|  |
| --- |
| *ЮГО-ЗАПАДНОЕ ОКРУЖНОЕ УПРАВЛЕНИЕ ОБРАЗОВАНИЯ*  *ДЕПАРТАМЕНТА ОБРАЗОВАНИЯ*  *ГОРОДА МОСКВЫ*  *Лицей №1533 (информационных технологий)* |

**ВЫПУСКНОЙ ПРОЕКТ**

учащегося группы П-11.4 Копылова Олега Павловича

**Обучающая программа для создания логических схем с использованием графического редактора**

|  |
| --- |
| Заказчик: Коган Андрей Горациевич |
| Руководитель: Завриев Николай Константинович |

Москва 2014

Оглавление

[Введение 3](#_Toc403772909)

[Цифровая электроника и интегральная схема 4](#_Toc403772910)

[Постановка задачи 6](#_Toc403772911)

[Анализ предметной области. Логическая схема 7](#_Toc403772912)

[Обзор аналогов 8](#_Toc403772913)

[Scheme 8](#_Toc403772914)

[Тренажёр «Логика» 9](#_Toc403772915)

[Microsoft Visio 9](#_Toc403772916)

[Способы реализации 10](#_Toc403772917)

[Программная реализация 11](#_Toc403772918)

[Ход работы 12](#_Toc403772919)

[Результаты 13](#_Toc403772920)

[Направления дальнейших разработок 17](#_Toc403772921)

[Литература 18](#_Toc403772922)

[Код программы 19](#_Toc403772923)

# Введение

Мировые тенденции развития рынка товаров в целом, как следствие приводят к активному развитию электроники. Она же в свою очередь, на данный момент времени, получила применение во всех устройствах и различного рода товаров, начиная от микроволновых плит, и заканчивая автомобилями.

Электроника делится на аналоговую и цифровую, причем последняя практически по всем позициям вытеснила аналоговую. Для специалистов в словах «цифровая» и «аналоговая» нет никакой тайны, но для людей несведущих разница между «цифрой» и «аналогом» может быть совсем неведомой. А между тем разница есть и весьма существенная.

Когда мы говорим о сигнале, то обычно подразумеваем электромагнитные колебания, наводящие ЭДС и вызывающие колебания тока в антенне приемника. По этим колебаниям приемное устройство – телевизор, радиоприемник, рация или сотовый телефон – составляет «представление» о том, какое изображение вывести на экран (при наличии видеосигнала) и какими звуками этот видеосигнал сопроводить.

В процессе передачи аналогового сигнала может произойти всякое. Могут возникнуть сторонние помехи, частота и амплитуда могут измениться, наконец, и сами передатчик и приемник во время преобразования сигнала вносят некоторую погрешность. Поэтому аналоговый сигнал всегда имеет некоторые искажения.

Цифровой сигнал считается более защищенными от помех и от внешних воздействий. Все дело в том, что при использовании «цифры» аналоговый сигнал зашифровывается в цифровой код. Нет, конечно, в окружающее пространство не распространяется поток цифр и чисел. Просто звуку определенной частоты и громкости или цвету картинки присваивается код из электрических импульсов. Продолжительность и частота импульсов задана заранее – она одна и у передатчика, и у приемника. Наличие импульса соответствует единице, отсутствие – нулю. Поэтому такая связь и получила название «цифровая».

Для обслуживания цифровой техники, тем более, для её ремонта и разработки, требуются специалисты, знающие принципы работы цифровых устройств и систем, базовые элементы цифровой электроники, типовые схемы их включения, способы построения наиболее типичных цифровых устройств.

# Цифровая электроника и интегральная схема

Перед созданием любого электронного устройства сначала создают его проект, схему, на которой отображают все элементы, из которых состоит устройство.

Интегральная схема - микроэлектронное устройство - электронная схема, обладающая законченным, сколь угодно сложным функционалом. Интегральные схемы состоят из различных транзисторов, резисторов, диодов и других электронных компонентов и отвечают за правильную работу устройств. Интегральной схемой называют как само электронное устройство, так и его схематичное представление.

Интегральные схемы обладают целым рядом преимуществ перед своими предшественниками – схемами, которые собирались из отдельных компонентов, монтируемых на шасси. Интегральные схемы имеют меньшие размеры, более высокие быстродействие и надежность; они, кроме того, дешевле и в меньшей степени подвержены отказам, вызываемым воздействиями вибраций, влаги и старения.

Начальным этапом разработки интегральной схемы является создание её логической составляющей. Для этого используют логические схемы – первый уровень проектирования интегральных схем, наглядно показывающий логику интегральных схем.

Таким образом, логические схемы намного проще интегральных схем. Именно в этой простоте состоит их отличие от других микросхем. Собранные из электронных компонентов логические схемы называются логическими компонентами. Логические схемы не несут никакой информации об электронных компонентах, которые будут использоваться при их реализации, поэтому при создании логических схем об этом можно не задумываются. Однако если заранее известно, какие электронные компоненты будут использоваться, можно создавать логическую схему так, чтобы она была наиболее проста для реализации.

Главные достоинства логических компонентов, по сравнению с другими цифровыми микросхемами, — это их высокое быстродействие (малые времена задержек), а также малая потребляемая мощность (малый ток потребления). Поэтому в тех случаях, когда требуемую функцию можно реализовать исключительно на логических элементах, всегда имеет смысл проанализировать этот вариант. Недостаток же их состоит в том, что на их основе довольно трудно реализовать сколько-нибудь сложные функции. Поэтому чаще всего логические элементы используются только в качестве дополнения к более сложным, к более «умным» микросхемам. И любой разработчик обычно стремится использовать их как можно меньше и как можно реже. Существует даже мнение, что мастерство разработчика обратно пропорционально количеству используемых им логических элементов. Однако это верно далеко не всегда.

# Постановка задачи

Задача – создать обучающую программу для создания логических схем с использованием графического редактора для персональных компьютеров и планшетов на базе операционных систем семейства Microsoft Windows.

Программа должна:

• дать возможность создавать логические схемы;

• запускать созданные логические схемы;

• сохранять и открывать ранее созданные логические схемы;

• дать возможность использовать в новых логических схемах другие ранее созданные логические схемы;

• обучать созданию логических схем с нуля, а именно:

• иметь справку, содержащую всю теорию логических схем, понятную любому человеку (ученику средней школы и старше);

• иметь базовый список задач на создание типичных логических схем разной сложности и проверку правильности решения задач;

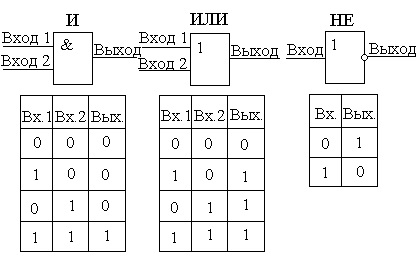
• дать возможность создавать новые задачи по созданию логических схем;

• иметь справку, содержащую полную инструкцию работы с программой.

# Анализ предметной области. Логическая схема

Логические схемы состоят из логических элементов (или, как их еще называют, вентили, «gates») и проводов, соединяющих их. Но если логическая схема используется в качестве составляющего другой логической схемы, то она также является логическим элементов. На практике почти всегда логические схемы используются для создания других схем. Поэтому слова «логическая схема» и «логический элемент» могут являться синонимами. Однако существует набор базовых логических элементов, не состоящих из других логический элементов. Эти логические элементы – И, ИЛИ и НЕ.

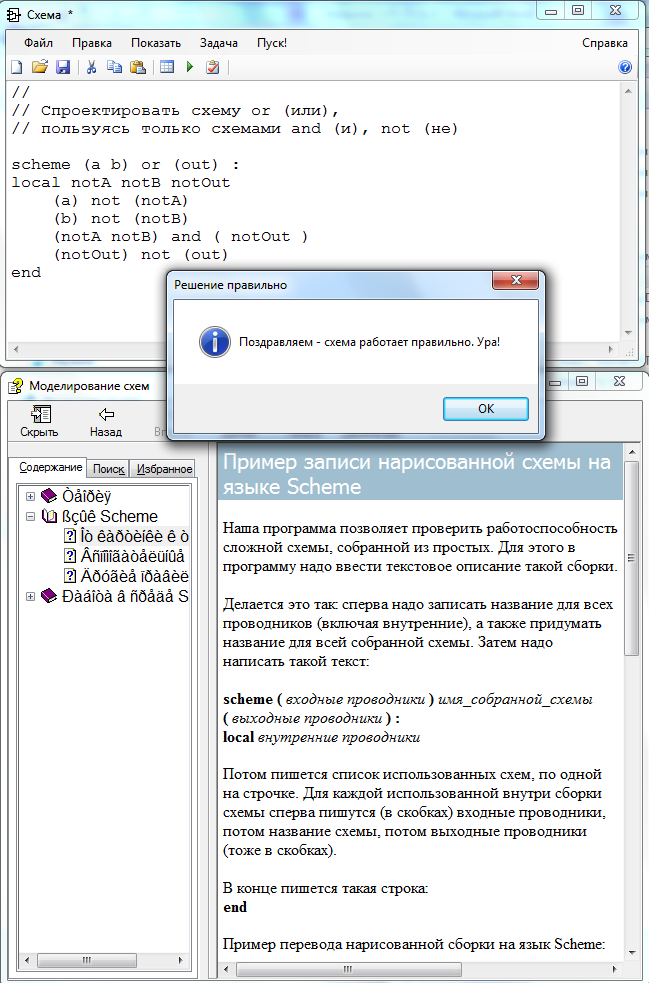
Логические элементы состоят из входов и выходов, на которые подаются сигналы в виде нулей и единиц (ведь, как вы уже знаете, логические схемы относятся к цифровой электронике). Обычно каждый логический элемент имеет несколько входов (от одного до двенадцати) и один выход. При этом связь между выходным сигналом и входными сигналами (таблица истинности) предельно проста. Каждой комбинации входных сигналов элемента соответствует заранее известное сочетание нулей или единиц на его выходах. Никакой внутренней памяти у логических элементов нет.

Базовые логические элементы И, ИЛИ и НЕ – самые простые и часто используемые логические элементы. Любой человек, желающий создавать логические схемы, должен знать их таблицы истинности и их схематичные обозначения наизусть.

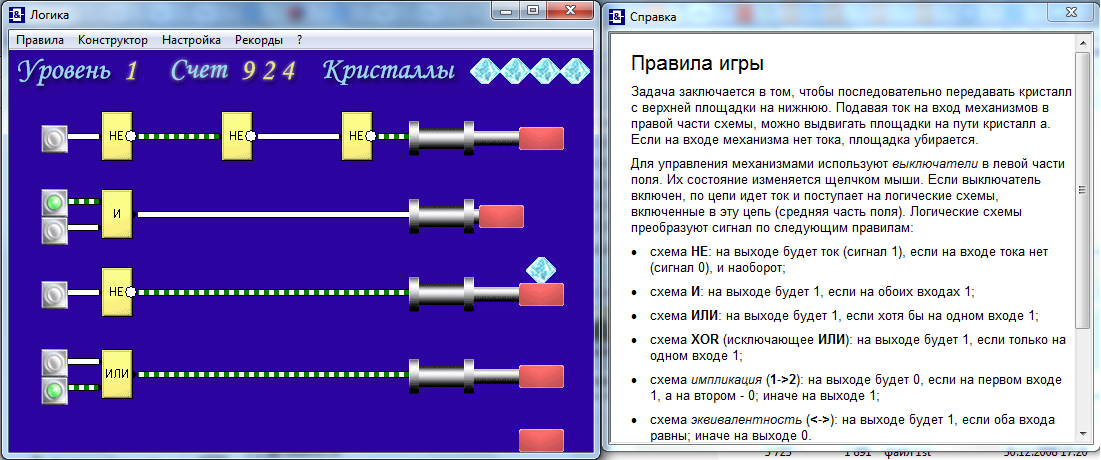
# Обзор аналогов

У требуемой программы есть несколько аналогов. Все они имеют свои особенности и отличия, недостатки и преимущества.

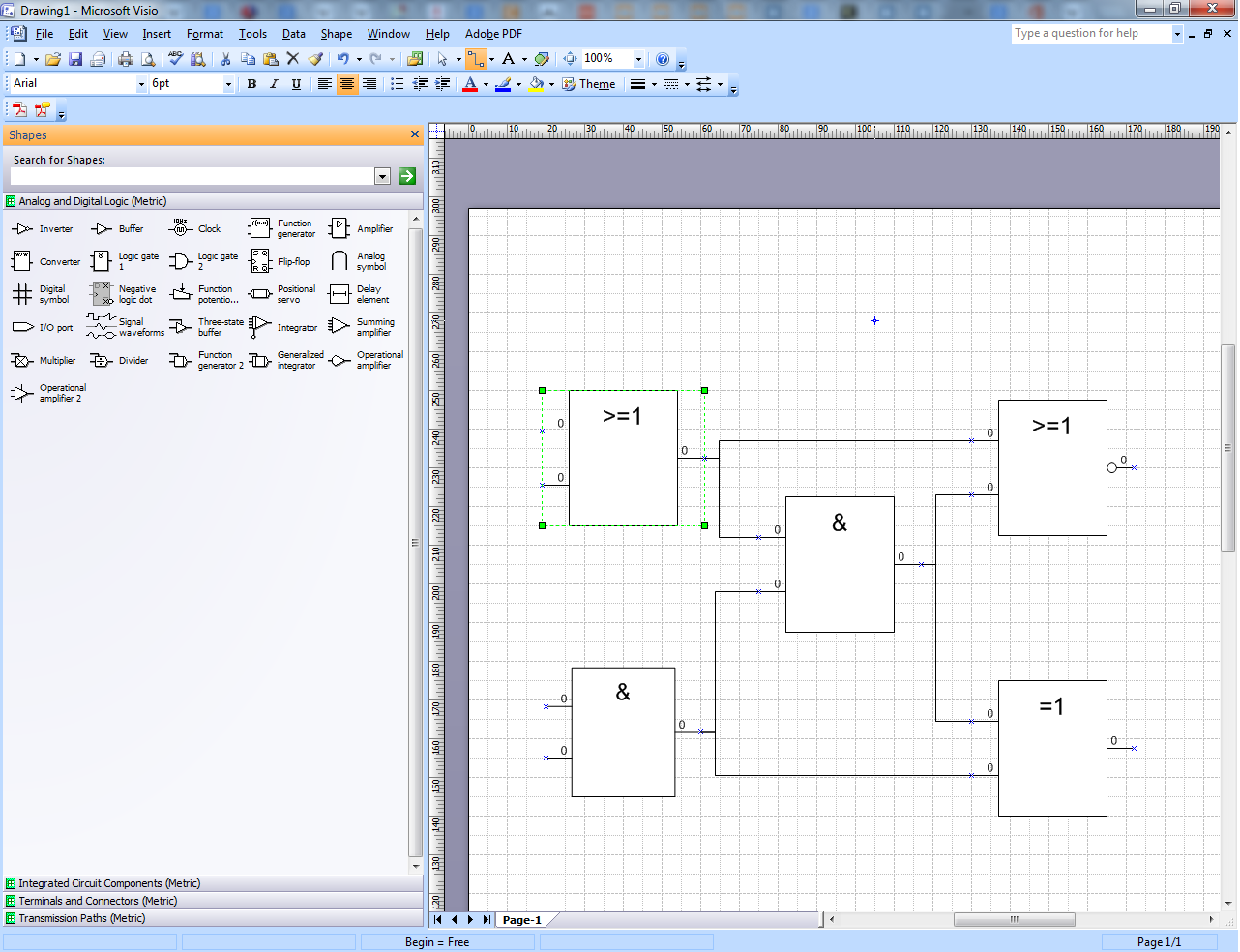
## Scheme

Scheme — это функциональный язык программирования, а также его одноимённая интегрированная среда разработки. На этом языке можно создавать логические схемы в текстовом виде и запускать их. В одной из версий этой программы так реализована обучающая составляющая. Однако факт необходимости изучения отдельного языка Scheme является основным минусом данного аналога, так как это создаёт лишние трудности и может отпугнуть от использования некоторую часть людей. Также из-за этого программа пригодна только для учеников средней школы и людей старше.

