следует принимать такую, которая занимает не менее 50 % времени в течение рабочей смены или рабочего дня.

Для обеспечения оптимальной работоспособности и сохранения здоровья профессиональных пользователей на протяжении рабочей смены должны устанавливаться регламентированные перерывы. Время регламентированных перерывов в течение рабочей смены следует устанавливать в зависимости от ее продолжительности, вида и категории трудовой деятельности (таблица 7.5).

Таблица 7.5 – Время регламентированных перерывов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория  работы ВДТ, ЭВМ и ПЭВМ | Уровень нагрузки за рабочую смену при видах работ с ВДТ | | | Суммарное время регламентированных перерывов, мин | |
|  | группа А,  количество  знаков | группа Б,  количество  знаков | группа В, ч | При 8-  часовой  смене | при  12-часовой смене |
| I | до 20000 | до 15000 | до 2,0 | 30 | 70 |
| II | до 40000 | до 30000 | до 4,0 | 50 | 90 |
| III | до 60000 | до 40000 | ДО 6,0 | 70 | 120 |

Продолжительность непрерывной работы с ВДТ без регламентированного перерыва не должна превышать 2 ч.

При работе с ВДТ, ЭВМ и ПЭВМ в ночную смену (с 22 до 6 ч), независимо от категории и вида трудовой деятельности, продолжительность регламентированных перерывов должна увеличиваться на 60 мин.

При 8-часовой рабочей смене и работе на ВДТ, ЭВМ и ПЭВМ регламентированные перерывы следует устанавливать:

* для I категории работ – через 2 ч от начала рабочей смены и через 2 ч после обеденного перерыва продолжительностью 15 мин каждый;
* для II категории работ – через 2 ч от начала рабочей смены и через 1,5-2 ч после обеденного перерыва продолжительностью 15 мин каждый или продолжительностью 10 мин через каждый час работы;
* для III категории работ – через 1,5-2 ч от начала рабочей смены и через 1,5-2 ч после обеденного перерыва продолжительностью 20 мин каждым или продолжительностью 15 мин через каждый час работы.

При 12-часовой рабочей смене регламентированные перерывы должны устанавливаться в первые 8 часов работы аналогично перерывам при 8-часовой рабочей смене, а в течение последних 4 ч работы, независимо от категории и вида работ, каждый час продолжительностью 15 мин.

Во время регламентированных перерывов с целью снижения нервно-эмоционального напряжения, утомления зрительного анализатора, устранения влияния гиподинамии и гипокинезии целесообразно выполнять комплексы упражнений.

Работающим на ВДТ, ЭВМ и ПЭВМ с высоким уровнем напряженности во время регламентированных перерывов и в конце рабочего дня показана психологическая разгрузка в специально оборудованных помещениях (комната психологической разгрузки).

Профессиональные пользователи ВДТ, ЭВМ и ПЭВМ должны проходить обязательные предварительные и периодические медицинские осмотры в установленном порядке и не иметь медицинских противопоказаний.

Женщины со времени установления беременности и в период кормления ребенка грудью к выполнению всех видов работ, связанных с использованием ВДТ, ЭВМ и ПЭВМ, не допускаются.

Эффективность перерывов повышается при сочетании с производственной гимнастикой или организации специального помещения для отдыха персонала с удобной мягкой мебелью, аквариумом, зеленой зоной и т.п.

## 7.3 Эргономика пользовательских интерфейсов

### 7.3.1 Основные принципы проектирования интерфейсов

Масштабы распространения компьютеров, все возрастающая интенсификация человеческого труда требуют повышения внимания к проектированию интерфейсов, с тем, чтобы по возможности способствовать устранению или уменьшению стресса, который испытывает человек при работе с компьютером.

Под пользовательским интерфейсом (ПИ) понимают совокупность элементов, позволяющих пользователю программы управлять ее работой и получать требуемые результаты. Это система правил и средств, регламентирующая и обеспечивающая взаимодействие программы с пользователем. При проектировании интерфейсов главная задача - облегчить процесс восприятия и переработки информации.

Одним из важных показателей качества программного обеспечения является удобство его использования. Удобство использования пользовательского интерфейса (usability) - показатель его качества, который определяет общую концепцию удобства при использовании программного обеспечения, логичность и простоту в расположении элементов управления. Термин «юзабилити» можно рассматривать как синоним слова «эргономичность» с той разницей, что последняя определяет минимальность конкретных физических усилий при пользовании вещью, а первая – конечную суммарную степень удобства, меру интеллектуального усилия необходимого для получения полезных качеств этой вещи и скорость достижения положительного результата при управлении ею.

Согласно международному стандарту качества интерфейсов ISO DIS 9241-11 Эргономические требования к офисной работе с визуальными терминалами [11], юзабилити определяется как «степень, в которой продукт может быть использован определенными пользователями для достижения поставленных целей эффективно, экономично и с удовлетворением в заданном контексте использования».

