Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

**ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ПОСТРОЕНИЯ**

**ЛОГИЧЕСКИХ СХЕМ**

БГУИР КП I–40 01 01 421 ПЗ

Выполнил

студент: гр. 851004 Пашкевич А.Л.

Проверил: Варфоломеев А. В.

Минск 2019

Учреждение образования

«Белорусский государственный университет информатики и

радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2019г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Пашкевичу Антону Леонтьевичу* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы  *Программное средство для построения логических схем* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *31.12.2019г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Среда программирования C++ Builder. Разработать программное средство для построения логических схем. Реализовать набор инструментов, посредством которых пользователь осуществляет построение, включая расположение логических элементов на полотне, соединение элементов проводами, а также механизм ветвления проводов. Обеспечить возможность модификации схемы путём удаления элементов и проводов, перемещения их по полотну. Обеспечить корректность построение схемы, посредством валидации места расположения элементов и концов проводов. Реализовать режим моделирования, который предоставляет возможность проверить корректность функционирования построенной схемы, а также необходимые для него логические узлы. Реализовать экспорт созданного изображения в формат векторной графики SVG.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ литературных источников\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Постановка задачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*3 Разработка программного средства\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*4 Руководство по установке и использованию программного средства\_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Варфоломеев А.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *10.09.2019г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

*Раздел 1. Введение к 20.09.2019г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 10.10.2019г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 30.10.2019г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4 к 30.11.2019г. – 80% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Заключение. Приложения к 10.12.2019г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 20.12.2019г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 20.12.2019г. по 31.12.2019г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Варфоломеев А.В.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_Пашкевич А. Л. 10.09.2019г.*

*(дата и подпись студента)*

Содержание

[Содержание 4](#_Toc28429369)

[Введение 5](#_Toc28429370)

[1 Анализ предметной области 6](#_Toc28429371)

[1.1 Элементы теории логических схем 6](#_Toc28429372)

[1.2 Анализ существующих аналогов 8](#_Toc28429373)

[1.3 Представление векторного изображения в формате SVG 10](#_Toc28429374)

[2 Постановка задачи 12](#_Toc28429375)

[3 Разработка программного средства 13](#_Toc28429376)

[3.1 Структура программного средства 13](#_Toc28429377)

[3.2 Структуры данных программного средства 13](#_Toc28429378)

[3.3 Разработка алгоритмов программного средства 14](#_Toc28429379)

[3.4 Схемы алгоритмов по ГОСТ 19.701-90 20](#_Toc28429386)

[4 Руководство по использованию программного средства 23](#_Toc28429390)

[Заключение 28](#_Toc28429391)

[Список использованной литературы 29](#_Toc28429392)

[Приложение А 30](#_Toc28429393)

Введение

Сегодняшний день просто немыслим без электронной вычислительной техники. Большинство таких приборов используют цифровую логику. При этом, сложные цифровые логические устройства, входящие в состав компьютера, состоят из ряда элементарных логических элементов, построенных на базе средств электронной техники.

Логические элементы позволяют реализовать любую логическую функцию. Входные и выходные сигналы логических элементов, соответствующие двум логическим состояниям 1 и 0, могут иметь один из двух установленных уровней электрического напряжения.

При проектировании цифровых логических устройств часто возникает задача по заданной таблице истинности записать выражение для логической функции и реализовать ее в виде логической схемы, состоящей из функционально полного набора логических элементов. Данная задача также называется задачей синтеза логических схем или логических устройств.

Синтез логических схем на основе функционально полного набора логических элементов состоит из представления логических функций, описывающих данные логические схемы в нормальных формах.

Данный курсовой проект направлен на создание эффективного приложения для построения логических схем, использующих различные функционально полные наборы логических элементов.

Актуальность такого продукта также подкрепляется широким распространением использования логических схем, а также небольшим выбором приложений, обеспечивающих простое, быстрое и наиболее комфортное построение данных схем.

В реализации проекта попутно решаются следующие задачи:

* разработка механизма моделирования процесса функционирования схемы;
* сохранение и открытие файлов, созданных с помощью данного приложения;
* предоставление возможности экспорта изображения в формат векторной графики SVG.

# Анализ предметной области

## Элементы теории логических схем

Логические схемы создаются для реализации в цифровых устройствах булевых функций (функций алгебры логики).

Булева функция в дискретной математике – это отображение Bn → B, где B = {0,1} — булево множество. Неотрицательное целое число n называют арностью или местностью функции, в случае n = 0 булева функция превращается в булеву константу. Каждая булева функция арности n полностью определяется заданием своих значений на своей области определения, то есть на всех булевых векторах длины n. Число таких векторов равно 2n. Поскольку на каждом векторе булева функция может принимать значение либо 0, либо 1, то количество всех n-арных булевых функций равно [2].

Булева функция задаётся конечным набором значений, что позволяет представить её в виде таблицы истинности, например как на рисунке 1.1.

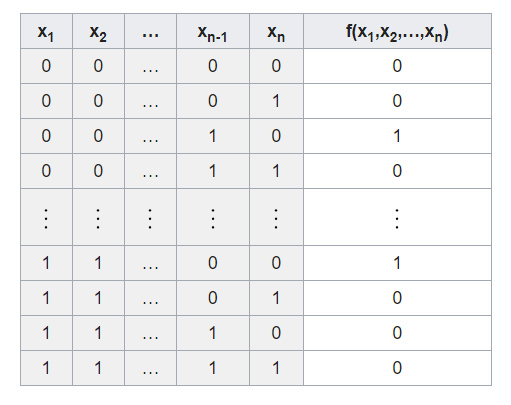


Рисунок 1.1 – Таблица истинности булевой функции

К основным булевым функциям относятся:

* отрицание;
* конъюнкция;
* дизъюнкция;
* сложение по модулю 2;
* импликация;
* эквивалентность;
* штрих Шеффера;
* стрелка Пирса.

Все сложные булевы функции возможно представлять таблицами истинности, но это не совсем удобно. Гораздо более простой способ их представления – это ДНФ и КНФ.

Дизъюнктивной нормальной формой или ДНФ называется дизъюнкция простых конъюнкций. При этом, простой конъюнкцией называется конъюнкция некоторого конечного набора переменных или их отрицаний, причём каждая переменная встречается не более одного раза [2]. Например функция – ДНФ.

Совершенной дизъюнктивной нормальной формой или СДНФ относительно некоторого заданного конечного набора переменных называется такая ДНФ, у которой в каждую конъюнкцию входят все переменные данного набора, причём в одном и том же порядке.

Необходимо рассмотреть понятие функционального элемента. В общем случае, это устройство, предназначенное для обработки информации в цифровой форме. Функциональный элемент имеет входы и выходы. Сигналы на входах функционального элемента — аргументы функции, которую реализует функциональный элемент, сигналы на выходах — значение функции от аргументов. Если входные и выходные сигналы — являются нулями и единицами, элемент называется логическим. При подаче на входы логического элемента любой комбинации двоичных сигналов, на выходах также возникает сигнал — значение булевой функции [3].

На данном этапе стоит заметить, что в большинстве случаев логические схемы строятся именно по алгебраическому представлению булевой функции, которая прошла процесс минимизации и представлена в виде минимальной ДНФ. Методов минимизации булевых функции довольно много, но у всех них одна цель – сократить число логических операций (а это значит и логических элементов на схеме) до того минимума, при котором функция ещё не теряет своего исходного логического значения. Этот процесс, по сути своей, необходим в целях рационального использования материальных и технических ресурсов при производстве физической схемы, а также с целью уменьшения её размеров.

При построении логической схемы необходимо учитывать установленные в алгебре логики правила (приоритеты) для выполнения логических операций, которые в булевом базисе имеют следующую очерёдность: НЕ – И – ИЛИ.