## Тренажёр «Логика»

Тренажер «Логика» предназначен для проведения изучения материала в игровой форме. Его простота позволяет изучить материал даже детям. Однако самой возможности создавать какие-либо логические схемы в ней нет, что не позволяет получить необходимый опыт в создании логических схем. 

## Microsoft Visio

Microsoft Visio — векторный графический редактор диаграмм и блок-схем для Windows. Он входит в стандартный набор программ MS Office. В нём можно создавать любые схему, но его минус в том, что по факту схемы являются рисунками, то есть их нельзя запускать. Обучение составлению логических схем в Visio также отсутствует.

# Способы реализации

При запуске созданной схемы программа сначала переводить схему на язык «Scheme», после чего анализирует полученный код. У такого способа есть несколько преимуществ.

Во-первых, это упрощает процесс создания программы, так как вместо того, чтобы придумывать некий новый алгоритм, используется уже придуманный способ. Имея схему в текстовом виде на языке «Scheme», программа просто идёт по тексту и подсчитывает значения на выходах схемы.

Во-вторых, этот способ решает проблему хранения схемы в компьютере, позволяя хранить схему в текстовом виде (переменные типа string могут занимать места до 2 GB).

Сохранять схему можно тремя способами:

• "Сохранить всю схему" - схема сохраняется полностью, и в дальнейшем вы можете, как редактировать её, так и добавлять в другие схемы;

• "Сохранить только логическую составляющею" - схема сохраняется, и в дальнейшем вы не можете её редактировать, но можете использовать в других схемах. Такой метод подойдёт, если вы не хотите, чтобы Ваши схему меняли, или не хотите, чтобы другие люди могли узнать, как Ваша схема работает. Такой файл будет занимать меньше места, чем при сохранении схемы полностью;

• "Сохранить как задачу" - результаты, которые даёт ваша схема на выходах сохраняются, и в дальнейшем они могут быть использованы как условия задачи, цель которой - составить схему, аналогичную Вашей. Сохраняя схему как задачу, Вы также можете добавить текстовую формулировку задачи и выбрать условия её выполнения.

# Программная реализация

Программа создана при помощи интегрированной среды разработки Microsoft Visual Studio 2010 версии 10.0.40219.1 SP1 Rel на объектно-ориентированном языке программирования C# 4.0.

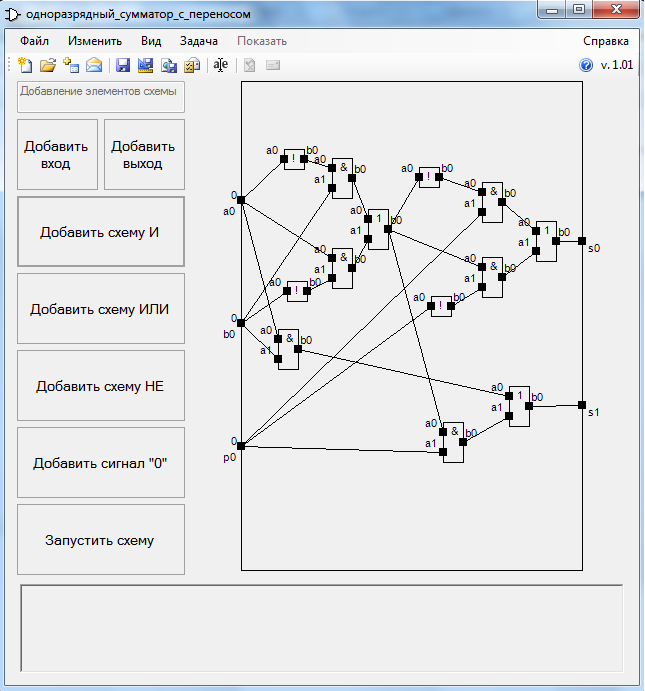
Справочный материал создан при помощи генератора документации HelpNDoc версии 4.6.2.573.

# Ход работы

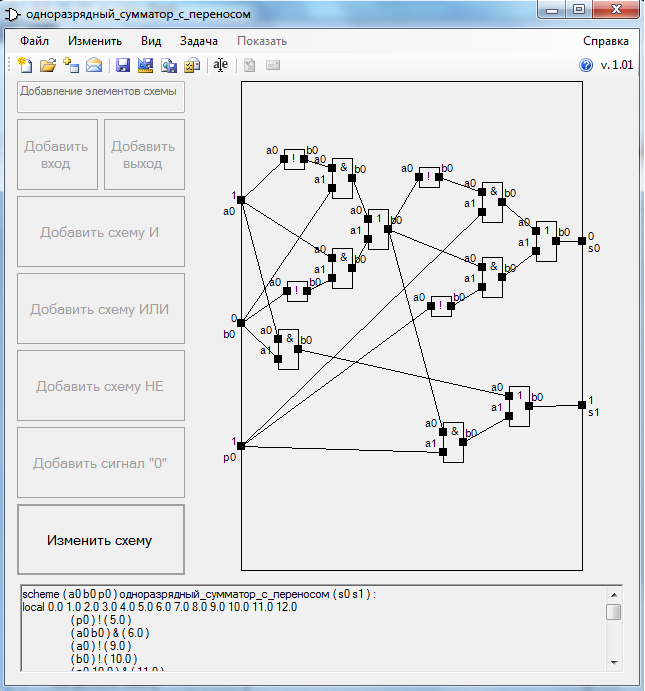
1. Была создана программа, в которой можно было создавать/удалять логические элементы, передвигать их, соединять/разъединять и т.д., то есть была создана графическая составляющая:
   1. Была создана программа, в которой можно было создавать логические элементы;
   2. Была добавлена возможность перемещать и удалять логические элементы;
   3. Была добавлена возможность соединять и разъединять логические элементы;
   4. Была добавлена возможность изменять количество входов и выходов и соединять их с логическими элементами;
   5. Были добавлены оставшиеся элементы графического редактора;
2. Было решено переводить схему на язык Scheme, что и было реализовано в программе;
3. Создан и реализован в программе алгоритм запуска схемы по коду на языке Scheme;
4. Было решено дать возможность сохранять схему двумя разными способами, добавлена возможность сохранять и загружать схему;
5. Была добавлена возможность использовать одну схему в качестве элемента другой схемы;
6. Добавлен режим выполнения заданий, созданы базовые задачи;
7. Добавлена возможность создавать свои собственные задания и потом решать их;
8. Был выбран генератор документации HelpNDoc. Была создана и добавлена в программу справка, содержащая теорию логических схем и инструкцию по работе с программой.

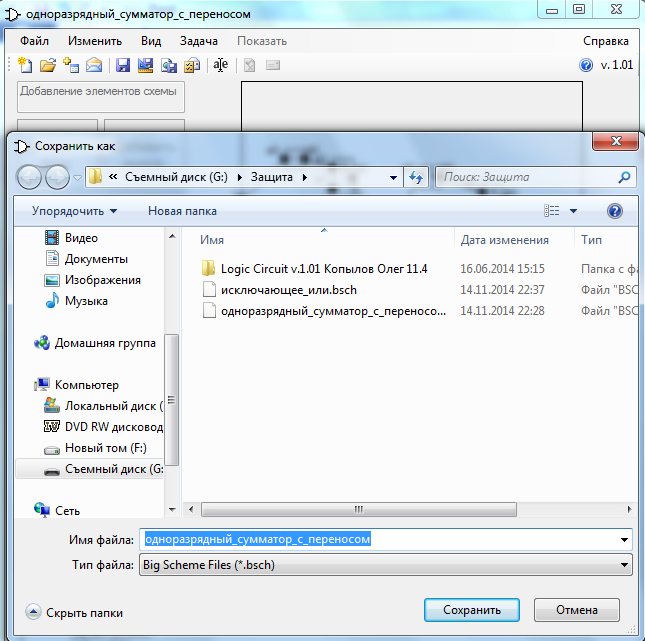
# Результаты

В итоге мною была полностью создана требуемая программа. В ней пользователь имеет возможность:

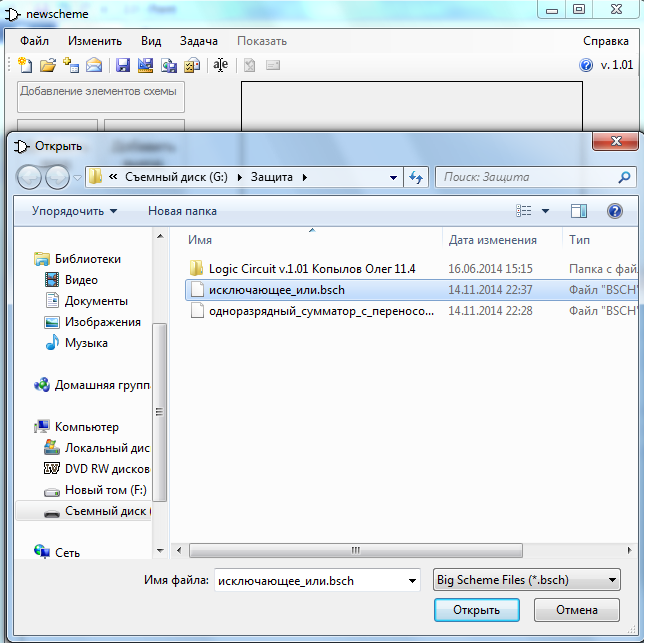
• создавать новую логическую схему;

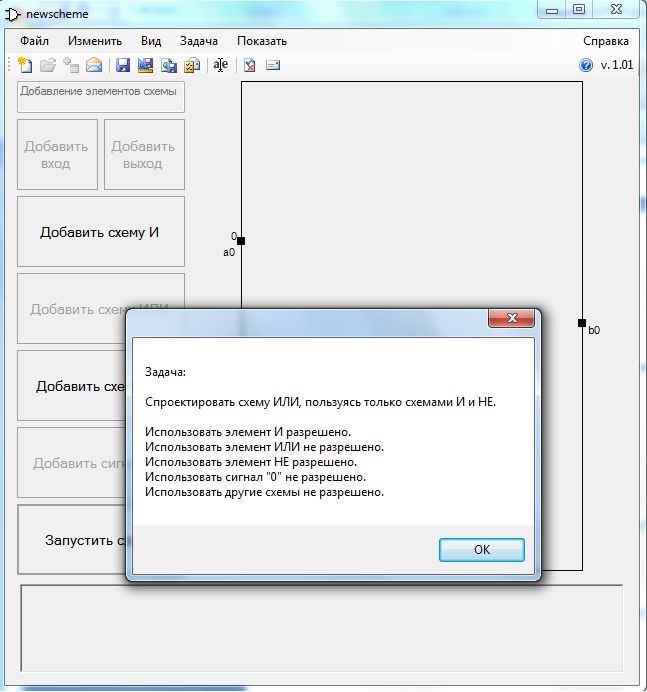
• запускать созданную логическую схему;



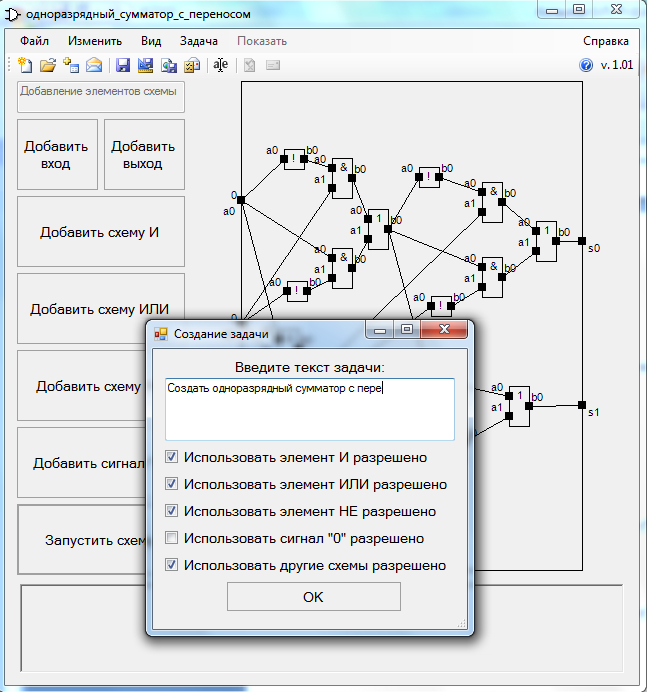
• сохранять созданную логическую схему;

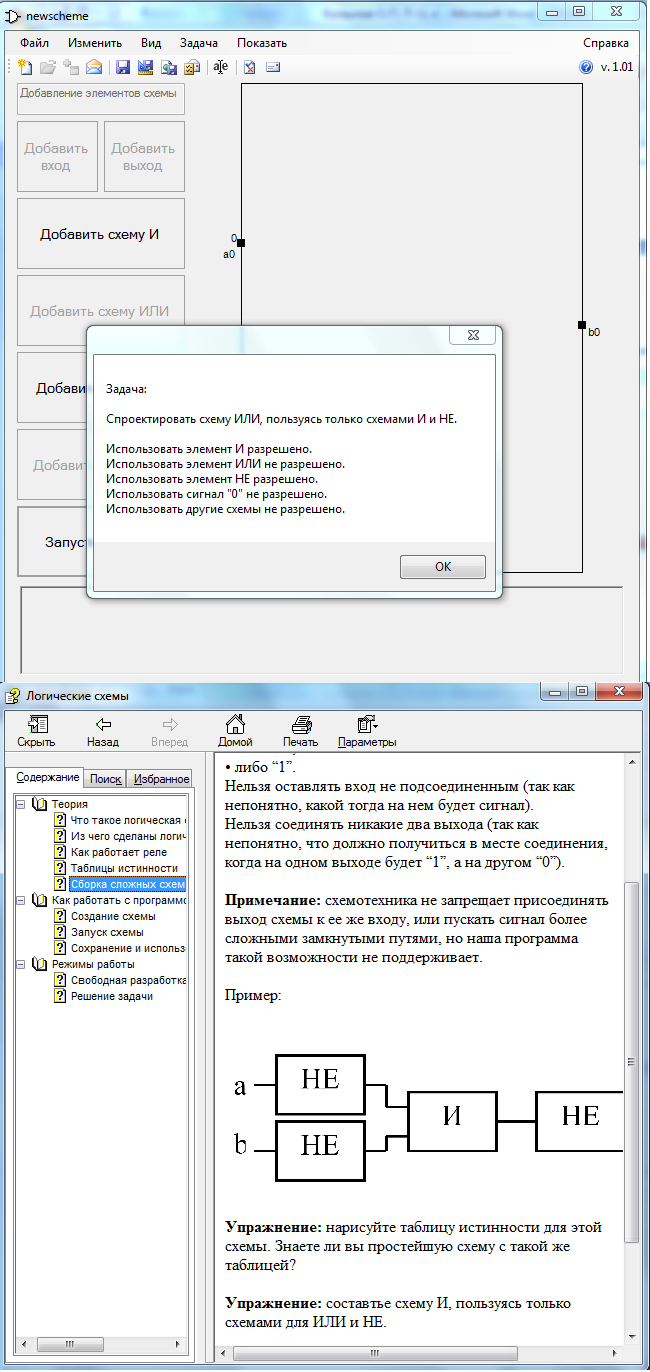
• загружать сохранённую логическую схему;



• выполнять задание по созданию определённой логической схемы;

• создавать своё собственное задание по созданию определённой логической схемы;



• читать справку с теорией логических схем и инструкцией по работе с программой.

# Направления дальнейших разработок

В дальнейшем программу можно улучшить несколькими способами:

• повышение удобства использования:

• сделать проводку проводов между логическими элементами в виде ломанных;

• выводить таблицу истинности созданной схемы в виде таблицы;

• дать возможность использовать в качестве решения задачи ранее созданную схему;

• автоматически открывать справку при первом использовании;

• при закрытии программы предлагать сохранить созданную схему (если она не была только что сохранена);

• добавление возможности ограничения количества попыток на выполнение задания;

• создавать логические схемы по введённому пользователем коду на языке Scheme;

• сохранять данные о выполнении заданий.