Для того чтобы любой пользователь мог легко освоить пользование программой, ее интерфейс должен быть выполнен в соответствии с выработанными эргономическими принципами. Эти принципы разрабатываются на основе опыта проектирования интерфейсов пользователей с учетом их пожеланий, на основе наблюдений за пользователями при их работе с программными средствами. Данные принципы позволяют рассматривать пользователя программного продукта и сам программный продукт как единую систему, между элементами которой происходят взаимодействия [4]. Человек рассматривается как компонент рабочей системы. Правильное проектирование рабочих систем приводит к снижению рисков при работе, а, следовательно, и стрессов, с одновременным увеличением производительности труда.

Для успешного проектирования рабочих систем необходимо точно определить назначение и требования к разрабатываемой системе, в данном случае к программному средству [7]. Интерфейс должен быть интуитивно понятным, не содержать элементов, которые вводят пользователя в заблуждение или заставляют обратиться в специализированные источники информации для уточнения смысла, должен позволять пользователю быстро получать доступ к имеющейся информации и удобно работать с программой.

Важным моментом при проектировании систем является распределение функций между человеком и системой. Для этого необходимо определить ограничения по функциям человека и машины, выбор исполнителя каждой функции и оптимизировать работу человека и машины. Распределение функций признается удовлетворительным, если рабочая нагрузка человека допустима, а работа осмысленна и мотивирована.

Хороший дизайн пользовательского интерфейса подразумевает, что программа соответствует ожиданиям пользователей о том, как она должна себя вести. Основное достоинство хорошего интерфейса пользователя заключается в том, что пользователь всегда чувствует, что он управляет программным обеспечением, а не программное обеспечение управляет им.

Для создания у пользователя такого ощущения «внутренней свободы» интерфейс должен обладать целым рядом свойств [2]:

1. Естественность интерфейса.Естественный интерфейс - такой, который не вынуждает пользователя существенно изменять привычные для него способы решения задачи. Это, в частности, означает, что сообщения и результаты, выдаваемые приложением, не должны требовать дополнительных пояснений. Использование знакомых пользователю понятий и образов (метафор) обеспечивает интуитивно понятный интерфейс при выполнении его заданий. Метафоры являются своего рода «мостиком», связывающим образы реального мира с теми действиями и объектами, которыми приходится манипулировать пользователю при его работе на компьютере; они обеспечивают «узнавание», а не «вспоминание». Пользователи запоминают действие, связанное со знакомым объектом, более легко, чем они запомнили бы имя команды, связанной с этим действием;
2. Согласованность интерфейса. Согласованность позволяет пользователям переносить имеющиеся знания на новые задания, осваивать новые аспекты быстрее, и благодаря этому фокусировать внимание на решаемой задаче, а не тратить время на уяснение различий в использовании тех или иных элементов управления, команд и т.д. Обеспечивая преемственность полученных ранее знаний и навыков, согласованность делает интерфейс узнаваемым и предсказуемым. Согласованность важна для всех аспектов интерфейса, включая имена команд, визуальное представление информации и поведение интерактивных элементов:

* согласованность в пределах продукта. Одна и та же команда должна выполнять одни и те же функции, где бы она ни встретилась, причем одним и тем же образом. Надо использовать одну и ту же команду, чтобы выполнить функции, которые кажутся подобными пользователю;
* согласованность в пределах рабочей среды. Поддерживая согласованность с интерфейсом, предоставляемым операционной системой, приложение может «опираться» на те знания и навыки пользователя, которые он получил ранее при работе с другими приложениями;
* согласованность в использовании метафор. Если поведение некоторого программного объекта выходит за рамки того, что обычно подразумевается под соответствующей ему метафорой, у пользователя могут возникнуть трудности при работе с таким объектом;

1. Дружественность интерфейса (принцип прощения пользователя). На каждом этапе работы эффективный интерфейс должен разрешать только соответствующий набор действий и предупреждать пользователей о тех ситуациях, где они могут повредить системе или данным; также, у пользователя должна быть возможность отменить или исправить выполненные действия. Даже при наличии хорошо спроектированного интерфейса пользователи могут делать те или иные ошибки. Эти ошибки могут быть как «физического» типа (случайный выбор неправильной команды или данных) так и «логического» (принятие неправильного решения на выбор команды или данных). Эффективный интерфейс должен позволять предотвращать ситуации, которые, вероятно, закончатся ошибками. Он также должен уметь адаптироваться к потенциальным ошибкам пользователя и облегчать ему процесс устранения последствий таких ошибок;
2. Принцип «обратной связи**».** Всегда должна существовать обратная связь для действий пользователя. Каждое действие пользователя должно получать визуальное, а иногда и звуковое подтверждение того, что программное обеспечение восприняло введенную команду; при этом вид реакции, по возможности, должен учитывать природу выполненного действия. Желательно, чтобы время отклика было сопоставимо с реакцией человека. Типичный пользователь способен вытерпеть только несколько секунд ожидания ответной реакции от своего электронного «собеседника». Задержку часто можно «спрятать» от пользователя через многозадачность, позволяя продолжать выполнять другую работу, пока производятся фоновые вычисления. В любом случае пользователь должен иметь подтверждение того, что программа не «зависла», а продолжает работать;
3. Простота интерфейса. Простота интерфейса предполагает не упрощенчество, а обеспечение легкости в его изучении и в использовании. Кроме того, интерфейс должен предоставлять доступ ко всему перечню функциональных возможностей, предусмотренных данным приложением. Реализация доступа к широким функциональным возможностям и обеспечение простоты работы противоречат друг другу. Разработка эффективного интерфейса призвана сбалансировать эти цели. Один из возможных путей поддержания простоты - представление на экране информации, минимально необходимой для выполнения пользователем очередного шага задания. Многословные командные имена или сообщения, непродуманные или избыточные фразы затрудняют пользователю извлечение существенной информации. Другой путь к созданию простого, но эффективного интерфейса -размещение и представление элементов на экране с учетом их смыслового значения и логической взаимосвязи. Это позволяет использовать в процессе работы ассоциативное мышление пользователя;
4. Гибкость интерфейса. Гибкость интерфейса – это его способность учитывать уровень подготовки и производительность труда пользователя. Гибкость предполагает возможность изменения структуры диалога и/или входных данных;
5. Эстетическая привлекательность. Проектирование визуальных компонентов является важнейшей составной частью разработки программного интерфейса. Корректное визуальное представление используемых объектов обеспечивает передачу весьма важной дополнительной информации о поведении и взаимодействии различных объектов. В то же время следует помнить, что каждый визуальный элемент, который появляется на экране, потенциально требует внимания пользователя, которое, как известно, не безгранично. Следует формировать на экране среду, которая не только содействовала бы пониманию пользователем представленной информации, но и позволяла бы сосредоточиться на наиболее важных ее аспектах.