При этом существует алгоритм построения логической схемы, на первом шаге которого определяется число логических переменных. Затем определяется количество логических операций и их порядок. На третьем шаге для каждой логической операции изображается соответствующий ей логический элемент. В конце логические элементы соединяются. Заметим, что соединение осуществляется в порядке выполнения операций. При этом, построение надо начинать с логической операции, которая должна выполняться последней [3].

## Анализ существующих аналогов

Как уже было замечено ранее, актуальность проектируемого продукта объясняется широким распространением использования логических схем, а также небольшим выбором приложений, обеспечивающих простое, быстрое и наиболее комфортное построение данных схем.

Среди ведущих аналогов выделяется среда моделирования Multisim от компании National Instruments. Данное программное обеспечение носит прикладной характер и служит для обучения схемотехнике, предназначается для курсов аналоговой, цифровой и силовой электроники, а также лабораторий.

Multisim включает в себя огромную стандартную библиотеку компонентов для эмулирования работы различных электронных схем. В Multisim встроен промышленный симулятор SPICE, а также MCU Module, позволяющий добавить в SPICE-эмулированную интегральную схему микроконтроллер и программировать его на С или Ассемблере. Модуль позволяет эмулировать работу интегральной схемы с микроконтроллером и различными дополнительными устройствами: RAM, ROM, клавиатурой, а также графическими и жидкокристаллическими дисплеями.

При этом построение конкретно логических схем – лишь верхушка айсберга возможностей данной среды моделирования. Вид схемы для функции , разработанной в Multisim представлен на рисунке 1.2.

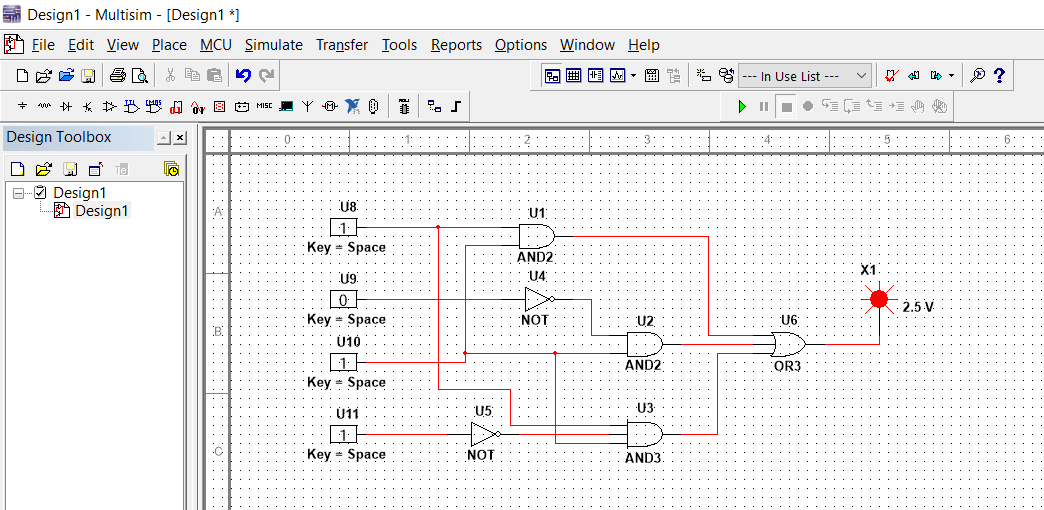


Рисунок 1.2 – Среда моделирования Multisim

К очевидным достоинствам приложения относятся широкие возможности моделирования и изучения работы электронных схем различного спектра назначения, а также опция программирования микроконтроллера на ассемблере. Однако помимо этого при анализе были выявлены и небольшие недостатки.

Во-первых, к ним относится избыточность приложения (если рассматривать его с целью построения конкретно логических схем булевых функций). Это значит, что Multisim настолько тяжеловесен, что для простой задачи создания небольшой схемы он не очень подходит. Во-вторых, с точки зрения пользователя процесс построения в некоторых местах неудобен из-за огрехов в режиме провода. Богатая библиотека также не способствуют быстрой адаптации работы с ней. В-третьих, в Multisim отсутствует экспорт изображения как в растровые, так и в векторные форматы. Наконец, Multisim не поддерживает логические элементы стандартов ГОСТ и МЭК, что является существенным недостатком для использования в странах СНГ.

Однако из данного приложения стоит позаимствовать такие элементы интерфейса схемы, как источник питания, лампочка, выделение компонента, подсветка входов компонента в режиме провода.

Ещё одним приложением такого же плана как Multisim является Proteus.

Proteus представляет собой систему схемотехнического моделирования, базирующуюся на основе моделей электронных компонентов, принятых в PSpice. Он предоставляет возможность моделирования работы программируемых устройств: микроконтроллеров, микропроцессоров, DSP. При этом в Proteus полностью реализована концепция сквозного проектирования, когда, например, инженер меняет что-то в логике работы схемотехники и программный пакет сразу же «подхватывает» данные изменения в системе трассировки. Библиотека компонентов содержит справочные данные. Дополнительно в Proteus входит система проектирования печатных плат. Proteus состоит из двух частей, двух подпрограмм: ISIS — программа синтеза и моделирования непосредственно электронных схем и ARES — программа разработки печатных плат.

Есть опция, позволяющая быстро написать программу для микроконтроллера, используемого в проекте, и скомпилировать.

Создадим схему функции и смоделируем её работу в приложении Proteus. Вид схемы представлен на рисунке 1.3.

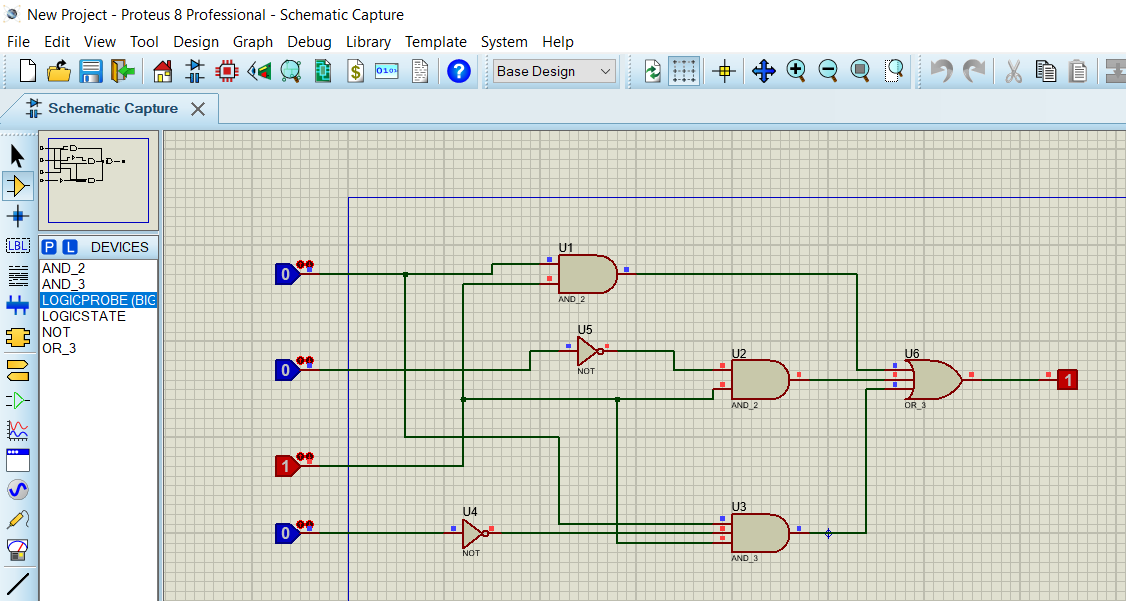


Рисунок 1.3 – Среда моделирования Proteus

В отличие от Multisim в Proteus присутствует возможность экспорта изображения в форматы BMP, PDF и SVG. Однако в остальном недостатки Multisim сохраняются. Приложение также не является простым в изучении и в процессе построения схемы. Поведение проводов не полностью контролируется пользователем.