# Литература

1. **Новиков, Ю. В.** Основы цифровой схемотехники. Базовые элементы и схемы. Методы проектирования. – М.: Мир, 2001. – 379 с., ил. – (Современная схемотехника). – ISBN 5-03-003449-8.
2. **Новиков, Ю. В.** Введение в цифровую схемотехнику / Ю. В. Новиков – М: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 343 с: ил., табл. – (Серия «Основы информационных технологий»). – ISBN 5-94774-600-X (БИНОМ.ЛЗ) ISBN 5-9556-0082-5 (ИНТУИТ.РУ).
3. **Вовк, Е. Т., Глинка, Н. В., Грацианова, Т. Ю. и др.** Информатика: пособие для подготовки к ЕГЭ / Е. Т. Вовк [и др.] ; под ред. Е. Т. Вовк. – М. : БИНОМ. Лаборатория знаний, 2013. – 322 с. : ил. – (ВМК МГУ – школе). – ISBN 978-5-9963-1224-5.

# Код программы

using System;

using System.Threading.Tasks;

using System.Threading;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System.Reflection;

using System.Security.Permissions;

namespace Logic\_Circuit

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

DoubleBuffered = true;

version = "1.01";

showinfoStripLabel.Text = "v. " + version;

schemesymbol = 's';

schemename = "newscheme";

this.Text = schemename;

GV.radius = 20; // кратно 10

GV.jackdiameter = 2 \* GV.radius / 5;

GV.jackradius = GV.jackdiameter / 2;

//GV.firstgap = GV.radius / 2 - GV.jackradius;

PanelX = 236;

PanelY = 52;

PanelWidth = 341;

PanelHeight = 489;

BigPanelX = PanelX - GV.jackradius;

BigPanelY = PanelY - GV.jackradius;

BigPanelWidth = PanelWidth + GV.jackdiameter;

BigPanelHeight = PanelHeight + GV.jackdiameter;

MouseX = 0;

MouseY = 0;

FirstPoint = new Point();

newInputs(2);

newOutputs(1);

maxjacksnumber = PanelHeight / GV.jackdiameter;

LogicGates = new LinkedList<LogicGate>();

Wires = new LinkedList<Wire>();

ChosenGate = null;

ChosenWire = null;

OperatorsMovement = false;

WiringFromBeginning = false;

WiringFromEnd = false;

TryingToCut = false;

GV.SpecialNotOutput = false;

GV.ShowingResults = false;

GV.FreeDevelopment = true;

this.Cursor = new Cursor(new Bitmap(Properties.Resources.ClosedHand).GetHicon());

this.Cursor = new Cursor(new Bitmap(Properties.Resources.OpenHand).GetHicon());

this.Cursor = new Cursor(new Bitmap(Properties.Resources.Pencil).GetHicon());

this.Cursor = new Cursor(new Bitmap(Properties.Resources.Scissors).GetHicon());

this.Cursor = DefaultCursor;

UsedSchemes = new LinkedList<SchemeContainer>();

openSchemeDialog.FileName = "newscheme";

openSchemeDialog.Filter = "Big Scheme Files (\*.bsch)|\*.bsch|All Files (\*.\*)|\*.\*";

addSchemeDialog.FileName = "newscheme";

addSchemeDialog.Filter = "Scheme Files (\*.bsch, \*.ssch)|\*.bsch;\*.ssch|All Files (\*.\*)|\*.\*";

openTaskDialog.FileName = "newtask";

openTaskDialog.Filter = "Task Scheme Files (\*.tsch)|\*.tsch|All Files (\*.\*)|\*.\*";

saveAllFileDialog.Filter = "Big Scheme Files (\*.bsch)|\*.bsch|All Files (\*.\*)|\*.\*";

saveLogicFileDialog.Filter = "Small Scheme Files (\*.ssch)|\*.ssch|All Files (\*.\*)|\*.\*";

saveTaskFileDialog.Filter = "Task Scheme Files (\*.tsch)|\*.tsch|All Files (\*.\*)|\*.\*";

StartOrStopFreeDevelopment();

свободнаяРазработкаToolStripMenuItem.CheckState = CheckState.Checked;

Basictasks();

Refresh();

}

string version;

char schemesymbol;

string schemename;

int PanelX;

int PanelY;

int PanelWidth;

int PanelHeight;

int BigPanelX;

int BigPanelY;

int BigPanelWidth;

int BigPanelHeight;

int MouseX;

int MouseY;

Point FirstPoint;

CircuitInput[] Inputs;

CircuitOutput[] Outputs;

int maxjacksnumber;

LinkedList<LogicGate> LogicGates;

LinkedList<Wire> Wires;

CircuitComponent ChosenGate;

Wire ChosenWire;

bool OperatorsMovement;

bool WiringFromBeginning;

bool WiringFromEnd;

bool TryingToCut;

LinkedList<SchemeContainer> UsedSchemes;

bool lastsavewasfull;

Task ActiveTask;

Task[] BasicTasks;

#region Отрисовка

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics graphReload = e.Graphics;

graphReload.Clear(BackColor);

DrawEverything(graphReload);

}

void DrawEverything(Graphics g)

{

g.DrawRectangle(new Pen(Color.Black), PanelX, PanelY, PanelWidth, PanelHeight);

if (LogicGates.Count != 0)

for (LinkedListNode<LogicGate> i = LogicGates.Last; ; i = i.Previous)

{

i.Value.Draw(g);

if (i.Previous == null)

break;

}

if (Wires.Count != 0)

foreach (Wire i in Wires)

{

if (i == Wires.Last.Value && (i.Beginning == null || i.End == null))

i.Draw(g, MouseX, MouseY);

else

i.Draw(g);

}

for (int i = 0; i < Inputs.Length; i++)

Inputs[i].Draw(g);

for (int i = 0; i < Outputs.Length; i++)

Outputs[i].Draw(g);

if (TryingToCut)

g.DrawLine(new Pen(Color.DarkRed), FirstPoint.X, FirstPoint.Y, MouseX, MouseY);

}

#endregion

#region Меню

private void создатьToolStripMenuItem\_Click(object sender, EventArgs e)

{

NewScheme();

}

private void открытьToolStripMenuItem\_Click(object sender, EventArgs e)

{

OpenScheme();

}

private void добавитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

AddScheme();

}

private void сохранитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveScheme();

}

private void всюСхемуToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveAll();

}

private void толькоЛогическуюСоставляющеюToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveLogic();

}

private void сохранитьКакЗадачуToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveTask();

}

private void закрытьToolStripMenuItem\_Click(object sender, EventArgs e)

{

Close();

}

private void переименоватьToolStripMenuItem\_Click(object sender, EventArgs e)

{

RenameScheme();

}

private void инверсныйВыходВСхемеНЕToolStripMenuItem\_Click(object sender, EventArgs e)

{

ToolStripMenuItem tsmi = this.инверсныйВыходВСхемеНЕToolStripMenuItem;

if (tsmi.CheckState == CheckState.Unchecked)

{

tsmi.CheckState = CheckState.Checked;

GV.SpecialNotOutput = true;

}

else

{

tsmi.CheckState = CheckState.Unchecked;

GV.SpecialNotOutput = false;

}

Refresh();

}

private void свободнаяРазработкаToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = true;

ClearCheckStates();

свободнаяРазработкаToolStripMenuItem.CheckState = CheckState.Checked;

NewScheme();

StartOrStopFreeDevelopment();

Refresh();

}

private void иЛИНаБазеИИНЕToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

иЛИНаБазеИИНЕToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[0];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void иЛИ3ToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

иЛИ3ToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[1];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void иЛИ4ToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

иЛИ4ToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[2];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void равенствоToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

равенствоToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[3];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void исключающееИЛИToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

исключающееИЛИToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[4];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void мультиплексорToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

мультиплексорToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[5];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void большинствоToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

большинствоToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[6];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void разрядСуммыToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

разрядСуммыToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[7];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void одноразрядныйСумматорToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

одноразрядныйСумматорToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[8];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void одноразрядныйСумматорСПереносомToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

одноразрядныйСумматорСПереносомToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[9];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void двухразрядныйСумматорСПереносомToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

двухразрядныйСумматорСПереносомToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[10];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void четырехразрядныйСумматорСПереносомToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

четырехразрядныйСумматорСПереносомToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[11];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void многоразрядныйУмножительToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

многоразрядныйУмножительToolStripMenuItem.CheckState = CheckState.Checked;

ActiveTask = BasicTasks[12];

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void решитьСозданнуюРанееЗадачуToolStripMenuItem\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

решитьСозданнуюРанееЗадачуToolStripMenuItem.CheckState = CheckState.Checked;

OpenTask();

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void заданиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

ShowGoal();

}

private void результатToolStripMenuItem\_Click(object sender, EventArgs e)

{

ShowResult();

}

private void содержаниеToolStripMenuItem\_Click(object sender, EventArgs e)

{

ShowHelp();

}

private void оПрограммеToolStripMenuItem\_Click(object sender, EventArgs e)

{

ShowInfo();

}

#endregion

#region Полоса инструментов

private void newschemeStripButton\_Click(object sender, EventArgs e)

{

NewScheme();

}

private void openschemeStripButton\_Click(object sender, EventArgs e)

{

OpenScheme();

}

private void addschemeStripButton\_Click(object sender, EventArgs e)

{

AddScheme();

}

private void savedinstructionStripButton\_Click(object sender, EventArgs e)

{

GV.FreeDevelopment = false;

ClearCheckStates();

решитьСозданнуюРанееЗадачуToolStripMenuItem.CheckState = CheckState.Checked;

OpenTask();

Starttask(ActiveTask);

StartOrStopFreeDevelopment();

Refresh();

}

private void saveschemeStripButton\_Click(object sender, EventArgs e)

{

SaveScheme();

}

private void saveallStripButton\_Click(object sender, EventArgs e)

{

SaveAll();

}

private void savelogicStripButton\_Click(object sender, EventArgs e)

{

SaveLogic();

}

private void saveastaskButton\_Click(object sender, EventArgs e)

{

SaveTask();

}

private void renameStripButton\_Click(object sender, EventArgs e)

{

RenameScheme();

}

private void instructionStripButton\_Click(object sender, EventArgs e)

{

ShowGoal();

}

private void showresultStripButton\_Click(object sender, EventArgs e)

{

ShowResult();

}

private void showhelpStripButton\_Click(object sender, EventArgs e)

{

ShowHelp();

}

private void showinfoStripLabel\_Click(object sender, EventArgs e)

{

ShowInfo();

}

#endregion

#region Подменю

private void удалитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

LogicGateDelete(sender, e);

Refresh();

}

private void инвертироватьЗначениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

(ChosenGate as CircuitInput).JackValue = !(ChosenGate as CircuitInput).JackValue;

Refresh();

}

private void изменитьНазваниеToolStripMenuItem\_Click(object sender, EventArgs e)

{

ChangeNameForm ChangeName = new ChangeNameForm((ChosenGate as CircuitJack).Name);

ChangeName.ShowDialog();

(ChosenGate as CircuitJack).Name = ChangeName.name;

Refresh();

}

private void переместитьВышеToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (Outputs.Contains(ChosenGate))

{

MoveLeftInMassive(Outputs, ChosenGate as CircuitOutput);

newOutputs(Outputs.Length);

}

else

{

MoveLeftInMassive(Inputs, ChosenGate as CircuitInput);

newInputs(Inputs.Length);

}

Refresh();

}

private void переместитьНижеToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (Outputs.Contains(ChosenGate))

{

MoveRightInMassive(Outputs, ChosenGate as CircuitOutput);

newOutputs(Outputs.Length);

}

else

{

MoveRightInMassive(Inputs, ChosenGate as CircuitInput);

newInputs(Inputs.Length);

}

Refresh();

}

private void удалитьToolStripMenuItem1\_Click(object sender, EventArgs e)

{

JackDelete(sender, e);

Refresh();

}

#endregion

#region Добавление элементов схемы

private void addinputbutton\_Click(object sender, EventArgs e)

{

if (Inputs.Length <= maxjacksnumber)

newInputs(Inputs.Length + 1);

Refresh();

}

private void addoutputbutton\_Click(object sender, EventArgs e)

{

if (Outputs.Length <= maxjacksnumber)

newOutputs(Outputs.Length + 1);

Refresh();

}

private void addandbutton\_Click(object sender, EventArgs e)

{

LogicGates.AddFirst(new LogicGate(PanelX + 2 \* GV.radius, PanelY + GV.radius, new string[2] { "a0", "a1" }, new string[1] { "b0" }, '&', "И"));

Refresh();

}

private void addorbutton\_Click(object sender, EventArgs e)

{

LogicGates.AddFirst(new LogicGate(PanelX + 2 \* GV.radius, PanelY + GV.radius, new string[2] { "a0", "a1" }, new string[1] { "b0" }, '1', "ИЛИ"));

Refresh();

}

private void addnotbutton\_Click(object sender, EventArgs e)

{

LogicGates.AddFirst(new LogicGate(PanelX + 2 \* GV.radius, PanelY + GV.radius, new string[1] { "a0" }, new string[1] { "b0" }, '!', "НЕ"));

Refresh();

}

private void addzerobutton\_Click(object sender, EventArgs e)

{

LogicGates.AddFirst(new LogicGate(PanelX + 2 \* GV.radius, PanelY + GV.radius, new string[0] { }, new string[1] { "b0" }, '0', "Сигнал \"0\""));

Refresh();

}

#endregion

#region Работа мышью

private void Form1\_MouseDown(object sender, MouseEventArgs e)

{

if ((e.Button == MouseButtons.Left || e.Button == MouseButtons.Right) && BigPanelHit(e.X, e.Y) && !OperatorsMovement && !WiringFromBeginning && !WiringFromEnd && !TryingToCut && !GV.ShowingResults)

{

bool jackhit = false;

for (int i = 0; i < Inputs.Length; i++)

if (Inputs[i].HitCheck(e.X, e.Y))

{

ChosenGate = Inputs[i];

if (e.Button == MouseButtons.Left)

{

ChosenWire = new Wire(ChosenGate, true);

Wires.AddLast(ChosenWire);

Inputs[i].Input.AddLast(ChosenWire);

MouseX = e.X;

MouseY = e.Y;

WiringFromBeginning = true;

this.Cursor = new Cursor(new Bitmap(Properties.Resources.Pencil).GetHicon());

}

else

{

инвертироватьЗначениеToolStripMenuItem.Enabled = true;

изменитьНазваниеToolStripMenuItem.Enabled = GV.FreeDevelopment;

переместитьВышеToolStripMenuItem.Enabled = GV.FreeDevelopment && !(Tools.GetNumberInArray(Inputs, ChosenGate as CircuitInput) == 0);

переместитьНижеToolStripMenuItem.Enabled = GV.FreeDevelopment && !(Tools.GetNumberInArray(Inputs, ChosenGate as CircuitInput) == Inputs.Length - 1);

удалитьToolStripMenuItem1.Enabled = GV.FreeDevelopment && (Inputs.Length > 1);

ContextMenuStripJack.Show(Cursor.Position.X, Cursor.Position.Y);

}

jackhit = true;

break;

}

if (!jackhit)

for (int i = 0; i < Outputs.Length; i++)

if (Outputs[i].HitCheck(e.X, e.Y))

{

ChosenGate = Outputs[i];

if (e.Button == MouseButtons.Left)

{

if (Outputs[i].Output == null)

{

ChosenWire = new Wire(ChosenGate, false);

Wires.AddLast(ChosenWire);

Outputs[i].Output = ChosenWire;

MouseX = e.X;

MouseY = e.Y;

WiringFromEnd = true;

this.Cursor = new Cursor(new Bitmap(Properties.Resources.Pencil).GetHicon());

}

}

else

{

инвертироватьЗначениеToolStripMenuItem.Enabled = false;