Качество интерфейса сложно оценить количественными характеристиками, однако более или менее объективную его оценку можно получить на основе приведенных ниже частных показателей:

* время, необходимое определенному пользователю для достижения заданного уровня знаний и навыков по работе с приложением (например, непрофессиональный пользователь должен освоить команды работы с файлами не более чем за 4 часа);
* сохранение полученных рабочих навыков по истечении некоторого времени (например, после недельного перерыва пользователь должен выполнить определенную последовательность операций за заданное время);
* скорость решения задачи с помощью данного приложения; при этом должно оцениваться не быстродействие системы и не скорость ввода данных с клавиатуры, а время, необходимое для достижения цели решаемой задачи. Исходя из этого, критерий оценки по данному показателю может быть сформулирован, например, так: пользователь должен обработать за час не менее 20 документов с ошибкой не более 1%;
* субъективная удовлетворенность пользователя при работе с системой (которая количественно может быть выражена в процентах или оценкой по n-бальной шкале).

В разработке интерфейса часто употребляются метафоры, чтобы помочь пользователю познакомиться с приложением, используя ассоциации с уже имеющимся знанием. Некоторые из наиболее известных - это метафора бухгалтерской книги VisiCalc, метафора рабочего стола, впервые примененная Xerox, и метафора чековой книжки из Quicken.

## 7.7 Вывод

В разделе подробно описаны вредные и опасные производственные факторы при работе с персональным компьютером в ходе проектирования и разработки программного средства Logical\_Formulas\_Converter.

Представлены требования к помещениям и рабочим местам, где эксплуатируются ПЭВМ касающиеся освещения, микроклимата, уровня шума и вибрации, излучений и электромагнитных полей, пожарной безопасности. Также описаны общие принципы мероприятий по безопасности жизнедеятельности.

# Заключение

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

89

*ДП 00.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консульт.*

*Акунович С.И.*

*Н. контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Заключение*

*Лит.*

*Листов*

1

*БГТУ 84419015, 2015*

В ходе выполнения дипломного проекта были рассмотрены и проанализированы различные языки описания алгоритмов логического управления.

Для упрощения создания функционалных блочных диаграмм на языке FBD на основе текстовых описаний логических формул были изучены и проанализированы файлы проекта в среде ISaGRAF 6.1. На основе этого анализа, полученных при этом данных и выявленных закономерностях была разработана программа Logical\_Formulas\_Converter, которая способна преобразовывать логические формулы, записанные в текстовом виде в исходном файле, в формат файла Prog1.ISAXML, на основании которого затем происходит построение функциональныз блочных диаграмм FBD в среде ISaGRAF 6,1, а также записываются при этом все необходимые переменные в файл словаря проекта. Автоматизация процесса построения блок-схем алгоритмов на основе логических формул позволяет во много раз ускорить процесс создания FBD-программ.

Программа Logical\_Formulas\_Converter была написана на языке программирования DELPHI в среде разработки Embarcadero RAD Studio XE7. Данная среда предоставляет все современные средства разработки программного обеспечения, в том числе и средства визуального программирования.

В экономическом разделе было рассчитано количество денежных затрат на разработку программного средства, планируемая отпускная цена, а также прибыль от реализации разработанного программного продукта. Все рассчитанные величины были сведены в результирующую таблицу, которая наглядно представляет все необходимые затраты на создание программного продукта.

После всего этого были определены мероприятия по охране труда и безопасности жизнедеятельности, так как работа за компьютером характеризуется значительным умственным напряжением и нервно-эмоциональной нагрузкой операторов, а также высокой напряженностью зрительной работы. Для организации работы были рассмотрены меры, обеспечивающие эргономические требования к организации рабочего места пользователя. Были предусмотрены меры защиты от возможного воздействия опасных и вредных факторов производства.