При этом Proteus имеет довольно приятный режим моделирования, из которого стоит позаимствовать отображение сигналов на входах и выходах компонентов.

## Представление векторного изображения в формате SVG

Векторная графика – это способ представления изображений и объектов, основанный на математическом описании элементарных географических объектов.

Одним из самых распространенных форматов файлов векторной графики является формат SVG. Формат SVG предназначен для описание двумерной векторной и смешанной графики в текстовом формате XML.

Опишем структуру документа. Первая строка – стандартный XML заголовок с указанием версии, кодировки. Пример:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

Вторая и третья строка – заголовок DOCTYPE, определяющий тип документа. Пример:

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

Четвертая строка – корневой элемент документ с указанием пространства имен SVG [8]. Пример:

<svg version="1.1"

baseProfile="full"

xmlns="http://www.w3.org/2000/svg"

xmlns:xlink="http://www.w3.org/1999/xlink"

xmlns:ev="http://www.w3.org/2001/xml-events"

width="100%" height="100%">

Далее идет остальной текст документа, завершающийся закрытием тега </svg>

Описание путей позволяет задать любую фигуру, описывая путь от начальной точки до конечной через промежуточные координаты. Строка с данными задается атрибутом d тега path и содержит команды, закодированные набором букв и чисел. Буквы – обозначают тип команды. Наиболее простые – M (англ. moveto – переместить), L (англ. lineto – нарисовать линию). Цифры, чаще всего, содержат координаты точек по осям X и Y [8].

Пример: линия из точки (100,100) в точку (100,200)

<path fill="none" stroke="black" d="M 100 100 L 100 200" />

Для прямоугольника строка задается 4-мя основными атрибутами тега rect: координаты X, Y левой верхней точки (атрибуты x, y), высота и ширина (атрибуты height и width соответственно). Пример:

<rect fill="white" x="400" y="600" width="300" height="200" />

Для окружности строка задается 3-мя основными атрибутами тега cricle: координаты центра (атрибуты cx, cy), радиус (атрибут r). Пример:

<circle cx="200px" cy="200px" r="104px" fill="red”/>

Выводимый текст заключается в тег text, в котором в качестве атрибутов задаются свойства. Элементарные свойства для вывода текста – координаты левой верхней точки текста (атрибуты x, y) [8]. Пример:

<text x="30" y="12" >LogicBuilder</text>

Каждому тегу можно задать дополнительные свойства, описания и примеры которых находятся в документации по формату SVG.

# Постановка задачи

Приложение LogicBuilder предназначается для построения логических схем линейных булевых функций и моделирования процесса их функционирования.

Таким образом, требуется разработать следующий функционал для построения логической схемы:

* структуру данных для хранения информации о компоненте;
* библиотеку компонентов стандарта ГОСТ;
* размещение компонентов на полотне;
* структуру данных для хранения информации о проводе;
* соединение компонентов проводами;
* перемещение компонентов по полотну;
* выделение и удаление компонентов и проводов;
* механизмы обновления информации о компонентах и проводах при удалении объекта;
* механизм валидации места нахождения компонента при его расположении и перемещении по полотну;
* корректное поведение подключенных к компоненту проводов при его перемещении;
* механизм валидации построения проводов посредством ограничения возможных стартовых и конечных точек проводов, а также выделением доступных точек;
* механизм ветвления проводов.

Для режима моделирования необходимо разработать:

* дополнительные компоненты «источник» и «лампа»;
* механизм распространения сигнала от источника по всей схеме;
* механизм вычисления логических значений на каждом компоненте, а также отображение входных и выходных сигналов. Для компонента -арной операции, где , упростить вычисление значения в случае лишь одного входного сигнала путём его пропуска и/или инвертирования;
* механизм определения присутствия в схеме циклов и, как следствие, неопределённого поведения.

Для работы приложения с файловой системой разработать:

* сохранение и открытие файлов с информацией о схеме;
* экспорт изображения схемы не в режиме моделирования в формат SVG.

# Разработка программного средства

## Структура программного средства

Программное средство LogicBuilder разбито на 7 модулей. Модульная декомпозиция была проведена с целью разграничения алгоритмов по их функциональному назначению и логическому значению.

Модуль uMain является главным и содержит обработчики событий формы приложения, обеспечивающие взаимодействие пользователя с программой, а также функции взаимодействия приложения с файловой системой.

Модуль uComponent содержит функции создания, удаления и перемещения компонента, а также функции, обеспечивающие связь компонентов и проводов при их соединении и удалении.

Модуль uWire содержит функции создания, удаления и построения проводов.

Модуль uValidation содержит функции подтверждения корректности построения проводов и расположения компонентов.

Модуль uView отвечает за отрисовку всех объектов, а также вспомогательных элементов на полотне.

Модуль uCommon включает в себя все глобальные переменные, необходимые для работы приложения, а также структуры данных для компонентов, моделируемых компонентов и проводов – классы Component, ModelComponent и Wire соответственно.

Последний модуль uModelMode отвечает за режим моделирования.

## Структуры данных программного средства

Для реализации заявленного функионала приложения были разработаны классы Component, ModelComponent и Wire, находящиеся в модуле uCommon.

Класс Component содержит информацию о логическом элементе схемы, который в терминологии данного курсового проекта приобретает название «компонент». Данная информация представлена следующими полями:

* type – тип компонента, задаётся посредством библиотеки;
* entry\_amount – количество входов компонента;
* x\_coord - x-координата верхнего левого угла компонента;
* y\_coord - y-координата верхнего левого угла компонента;
* out\_x – x-координата выходного проводка компонента;
* out\_y – y-координата выходного проводка компонента;
* in\_x – x-координата входных проводков компонента;
* in\_y – массив y-координат входных проводков компонента;
* in\_wires – массив номеров входных проводов;
* out\_wire – номер выходного провода.

Также класс Component содержит методы работы с представленными выше полями, включая их заполнение и получение информации из них.

Класс Wire содержит информацию о проводе на логической схеме и представлен следующими полями:

* lines – двумерный массив, содержащий координаты линий провода;
* lines\_amount – количество линий в проводе;
* in\_component – номер входного компонента;
* out\_component – номер выходного компонента;
* out\_component\_entry – номер входа выходного компонента;
* connected\_wires – массив с номерами проводов, подключенных к данному;
* connected\_wires\_amount – количество подключенных проводов;
* parent\_wire – родительский провод (в случае наличия).

Данный класс также содержит методы эффективной работы с полями.

Класс ModelComponent является производным от класса Component и содержит всего лишь два собственных поля:

* in\_charge – массив входных сигналов для компонента;
* out\_charge – выходной сигнал компонента.

Объекты представленных выше классов хранятся в статических массивах component\_array, model\_component\_array и wire\_array. Выбор этих структур данных обусловлен быстрым доступом к нужной сущности посредством индекса, а также простотой использования. Динамическое выделение памяти не используется ввиду отсутствия необходимости в данном подходе.

## Разработка алгоритмов программного средства

### Алгоритм добавления компонента на схему

Данный алгоритм вызывается обработчиком события нажатия левой кнопки мыши по полотну в режиме курсора и при выбранном компоненте. На первом этапе функцией round\_coords округляются координаты точки, в которой был выполнен клик, до значений, соответствующих гранулярности сетки полотна (по умолчанию – 5 пикселей). Затем данное место полотна проверяется функцией valid\_place на возможность расположения в нём компонента. Это необходимо для того, чтобы компоненты на схеме располагались с промежуточным пространством, не накладываясь друг на друга. Затем активируется функция add\_component, которая создаёт объект класса Component, инициализирует его соответствующей информацией и записывает в массив component\_array. На последнем шаге выполняется перерисовка полотна. Возможный результат работы данного алгоритма представлен на рисунке 3.1.