изменитьНазваниеToolStripMenuItem.Enabled = GV.FreeDevelopment;

переместитьВышеToolStripMenuItem.Enabled = GV.FreeDevelopment && !(Tools.GetNumberInArray(Outputs, ChosenGate as CircuitOutput) == 0);

переместитьНижеToolStripMenuItem.Enabled = GV.FreeDevelopment && !(Tools.GetNumberInArray(Outputs, ChosenGate as CircuitOutput) == Outputs.Length - 1);

удалитьToolStripMenuItem1.Enabled = GV.FreeDevelopment && (Outputs.Length > 1);

ContextMenuStripJack.Show(Cursor.Position.X, Cursor.Position.Y);

}

jackhit = true;

break;

}

if (!jackhit)

{

bool remove = false;

foreach (LogicGate i in LogicGates)

{

ChosenGate = i;

if (i.OutputHitNumber(e.X, e.Y) != i.Outputs.Length)

{

WiringFromBeginning = true;

this.Cursor = new Cursor(new Bitmap(Properties.Resources.Pencil).GetHicon());

break;

}

else if (i.InputHitNumber(e.X, e.Y) != i.Inputs.Length)

{

WiringFromEnd = true;

this.Cursor = new Cursor(new Bitmap(Properties.Resources.Pencil).GetHicon());

break;

}

else if (i.HitCheck(e.X, e.Y))

{

if (e.Button == MouseButtons.Right)

remove = true;

else

{

OperatorsMovement = true;

this.Cursor = new Cursor(new Bitmap(Properties.Resources.ClosedHand).GetHicon());

}

break;

}

}

if (e.Button == MouseButtons.Left)

{

if (WiringFromBeginning)

{

ChosenWire = new Wire(ChosenGate, true);

Wires.AddLast(ChosenWire);

(ChosenGate as LogicGate).Outputs[(ChosenGate as LogicGate).OutputHitNumber(e.X, e.Y)].AddLast(ChosenWire);

FirstPoint.X = e.X;

FirstPoint.Y = e.Y;

MouseX = e.X;

MouseY = e.Y;

}

else if (WiringFromEnd)

{

ChosenWire = new Wire(ChosenGate, false);

Wires.AddLast(ChosenWire);

(ChosenGate as LogicGate).Inputs[(ChosenGate as LogicGate).InputHitNumber(e.X, e.Y)] = ChosenWire;

FirstPoint.X = e.X;

FirstPoint.Y = e.Y;

MouseX = e.X;

MouseY = e.Y;

}

else if (OperatorsMovement)

{

LogicGates.Remove(LogicGates.Find(ChosenGate as LogicGate));

LogicGates.AddFirst(ChosenGate as LogicGate);

MouseX = e.X - ChosenGate.X;

MouseY = e.Y - ChosenGate.Y;

}

else

{

TryingToCut = true;

FirstPoint.X = e.X;

FirstPoint.Y = e.Y;

MouseX = e.X;

MouseY = e.Y;

this.Cursor = new Cursor(new Bitmap(Properties.Resources.Scissors).GetHicon());

}

}

else

{

WiringFromEnd = false;

WiringFromBeginning = false;

OperatorsMovement = false;

if (remove)

{

logicGateToolStripMenuItem.Text = (ChosenGate as LogicGate).SchemeName;

ContextMenuStripLogicGateDelete.Show(Cursor.Position.X, Cursor.Position.Y);

}

else

ChosenGate = null;

}

}

Refresh();

}

}

private void Form1\_MouseMove(object sender, MouseEventArgs e)

{

if (OperatorsMovement)

{

if (e.X < PanelX + MouseX)

ChosenGate.X = PanelX;

else if (e.X > PanelX + PanelWidth - (GV.radius - MouseX))

ChosenGate.X = PanelX + PanelWidth - GV.radius;

else ChosenGate.X = e.X - MouseX;

if (e.Y < PanelY + MouseY)

ChosenGate.Y = PanelY;

else if (e.Y > PanelY + PanelHeight - (Math.Max((ChosenGate as LogicGate).Inputs.Length, (ChosenGate as LogicGate).Outputs.Length) \* GV.radius - MouseY))

ChosenGate.Y = PanelY + PanelHeight - Math.Max((ChosenGate as LogicGate).Inputs.Length, (ChosenGate as LogicGate).Outputs.Length) \* GV.radius;

else ChosenGate.Y = e.Y - MouseY;

Refresh();

}

else if (WiringFromBeginning || WiringFromEnd || TryingToCut)

{

if (e.X < PanelX)

{

MouseX = PanelX;

//MouseY = (MouseX \* (e.Y - FirstPoint.Y) + (FirstPoint.Y \* e.X - e.Y \* FirstPoint.X)) / (e.X - FirstPoint.X);

}

else if (e.X > PanelX + PanelWidth)

{

MouseX = PanelX + PanelWidth;

//MouseY = (MouseX \* (e.Y - FirstPoint.Y) + (FirstPoint.Y \* e.X - e.Y \* FirstPoint.X)) / (e.X - FirstPoint.X);

}

else MouseX = e.X;

if (e.Y < PanelY)

{

MouseY = PanelY;

//MouseX = (MouseY \* (e.X - FirstPoint.X) + e.Y \* FirstPoint.X - FirstPoint.Y \* e.X) / (e.Y - FirstPoint.Y);

}

else if (e.Y > PanelY + PanelHeight)

{

MouseY = PanelY + PanelHeight;

//MouseX = (MouseY \* (e.X - FirstPoint.X) + e.Y \* FirstPoint.X - FirstPoint.Y \* e.X) / (e.Y - FirstPoint.Y);

}

else MouseY = e.Y;

Refresh();

}

else if (!GV.ShowingResults)

MakeRightCursor(e.X, e.Y);

}

private void Form1\_MouseUp(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left && !GV.ShowingResults)

{

if (WiringFromBeginning)

{

foreach (CircuitOutput i in Outputs)

{

if (i != ChosenGate && i.HitCheck(MouseX, MouseY) && i.Output == null)

{

i.Output = ChosenWire;

ChosenWire.End = i;

ChosenWire = null;

break;

}

}

if (ChosenWire != null)

foreach (LogicGate i in LogicGates)

{

if (i != ChosenGate && i.InputHitNumber(MouseX, MouseY) != i.Inputs.Length)

{

i.Inputs[i.InputHitNumber(MouseX, MouseY)] = ChosenWire;

ChosenWire.End = i;

ChosenWire = null;

break;

}

}

if (ChosenWire != null)

{

Wires.RemoveLast();

if (ChosenGate is CircuitInput)

(ChosenGate as CircuitInput).Input.RemoveLast();

else

(ChosenGate as LogicGate).Outputs[(ChosenGate as LogicGate).OutputHitNumber(ChosenWire)].RemoveLast();

ChosenWire = null;

}

WiringFromBeginning = false;

MakeRightCursor(e.X, e.Y);

}

else if (WiringFromEnd)

{

foreach (CircuitInput i in Inputs)

{

if (i != ChosenGate && i.HitCheck(MouseX, MouseY))

{

i.Input.AddLast(ChosenWire);

ChosenWire.Beginning = i;

ChosenWire = null;

break;

}

}

if (ChosenWire != null)

foreach (LogicGate i in LogicGates)

{

if (i != ChosenGate && i.OutputHitNumber(MouseX, MouseY) != i.Outputs.Length)

{

i.Outputs[i.OutputHitNumber(MouseX, MouseY)].AddLast(ChosenWire);

ChosenWire.Beginning = i;

ChosenWire = null;

break;

}

}

if (ChosenWire != null)

{

Wires.RemoveLast();

if (ChosenGate is CircuitOutput)

(ChosenGate as CircuitOutput).Output = null;

else

(ChosenGate as LogicGate).Inputs[(ChosenGate as LogicGate).InputHitNumber(ChosenWire)] = null;

ChosenWire = null;

}

WiringFromEnd = false;

MakeRightCursor(e.X, e.Y);

}

else if (TryingToCut)

{

if (Wires.Count != 0)

{

LinkedListNode<Wire> i = Wires.Last;

while (true)

{

if (CrossingCheck(FirstPoint.X, FirstPoint.Y, MouseX, MouseY, i.Value.Beginning.X + ((i.Value.Beginning is LogicGate) ? GV.radius : 0), i.Value.Beginning.GetOutputCentralYCoordinate(i.Value), i.Value.End.X, i.Value.End.GetInputCentralYCoordinate(i.Value)))

{

if (i.Value.Beginning is CircuitInput)

(i.Value.Beginning as CircuitInput).Input.Remove(i.Value);

else

(i.Value.Beginning as LogicGate).Outputs[(i.Value.Beginning as LogicGate).OutputHitNumber(i.Value)].Remove(i.Value);

if (i.Value.End is CircuitOutput)

(i.Value.End as CircuitOutput).Output = null;

else

(i.Value.End as LogicGate).Inputs[(i.Value.End as LogicGate).InputHitNumber(i.Value)] = null;

Wires.Remove(i.Value);

if (Wires.Count != 0)

{

i = Wires.Last;

continue;

}

}

if (i.Previous == null)

break;

i = i.Previous;

}

}

MouseX = 0;

MouseY = 0;

TryingToCut = false;

MakeRightCursor(e.X, e.Y);

}

else if (OperatorsMovement)

{

/\*

if (e.X < PanelX + MouseX)

ChosenGate.X = PanelX;

else if (e.X > PanelX + PanelWidth - (GV.radius - MouseX))

ChosenGate.X = PanelX + PanelWidth - GV.radius;

else ChosenGate.X = e.X - MouseX;

if (e.Y < PanelY + MouseY)

ChosenGate.Y = PanelY;

else if (e.Y > PanelY + PanelHeight - (2 \* GV.radius - MouseY))

ChosenGate.Y = PanelY + PanelHeight - 2 \* GV.radius;

else ChosenGate.Y = e.Y - MouseY;

\*/

MouseX = 0;

MouseY = 0;

OperatorsMovement = false;

MakeRightCursor(e.X, e.Y);

}

ChosenGate = null;

Refresh();

Refresh();

}

}

#endregion

#region Работа схемы

private void runschemebutton\_Click(object sender, EventArgs e)

{

if (!GV.ShowingResults)

{

string scheme = Scheme.ConvertToScheme(schemename, Inputs, Outputs, LogicGates, Wires);

foreach (LogicGate i in LogicGates)

{

i.Valueisknown = false;

}

if (scheme == null)

MessageBox.Show("Ошибка при обработке схемы!");

else

{

richTextBox1.Text = scheme;

bool[] b = Scheme.RunScheme(scheme, UsedSchemes, CircuitInput.GetBoolArray(Inputs));

if (b == null || b.Length != Outputs.Length)

MessageBox.Show("Ошибка при попытке запустить схему!");

else

for (int i = 0; i < Outputs.Length; i++)

Outputs[i].JackValue = b[i];

}

GV.ShowingResults = true;

}

else

{

richTextBox1.Text = "";

GV.ShowingResults = false;

}

StartOrStopShowingResults();

Refresh();

}

#endregion

#region Инструменты

void ClearCheckStates()

{

свободнаяРазработкаToolStripMenuItem.CheckState = CheckState.Unchecked;

иЛИНаБазеИИНЕToolStripMenuItem.CheckState = CheckState.Unchecked;

иЛИ3ToolStripMenuItem.CheckState = CheckState.Unchecked;

иЛИ4ToolStripMenuItem.CheckState = CheckState.Unchecked;

равенствоToolStripMenuItem.CheckState = CheckState.Unchecked;

исключающееИЛИToolStripMenuItem.CheckState = CheckState.Unchecked;

мультиплексорToolStripMenuItem.CheckState = CheckState.Unchecked;

большинствоToolStripMenuItem.CheckState = CheckState.Unchecked;

разрядСуммыToolStripMenuItem.CheckState = CheckState.Unchecked;

одноразрядныйСумматорToolStripMenuItem.CheckState = CheckState.Unchecked;

одноразрядныйСумматорСПереносомToolStripMenuItem.CheckState = CheckState.Unchecked;

двухразрядныйСумматорСПереносомToolStripMenuItem.CheckState = CheckState.Unchecked;

четырехразрядныйСумматорСПереносомToolStripMenuItem.CheckState = CheckState.Unchecked;

многоразрядныйУмножительToolStripMenuItem.CheckState = CheckState.Unchecked;

решитьСозданнуюРанееЗадачуToolStripMenuItem.CheckState = CheckState.Unchecked;

}

void LogicGateDelete(object sender, EventArgs e)

{

for (int i = 0; i < (ChosenGate as LogicGate).Outputs.Length; i++)

{

foreach (Wire j in (ChosenGate as LogicGate).Outputs[i])

{

if (j.End is CircuitOutput)

(j.End as CircuitOutput).Output = null;

else if (j.End is CircuitInput)

(j.End as CircuitInput).Input.Remove(j);

else

(j.End as LogicGate).Inputs[(j.End as LogicGate).InputHitNumber(j)] = null;

Wires.Remove(j);

}

}

for (int i = 0; i < (ChosenGate as LogicGate).Inputs.Length; i++)

{

if ((ChosenGate as LogicGate).Inputs[i] != null)

{

if ((ChosenGate as LogicGate).Inputs[i].Beginning is CircuitOutput)

(ChosenGate as LogicGate).Inputs[i].Beginning = null;

else if ((ChosenGate as LogicGate).Inputs[i].Beginning is CircuitInput)

((ChosenGate as LogicGate).Inputs[i].Beginning as CircuitInput).Input.Remove((ChosenGate as LogicGate).Inputs[i]);

else

((ChosenGate as LogicGate).Inputs[i].Beginning as LogicGate).Outputs[((ChosenGate as LogicGate).Inputs[i].Beginning as LogicGate).OutputHitNumber((ChosenGate as LogicGate).Inputs[i])].Remove((ChosenGate as LogicGate).Inputs[i]);

Wires.Remove((ChosenGate as LogicGate).Inputs[i]);

}

}

//LogicalOperators.Remove(LogicalOperators.Find(ChosenOperator));

LogicGates.Remove(ChosenGate as LogicGate);

if (UsedSchemes.Count != 0)

for (LinkedListNode<SchemeContainer> i = UsedSchemes.Last; ; i = i.Previous)

{

if (i.Value.SchemeName == (ChosenGate as LogicGate).SchemeName)

{

UsedSchemes.Remove(i);

break;

}

if (i.Previous == null)

break;

}

ChosenGate = null;

Refresh();

}

void JackDelete(object sender, EventArgs e)

{

if (ChosenGate is CircuitInput)

{

foreach (Wire i in (ChosenGate as CircuitInput).Input)

{

if (i.End is CircuitOutput)

(i.End as CircuitOutput).Output = null;

else if (i.End is CircuitInput)

(i.End as CircuitInput).Input.Remove(i);

else

(i.End as LogicGate).Inputs[(i.End as LogicGate).InputHitNumber(i)] = null;

Wires.Remove(i);

}

//Inputs = DeleteFromMassive(Inputs, (ChosenGate as CircuitJack));

CircuitInput[] c = new CircuitInput[Inputs.Length - 1];

byte isalreadydeleted = 0;

for (int i = 0; i < Inputs.Length; i++)

{

if (Inputs[i] == ChosenGate)

isalreadydeleted = 1;

else

c[i - isalreadydeleted] = Inputs[i];

}

Inputs = c;

newInputs(Inputs.Length);

}

else

{

if ((ChosenGate as CircuitOutput).Output != null)

{

Wire i = (ChosenGate as CircuitOutput).Output;

if (i.Beginning is CircuitOutput)

(i.Beginning as CircuitOutput).Output = null;

else if (i.Beginning is CircuitInput)

(i.Beginning as CircuitInput).Input.Remove(i);

else

(i.Beginning as LogicGate).Outputs[(i.Beginning as LogicGate).OutputHitNumber(i)].Remove(i);

Wires.Remove(i);

}

//Outputs = DeleteFromMassive(Outputs, (ChosenGate));