# Список использованных источников

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

1

90

*ДП 00.00.ПЗ*

*Разраб.*

*Микулич Е.В.*

*Провер.*

*.*

*Акунович С.И.*

*Консульт.*

*Акунович С.И.*

*Н. контр.*

*Жиляк Н.А.*

*Утверд.*

*Урбанович П.П.*

*Список использованных источников*

*Лит.*

*Листов*

2

*БГТУ 84419015, 2015*

1 Безопасность жизнедеятельности: учебник для вузов / С.В. Белов [и др.]; под общ. ред. С.В. Белова. – 2-е изд. – М.: Высшая школа, 1999. – 448 с.

2 Гультяев, А.К. Проектирование и дизайн пользовательского интерфейса / А.К. Гультяев, В.А. Машин. − М.: Корона принт, 2010. − 190 с.

3 Зинченко, В.П. Основы эргономики / В.П. Зинченко, В.М. Мунипов. – М.: МГУ, 1979. – 179 с.

4 Зинченко, В.П. Эргономика: человекоориентированное проектирование техники, программных средств и среды: учебник. / В.П. Зинченко, В.М. Мунипов – М.: Логос, 2001. – 356 с.

5 Лазаренков, А.М. Охрана труда в энергетической отрасли: учебник / А.М. Лазаренков, Л.П. Филянович, В.П. Бубнов. – Минск: ИВЦ Минфина, 2010. – 655 с.

6 Михнюк, Т.Ф. Охрана труда: учебник для студентов высших учебных заведений / Т.Ф. Михнюк. – Минск: ИВЦ Минфина, 2009. − 345 с.

7 Основы инженерной психологии: учебник для технических вузов / Б.Ф. Ломов [и др.]; под общ. ред. Б.Ф. Ломова. – М.: Высшая школа, 1986. – 356 с.

8 СанПиН 9-131 РБ 2000. Санитарные правила и нормы «Гигиенические требования к видеодисплейным терминалам, электронно-вычислительным машинам и организации работы». – Минск: Министерство здравоохранения РБ, 10.11.2000. – 64 с.

9 Самгин, Э.Б. Освещение рабочих мест / Э.Б. Самгин. – М.: МИРЭА, 1989. – 186 с.

10 Юдин, Е.Я. Борьба с шумом на производстве: справочник / Е.Я. Юдин, Л.А. Борисов; под ред. Е.Я. Юдина. – М.: Машиностроение, 1985. – 400 с.

11 Каштелян, Т.В. Технико-экономическое обоснование дипломных проектов / Т.В. Каштелян. – Минск: БГТУ, 2013. – 80 с.

12 Шалыто, А. А. Алгоритмизация и программирование задач логического управления / А. А. Шалыто. – СПб.: СПбГУ ИТМО, 1998. – 55 с.

13 Embarcadero RAD Studio XE2 // Softline – программное обеспечение, лицензирование, обучение, консалтинг [Электронный ресурс]. – 2011. – Режим доступа: http://store.softline.ru/embarcadero/embarcadero-rad-studio-xe2/. – Дата доступа: 16.04.2013.

14 Флёнов, М.Е. Библия Delphi / Флёнов М.Е. – 3-е изд. – СПб.: БХВ-Петербург, 2011. – 688 с.

15 Насыров, И.А. Введение в современные спутниковые радионавигационные системы: учеб. пособие / Насыров И.А. – Казань: КГУ, 2005. – 43 с.

16 Карасев, В.В. Современные спутниковые радионавигационные системы: учеб. пособие / В.В. Карасев. – Владивосток: ДГТРУ, 2006. – 63 с.

17 Золотарев, С. ISaGRAF 6.1: динамичное развитие и концептуальные новшества / С. Золотарев, А. Колтунцев // Компания «ФИОРД» – современные аппаратные и программные средства для встраиваемых систем, АСУ ТП и автоматизации зданий [Электронный ресурс]. – 2012. – Режим доступа: http://www.fiord.com/download/New\_articlies/AVProm%20N8%202012.pdf. – Дата доступа: 07.04.2013.

18 Акунович, С.И. Дискретные системы логического управления технологических машин / С.И. Акунович, А.А. Гончаров, Ю.Н. Петренко. – Минск: Юнипак, 2006. – 334 с.

19 Акунович С.И. Об использовании визуального программирования при изучении алгоритмов управления дискретных устройств / С.И. Акунович // МНПК «Информатизация образовательных процессов: автоматизация управления, технологии, дистанционное обучение». – Минск, 2001. – 162 с.

20 Информационные технологии: учеб.-метод. пособие / Институт КВШУ; авт.-сост. О.П. Пономарев. – Калининград, 2004. – 122 с.