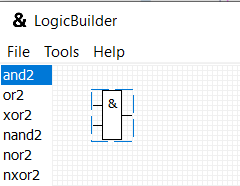


Рисунок 3.1 – Результат алгоритма добавления компонента

### Алгоритм соединения компонентов проводом

Построение провода между компонентами логически разбивается на три этапа: нахождение стартовой точки провода, нахождение промежуточных точек, и конечной. При этом, стартовой точкой может быть только конец выходного проводка компонента, а конечной – начало входного проводка любого компонента. Нельзя как выводить несколько проводов из одного вывода, так и вводить несколько проводов в один вход. При этом линии проводов должны быть горизонтальными или вертикальными.

На первом этапе алгоритма пользователю должны отображаться доступные стартовые точки провода (незанятые выходы компонентов). Это достигается обработкой события движения мыши, посредством подсветки доступной точки при наведении на неё (проверяется функцией valid\_wire\_start). При нажатии левой кнопки мыши активируется соответствующий обработчик, записывающий стартовые координаты во временный массив и переводящий построение на следующий этап. На втором этапе при каждом нажатии мыши по полотну координаты выбранной точки корректируются функцией correct\_with\_angle, для того, чтобы они представляли горизонтальную или вертикальную линию. При этом происходит динамическое отображение рисуемого провода, посредством записи координат в массив move\_line\_buffer и перерисовкой полотна по нему. На третьем этапе подсвечиваются доступные точки для завершения провода (при наведении на незанятые входы компонентов). При нажатии в такой точке временный массив дозаполняется этими координатами, проходит автокоррекцию высоты последней линии провода и передаётся в функцию add\_wire, которая создаёт объект Wire, инициализирует его поля и записывает в массив wire\_array. После этого происходит перерисовка полотна. Возможный результат работы данного алгоритма представлен на рисунке 3.2.

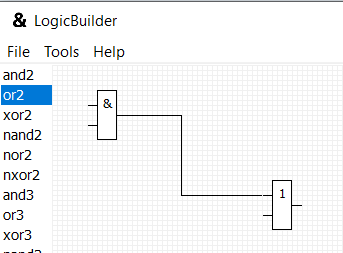


Рисунок 3.2 – Результат алгоритма соединения

компонентов проводом

### Алгоритм ветвления проводов

Данный алгоритм почти аналогичен предыдущему алгоритму соединения компонентов проводом, за исключением того, что стартовой точкой провода может быть только внутренняя точка одной из линий другого провода.

Для валидации этой опции используется функция valid\_branch\_wire\_start, а также подсветка доступных точек на проводе. На последнем этапе в функции add\_wire предусмотрена дополнительная обработка для проводов-«наследников»: в поле in\_component записывается номер начального компонента родительского провода и текущий провод записывается в список подключенных проводов родителя. Возможный результат работы данного алгоритма представлен на рисунке 3.3.

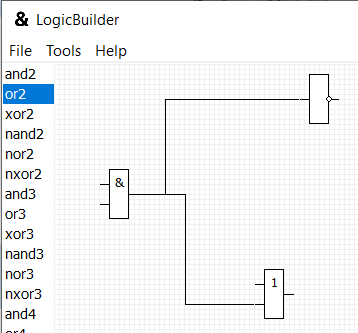


Рисунок 3.3 – Результат алгоритма ветвления проводов

### Алгоритм перемещения компонента

Компонент может перемещаться по полотну вверх, вниз, влево, вправо по дискретным значениям сетки полотна. Данные значения регулируются вручную пользователем.

При нажатии любой управляющей клавиши сначала предполагаемое следующее место расположения компонента проходит валидацию функцией valid\_place. В случае успеха компонент с новыми координатами передаётся в функцию modify\_component\_position, где происходит перезапись координат компонента и входов и выходов компонента, а также координат исходящих или входящих линий проводов, подключённых к компоненту. При этом происходит проверка функцией valid\_local\_line\_is\_alone случая, когда из «локальной» линии провода исходит дочерний провод. Если хотя бы на одном из входов или выходе компонента такая ситуация присутствует, то компонент не передвигается. Это делается с целью сохранения корректной топологии схемы.

Стоит заметить, что при передвижении компонента линии подключённых проводов становятся наклонными, что не рекомендуется для логических схем. С целью исключения загромождения алгоритма, данная ситуация не обрабатывается.

### Алгоритмы удаления компонента и провода

Данные два алгоритма объединены в один раздел из-за схожести функционирования. Для логического удаления объекта схемы необходимо в первую очередь удалить всю информацию об этом объекте в других, связанных с ним объектах.

При удалении компонента (посредством функции delete\_component), в связанных с ним проводах в соответствующих полях удаляются номера этого компонента. При этом, для выходного провода, данный процесс осуществляется рекурсивной функцией обхода схемы delete\_in\_component\_in\_all\_connected\_wires, которая удаляет номер входного компонента у всего дерева проводов, связанных с этим компонентом. Затем компонент физически удаляется в массиве методом «затирания». В конце функция decrease\_components\_index корректирует индексы всех объектов схемы.

Схема удаления провода такая же: удаляется информация о проводе в связанных компонентах, объект в массиве «затирается», индексы объектов схемы обновляются. Однако в функции delete\_wire присутствует дополнительная обработка дочерних проводов. Она заключается в следующем:

* для всего дерева дочерних проводов удаляется входной компонент;
* если провод сам дочерний, то у его родителя корректируется список подключённых проводов и их количество.

Для схемы, изображённой на рисунке 3.3 удалим компонент «or2» и провод, ведущий к компоненту «not». Вид полученной схемы представлен на рисунке 3.4.

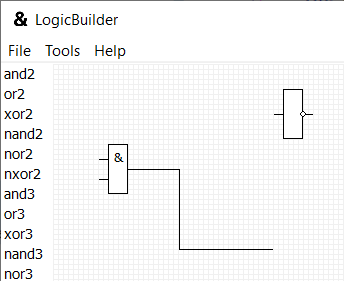


Рисунок 3.4 – Результат алгоритма удаления провода и компонента

### Алгоритмы режима моделирования

Алгоритмы этой группы располагаются в модуле uModelMode. Перечислим их названия и функциональное назначение:

* init\_model\_array – инициализирует массив моделируемых компонентов и устанавливает стартовые сигналы на источниках;
* reset\_charges – сбрасывает сигналы на всех входах и выходах компонентов, кроме источника;
* model\_scheme – алгоритм, собственно моделирующий работу схемы. Включает несколько внутренних алгоритмов;
* spread\_charge\_by\_wire – рекурсивный алгоритм распространения сигнала по проводу и его потомкам;
* is\_ready\_to\_generate – проверяет готовность компонента к вычислению выходного значения;
* generate\_out\_charge – вычисляет выходное значение компонента. использует вспомогательные функции «and\_function», «or\_function» и «xor\_function»;
* is\_trash\_component – проверяет, является ли компонент бесполезным (подключён ли к источнику питания по схеме);
* break\_in\_wires\_in\_connected\_components – в случае, если компонент бесполезный, то отключает все связанные с ним провода от выходных компонентов (в массиве model\_component\_array);
* is\_cycle\_exist – проверяет наличия цикла в схеме. В случае существования такого приводит к остановке моделирования.