CircuitOutput[] c = new CircuitOutput[Outputs.Length - 1];

byte isalreadydeleted = 0;

for (int i = 0; i < Outputs.Length; i++)

{

if (Outputs[i] == ChosenGate)

isalreadydeleted = 1;

else

c[i - isalreadydeleted] = Outputs[i];

}

Outputs = c;

newOutputs(Outputs.Length);

}

ChosenGate = null;

Refresh();

}

bool PanelHit(int x, int y)

{

if (PanelX <= x && x <= PanelX + PanelWidth && PanelY <= y && y <= PanelY + PanelHeight)

return true;

return false;

}

bool BigPanelHit(int x, int y)

{

if (BigPanelX <= x && x <= BigPanelX + BigPanelWidth && BigPanelY <= y && y <= BigPanelY + BigPanelHeight)

return true;

return false;

}

bool CrossingCheck(int l2x1, int l2y1, int l2x2, int l2y2, int l1x1, int l1y1, int l1x2, int l1y2)

{

if (Math.Min(l2x1, l2x2) > Math.Max(l1x1, l1x2) || Math.Min(l1x1, l1x2) > Math.Max(l2x1, l2x2) || Math.Min(l2y1, l2y2) > Math.Max(l1y1, l1y2) || Math.Min(l1y1, l1y2) > Math.Max(l2y1, l2y2))

return false;

if (l1x1 == l1x2 && l2x1 == l2x2)

return false;

else if (l1x1 == l1x2)

{

double k2 = (l2y1 - l2y2) / (double)(l2x1 - l2x2);

double b2 = (l2x1 \* l2y2 - l2y1 \* l2x2) / (double)(l2x1 - l2x2);

double y0 = l1x1 \* k2 + b2;

if (Math.Min(l1y1, l1y2) > y0 || Math.Max(l1y1, l1y2) < y0 || Math.Min(l2y1, l2y2) > y0 || Math.Max(l2y1, l2y2) < y0)

return false;

}

else if (l2x1 == l2x2)

{

double k1 = (l1y1 - l1y2) / (double)(l1x1 - l1x2);

double b1 = (l1x1 \* l1y2 - l1y1 \* l1x2) / (double)(l1x1 - l1x2);

double y0 = l2x1 \* k1 + b1;

if (Math.Min(l1y1, l1y2) > y0 || Math.Max(l1y1, l1y2) < y0 || Math.Min(l2y1, l2y2) > y0 || Math.Max(l2y1, l2y2) < y0)

return false;

}

else

{

double k1 = (l1y1 - l1y2) / (double)(l1x1 - l1x2);

double b1 = (l1x1 \* l1y2 - l1y1 \* l1x2) / (double)(l1x1 - l1x2);

double k2 = (l2y1 - l2y2) / (double)(l2x1 - l2x2);

double b2 = (l2x1 \* l2y2 - l2y1 \* l2x2) / (double)(l2x1 - l2x2);

double x0 = (b2 - b1) / (k1 - k2);

if (Math.Min(l1x1, l1x2) > x0 || Math.Max(l1x1, l1x2) < x0 || Math.Min(l2x1, l2x2) > x0 || Math.Max(l2x1, l2x2) < x0)

return false;

/\*if ((l2y1 >= l2x1 \* k + b) == (l2y2 >= l2x2 \* k + b))

return false;\*/

}

return true;

}

//Point

void newInputs(int inputsnumber)

{

CircuitInput[] a = Inputs;

Inputs = new CircuitInput[inputsnumber];

int gapbetweeninputs = (PanelHeight - GV.jackradius \* inputsnumber) / (inputsnumber + 1);

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

{

if (a == null || i >= a.Length)

{

int name = 0;

for (int j = 0; j < Inputs.Length; j++)

if (Inputs[j] != null && Inputs[j].Name == "a" + name.ToString())

{

name++;

j = -1;

}

Inputs[i] = new CircuitInput(PanelX, PanelY + (i + 1) \* gapbetweeninputs + i \* GV.jackradius, "a" + name.ToString());

}

else

{

Inputs[i] = a[i];

Inputs[i].Y = PanelY + (i + 1) \* gapbetweeninputs + i \* GV.jackradius;

}

}

}

void newInputs(string[] inputs)

{

Inputs = new CircuitInput[inputs.Length];

int gapbetweeninputs = (PanelHeight - GV.jackradius \* inputs.Length) / (inputs.Length + 1);

for (int i = 0; i < inputs.Length; i++)

Inputs[i] = new CircuitInput(PanelX, PanelY + (i + 1) \* gapbetweeninputs + i \* GV.jackradius, inputs[i]);

}

void newOutputs(int outputsnumber)

{

CircuitOutput[] b = Outputs;

Outputs = new CircuitOutput[outputsnumber];

int gapbetweenoutputs = (PanelHeight - GV.jackradius \* outputsnumber) / (outputsnumber + 1);

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

{

if (b == null || i >= b.Length)

{

int name = 0;

for (int j = 0; j < Outputs.Length; j++)

if (Outputs[j] != null && Outputs[j].Name == "b" + name.ToString())

{

name++;

j = -1;

}

Outputs[i] = new CircuitOutput(PanelX + PanelWidth, PanelY + (i + 1) \* gapbetweenoutputs + i \* GV.jackradius, "b" + name.ToString());

}

else

{

Outputs[i] = b[i];

Outputs[i].Y = PanelY + (i + 1) \* gapbetweenoutputs + i \* GV.jackradius;

}

}

}

void newOutputs(string[] outputs)

{

CircuitOutput[] b = Outputs;

Outputs = new CircuitOutput[outputs.Length];

int gapbetweenoutputs = (PanelHeight - GV.jackradius \* outputs.Length) / (outputs.Length + 1);

for (int i = 0; i < outputs.Length; i++)

Outputs[i] = new CircuitOutput(PanelX + PanelWidth, PanelY + (i + 1) \* gapbetweenoutputs + i \* GV.jackradius, outputs[i]);

}

CircuitJack[] MoveLeftInMassive(CircuitJack[] a, CircuitJack b)

{

int movedfromnumber = (int)Tools.GetNumberInArray(a, b);

CircuitJack c = a[movedfromnumber - 1];

a[movedfromnumber - 1] = a[movedfromnumber];

a[movedfromnumber] = c;

return a;

}

CircuitJack[] MoveRightInMassive(CircuitJack[] a, CircuitJack b)

{

int movedfromnumber = (int)Tools.GetNumberInArray(a, b);

CircuitJack c = a[movedfromnumber + 1];

a[movedfromnumber + 1] = a[movedfromnumber];

a[movedfromnumber] = c;

return a;

}

/\* CircuitJack[] DeleteFromMassive(CircuitJack[] a, CircuitJack b)

{

CircuitJack[] c = new CircuitJack[a.Length - 1];

byte isalreadydeleted = 0;

for (int i = 0; i < a.Length; i++)

{

if (a[i] == b)

isalreadydeleted = 1;

else

c[i - isalreadydeleted] = a[i];

}

return c;

}\*/

void MakeRightCursor(int x, int y)

{

bool abletomove = false;

if (PanelHit(x, y))

{

foreach (LogicGate i in LogicGates)

{

if (i.HitCheck(x, y) && i.InputHitNumber(x, y) == i.Inputs.Length && i.OutputHitNumber(x, y) == i.Outputs.Length)

{

this.Cursor = new Cursor(new Bitmap(Properties.Resources.OpenHand).GetHicon());

abletomove = true;

break;

}

}

}

if (this.Cursor != DefaultCursor && !abletomove)

this.Cursor = DefaultCursor;

}

BigSchemeContainer BigDataSave()

{

return new BigSchemeContainer(schemesymbol, schemename, Inputs, Outputs, LogicGates, Wires, GV.SpecialNotOutput, UsedSchemes);

}

SchemeContainer DataSave()

{

return new SchemeContainer(schemesymbol, schemename, Scheme.ConvertToScheme(schemename, Inputs, Outputs, LogicGates, Wires), CircuitInput.GetStringArray(Inputs), CircuitOutput.GetStringArray(Outputs), UsedSchemes);

}

void BigDataOpen(BigSchemeContainer Data)

{

schemesymbol = Data.SchemeSymbol;

schemename = Data.SchemeName;

this.Text = schemename;

Inputs = Data.Inputs;

Outputs = Data.Outputs;

LogicGates = Data.LogicGates;

Wires = Data.Wires;

GV.SpecialNotOutput = Data.SpecialNotOutput;

UsedSchemes = Data.UsedSchemes;

if (!GV.FreeDevelopment)

{

for (int i = 0; i < Inputs.Length; i++)

Inputs[i].Name = ActiveTask.Inputs[i];

for (int i = 0; i < Outputs.Length; i++)

Outputs[i].Name = ActiveTask.Outputs[i];

}

Refresh();

}

void NewScheme()

{

schemesymbol = 's';

schemename = "newscheme";

SchemeNameForm SchemeName = new SchemeNameForm(schemesymbol, schemename);

SchemeName.ShowDialog();

schemesymbol = SchemeName.formschemesymbol;

schemename = SchemeName.formschemename;

this.Text = schemename;

FirstPoint = new Point();

Inputs = null;

Outputs = null;

newInputs(2);

newOutputs(1);

LogicGates = new LinkedList<LogicGate>();

Wires = new LinkedList<Wire>();

ChosenGate = null;

ChosenWire = null;

OperatorsMovement = false;

WiringFromBeginning = false;

WiringFromEnd = false;

TryingToCut = false;

richTextBox1.Text = "";

GV.ShowingResults = false;

StartOrStopShowingResults();

UsedSchemes = new LinkedList<SchemeContainer>();

Refresh();

}

void OpenScheme()

{

openSchemeDialog.ShowDialog();

}

private void openSchemeDialog\_FileOk(object sender, CancelEventArgs e)

{

OpenSchemeOk();

}

void OpenSchemeOk()

{

BinaryFormatter bf = new BinaryFormatter();

FileStream fs = new FileStream(openSchemeDialog.FileName, FileMode.Open, FileAccess.Read);

object ob = bf.Deserialize(fs);

if (ob is BigSchemeContainer)

{

BigSchemeContainer Data = (BigSchemeContainer)ob;

fs.Close();

BigDataOpen(Data);

}

else

MessageBox.Show("Выбран файл неправильного типа!");

}

void AddScheme()

{

addSchemeDialog.ShowDialog();

}

private void addSchemeDialog\_FileOk(object sender, CancelEventArgs e)

{

AddSchemeOk();

}

void AddSchemeOk()

{

BinaryFormatter bf = new BinaryFormatter();

FileStream fs = new FileStream(addSchemeDialog.FileName, FileMode.Open, FileAccess.Read);

object ob = bf.Deserialize(fs);

if (ob is BigSchemeContainer)

{

BigSchemeContainer Data = (BigSchemeContainer)ob;

fs.Close();

UsedSchemes.AddLast(new SchemeContainer(Data.SchemeSymbol, Data.SchemeName, Scheme.ConvertToScheme(Data.SchemeName, Data.Inputs, Data.Outputs, Data.LogicGates, Data.Wires), CircuitInput.GetStringArray(Data.Inputs), CircuitOutput.GetStringArray(Data.Outputs), Data.UsedSchemes));

LogicGates.AddFirst(new LogicGate(PanelX + 2 \* GV.radius, PanelY + GV.radius, CircuitInput.GetStringArray(Data.Inputs), CircuitOutput.GetStringArray(Data.Outputs), Data.SchemeSymbol, Data.SchemeName));

}

else if (ob is SchemeContainer)

{

SchemeContainer Data = (SchemeContainer)ob;

fs.Close();

UsedSchemes.AddLast(Data);

LogicGates.AddFirst(new LogicGate(PanelX + 2 \* GV.radius, PanelY + GV.radius, Data.Inputs, Data.Outputs, Data.SchemeSymbol, Data.SchemeName));

}

else

{

MessageBox.Show("Выбран файл неправильного типа!");

}

}

void OpenTask()

{

openTaskDialog.ShowDialog();

}

private void openTaskDialog\_FileOk(object sender, CancelEventArgs e)

{

OpenTaskOk();

}

void OpenTaskOk()

{

BinaryFormatter bf = new BinaryFormatter();

FileStream fs = new FileStream(openTaskDialog.FileName, FileMode.Open, FileAccess.Read);

object ob = bf.Deserialize(fs);

if (ob is Task)

{

Task Data = (Task)ob;

fs.Close();

ActiveTask = Data;

}

else

MessageBox.Show("Выбран файл неправильного типа!");

}

void SaveScheme()

{

if (lastsavewasfull && saveAllFileDialog.FileName != "")

SaveAllOk();

else if (!lastsavewasfull && saveLogicFileDialog.FileName != "")

SaveLogicOk();

else

SaveAll();

}

void SaveAll()

{

saveAllFileDialog.FileName = schemename;

saveAllFileDialog.ShowDialog();

}

private void saveAllFileDialog\_FileOk(object sender, CancelEventArgs e)

{

SaveAllOk();

}

void SaveAllOk()

{

BigSchemeContainer Data = new BigSchemeContainer(BigDataSave());

BinaryFormatter bf = new BinaryFormatter();

FileStream fs = new FileStream(saveAllFileDialog.FileName, FileMode.Create, FileAccess.Write);

bf.Serialize(fs, Data);

fs.Close();

lastsavewasfull = true;

}

void SaveLogic()

{

saveLogicFileDialog.FileName = schemename;

saveLogicFileDialog.ShowDialog();

}

private void saveLogicFileDialog\_FileOk(object sender, CancelEventArgs e)

{

SaveLogicOk();

}

void SaveLogicOk()

{

SchemeContainer Data = new SchemeContainer(DataSave());

BinaryFormatter bf = new BinaryFormatter();

FileStream fs = new FileStream(saveLogicFileDialog.FileName, FileMode.Create, FileAccess.Write);

bf.Serialize(fs, Data);

fs.Close();

lastsavewasfull = false;

}

void SaveTask()

{

string scheme = Scheme.ConvertToScheme(schemename, Inputs, Outputs, LogicGates, Wires);

foreach (LogicGate i in LogicGates)

{

i.Valueisknown = false;

}

if (scheme == null)

MessageBox.Show("Ошибка при обработке схемы!");

else

{

bool everythingisawesome = true;

bool[] a = new bool[Inputs.Length];

int numberofvariants = (int)Math.Pow(2, Inputs.Length);

bool[,] b = new bool[numberofvariants, Outputs.Length];

bool[] storage;

int i, i2, j;

for (i = 0; i < numberofvariants; i++)

{

i2 = i;

for (j = 0; j < a.Length; j++)

{

a[j] = i2 % 2 == 1;

i2 >>= 1;

}

storage = Scheme.RunScheme(scheme, UsedSchemes, a);

if (storage == null || storage.Length != Outputs.Length)

{

everythingisawesome = false;

MessageBox.Show("Ошибка при попытке запустить схему!");

break;

}

for (j = 0; j < storage.Length; j++)

b[i, j] = storage[j];

}

if (everythingisawesome)

{

TaskForm taskform = new TaskForm();

taskform.ShowDialog();

ActiveTask = new Task(taskform.stask, CircuitInput.GetStringArray(Inputs), CircuitOutput.GetStringArray(Outputs), b, taskform.andisallowed, taskform.orisallowed, taskform.notisallowed, taskform.zeroisallowed, taskform.otherschemesareallowed);

saveTaskFileDialog.FileName = schemename;

saveTaskFileDialog.ShowDialog();

}

}

}

private void saveTaskFileDialog\_FileOk(object sender, CancelEventArgs e)

{

SaveTaskOk();

}

void SaveTaskOk()

{

Task Data = ActiveTask;

BinaryFormatter bf = new BinaryFormatter();

FileStream fs = new FileStream(saveTaskFileDialog.FileName, FileMode.Create, FileAccess.Write);

bf.Serialize(fs, Data);

fs.Close();

}

void RenameScheme()

{

SchemeNameForm SchemeName = new SchemeNameForm(schemesymbol, schemename);

SchemeName.ShowDialog();

schemesymbol = SchemeName.formschemesymbol;

schemename = SchemeName.formschemename;

this.Text = schemename;