# ПРИЛОЖЕНИЕ А. Структурная схема БУ БКУ

# ПРИЛОЖЕНИЕ Б. Преобразование блок-схемы алгоритма

# ПРИЛОЖЕНИЕ В. Алгоритм программы Gerakl to SFC Converter

# ПРИЛОЖЕНИЕ Г. Исходные блок-схемы алгоритмов

# ПРИЛОЖЕНИЕ Д. Визуальная программа на языке SFC

# ПРИЛОЖЕНИЕ Е

(обязательное)

**Листинг программы Logical\_Formulas\_Converter**

unit Unit1;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ComCtrls, Vcl.StdCtrls;

type

Node = class

public

symbol: String;

\_not: bool;

left, right: Node;

id: Integer;

published

constructor Create;

end;

type

Tree = class

public

root: Node;

value: String;

published

constructor Create;

end;

type

TForm1 = class(TForm)

PageControl1: TPageControl;

TabSheet1: TTabSheet;

TabSheet2: TTabSheet;

Label2: TLabel;

Edit1: TEdit;

Label3: TLabel;

Edit2: TEdit;

Button2: TButton;

Button3: TButton;

Button4: TButton;

Label4: TLabel;

Edit3: TEdit;

Button5: TButton;

Button1: TButton;

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

procedure Button4Click(Sender: TObject);

procedure Button5Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure Button1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

view: String;

count\_vars, count\_box, id, width, width\_temp, height, from, \_to: Integer;

vars, boxs, variables, arcs, buff: TStringList;

from\_size, to\_size: Extended;

function TranslitRus2Lat(const Str: string): string;

procedure buildTree(str: String; \_node: Node; list: TStringList; \_not: bool);

function brackets(str: String): TStringList;

function IndexOfAny(str: String; sep: String): Integer;

procedure drawTree(\_node: Node);

procedure findVariable(\_node: Node);

procedure drawFormula(\_node: Node);

procedure functionWriteFilePrepare(\_tree: Tree);

procedure functionWriteFile();

procedure setIDs(\_node: Node);

implementation

uses Unit2;

{$R \*.dfm}

constructor Node.Create;

begin

inherited;

self.symbol := '';

self.\_not := false;

self.left := nil;

self.right := nil;

end;

constructor Tree.Create;

begin

inherited;

self.root := nil;

self.value := '';

end;

////////////////////////////////////////////////////////////////////////////////////////

function TranslitRus2Lat(const Str: string): string;

const

RArrayL = 'абвгдеёжзийклмнопрстуфхцчшщьыъэюя';

RArrayU = 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯ';

colChar = 33;

arr: array[1..2, 1..ColChar] of string =

(('a', 'b', 'v', 'g', 'd', 'e', 'yo', 'zh', 'z', 'i', 'y',

'k', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'f',

'kh', 'ts', 'ch', 'sh', 'shch', '''', 'y', '''', 'e', 'yu', 'ya'),

('A', 'B', 'V', 'G', 'D', 'E', 'Yo', 'Zh', 'Z', 'I', 'Y',

'K', 'L', 'M', 'N', 'O', 'P', 'R', 'S', 'T', 'U', 'F',

'Kh', 'Ts', 'Ch', 'Sh', 'Shch', '''', 'Y', '''', 'E', 'Yu', 'Ya'));

var

i: Integer;

LenS: Integer;

p: integer;

d: byte;

begin

result := '';

LenS := length(str);

for i := 1 to lenS do

begin

d := 1;

p := pos(str[i], RArrayL);

if p = 0 then

begin

p := pos(str[i], RArrayU);

d := 2

end;

if p <> 0 then

result := result + arr[d, p]

else

result := result + str[i]; //если не русская буква, то берем исходную

end;

end;

////////////////////////////////////////////////////////////////////////////////////////

function IndexOfAny(str: String; sep: String): Integer;

var

i, index: Integer;

begin

index := MaxInt;

for i := 1 to Length(sep) do

if (AnsiPos(sep[i], str) <> 0) and (AnsiPos(sep[i], str) < index) then index := AnsiPos(sep[i], str);

if index = MaxInt then index := 0;

IndexOfAny := index;

end;

////////////////////////////////////////////////////////////////////////////////////////

function brackets(str: String): TStringList;

var

input, output, list: TStringList;

i, j, max: Integer;

begin

// индексы всех открывающихся скобок / закрывающихся скобок

input := TStringList.Create;

output := TStringList.Create;

// сопоставление каждой открывающейся скобки и закрывающейся скобки

list := TStringList.Create;

for i := 0 to Length(str) do begin

if str[i] = '(' then

begin

input.Append(IntToStr(i));

end;

if str[i] = ')' then

begin

output.Append(IntToStr(i));

end;

end;

for i := 0 to output.Count - 1 do

begin

max := 0;

for j := 0 to input.Count - 1 do

begin

if (StrToInt(input.Strings[j]) < StrToInt(output.Strings[i])) and

(StrToInt(input.Strings[j]) > max) then

max := StrToInt(input[j]);

end;

input.Delete(input.IndexOf(IntToStr(max)));

list.Add(Format('%s=%s', [IntToStr(max), output[i]]))

end;

brackets := list;