Разработан ряд требований к моделированию работы схемы:

* вычисления производятся от источников тока до тех пор, пока выходные значения на всех компонентах не будут вычислены;
* для компонента -арной операции, где , вычисление упрощается для случая, когда присутствует лишь один входной сигнал. В таком случае этот сигнал просто пропускается через компонент, при чём, если компонент инвертирующий, то сигнал инвертируется;
* при нахождении цикла, моделирование прекращается ввиду отсутствия предиктивных вычислений;
* на одном полотне могут моделироваться сразу несколько независимых схем.

Процесс моделирования заключается в заполнении массивов входных и выходных значений на всех компонентах, включённых в схему. Создаётся массив check\_array с информацией о посещённых компонентах. Для каждой схемы существует неопределяемое количество итераций, на каждой из которых просматривается весь моделируемый массив и ищутся непроверенные компоненты, при чём бесполезные компоненты (определяемые функцией is\_trash\_component) исключаются из рассмотрения. После этого сигнал распространяется по схеме от каждого компонента с определённым выходным значением. Индексы этих компонентов отмечаются в массиве check\_array. Затем функцией is\_ready\_to\_generate определяются компоненты, готовые к вычислению. На этих компонентах вычисляются выходные значения посредством функции generate\_out\_charge.

Когда алгоритм моделирования заканчивается, полотно перерисовывается, при этом на нём отображаются определённые входные и выходные значения на всех значащих компонентах, а также состояния ламп. Пример смоделированной схемы для функции приведён на рисунке 3.5.