}

void StartOrStopShowingResults()

{

addinputbutton.Enabled = !GV.ShowingResults && GV.FreeDevelopment;

addoutputbutton.Enabled = !GV.ShowingResults && GV.FreeDevelopment;

addandbutton.Enabled = !GV.ShowingResults && (GV.FreeDevelopment || ActiveTask.AndIsAllowed);

addorbutton.Enabled = !GV.ShowingResults && (GV.FreeDevelopment || ActiveTask.OrIsAllowed);

addnotbutton.Enabled = !GV.ShowingResults && (GV.FreeDevelopment || ActiveTask.NotIsAllowed);

addzerobutton.Enabled = !GV.ShowingResults && (GV.FreeDevelopment || ActiveTask.ZeroIsAllowed);

runschemebutton.Text = (GV.ShowingResults) ? "Изменить схему" : "Запустить схему";

}

void StartOrStopFreeDevelopment()

{

показатьToolStripMenuItem.Enabled = !GV.FreeDevelopment;

instructionStripButton.Enabled = !GV.FreeDevelopment;

showresultStripButton.Enabled = !GV.FreeDevelopment;

добавитьToolStripMenuItem.Enabled = GV.FreeDevelopment || ActiveTask.OtherSchemesAreAllowed;

addschemeStripButton.Enabled = GV.FreeDevelopment || ActiveTask.OtherSchemesAreAllowed;

открытьToolStripMenuItem.Enabled = GV.FreeDevelopment;

openschemeStripButton.Enabled = GV.FreeDevelopment;

addinputbutton.Enabled = GV.FreeDevelopment;

addoutputbutton.Enabled = GV.FreeDevelopment;

addandbutton.Enabled = GV.FreeDevelopment || ActiveTask.AndIsAllowed;

addorbutton.Enabled = GV.FreeDevelopment || ActiveTask.OrIsAllowed;

addnotbutton.Enabled = GV.FreeDevelopment || ActiveTask.NotIsAllowed;

addzerobutton.Enabled = GV.FreeDevelopment || ActiveTask.ZeroIsAllowed;

}

void Basictasks()

{

BasicTasks = new Task[13];

bool[,] truthtable;

truthtable = new bool[4, 1];

truthtable[0, 0] = false;

truthtable[1, 0] = true;

truthtable[2, 0] = true;

truthtable[3, 0] = true;

BasicTasks[0] = new Task(@"

Спроектировать схему ИЛИ, пользуясь только схемами И и НЕ.

", new string[2] { "a0", "a1" }, new string[1] { "b0" }, truthtable, true, false, true, false, false);

truthtable = new bool[8, 1];

truthtable[0, 0] = false;

for (int i = 1; i < 8; i++)

truthtable[i, 0] = true;

BasicTasks[1] = new Task(@"

Составить схему, вычисляющую ИЛИ от трех входов (на выходе 1, если хотя бы на одном из входов 1).

", new string[3] { "a0", "a1", "a2" }, new string[1] { "b0" }, truthtable, true, true, true, true, false);

truthtable = new bool[16, 1];

truthtable[0, 0] = false;

for (int i = 1; i < 16; i++)

truthtable[i, 0] = true;

BasicTasks[2] = new Task(@"

Составить схему, вычисляющую ИЛИ от четырех входов (на выходе 1, если хотя бы на одном из входов 1).

", new string[4] { "a0", "a1", "a2", "a3" }, new string[1] { "b0" }, truthtable, true, true, true, true, false);

truthtable = new bool[4, 1];

truthtable[0, 0] = true;

truthtable[1, 0] = false;

truthtable[2, 0] = false;

truthtable[3, 0] = true;

BasicTasks[3] = new Task(@"

Составить схему РАВЕНСТВО с двумя входами и одним выходом. На выходе 1, если входы равны, и 0, если входы разные.

", new string[2] { "a0", "a1" }, new string[1] { "b0" }, truthtable, true, true, true, true, false);

truthtable = new bool[4, 1];

truthtable[0, 0] = false;

truthtable[1, 0] = true;

truthtable[2, 0] = true;

truthtable[3, 0] = false;

BasicTasks[4] = new Task(@"

Составить схему ИСКЛЮЧАЮЩЕЕ ИЛИ с двумя входами и одним выходом. На выходе 0, если входы равны, и 1, если входы разные.

", new string[2] { "a0", "a1" }, new string[1] { "b0" }, truthtable, true, true, true, true, false);

truthtable = new bool[8, 1];

truthtable[0, 0] = false;

truthtable[1, 0] = false;

truthtable[2, 0] = false;

truthtable[3, 0] = true;

truthtable[4, 0] = true;

truthtable[5, 0] = false;

truthtable[6, 0] = true;

truthtable[7, 0] = true;

BasicTasks[5] = new Task(@"

Составить схему с тремя входами (a0, a1, select), на выходе которой a0 если select=0 и a1 если select=1.

", new string[3] { "a0", "a1", "select" }, new string[1] { "b0" }, truthtable, true, true, true, true, false);

truthtable = new bool[8, 1];

truthtable[0, 0] = false;

truthtable[1, 0] = false;

truthtable[2, 0] = false;

truthtable[3, 0] = true;

truthtable[4, 0] = false;

truthtable[5, 0] = true;

truthtable[6, 0] = true;

truthtable[7, 0] = true;

BasicTasks[6] = new Task(@"

Составить схему с тремя входами, у которой выход равен 1 или 0 в зависимости от того, каких входов больше (то есть на выходе 1, если на входах хотя бы две 1, и 0, если на входах хотя бы два 0).

", new string[3] { "a0", "a1", "a2" }, new string[1] { "b0" }, truthtable, true, true, true, true, false);

truthtable = new bool[4, 1];

truthtable[0, 0] = false;

truthtable[1, 0] = true;

truthtable[2, 0] = true;

truthtable[3, 0] = false;

BasicTasks[7] = new Task(@"

Составить схему, на выходе которой сумма двух двоичных разрядов (перенос в старший разряд отбрасывается; то есть, на выходе 1, если ровно на одном входе 1).

", new string[2] { "a0", "a1" }, new string[1] { "b0" }, truthtable, true, true, true, true, false);

truthtable = new bool[4, 2];

truthtable[0, 0] = false;

truthtable[1, 0] = true;

truthtable[2, 0] = true;

truthtable[3, 0] = false;

truthtable[0, 1] = false;

truthtable[1, 1] = false;

truthtable[2, 1] = false;

truthtable[3, 1] = true;

BasicTasks[8] = new Task(@"

Составить схему с двумя входами и двумя выходами. На выходах схемы - сумма двух входов (оба разряда).

", new string[2] { "a0", "b0" }, new string[2] { "s0", "p1" }, truthtable, true, true, true, true, false);

truthtable = new bool[8, 2];

truthtable[0, 0] = false;

truthtable[1, 0] = true;

truthtable[2, 0] = true;

truthtable[3, 0] = false;

truthtable[4, 0] = true;

truthtable[5, 0] = false;

truthtable[6, 0] = false;

truthtable[7, 0] = true;

truthtable[0, 1] = false;

truthtable[1, 1] = false;

truthtable[2, 1] = false;

truthtable[3, 1] = true;

truthtable[4, 1] = false;

truthtable[5, 1] = true;

truthtable[6, 1] = true;

truthtable[7, 1] = true;

BasicTasks[9] = new Task(@"

Составить схему с тремя входами и двумя выходами. На выходах схемы - сумма трех входов (оба разряда).

", new string[3] { "a0", "b0", "p0" }, new string[2] { "s0", "p1" }, truthtable, true, true, true, true, true);

truthtable = new bool[32, 3];

int j = 0;

int storage = 0;

for (byte p0 = 0; p0 < 2; p0++)

for (byte b1 = 0; b1 < 2; b1++)

for (byte b0 = 0; b0 < 2; b0++)

for (byte a1 = 0; a1 < 2; a1++)

for (byte a0 = 0; a0 < 2; a0++)

{

storage = a0 + b0 + p0;

truthtable[j, 0] = storage % 2 == 1;

storage = a1 + b1 + ((storage > 1) ? 1 : 0);

truthtable[j, 1] = storage % 2 == 1;

truthtable[j, 2] = storage > 1;

j++;

}

BasicTasks[10] = new Task(@"

Составить схему, на входах которой два 2-битовых числа и еще один бит переноса, а на выходах - сумма этих трех слагаемых (3 бита).

", new string[5] { "a0", "a1", "b0", "b1", "p0" }, new string[3] { "s0", "s1", "p2" }, truthtable, true, true, true, true, true);

truthtable = new bool[512, 5];

j = 0;

storage = 0;

for (byte p0 = 0; p0 < 2; p0++)

for (byte b3 = 0; b3 < 2; b3++)

for (byte b2 = 0; b2 < 2; b2++)

for (byte b1 = 0; b1 < 2; b1++)

for (byte b0 = 0; b0 < 2; b0++)

for (byte a3 = 0; a3 < 2; a3++)

for (byte a2 = 0; a2 < 2; a2++)

for (byte a1 = 0; a1 < 2; a1++)

for (byte a0 = 0; a0 < 2; a0++)

{

storage = a0 + b0 + p0;

truthtable[j, 0] = storage % 2 == 1;

storage = a1 + b1 + ((storage > 1) ? 1 : 0);

truthtable[j, 1] = storage % 2 == 1;

storage = a2 + b2 + ((storage > 1) ? 1 : 0);

truthtable[j, 2] = storage % 2 == 1;

storage = a3 + b3 + ((storage > 1) ? 1 : 0);

truthtable[j, 3] = storage % 2 == 1;

truthtable[j, 4] = storage > 1;

j++;

}

BasicTasks[11] = new Task(@"

Составить схему, на входах которой два 4-битовых числа и еще один бит переноса, а на выходах - сумма этих трех слагаемых (5 битов).

", new string[9] { "a0", "a1", "a2", "a3", "b0", "b1", "b2", "b3", "p0" }, new string[5] { "s0", "s1", "s2", "s3", "p4" }, truthtable, true, true, true, true, true);

truthtable = new bool[256, 8];

j = 0;

storage = 0;

for (byte b3 = 0; b3 < 2; b3++)

for (byte b2 = 0; b2 < 2; b2++)

for (byte b1 = 0; b1 < 2; b1++)

for (byte b0 = 0; b0 < 2; b0++)

for (byte a3 = 0; a3 < 2; a3++)

for (byte a2 = 0; a2 < 2; a2++)

for (byte a1 = 0; a1 < 2; a1++)

for (byte a0 = 0; a0 < 2; a0++)

{

storage = a0 \* b0;

truthtable[j, 0] = storage % 2 == 1;

storage = a1 \* b0 + a0 \* b1 + ((storage > 1) ? 1 : 0);

truthtable[j, 1] = storage % 2 == 1;

storage = a2 \* b0 + a1 \* b1 + a0 \* b2 + ((storage > 1) ? 1 : 0);

truthtable[j, 2] = storage % 2 == 1;

storage = a3 \* b0 + a2 \* b1 + a1 \* b2 + a0 \* b3 + ((storage > 1) ? 1 : 0);

truthtable[j, 3] = storage % 2 == 1;

storage = a3 \* b1 + a2 \* b2 + a1 \* b3 + ((storage > 1) ? 1 : 0);

truthtable[j, 4] = storage % 2 == 1;

storage = a3 \* b2 + a2 \* b3 + ((storage > 1) ? 1 : 0);

truthtable[j, 5] = storage % 2 == 1;

storage = a3 \* b3 + ((storage > 1) ? 1 : 0);

truthtable[j, 6] = storage % 2 == 1;

truthtable[j, 7] = storage > 1;

j++;

}

BasicTasks[12] = new Task(@"

Построить схему, умножающую два 4-разрядных числа.

", new string[8] { "a0", "a1", "a2", "a3", "b0", "b1", "b2", "b3" }, new string[8] { "m0", "m1", "m2", "m3", "m4", "m5", "m6", "m7" }, truthtable, true, true, true, true, true);

}

void Starttask(Task a)

{

NewScheme();

newInputs(a.Inputs);

newOutputs(a.Outputs);

ShowGoal();

}

void ShowGoal()

{

string goal = "Задача:\n";

goal += ActiveTask.STask;

goal += "\nИспользовать элемент И ";

if (!ActiveTask.AndIsAllowed) goal += "не ";

goal += "разрешено.";

goal += "\nИспользовать элемент ИЛИ ";

if (!ActiveTask.OrIsAllowed) goal += "не ";

goal += "разрешено.";

goal += "\nИспользовать элемент НЕ ";

if (!ActiveTask.NotIsAllowed) goal += "не ";

goal += "разрешено.";

goal += "\nИспользовать сигнал \"0\" ";

if (!ActiveTask.ZeroIsAllowed) goal += "не ";

goal += "разрешено.";

goal += "\nИспользовать другие схемы ";

if (!ActiveTask.OtherSchemesAreAllowed) goal += "не ";

goal += "разрешено.";

MessageBox.Show(goal);

}

void ShowResult()

{

string scheme = Scheme.ConvertToScheme(schemename, Inputs, Outputs, LogicGates, Wires);

foreach (LogicGate i in LogicGates)

{

i.Valueisknown = false;

}

if (scheme == null)

MessageBox.Show("Ошибка при обработке схемы!");

else

{

bool everythingisawesome = true;

bool[] a = new bool[Inputs.Length];

int numberofvariants = (int)Math.Pow(2, Inputs.Length);

bool[][] b = new bool[numberofvariants][];

int i, i2, j;

for (i = 0; i < numberofvariants; i++)

{

i2 = i;

for (j = 0; j < a.Length; j++)

{

a[j] = i2 % 2 == 1;

i2 >>= 1;

}

b[i] = Scheme.RunScheme(scheme, UsedSchemes, a);

if (b[i] == null || b[i].Length != Outputs.Length)

{

everythingisawesome = false;

MessageBox.Show("Ошибка при попытке запустить схему!");

}

else

{

for (j = 0; j < Outputs.Length; j++)

{

if (b[i][j] != ActiveTask.Truthtable[i, j])

{

everythingisawesome = false;

break;

}

}

}

if (!everythingisawesome)

break;

}

if (everythingisawesome)

MessageBox.Show("Задача решена верно!");

else

MessageBox.Show("Задача не решена!");

}

}

#endregion

#region Справка

void ShowHelp()

{

//Logic Circuit Help.hnd

//Help.ShowHelp(this, @"./Resources/efts.chm");

//Properties.Resources.Логические\_схемы

Help.ShowHelp(this, @"./Resources/LCHelp.chm");

}

void ShowInfo()

{

string message =

"Данная программа - обучающая программа для создания логических схем с использованием графического редактора для персональных компьютеров и планшетов на базе операционных систем семейства Microsoft Windows.\n\n" +

"Автор: Копылов Олег (2014)\n\n" +

//"Application path: " + Path.Combine(Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location), Path.GetFileNameWithoutExtension(System.Reflection.Assembly.GetExecutingAssembly().Location) + ".exe") + Environment.NewLine +

"Базовая директория приложения: " + Directory.GetCurrentDirectory();

MessageBox.Show(message, "Версия программы v. " + version, MessageBoxButtons.OK, MessageBoxIcon.Information);

}

#endregion

}

public partial class ChangeNameForm : Form

{

public ChangeNameForm(string name)

{

InitializeComponent();

textBox1.Text = name;

this.name = name;

}

public string name;

private void savenamebutton\_Click(object sender, EventArgs e)

{

textBox1.Text = textBox1.Text.Replace(" ", string.Empty);

if (textBox1.Text != "")

name = textBox1.Text;

//name = (textBox1.Text == null) ? "" : textBox1.Text;

this.Close();

}

}

public partial class SchemeNameForm : Form

{

public SchemeNameForm(char schemesymbol, string schemename)

{

InitializeComponent();

textBox1.Text = schemename;

textBox2.Text = schemesymbol.ToString();

this.formschemesymbol = schemesymbol;

this.formschemename = schemename;