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure buildTree(str: String; \_node: Node; list: TStringList; \_not: bool);

var

index1, index2, last, index: Integer;

left, right, value, tmp: String;

leftNot, rightNot: Bool;

leftNode, rightNode: Unit1.Node;

begin

left := ''; right := ''; value := '';

leftNot := false; rightNot := false;

index1 := IndexOfAny(str, '+\*');

index2 := IndexOfAny(str, '(');

if (index1 >= 1) and (index2 >= 1) and (index2 < index1) then // если первые скобки в левой части

begin

if (index2 - 1 >= 1) and (str[index2 - 1] = '^') then leftNot := true;

tmp := IntToStr(index2);

last := StrToInt(list.Values[tmp]);

left := Copy(str, index2 + 1, (last - index2) - 1);

right := Copy(str, last + 2, MaxInt);

if ((last + 1) >= 0) and ((last + 1) < Length(str)) then value := str[last + 1];

index := last + 2; // удаляем вторые скобки в правой части при их налич

if (index >= 0) and (index < Length(str)) and (str[index] = '^') then

begin

index := last + 3;

rightNot := true;

end;

if (list.IndexOfName(IntToStr(index)) >= 0) and (list.IndexOfName(IntToStr(index)) < (list.Count)) then

begin

last := StrToInt(list.Values[IntToStr(index)]);

if (last >= 0) and (last = Length(str) - 1) then

begin

if (rightNot = false) then right := Copy(right, 1, Length(right) - 2)

else right := Copy(right, 2, Length(right) - 3);

end;

end

else if (IndexOfAny(right, '+\*') = 0) then // удаляем ^ перед вторым числом при его наличии

begin

if(index >= 0) and (index < Length(str)) then

begin

index := last + 2;

if str[index] = '^' then

begin

rightNot := true;

right := Copy(right, 2, length(right) - 1);

end;

end;

end

else rightNot := false;

end

else if IndexOfAny(str, '+\*') = 0 then // если это простое присвоение

begin

if (str[1] <> '^') then value := Copy(str, 1, MaxInt)

else

begin

value := Copy(str, 2, MaxInt); \_not := true;

end;

end

else // если первое число в левой части

begin

if (str[1] <> '^') then left := Copy(str, 1, index1 - 1)

else if (index1 - 1 >= 0) and (index1 -1 < Length(str)) then

begin

left := Copy(str, 2, index1 - 2); leftNot := true;

end;

right := Copy(str, index1 + 1, MaxInt);

if (index1 >= 0) and (index1 < Length(str)) then value := str[index1];

index := index1 + 1; // удаляем вторые скобки в правой части при их наличии

if (index >= 0) and (index < Length(str)) and (str[index] = '^') then

begin

index := index1 + 2; rightNot := true;

end;

if (list.IndexOfName(IntToStr(index)) >= 0) and (list.IndexOfName(IntToStr(index)) < (list.Count)) then

begin

last := StrToInt(list.Values[IntToStr(index)]);

if (last >= 0) and (last = Length(str)) then

begin

if (rightNot = false) then right := Copy(right, 2, Length(right) - 2)

else right := Copy(right, 3, Length(right) - 3);

end;

end

else if IndexOfAny(right, '+\*') = 0 then // удаляем ^ перед вторым числом при его наличии

begin

if (index >= 0) and (index < Length(str)) then

begin

index := index1 + 1;

if (str[index] = '^') then

begin

rightNot := true; right := Copy(right, 2, Length(right) - 1);

end;

end;

end

else rightNot := false;

end;

\_node.\_not := \_not;

\_node.symbol := value;

if (IndexOfAny(left, '+\*(') = 0) and (left <> '') then

begin

leftNode := Node.Create;

leftNode.symbol := left;

leftNode.left := nil;

leftNode.right := nil;

leftNode.\_not := leftNot;

leftNot := false;

\_node.left := leftNode;

end;

if (IndexOfAny(right, '+\*(') = 0) and (right <> '') then

begin

rightNode := Node.Create;

rightNode.symbol := right;

rightNode.left := nil;

rightNode.right := nil;

rightNode.\_not := rightNot;

rightNot := false;

\_node.right := rightNode;

end;

if IndexOfAny(left, '+\*(') <> 0 then

begin

//\_node.\_not := leftNot;

\_node.left := Node.Create;

buildTree(left, \_node.left, brackets(left), {false}leftNot);

end;

if IndexOfAny(right, '+\*(') <> 0 then

begin

\_node.right := Node.Create;

buildTree(right, \_node.right, brackets(right), rightNot);

end;

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure drawTree(\_node: Node);

begin

if (\_node.left = nil) and (\_node.right = nil) then

begin

if (\_node.\_not = true) then view := view + ' НЕ (' + \_node.symbol + ')'

else view := view + \_node.symbol;

end;

if (\_node.left <> nil) then

begin

if (\_node.left.left = nil) and (\_node.left.right = nil) then

if \_node.\_not = true then view := view + ' НЕ ('; // 2 вариант

if (\_node.symbol = '\*') then view := view + ' И ('

else if \_node.symbol = '+' then view := view + ' ИЛИ (';

if (\_node.left.left <> nil) or (\_node.left.right <> nil) then

if (\_node.\_not = true) then view := view + ' НЕ (('; // 1 вариант

drawTree(\_node.left);

end;

if (\_node.right <> nil) then

begin

if (\_node.left.left <> nil) or (\_node.left.right <> nil) then

if (\_node.\_not = true) then view := view + ')'; // 1 вариант

if (\_node.symbol = '\*') then view := view + ';'

else if (\_node.symbol = '+') then view := view + ';';

drawTree(\_node.right);

if (\_node.symbol = '\*') then view := view + ')'

else if (\_node.symbol = '+') then view := view + ')';

if (\_node.left.left = nil) and (\_node.left.right = nil) then

if (\_node.\_not = true) then view := view + ')'; // 2 вариант

end;