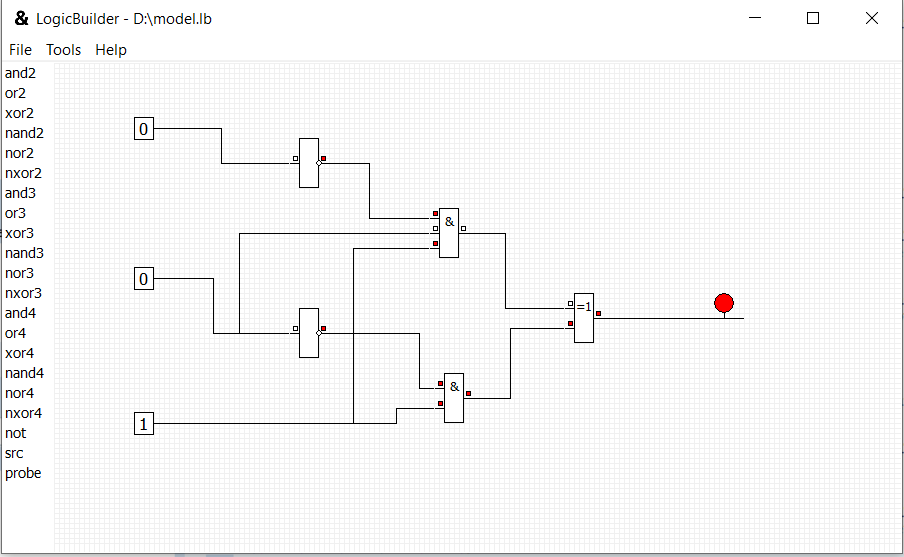


Рисунок 3.5 – Результат режима моделирования схемы

## Схемы алгоритмов по ГОСТ 19.701-90

### Схема алгоритма is\_cycle\_exist

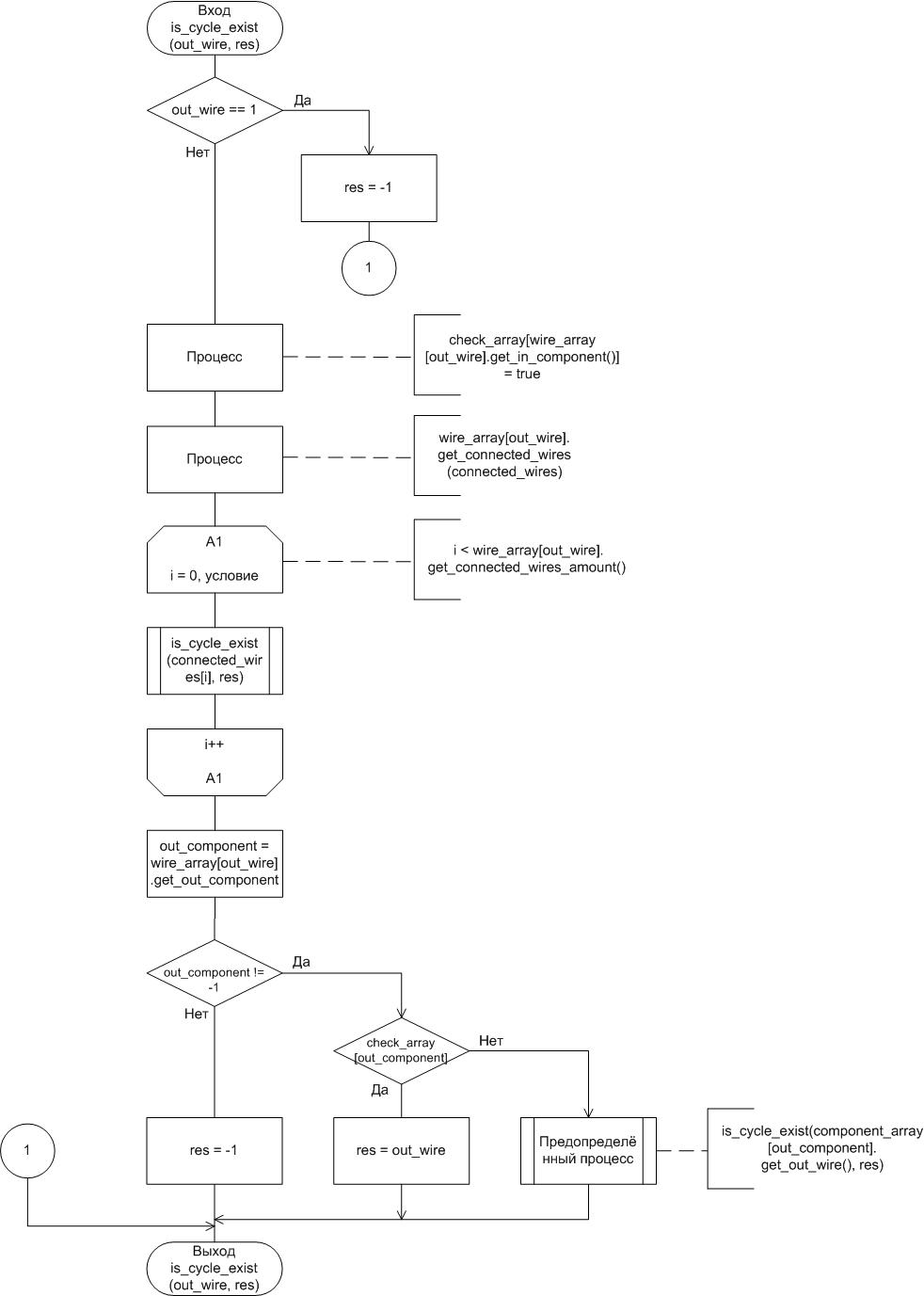


Рисунок 3.6 – Схема алгоритма is\_cycle\_exist

### Схема алгоритма delete\_component

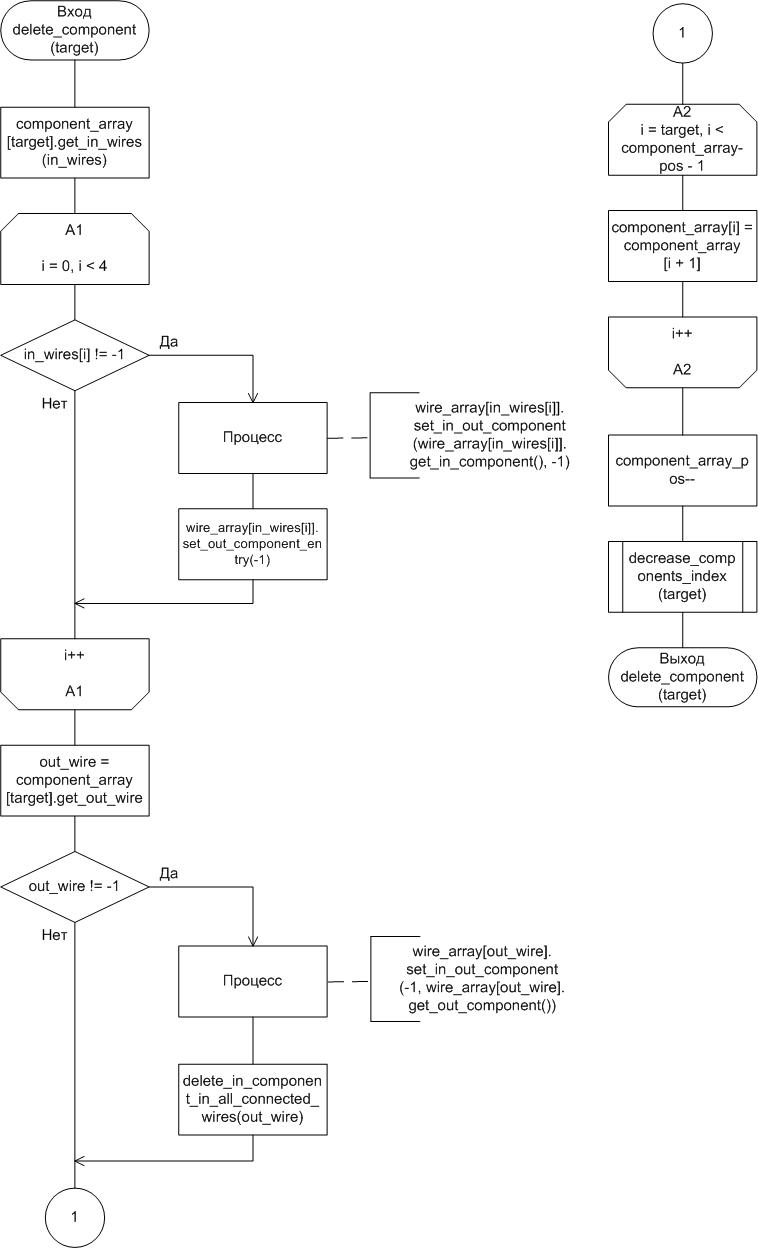


Рисунок 3.7 – Схема алгоритма delete\_component

### Схема алгоритма valid\_wire\_end

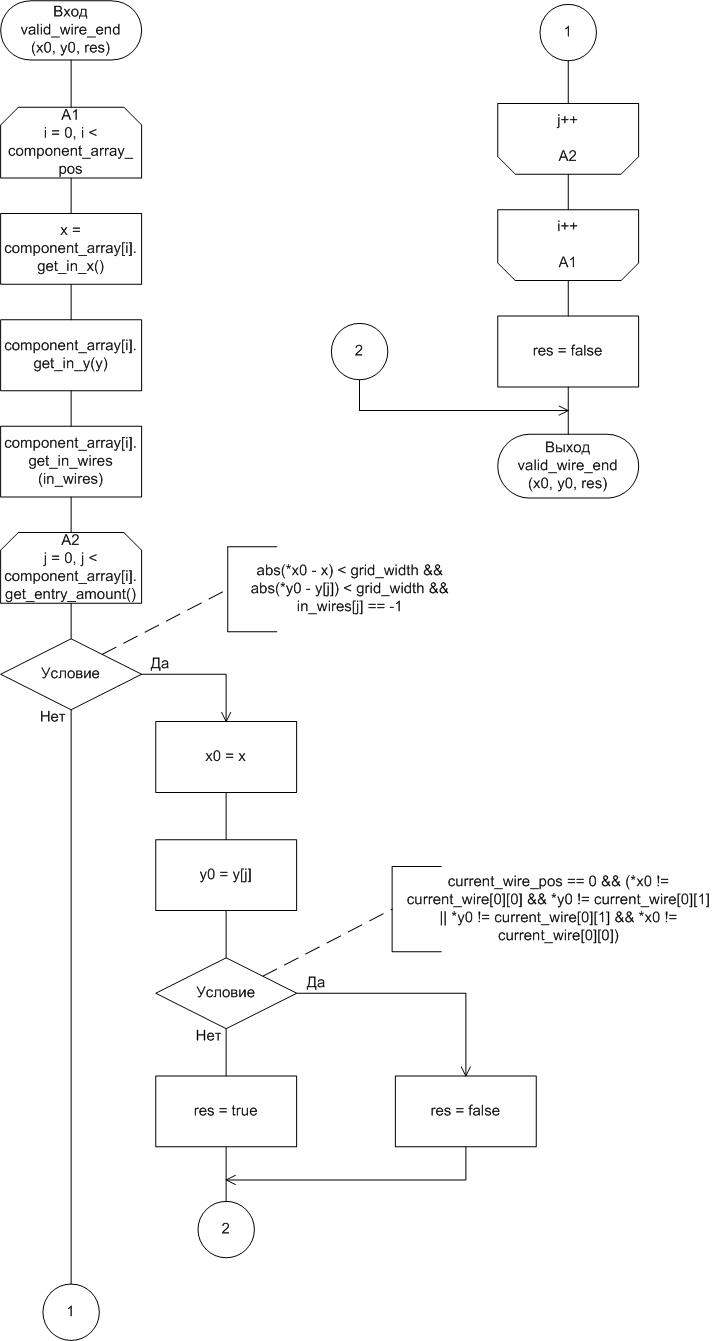


Рисунок 3.8 – Схема алгоритма valid\_wire\_end

# Руководство по использованию программного средства

Приложение LogicBuilder предназначается для построения логических схем линейных булевых функций и моделирования процесса их функционирования.

Процесс работы в приложении разбит на несколько режимов:

* cursor mode – режим курсора, в котором на полотне размещаются компоненты, производится выделение, перемещение и удаление компонентов и проводов;
* model mode – режим моделирования работы построенной схемы. Позволяет пользователю менять значения на источнике и следить за изменениями на схеме;
* wire mode – режим построения провода для соединения компонентов;
* branch wire mode – режим построения ответвляющегося провода, стартовой точкой которого является другой провод.

Режим курсора включается нажатием клавиши «C» и является активированным по умолчанию при открытии приложения. В данном режиме двойным щелчком мыши по названию компонента из списка слева выбирается желаемый компонент. Далее пользователь может свободно располагать компоненты этого типа по полотну одинарным щелчком мыши. При этом, последний добавленный компонент остаётся выделенным, что позволяет сразу же его перемещать.

Перемещение происходит с помощью управляющих клавиш «W», «A», «S», «D». По умолчанию шагом перемещения является значение 10 пикселей, что является удвоенным значением ширины сетки полотна. Для изменения этого параметра используется клавиша «shift». При её нажатии шаг перемещения изменяется циклически по схеме 10-15-20-5. Таким образом, компоненты можно расположить с любой необходимой точностью. Заметим, что перемещается лишь один выделенный компонент. Процесс работы в режиме курсора отображён на рисунке 4.1.

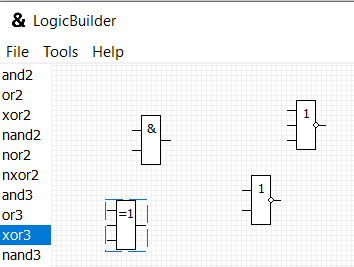


Рисунок 4.1 – Режим курсора

Также в режиме курсора происходит выделение и удаление проводов. Вид выделенного провода представлен на рисунке 4.2.