}

public char formschemesymbol;

public string formschemename;

private void changenameschemebutton\_Click(object sender, EventArgs e)

{

textBox1.Text = textBox1.Text.Replace(" ", string.Empty);

if (textBox1.Text != "")

formschemename = textBox1.Text;

if (textBox2.Text != "" || textBox2.Text != " ")

formschemesymbol = char.Parse(textBox2.Text);

this.Close();

}

}

public partial class TaskForm : Form

{

public TaskForm()

{

InitializeComponent();

stask = "";

}

public string stask;

public bool andisallowed;

public bool orisallowed;

public bool notisallowed;

public bool zeroisallowed;

public bool otherschemesareallowed;

private void changenameschemebutton\_Click(object sender, EventArgs e)

{

stask = textBox1.Text;

andisallowed = checkBox1.Checked;

orisallowed = checkBox2.Checked;

notisallowed = checkBox3.Checked;

zeroisallowed = checkBox4.Checked;

otherschemesareallowed = checkBox5.Checked;

this.Close();

}

}

[Serializable]

class BigSchemeContainer

{

char schemesymbol;

string schemename;

CircuitInput[] inputs;

CircuitOutput[] outputs;

LinkedList<LogicGate> logicgates;

LinkedList<Wire> wires;

bool specialnotoutput;

LinkedList<SchemeContainer> usedschemes;

public char SchemeSymbol

{

get { return schemesymbol; }

set { schemesymbol = value; }

}

public string SchemeName

{

get { return schemename; }

set { schemename = value; }

}

public CircuitInput[] Inputs

{

get { return inputs; }

}

public CircuitOutput[] Outputs

{

get { return outputs; }

}

public LinkedList<LogicGate> LogicGates

{

get { return logicgates; }

}

public LinkedList<Wire> Wires

{

get { return wires; }

}

public bool SpecialNotOutput

{

get { return specialnotoutput; }

}

public LinkedList<SchemeContainer> UsedSchemes

{

get { return usedschemes; }

}

public BigSchemeContainer(char SchemeSymbol, string SchemeName, CircuitInput[] Inputs, CircuitOutput[] Outputs, LinkedList<LogicGate> LogicGates, LinkedList<Wire> Wires, bool SpecialNotOutput, LinkedList<SchemeContainer> UsedSchemes)

{

schemesymbol = SchemeSymbol;

schemename = SchemeName;

inputs = Inputs;

outputs = Outputs;

logicgates = LogicGates;

wires = Wires;

specialnotoutput = SpecialNotOutput;

usedschemes = UsedSchemes;

}

public BigSchemeContainer(BigSchemeContainer OldData)

{

schemesymbol = OldData.SchemeSymbol;

schemename = OldData.SchemeName;

inputs = OldData.Inputs;

outputs = OldData.Outputs;

logicgates = OldData.LogicGates;

wires = OldData.Wires;

specialnotoutput = OldData.SpecialNotOutput;

usedschemes = OldData.UsedSchemes;

}

}

class BooleanString

{

bool? bvalue;

string svalue;

public bool? BoolValue

{

get { return bvalue; }

set { bvalue = value; }

}

public string StringValue

{

get { return svalue; }

set { svalue = value; }

}

public static int? GetNumberInArray(BooleanString[] a, string b)

{

for (int i = 0; i < a.Length; i++)

{

if (a[i].StringValue == b)

return i;

}

return null;

}

public BooleanString(string stringvalue)

{

svalue = stringvalue;

bvalue = null;

}

}

[Serializable]

abstract class CircuitComponent

{

public abstract int X { get; set; }

public abstract int Y { get; set; }

public abstract int GetInputCentralYCoordinate(Wire thisWire);

public abstract int GetOutputCentralYCoordinate(Wire thisWire);

public abstract void Draw(Graphics graph);

public abstract bool HitCheck(int mouseX, int mouseY);

}

[Serializable]

class CircuitInput : CircuitJack

{

bool jackvalue;

int x;

int y;

string name;

LinkedList<Wire> input;

public override bool JackValue

{

get { return jackvalue; }

set { jackvalue = value; }

}

public override int X

{

get { return x; }

set { x = value; }

}

public override int Y

{

get { return y; }

set { y = value; }

}

public override int GetInputCentralYCoordinate(Wire thisWire)

{

return y;

}

public override int GetOutputCentralYCoordinate(Wire thisWire)

{

return y;

}

public override string Name

{

get { return name; }

set { name = value; }

}

public LinkedList<Wire> Input

{

get { return input; }

}

public override void Draw(Graphics graph)

{

graph.FillRectangle(new SolidBrush(Color.Black), x - GV.jackradius, y - GV.jackradius, GV.jackdiameter, GV.jackdiameter);

graph.DrawString((jackvalue) ? "1" : "0", new Font("Arial", 2 \* GV.radius / 5), new SolidBrush(Color.Black), x - 3 \* GV.jackradius, y - 3 \* GV.jackradius);

graph.DrawString(name, new Font("Arial", 2 \* GV.radius / 5), new SolidBrush(Color.Black), x - (2 + 3 \* name.Length / 2) \* GV.jackradius, y + GV.jackradius);

}

public override bool HitCheck(int mouseX, int mouseY)

{

if ((x - GV.jackradius <= mouseX && mouseX <= x + GV.jackradius) && (y - GV.jackradius <= mouseY && mouseY <= y + GV.jackradius))

return true;

return false;

}

public static bool[] GetBoolArray(CircuitInput[] Inputs)

{

bool[] a = new bool[Inputs.Length];

for (int i = 0; i < a.Length; i++)

a[i] = Inputs[i].JackValue;

return a;

}

public static string[] GetStringArray(CircuitInput[] Inputs)

{

string[] a = new string[Inputs.Length];

for (int i = 0; i < a.Length; i++)

a[i] = Inputs[i].Name;

return a;

}

public CircuitInput(int x, int y, string name)

{

jackvalue = false;

this.x = x;

this.y = y;

this.name = name;

input = new LinkedList<Wire>();

}

}

[Serializable]

abstract class CircuitJack : CircuitComponent

{

public abstract bool JackValue { get; set; }

public abstract string Name { get; set; }

}

[Serializable]

class CircuitOutput : CircuitJack

{

bool jackvalue;

int x;

int y;

string name;

Wire output;

public override bool JackValue

{

get { return jackvalue; }

set { jackvalue = value; }

}

public override int X

{

get { return x; }

set { x = value; }

}

public override int Y

{

get { return y; }

set { y = value; }

}

public override int GetInputCentralYCoordinate(Wire thisWire)

{

return y;

}

public override int GetOutputCentralYCoordinate(Wire thisWire)

{

return y;

}

public override string Name

{

get { return name; }

set { name = value; }

}

public Wire Output

{

get { return output; }

set { output = value; }

}

public override void Draw(Graphics graph)

{

graph.FillRectangle(new SolidBrush(Color.Black), x - GV.jackradius, y - GV.jackradius, GV.jackdiameter, GV.jackdiameter);

if (GV.ShowingResults)

graph.DrawString((jackvalue) ? "1" : "0", new Font("Arial", 2 \* GV.radius / 5), new SolidBrush(Color.Black), x + GV.jackradius, y - 3 \* GV.jackradius);

graph.DrawString(name, new Font("Arial", 2 \* GV.radius / 5), new SolidBrush(Color.Black), x + GV.jackradius, y);

}

public override bool HitCheck(int mouseX, int mouseY)

{

if ((x - GV.jackradius <= mouseX && mouseX <= x + GV.jackradius) && (y - GV.jackradius <= mouseY && mouseY <= y + GV.jackradius))

return true;

return false;

}

public static string[] GetStringArray(CircuitOutput[] Outputs)

{

string[] a = new string[Outputs.Length];

for (int i = 0; i < a.Length; i++)

a[i] = Outputs[i].Name;

return a;

}

public CircuitOutput(int x, int y, string name)

{

jackvalue = false;

this.x = x;

this.y = y;

this.name = name;

name = "";

}

}

class GV // GlobalVariables

{

public static int radius;

public static int jackdiameter;

public static int jackradius;

//public static int firstgap;

public static bool SpecialNotOutput;

public static bool ShowingResults;

public static bool FreeDevelopment;

}

[Serializable]

class LogicGate : CircuitComponent

{

int x;

int y;

char schemesymbol;

string schemename;

Wire[] inputs;

string[] inputsnames;

LinkedList<Wire>[] outputs;

string[] outputsnames;

bool valueisknown;

public override int X

{

get { return x; }

set { x = value; }

}

public override int Y

{

get { return y; }

set { y = value; }

}

public override int GetInputCentralYCoordinate(Wire thisWire)

{

int maxnumber = Math.Max(inputs.Length, outputs.Length);

int i;

for (i = 0; i < inputs.Length; i++)

if (inputs[i] == thisWire)

break;

return y + i \* maxnumber \* GV.radius / inputs.Length + ((maxnumber \* GV.radius / inputs.Length) / 2 - GV.jackradius) + GV.jackradius;

}

public override int GetOutputCentralYCoordinate(Wire thisWire)

{

int maxnumber = Math.Max(inputs.Length, outputs.Length);

int i;

for (i = 0; i < outputs.Length; i++)

if (outputs[i].Contains(thisWire))

break;

return y + i \* maxnumber \* GV.radius / outputs.Length + ((maxnumber \* GV.radius / outputs.Length) / 2 - GV.jackradius) + GV.jackradius;

}

public char SchemeSymbol

{

get { return schemesymbol; }

set { schemesymbol = value; }

}

public string SchemeName

{

get { return schemename; }

set { schemename = value; }

}

public Wire[] Inputs

{

get { return inputs; }

}

public LinkedList<Wire>[] Outputs

{

get { return outputs; }

}

public bool Valueisknown

{

get { return valueisknown; }

set { valueisknown = value; }

}

public override void Draw(Graphics graph)

{

int maxnumber = Math.Max(inputs.Length, outputs.Length);

graph.FillRectangle(new SolidBrush(Color.FromKnownColor(KnownColor.Control)), x, y, GV.radius, maxnumber \* GV.radius);

graph.DrawRectangle(new Pen(Color.Black), x, y, GV.radius, maxnumber \* GV.radius);

for (int i = 0; i < inputs.Length; i++)

{

graph.FillRectangle(new SolidBrush(Color.Black), x - GV.jackradius, y + i \* maxnumber \* GV.radius / inputs.Length + ((maxnumber \* GV.radius / inputs.Length) / 2 - GV.jackradius), GV.jackdiameter, GV.jackdiameter);

graph.DrawString(inputsnames[i], new Font("Arial", 2 \* GV.radius / 5), new SolidBrush(Color.Black), x - (2 + 3 \* inputsnames[i].Length / 2) \* GV.jackradius, y + i \* maxnumber \* GV.radius / inputsnames.Length + ((maxnumber \* GV.radius / inputsnames.Length) / 2 - GV.jackradius) - 3 \* GV.jackradius);

}

for (int i = 0; i < outputs.Length; i++)

{

if (schemesymbol == '!' && GV.SpecialNotOutput)

{

graph.FillRectangle(new SolidBrush(Color.FromKnownColor(KnownColor.Control)), x + GV.radius - GV.jackradius, y + i \* maxnumber \* GV.radius / outputs.Length + ((maxnumber \* GV.radius / outputs.Length) / 2 - GV.jackradius), GV.jackdiameter, GV.jackdiameter);

graph.DrawRectangle(new Pen(Color.Black), x + GV.radius - GV.jackradius, y + i \* maxnumber \* GV.radius / outputs.Length + ((maxnumber \* GV.radius / outputs.Length) / 2 - GV.jackradius), GV.jackdiameter, GV.jackdiameter);

}

else

graph.FillRectangle(new SolidBrush(Color.Black), x + GV.radius - GV.jackradius, y + i \* maxnumber \* GV.radius / outputs.Length + ((maxnumber \* GV.radius / outputs.Length) / 2 - GV.jackradius), GV.jackdiameter, GV.jackdiameter);

graph.DrawString(outputsnames[i], new Font("Arial", 2 \* GV.radius / 5), new SolidBrush(Color.Black), x + GV.radius, y + i \* maxnumber \* GV.radius / outputsnames.Length + ((maxnumber \* GV.radius / outputsnames.Length) / 2 - GV.jackradius) - 3 \* GV.jackradius);

}

graph.DrawString(schemesymbol.ToString(), new Font("Arial", 2 \* GV.radius / 5), new SolidBrush(Color.Black), x + 3 \* GV.radius / 10, y + GV.radius / 10);

}

public override bool HitCheck(int mouseX, int mouseY)

{

if (x <= mouseX && mouseX <= x + GV.radius && y <= mouseY && mouseY <= y + Math.Max(inputs.Length, outputs.Length) \* GV.radius)

return true;

return false;

}

public int InputHitNumber(int mouseX, int mouseY)

{

int inputnumber = 0;

int maxnumber = Math.Max(inputs.Length, outputs.Length);

if (x - GV.jackradius <= mouseX && mouseX <= x + GV.jackradius)

for (; inputnumber < inputs.Length; inputnumber++)

if (inputs[inputnumber] == null && y + inputnumber \* maxnumber \* GV.radius / inputs.Length + ((maxnumber \* GV.radius / inputs.Length) / 2 - GV.jackradius) <= mouseY && mouseY <= y + inputnumber \* maxnumber \* GV.radius / inputs.Length + ((maxnumber \* GV.radius / inputs.Length) / 2 - GV.jackradius) + GV.jackdiameter)

return inputnumber;

return inputs.Length;

}

public int InputHitNumber(Wire thisWire)

{

int inputnumber = 0;

for (; inputnumber < inputs.Length; inputnumber++)

if (inputs[inputnumber] == thisWire)

return inputnumber;

return inputs.Length;

}

public int OutputHitNumber(int mouseX, int mouseY)

{

int outputnumber = 0;

int maxnumber = Math.Max(inputs.Length, outputs.Length);

if (x + GV.radius - GV.jackradius <= mouseX && mouseX <= x + GV.radius + GV.jackradius)

for (; outputnumber < outputs.Length; outputnumber++)

if (y + outputnumber \* maxnumber \* GV.radius / outputs.Length + ((maxnumber \* GV.radius / outputs.Length) / 2 - GV.jackradius) <= mouseY && mouseY <= y + outputnumber \* maxnumber \* GV.radius / outputs.Length + ((maxnumber \* GV.radius / outputs.Length) / 2 - GV.jackradius) + GV.jackdiameter)

return outputnumber;

return outputs.Length;

}

public int OutputHitNumber(Wire thisWire)

{

int outputnumber = 0;

for (; outputnumber < outputs.Length; outputnumber++)

if (outputs[outputnumber].Contains(thisWire))

return outputnumber;

return outputs.Length;

}

public LogicGate(int x, int y, string[] inputsnames, string[] outputsnames, char schemesymbol, string schemename)

{

this.x = x;

this.y = y;

this.schemesymbol = schemesymbol;

this.schemename = schemename;

valueisknown = false;

inputs = new Wire[inputsnames.Length];

this.inputsnames = inputsnames;

outputs = new LinkedList<Wire>[outputsnames.Length];

for (int i = 0; i < outputs.Length; i++)

outputs[i] = new LinkedList<Wire>();

this.outputsnames = outputsnames;

}

}

class Scheme

{

public static string ConvertToScheme(string schemename, CircuitInput[] Inputs, CircuitOutput[] Outputs, LinkedList<LogicGate> LogicGates, LinkedList<Wire> Wires)

{

string scheme = "scheme";// "Максимальный размер объекта String в 2 ГБ памяти, или около 1 миллиард символов." - MSDN

LogicGate[] logicgates = new LogicGate[LogicGates.Count];

LogicGates.CopyTo(logicgates, 0);

scheme += " (";

for (int i = 0; i < Inputs.Length; i++)

scheme += " " + Inputs[i].Name;

scheme += " ) " + schemename + " (";

for (int i = 0; i < Outputs.Length; i++)

scheme += " " + Outputs[i].Name;

scheme += " ) :" + Environment.NewLine;

scheme += "local";

for (int i = 0; i < logicgates.Length; i++)

for (int j = 0; j < logicgates[i].Outputs.Length; j++)

if (logicgates[i].Outputs[j].Count != 0)