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure findVariable(\_node: Node);

begin

if (\_node.left = nil) and (\_node.right = nil) then

begin

count\_vars := count\_vars + 1;

vars.Add(\_node.symbol);

end;

if (\_node.left <> nil) or (\_node.right <> nil) then

begin

count\_box := count\_box + 1;

if (\_node.\_not = true) then count\_box := count\_box + 1;

if (\_node.left <> nil) then findVariable(\_node.left);

if (\_node.right <> nil) then findVariable(\_node.right);

end;

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure drawFormula(\_node: Node);

begin

if (\_node.left = nil) and (\_node.right = nil) then

begin

if (\_node.\_not = true) then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id - 1) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 4) + '),S=(24,8),C=(1,1),X=NOT' + #13 + 'T=0');

arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id - 1) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

width := width - 40;

end;

variables.Add('@VAR:' + IntToStr(\_node.id) + ',P=(' + IntToStr(width) + ',' + IntToStr(height) + '),S=(32,4),C=(1,1),X=' + \_node.symbol);

height := height + 4;

end;

if (\_node.left <> nil) or (\_node.right <> nil) then

begin

if (\_node.\_not = true) then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id - 1) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 4) + '),S=(24,8),C=(1,1),X=NOT' + #13 + 'T=0');

arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id - 1) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

width := width - 40;

end;

if (\_node.symbol = '\*') then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 2) + '),S=(24,12),C=(2,1),X=AND' + #13 + 'T=0');

if \_node.left.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

if \_node.right.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

end

else if (\_node.symbol = '+') then

begin

boxs.Add('@BOX:' + IntToStr(\_node.id) + ',P=(' + IntToStr(width) + ',' + IntToStr(height + 2) + '),S=(24,12),C=(2,1),X=OR' + #13 + 'T=0');

if \_node.left.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.left.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

if \_node.right.\_not = true then arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id - 1) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')')

else arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_node.right.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_node.id) + ',' + IntToStr(1) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

end;

width := width - 40;

if (\_node.left <> nil) then drawFormula(\_node.left);

if (\_node.right <> nil) then drawFormula(\_node.right);

end;

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure setIDs(\_node: Node);

begin

\_node.id := id;

if \_node.\_not = true then begin \_node.id := id + 1; id := id + 2; end;

if \_node.\_not = false then id := id + 1;

if \_node.left <> nil then setIDs(\_node.left);

if \_node.right <> nil then setIDs(\_node.right);

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure functionWriteFilePrepare(\_tree: Tree);

var

k: Integer;

begin

id := id + 1;

setIDs(\_tree.root);

findVariable(\_tree.root);

vars.Add(\_tree.value);

if (\_tree.root.\_not = false) then for k := 0 to (count\_box + count\_vars + 1) do buff.Add('0')

else for k := 0 to (count\_box + count\_vars + 2) do buff.Add('0');

width := ((count\_box + 3 - width\_temp) \* 40);

width\_temp := count\_box;

height := height + 20;

variables.Add('@VAR:' + IntToStr(\_tree.root.id - 1) + ',P=(' + IntToStr(width) + ',' + IntToStr(height) + '),S=(32,4),C=(1,1),X=' + \_tree.value);

arcs.Add('@ARC:D=0,Z=(0,0),F=(' + IntToStr(\_tree.root.id) + ',' + IntToStr(0) + '),T=(' + IntToStr(\_tree.root.id - 1) + ',' + IntToStr(0) + ')' + #13 + 'P=(' + FloatToStr(width - 40 + 0.4) + ';' + FloatToStr(width + 0.4) + ')');

width := width - 32;

drawFormula(\_tree.root);

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure functionWriteFile();

var

str, str\_out: TStringList;

f: TextFile;

j: Integer;

filename1: String;

begin

str := TStringList.Create;

str\_out := TStringList.Create;

filename1 := Unit1.Form1.Edit3.Text;

AssignFile(f, filename1); // Prog1.isaxml для записи

str.LoadFromFile(filename1, TEncoding.Default); // Prog1.isaxml для чтения

str\_out.Add('<?xml version="1.0" encoding="utf-8"?>'); // <?xml version="1.0" encoding="utf-8"?>

str\_out.Add(str[1]); // <Pou FileVersion= ........>

str\_out.Add(str[2]); // <Program />

str\_out.Add(' <LocalVars>'); // <LocalVars />

for j := 0 to (vars.Count - 1) do

begin

if (str\_out.IndexOf(' <Variable Name="' + vars[j] + '" DataType="BOOL" InitialValue="" Comment="" Address="" Kind="Var" Alias="" AccessRights="ReadWrite" StringSize="0" />') = -1) then

str\_out.Add(' <Variable Name="' + vars[j] + '" DataType="BOOL" InitialValue="" Comment="" Address="" Kind="Var" Alias="" AccessRights="ReadWrite" StringSize="0" />');

end;

str\_out.Add(' </LocalVars>');

str\_out.Add(' <PouBody><![CDATA[PROGRAM Prog1');

str\_out.Add('#info= FBD');

str\_out.Add('@@NBID=' + IntToStr(count\_box + count\_vars + 1));