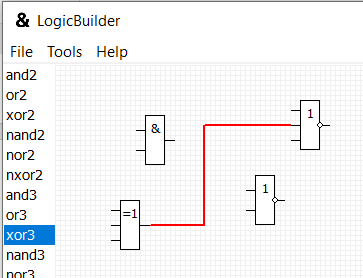


Рисунок 4.2 – Выделенный провод

Выделение и компонентов, и проводов происходит в результате щелчка по объекту. При щелчке по любому свободному месту полотна выделение пропадает. Также стоит упомянуть возможность включения и отключения сетки полотна клавишей «G». К слову, данная опция работает во всех режимах.

Режим провода активируется нажатием клавиши «X». Провода строятся строго из выходов компонентов во входы других компонентов. При этом, при наведении на доступные стартовые точки для провода, они подсвечиваются. Например, как на рисунке 4.3.

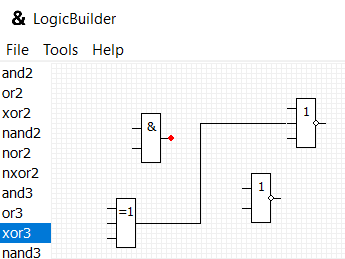


Рисунок 4.3 – Стартовая точка провода

При этом занятые входы или выходы не доступны для построения провода. Далее начинается собственно построение провода и его временное отображение. Для того чтобы сбросить построение, необходимо перейти в режим курсора и щёлкнуть по свободному месту на полотне. Пример построения провода показан на рисунке 4.4.

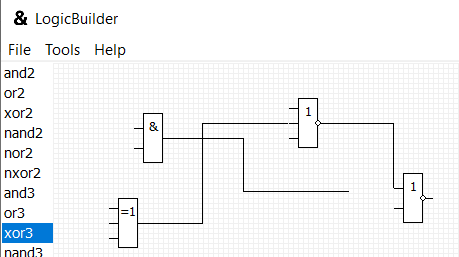


Рисунок 4.4 – Построение провода

Для окончания провода во входе компонента необходимо нажать клавишу «Alt». После этого активируются доступные конечные точки. Данная ситуация отображена на рисунке 4.5.

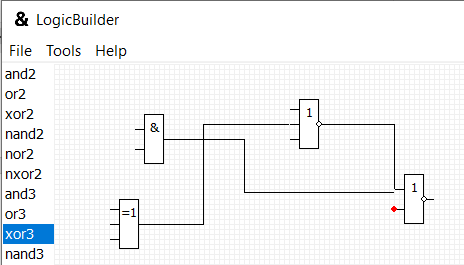


Рисунок 4.5 – Конечная точка провода

Заметим, что произойдёт автоматическая коррекция высоты провода и он будет подключён ко входу компонента.

В режиме branch wire mode стартовой точкой провода служит другой провод, а активируется он нажатием клавиши «Z». Вся остальная механика построения такая как и в wire mode. Пример ситуации показан на рисунке 4.6.

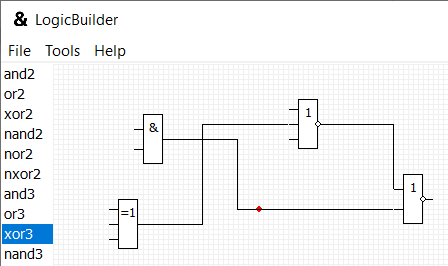


Рисунок 4.6 – Branch wire mode

Режим моделирования активируется и деактивируется клавишей «V». При этом стандартными значениями на источниках являются единицы. Сразу после включения схема начинает работать. Пользователь может нажимать на источники, изменяя их выходные значения. При обнаружении в схеме цикла приложение сообщает о невозможности обработки такой схемы. Пример режима моделирования иллюстрирует рисунок 4.7.

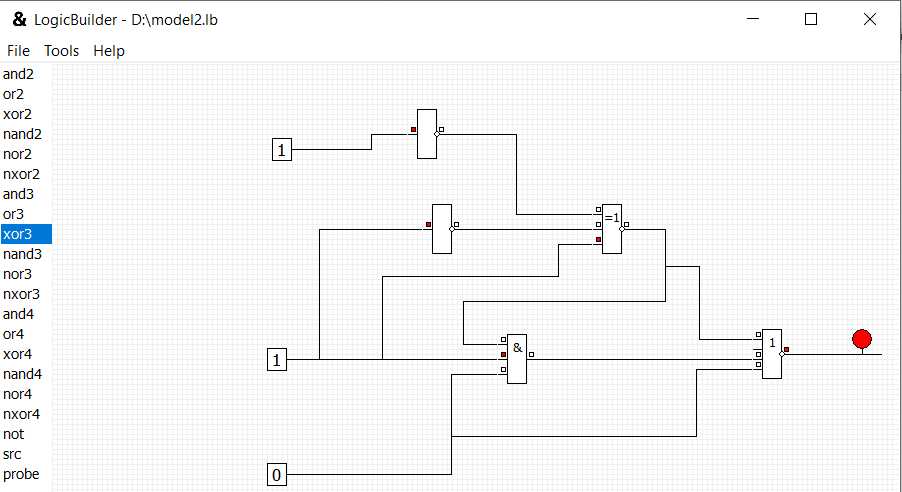


Рисунок 4.7 – Режим моделирования

Все управляющие действия находятся также на вкладке главного меню «Tools». Вид вкладки на рисунке 4.8.

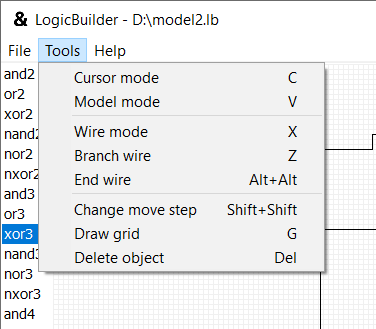


Рисунок 4.8 – Вкладка инструментов

На вкладке главного меню «File» находятся действия, отвечающие за сохранение и открытие файла, создание нового файла, а также экспорт изображения в формат SVG. Вид данной вкладки представлен на рисунке 4.9.

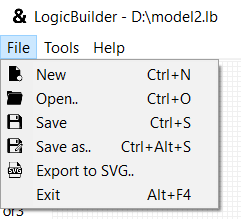


Рисунок 4.9 – Вкладка файла в главном меню

Также в главном меню есть вкладка со справочной информацией по использованию приложения. Открывающееся при этом окно показано на рисунке 4.10.