{

/\*

int l = 0;

foreach (Wire k in logicgates[i].Outputs[j])

if (k.End is LogicGate)

{

k.Name = i.ToString() + "." + j.ToString() + "." + l.ToString();

scheme += " " + k.Name;

l++;

}

\*/

scheme += " " + i.ToString() + "." + j.ToString();

}

scheme += Environment.NewLine;

bool isdone = false;

bool smthisdone = false;

while (!isdone)

{

isdone = true;

for (int i = 0; i < logicgates.Length; i++)

{

if (!logicgates[i].Valueisknown)

{

isdone = false;

bool allinputsareknown = true;

for (int j = 0; j < logicgates[i].Inputs.Length; j++)

{

if (logicgates[i].Inputs[j] == null)

return null;

if (logicgates[i].Inputs[j].Beginning is LogicGate && !(logicgates[i].Inputs[j].Beginning as LogicGate).Valueisknown)

{

allinputsareknown = false;

break;

}

}

if (allinputsareknown)

{

smthisdone = true;

logicgates[i].Valueisknown = true;

if (logicgates[i].Inputs != null)

{

scheme += "\t(";

for (int j = 0; j < logicgates[i].Inputs.Length; j++)

{

scheme += " ";

if (logicgates[i].Inputs[j].Beginning is CircuitInput)

scheme += (logicgates[i].Inputs[j].Beginning as CircuitInput).Name;

else

//scheme += logicgates[i].Inputs[j].Name;

scheme += Tools.GetNumberInArray(logicgates, logicgates[i].Inputs[j].Beginning).ToString() + "." + Tools.GetNumberInArray((logicgates[i].Inputs[j].Beginning as LogicGate).Outputs, logicgates[i].Inputs[j]);

/\*if (logicgates[(int)Tools.GetNumberInArray(logicgates, logicgates[i].Inputs[j].Beginning)].Outputs[(int)Tools.GetNumberInArray((logicgates[i].Inputs[j].Beginning as LogicGate).Outputs, logicgates[i].Inputs[j])].Count != 0)

{

int l = 0;

foreach (Wire k in logicgates[(int)Tools.GetNumberInArray(logicgates, logicgates[i].Inputs[j].Beginning)].Outputs[(int)Tools.GetNumberInArray((logicgates[i].Inputs[j].Beginning as LogicGate).Outputs, logicgates[i].Inputs[j])])

if (k.End is CircuitOutput)

scheme += " " + (k.End as CircuitOutput).Name;

else

{

scheme += " " + Tools.GetNumberInArray(logicgates, logicgates[i].Inputs[j].Beginning).ToString() + "." + Tools.GetNumberInArray((logicgates[i].Inputs[j].Beginning as LogicGate).Outputs, logicgates[i].Inputs[j]).ToString() + "." + l.ToString();

l++;

}

} \*/

}

scheme += " ) " + logicgates[i].SchemeSymbol.ToString() + " (";

for (int j = 0; j < logicgates[i].Outputs.Length; j++)

if (logicgates[i].Outputs[j].Count != 0)

{

/\*

foreach (Wire k in logicgates[i].Outputs[j])

{

scheme += " ";

if (k.End is CircuitOutput)

scheme += (k.End as CircuitOutput).Name;

else

scheme += k.Name;

}

\*/

scheme += " " + i.ToString() + "." + j.ToString();

}

scheme += " )" + Environment.NewLine;

}

}

}

}

if (!smthisdone)

return null;

}

for (int i = 0; i < Outputs.Length; i++)

{

if (Outputs[i].Output == null)

return null;

scheme += "\t( ";

if (Outputs[i].Output.Beginning is CircuitInput)

scheme += (Outputs[i].Output.Beginning as CircuitInput).Name;

else

scheme += Tools.GetNumberInArray(logicgates, Outputs[i].Output.Beginning).ToString() + "." + Tools.GetNumberInArray((Outputs[i].Output.Beginning as LogicGate).Outputs, Outputs[i].Output);

scheme += " ) = ( ";

scheme += Outputs[i].Name;

scheme += " )" + Environment.NewLine;

}

/\* for (int i = 0; i < logicgates.Length; i++)

logicgates[i].Valueisknown = false;\*/

scheme += "end";

return scheme;

}

public static bool[] RunScheme(string scheme, LinkedList<SchemeContainer> usedschemes, bool[] a)

{

char[] array = new char[scheme.Length];

string schemecopy = "";

for (int i = 0; i < scheme.Length; i++)

schemecopy += scheme[scheme.Length - 1 - i];

//shemecopy = array.ToString();

if (schemecopy[schemecopy.Length - 1] == 's') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'c') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'h') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'e') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'm') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'e') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == '(') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

LinkedList<string> InputsNamesList = new LinkedList<string>();

while (schemecopy[schemecopy.Length - 1] != ')')

{

InputsNamesList.AddLast("");

while (schemecopy[schemecopy.Length - 1] != ' ')

{

InputsNamesList.Last.Value += schemecopy[schemecopy.Length - 1];

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

if (InputsNamesList.Count == 0 || InputsNamesList.Count != a.Length)

return null;

string[] InputsNamesArray = new string[InputsNamesList.Count];

InputsNamesList.CopyTo(InputsNamesArray, 0);

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

while (schemecopy[schemecopy.Length - 1] != ' ')

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

if (schemecopy[schemecopy.Length - 1] == '(') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

LinkedList<string> OutputsNamesList = new LinkedList<string>();

while (schemecopy[schemecopy.Length - 1] != ')')

{

OutputsNamesList.AddLast("");

while (schemecopy[schemecopy.Length - 1] != ' ')

{

OutputsNamesList.Last.Value += schemecopy[schemecopy.Length - 1];

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

if (OutputsNamesList.Count == 0)

return null;

string[] OutputsNamesArray = new string[OutputsNamesList.Count];

OutputsNamesList.CopyTo(OutputsNamesArray, 0);

bool[] b = new bool[OutputsNamesList.Count];

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == ':') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == '\r') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == '\n') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'l') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'o') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'c') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'a') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'l') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

//if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

LinkedList<BooleanString> WiresList = new LinkedList<BooleanString>();

while (schemecopy[schemecopy.Length - 1] != '\r')

{

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

WiresList.AddLast(new BooleanString(""));

while (schemecopy[schemecopy.Length - 1] != '\r' && schemecopy[schemecopy.Length - 1] != ' ')

{

WiresList.Last.Value.StringValue += schemecopy[schemecopy.Length - 1];

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

if (WiresList.Last.Value.StringValue == "")

return null;

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

BooleanString[] WiresArray = new BooleanString[WiresList.Count];

WiresList.CopyTo(WiresArray, 0);

if (schemecopy[schemecopy.Length - 1] == '\n') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

string circuitcomponentname;

LinkedList<bool> inputs;// всё false?

bool[] outputs = new bool[1];

int outputnumber;

while (schemecopy[schemecopy.Length - 1] == '\t')

{

inputs = new LinkedList<bool>();

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

if (schemecopy[schemecopy.Length - 1] == '(') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

while (schemecopy[schemecopy.Length - 1] != ')')

{

circuitcomponentname = "";

while (schemecopy[schemecopy.Length - 1] != ' ')

{

circuitcomponentname += schemecopy[schemecopy.Length - 1];

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

if (InputsNamesArray.Contains(circuitcomponentname))

inputs.AddLast(a[(int)Tools.GetNumberInArray(InputsNamesArray, circuitcomponentname)]);

else if (BooleanString.GetNumberInArray(WiresArray, circuitcomponentname) != null)

//if (WiresArray[(int)BooleanString.GetNumberInArray(WiresArray, circuitcomponentname)].StringValue == "0")

//inputs.AddLast(false);

//else

inputs.AddLast((bool)WiresArray[(int)BooleanString.GetNumberInArray(WiresArray, circuitcomponentname)].BoolValue);

else return null;

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

switch (schemecopy[schemecopy.Length - 1])

{

case '&':

if (inputs.Count != 2)

return null;

outputs = new bool[1];

outputs[0] = inputs.First.Value & inputs.First.Next.Value;

break;

case '1':

if (inputs.Count != 2)

return null;

outputs = new bool[1];

outputs[0] = inputs.First.Value | inputs.First.Next.Value;

break;

case '!':

if (inputs.Count != 1)

return null;

outputs = new bool[1];

outputs[0] = !inputs.First.Value;

break;

/\*case '0':

if (inputs.Count != 0)

return null;

outputs = new bool[1];

outputs[0] = false;

break;\*/

case '=':

if (inputs.Count != 1)

return null;

outputs = new bool[1];

outputs[0] = inputs.First.Value;

break;

default:

SchemeContainer i = SchemeContainer.NecessaryNode(usedschemes, schemecopy[schemecopy.Length - 1]);

if (i != null)

{

bool[] inputsarray = new bool[inputs.Count];

inputs.CopyTo(inputsarray, 0);

outputs = RunScheme(i.SchemeText, i.UsedSchemes, inputsarray);

}

break;

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == '(') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == ' ') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

outputnumber = -1;

while (schemecopy[schemecopy.Length - 1] != ')')

{

outputnumber++;

if (outputs.Length == outputnumber)

return null;

circuitcomponentname = "";

while (schemecopy[schemecopy.Length - 1] != ' ')

{

circuitcomponentname += schemecopy[schemecopy.Length - 1];

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

}

if (OutputsNamesArray.Contains(circuitcomponentname))

b[(int)Tools.GetNumberInArray(OutputsNamesArray, circuitcomponentname)] = outputs[outputnumber];

else if (BooleanString.GetNumberInArray(WiresArray, circuitcomponentname) != null)

WiresArray[(int)BooleanString.GetNumberInArray(WiresArray, circuitcomponentname)].BoolValue = outputs[outputnumber];

else return null;

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

//outputnumber++;

}

schemecopy = schemecopy.Remove(schemecopy.Length - 1);

if (schemecopy[schemecopy.Length - 1] == '\r') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == '\n') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

}

if (schemecopy[schemecopy.Length - 1] == 'e') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'n') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

if (schemecopy[schemecopy.Length - 1] == 'd') schemecopy = schemecopy.Remove(schemecopy.Length - 1); else return null;

return b;

}

}

[Serializable]

class SchemeContainer

{

char schemesymbol;

string schemename;

string schemetext;

string[] inputs;

string[] outputs;

LinkedList<SchemeContainer> usedschemes;

public char SchemeSymbol

{

get { return schemesymbol; }

set { schemesymbol = value; }

}

public string SchemeName

{

get { return schemename; }

set { schemename = value; }

}

public string SchemeText

{

get { return schemetext; }

set { schemetext = value; }

}

public string[] Inputs

{

get { return inputs; }

}

public string[] Outputs

{

get { return outputs; }

}

public LinkedList<SchemeContainer> UsedSchemes

{

get { return usedschemes; }

set { usedschemes = value; }

}

public static SchemeContainer NecessaryNode(LinkedList<SchemeContainer> l, char c)

{

foreach (SchemeContainer i in l)

if (i.schemesymbol == c)

return i;

return null;

}

public SchemeContainer(char SchemeSymbol, string SchemeName, string SchemeText, string[] Inputs, string[] Outputs, LinkedList<SchemeContainer> UsedSchemes)

{

schemesymbol = SchemeSymbol;

schemename = SchemeName;

schemetext = SchemeText;

inputs = Inputs;

outputs = Outputs;

usedschemes = UsedSchemes;

}

public SchemeContainer(SchemeContainer OldData)

{

schemesymbol = OldData.SchemeSymbol;

schemename = OldData.SchemeName;

schemetext = OldData.SchemeText;

inputs = OldData.Inputs;

outputs = OldData.Outputs;

usedschemes = OldData.UsedSchemes;

}

}

[Serializable]

class Task

{

string stask;

string[] inputs;

string[] outputs;

bool[,] truthtable;

/\*

\* 1 вход:

\* 0ой вариант - 0

\* 1ый вариант - 1

\* 2 входа:

\* 0ой вариант - 00

\* 1ый вариант - 01

\* 2ой вариант - 10

\* 3ий вариант - 11

\* 1 вход:

\* 0ой вариант - 000

\* 1ый вариант - 001

\* 2ой вариант - 010

\* 3ий вариант - 011

\* 4ый вариант - 100

\* 5ый вариант - 101

\* 6ой вариант - 110

\* 7ой вариант - 111

\* и т.д.

\*/

bool andisallowed;

bool orisallowed;

bool notisallowed;

bool zeroisallowed;

bool otherschemesareallowed;

public string STask

{

get { return stask; }

}

public string[] Inputs

{

get { return inputs; }

}

public string[] Outputs

{

get { return outputs; }

}

public bool[,] Truthtable

{

get { return truthtable; }

}

public bool AndIsAllowed

{

get { return andisallowed; }

}

public bool OrIsAllowed

{

get { return orisallowed; }

}

public bool NotIsAllowed

{

get { return notisallowed; }

}

public bool ZeroIsAllowed

{

get { return zeroisallowed; }

}

public bool OtherSchemesAreAllowed

{

get { return otherschemesareallowed; }

}

public Task(string stask, string[] inputs, string[] outputs, bool[,] truthtable, bool andisallowed, bool orisallowed, bool notisallowed, bool zeroisallowed, bool otherschemesareallowed)

{

this.inputs = inputs;

this.outputs = outputs;

this.truthtable = truthtable;

this.stask = stask;

this.andisallowed = andisallowed;

this.orisallowed = orisallowed;

this.notisallowed = notisallowed;

this.zeroisallowed = zeroisallowed;

this.otherschemesareallowed = otherschemesareallowed;

}

}

class Tools

{

public static int? GetNumberInArray(CircuitComponent[] a, CircuitComponent b)

{

for (int i = 0; i < a.Length; i++)

if (a[i] == b)

return i;

return null;

}

public static int? GetNumberInArray(LinkedList<Wire>[] a, Wire b)

{

for (int i = 0; i < a.Length; i++)

if (a[i].Contains(b))

return i;

return null;

}

public static int? GetNumberInArray(string[] a, string b)

{

for (int i = 0; i < a.Length; i++)

if (a[i].Contains(b))

return i;

return null;

}

/\*public static int? GetNumberInArray(LinkedList<Wire>[] a, Wire b)

{

for (int i = 0; i < a.Length; i++)

if (a[i].Contains(b))

return i;

return null;

}\*/

}

[Serializable]

class Wire

{

CircuitComponent beginning;

CircuitComponent end;

public CircuitComponent Beginning

{

get { return beginning; }

set { beginning = value; }

}

public CircuitComponent End

{

get { return end; }

set { end = value; }

}

public void Draw(Graphics graph)

{

if (beginning is CircuitJack)

graph.DrawLine(new Pen(Color.Black), beginning.X, beginning.GetOutputCentralYCoordinate(this), end.X, end.GetInputCentralYCoordinate(this));

else

graph.DrawLine(new Pen(Color.Black), beginning.X + GV.radius, beginning.GetOutputCentralYCoordinate(this), end.X, end.GetInputCentralYCoordinate(this));

}

public void Draw(Graphics graph, int mouseX, int mouseY)

{

if (beginning == null)

graph.DrawLine(new Pen(Color.Black), mouseX, mouseY, end.X, end.GetInputCentralYCoordinate(this));

else

{

if (beginning is CircuitJack)

graph.DrawLine(new Pen(Color.Black), beginning.X, beginning.Y, mouseX, mouseY);

else

graph.DrawLine(new Pen(Color.Black), beginning.X + GV.radius, beginning.GetOutputCentralYCoordinate(this), mouseX, mouseY);

}

}

public Wire(CircuitComponent Point, bool PointIsBeginning)

{

if (PointIsBeginning)

beginning = Point;

else

end = Point;

}

public Wire(CircuitComponent beginning, CircuitComponent end)

{

this.beginning = beginning;

this.end = end;

}

}

}