for j := 0 to (boxs.Count - 1) do

begin

str\_out.Add(boxs[j]);

end;

for j := 0 to (variables.Count - 1) do

begin

str\_out.Add(variables[j]);

end;

for j := 0 to (arcs.Count - 1) do

begin

str\_out.Add(arcs[j]);

end;

str\_out.Add('#end\_info');

str\_out.Add('#info= ID\_MAX');

str\_out.Add('NextId=' + IntToStr(count\_box + count\_vars + 1));

str\_out.Add('#end\_info');

str\_out.Add('END\_PROGRAM]]></PouBody>');

str\_out.Add('</Pou>');

for j := 0 to (str\_out.Count - 1) do

str\_out[j] := TranslitRus2Lat(str\_out[j]);

str\_out.SaveToFile (filename1, TEncoding.Default);

end;

////////////////////////////////////////////////////////////////////////////////////////

procedure TForm1.Button1Click(Sender: TObject);

var

\_tree: Tree;

\_message, output: String;

f: TextFile;

begin

AssignFile(f, edit1.Text);

Reset(f);

while not Eof(f) do

begin

readln(f, \_message);

\_message := StringReplace(\_message, ' ', '', [rfReplaceAll]);

output := Copy(\_message, 1, IndexOfAny(\_message, '=') - 1);

\_message := Copy(\_message, IndexOfAny(\_message, '=') + 1, MAXINT);

\_tree := Tree.Create;

\_tree.root := Node.Create;

\_tree.value := output;

view := '';

buildTree(\_message, \_tree.root, brackets(\_message), false);

// запись в файл схемы блоков (foreach tree = для каждой строки формул)

functionWriteFilePrepare(\_tree);

end;

functionWriteFile();

ShowMessage('Выполнено');

end;

procedure TForm1.Button2Click(Sender: TObject);

begin

DataModule2.OpenDialog1.Title := 'Выберите входной файл с логическими формулами';

DataModule2.OpenDialog1.Filter := 'Text Files(\*.txt)|\*.txt';

DataModule2.OpenDialog1.FilterIndex := 0;

DataModule2.OpenDialog1.FileName := '';

if (DataModule2.OpenDialog1.Execute()) then

Form1.Edit1.Text := DataModule2.OpenDialog1.FileName;

end;

////////////////////////////////////////////////////////////////////////////////////

procedure TForm1.Button3Click(Sender: TObject);

begin

DataModule2.SaveDialog1.Title := 'Выберите выходной файл для функциональных формул';

DataModule2.SaveDialog1.Filter := 'Text Files(\*.txt)|\*.txt';

DataModule2.SaveDialog1.FilterIndex := 0;

DataModule2.SaveDialog1.FileName := '';

if (DataModule2.SaveDialog1.Execute()) then

Form1.Edit2.Text := DataModule2.SaveDialog1.FileName;

end;

////////////////////////////////////////////////////////////////////////////////////

procedure TForm1.Button4Click(Sender: TObject);

var

\_message, \_message2 : TStringList;

i: Integer;

output: String;

\_tree: Unit1.Tree;

begin

\_message := TStringList.Create;

\_message2 := TStringList.Create;

\_message.LoadFromFile(edit1.Text, TEncoding.Default);

for i := 0 to (\_message.Count - 1) do

begin

\_message[i] := TranslitRus2Lat(\_message[i]);

\_message[i] := StringReplace(\_message[i], ' ', '', [rfReplaceAll]);

output := Copy(\_message[i], 1, IndexOfAny(\_message[i], '=') - 1);

\_message[i] := Copy(\_message[i], IndexOfAny(\_message[i], '=') + 1, MAXINT);

\_tree := Tree.Create;

\_tree.root := Node.Create;

\_tree.value := output;

view := '';

buildTree(\_message[i], \_tree.root, brackets(\_message[i]), false);

drawTree(\_tree.root);

\_message2.Add(\_tree.value + ' =' + view);

end;

\_message2.SaveToFile(edit2.Text);

ShowMessage('Выполнено');

end;

/////////////////////////////////////////////////////////////////////////////////////////

procedure TForm1.Button5Click(Sender: TObject);

begin

DataModule2.SaveDialog1.Title := 'Выберите выходной файл на языке FBD';

DataModule2.SaveDialog1.Filter := 'IsaXml Files(\*.isaxml)|\*.isaxml';

DataModule2.SaveDialog1.FilterIndex := 0;

DataModule2.SaveDialog1.FileName := '';

if (DataModule2.SaveDialog1.Execute()) then

Form1.Edit3.Text := DataModule2.SaveDialog1.FileName;

end;

procedure TForm1.FormCreate(Sender: TObject);

begin

width\_temp := 0;

FormatSettings.DecimalSeparator := '.';

view := '';

count\_vars := 0;

count\_box := 0;

vars := TStringList.Create;

vars.Duplicates := dupIgnore;

vars.Sorted := true;

id := 1;

Unit1.width := 8;

Unit1.height := 4;

boxs := TStringList.Create;

variables := TStringList.Create;

arcs := TstringList.Create;

buff := TstringList.Create;

from := 0;

\_to := 0;

from\_size := 0;

to\_size := 0;

end;

///////////////////////////////////////////////////////////////////////////////////////

end.