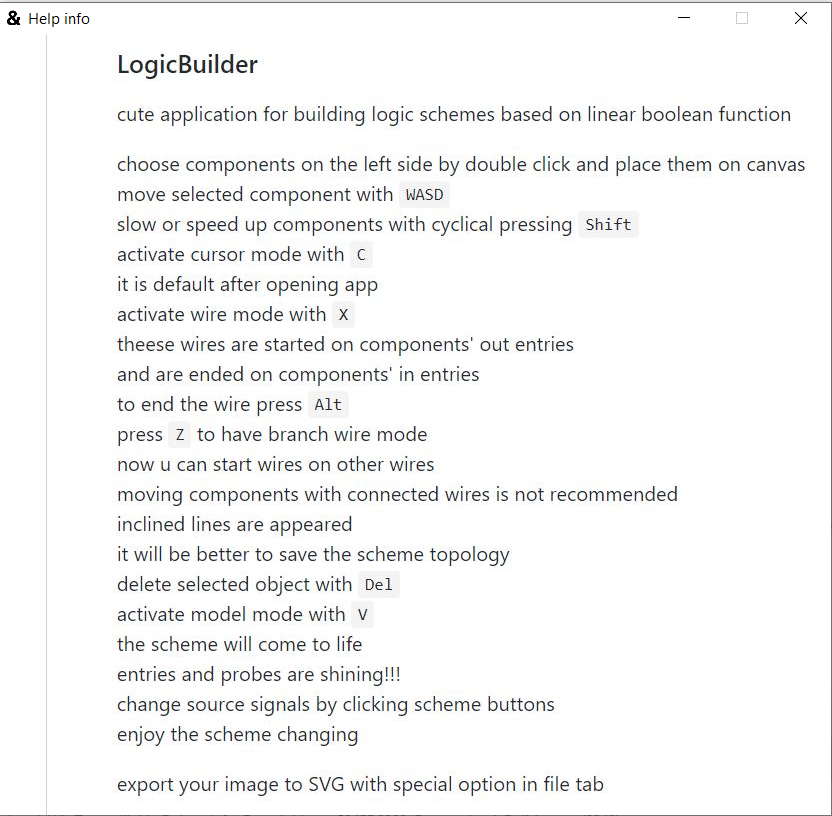


Рисунок 4.10 – Окно справочной информации

Заключение

В ходе разработки было создано эффективное приложение для построения логических схем, основанных на линейных булевых функциях. Помимо выполнения своего основного функционала, оно предоставляет пользователю возможность смоделировать процесс работы схемы, проследить изменение входных и выходных сигналов на каждом узле схемы при подаче на неё различных стартовых наборов. Таким образом, пользователь получает способ для проверки правильности построения схемы.

Данное приложение может быть улучшено посредством добавления в режим моделирования предиктивных вычислений и, как следствие, циклов в схеме. Это обстоятельство позволит моделировать такие устройства, как синхронные и асинхронные RS-триггеры, одно- и двухступенчатые D-триггеры и т.п. В процесс построения схемы может быть добавлено множественное выделение объектов, локальный буфер обмена, а также определённая гибкость по отношению к редактированию построенных проводов. Однако стоит заметить, что уже в настоящем своём виде приложение LogicBuilder хорошо справляется со своей задачей.

В ходе написания данного курсового проекта были получены базовые знания по основам объектно-ориентированного программирования, а также навыки модульной декомпозиции в языке C++. Приобретённые знания будут широко использоваться в дальнейшем. Также было сформировано представление о внутреннем устройстве формата векторной графики SVG.

Список использованной литературы

[1] Bjarne Stroustrup, The C++ Programming Language/ Addison-Wesley Professional; 4 edition (May 19, 2013)

[2] Шевелев Ю. П. Дискретная математика. Ч.1: Теория множеств. Булева алгебра: Учебное пособие — Томск: гос. ун-т систем управления и радиоэлектроники, 2003. — 118 с.

[3] Лысиков Б.Г., Арифметические и логические основы цифровых автоматов: учебник для вузов. 2-е изд., перераб. и доп. — Минск: Высш.шк., 1980 – 336 с.

[4] Нарышкин А. К. Цифровые устройства и микропроцессоры: Учеб. пособие для студ. вузов — М.: Издательский центр «Академия», 2006. – 320 с.

[5] В.А. Скляров. Язык C++ и объектно-ориентированное программирование: Справочное издание. - Минск: Вышэйшая школа, 1997. 480с.

[6] Прата, Стивен Язык программирования C++. Лекции и упражнения / Стивен Прата. - М.: Вильямс, 2015.

[7] Х. Дейтел, П. Дейтел. Как программировать на C++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с

[8] w3.org [Электронный ресурс] – Режим доступа:

<https://www.w3.org/Graphics/SVG/1.1/DTD/svg1.dtd> – Дата доступа: 17.12.2019.

Приложение А

(обязательное)

Исходный код программы

bool valid\_local\_line\_is\_alone(int target\_wire, int mode){

if (wire\_array[target\_wire].

get\_connected\_wires\_amount() != 0) {

int connected\_wires[5];

wire\_array[target\_wire].get\_connected\_wires(

connected\_wires);

int x01, y01, x02, y02;

if (mode == 1)

wire\_array[target\_wire].get\_first\_line(&x01, &y01,

&x02, &y02);

else

wire\_array[target\_wire].get\_last\_line(&x01, &y01,

&x02, &y02);

for (int j = 0; j < wire\_array[target\_wire].

get\_connected\_wires\_amount(); j++) {

int x1, y1, x2, y2;

wire\_array[connected\_wires[j]].get\_first\_line(

&x1, &y1, &x2, &y2);

if ((y1 >= y01 && y1 <= y02 ||

y1 <= y01 && y1 >= y01) && (x1 >= x01 &&

x1 <= x02 || x1 <= x01 && x1 >= x01)) {

return false;

}

}

}

return true;

}

void add\_wire(int item[10][4]){

if (wire\_array\_pos == 300)

return;

int start;

int end[2] = {-1, -1};

if (!branch\_wire\_mode)

start = find\_start\_component(item[0][0], item[0][1]);

else {

start = wire\_array[parent\_wire].get\_in\_component();

int temp = wire\_array[parent\_wire].

get\_connected\_wires\_amount();

if (temp == 5)

return;

wire\_array[parent\_wire].set\_connected\_wire(

wire\_array\_pos, temp++);

wire\_array[parent\_wire].

set\_connected\_wires\_amount(temp);

}

find\_end\_component(end, item[current\_wire\_pos][2],

item[current\_wire\_pos][3]);

if (!branch\_wire\_mode) {

component\_array[start].set\_out\_wire(wire\_array\_pos);

wire\_array[wire\_array\_pos].set\_parent\_wire(-1);

}

else

wire\_array[wire\_array\_pos].

set\_parent\_wire(parent\_wire);

component\_array[end[0]].set\_in\_wire(wire\_array\_pos, end[1]);

for (int i = 0; i < 5; i++) {

wire\_array[wire\_array\_pos].set\_connected\_wire(-1, i);

}

wire\_array[wire\_array\_pos].set\_connected\_wires\_amount(0);

wire\_array[wire\_array\_pos].set\_in\_out\_component(start,

end[0]);

wire\_array[wire\_array\_pos].set\_out\_component\_entry(end[1]);

wire\_array[wire\_array\_pos].set\_lines(item);

wire\_array[wire\_array\_pos++].set\_lines\_amount(

++current\_wire\_pos);

}

int generate\_out\_charge(int in\_charges[4],

int (\*bool\_func)(int, int)){

int res = -1;

for (int i = 0 ; i < 4; i++){

if (res == -1 && in\_charges[i] != -1){

res = in\_charges[i];

continue;

}

if (in\_charges[i] != -1)

res = (\*bool\_func)(res, in\_charges[i]);

}

return res;

}

bool is\_trash\_component(ModelComponent entity){

int in\_wires[4];

entity.get\_in\_wires(in\_wires);

for (int i = 0; i < 4; i++)

if (in\_wires[i] != -1)

if (wire\_array[in\_wires[i]].

get\_in\_component() != -1)

return false;

return true;

}

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнительные сведения* | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР КП 1–40 01 01 421 ПЗ | | | | | Пояснительная записка | | | | 32 с. | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.851004-01 СА | | | | | Режим моделирования работы логической схемы | | | | Формат А1 | |
|  | | | | | Схема алгоритма | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 421 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Программное средство для построения логических схем  Ведомость курсового  проекта | Литера | | | Лист | Листов |
| Разраб. | | Пашкевич А.Л. |  | 27.12 | Т |  |  | 32 | 32 |
| Провер. | | Варфоломеев А.В. |  | 28.12 | Кафедра ПОИТ  гр. 851004 | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |