ТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2024 г.

Автоматизация проверки работоспособности функциональных схем в курсовых проектах по теории автоматов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ В.А. Парасич
«___»_____2024 г.

Автор работы,
студент группы КЭ-405
_____Н.А. Кузнецов
«___»_____2024 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2024 г.

Челябинск-2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2024 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Кузнецов Никита Алексеевич
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Автоматизация проверки работоспособности функциональных схем в курсовых проектах по теории автоматов» утверждена приказом по университету от «22» 04 2024 г. № 764-13/12

2. **Срок сдачи студентом законченной работы:** 01 июня 2024 г.

3. **Исходные данные к работе:**

3.1. Курсовая работа по теории автоматов:

- автомат задается совмещенной таблицей переходов;
- элементы памяти: D – триггер, JK – триггер, RS – триггер, T – триггер;
- в результате синтеза получаем систему минимизированных уравнений и соответствующую функциональную схему.

3.2. Требования к функционалу разрабатываемого приложения:

- удобный и понятный интерфейс;

- построение функциональной схемы на основе минимизированных уравнений, которые записал пользователь;
- проверка функциональной схемы на всех типах триггеров.

3.3. Операционная система: Microsoft Windows 7 и выше.

3.4. Целевая аудитория приложения: студенты и преподаватели IT специальностей.

Перечень подлежащих разработке вопросов:

Выпускная квалификационная работа (ВКР) должна содержать разработку следующих вопросов:

1. Анализ и обзор программного обеспечения для проверки функциональных схем по теории автоматов.
2. Выбор средств разработки для реализации поставленной задачи.
3. Архитектурное проектирование приложения.
4. Программная реализация.

Дата выдачи задания: 1 декабря 2023 г.

Руководитель работы _____/ *В.А. Парасич/*

Студент _____/ Н.А. Кузнецов /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2024	
Анализ аналогов разрабатываемого приложения	12.03.2024	
Выбор среды разработки приложения	24.03.2024	
Проектирование архитектуры приложения	03.04.2024	
Реализация системы	18.04.2024	
Тестирование и отладка	02.05.2024	
Компоновка текста работы и сдача на нормоконтроль	15.05.2024	
Подготовка презентации и доклада	30.05.2024	

Руководитель работы _____/ *В.А. Парасич/*

Студент _____/ *Н.А. Кузнецов* /

АННОТАЦИЯ

Кузнецов Н.А. Автоматизация проверки работоспособности функциональных схем в курсовых проектах по теории автоматов – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2024, 49 с., библиогр. список – 16 наим.

В рамках выпускной квалификационной работы проведен аналитический обзор аналогов и программного обеспечения. Определены основные требования к проекту, как функциональные, так и нефункциональные.

Были спроектированы, разработаны, реализованы и протестированы алгоритмы проверки и построения функциональной схемы в курсовых проектах по теории автоматов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном информационном обществе, где цифровые технологии проникают во все сферы жизни, предмет "Теория автоматов" занимает важное место в образовательном процессе. Этот учебный курс предоставляет студентам ключевые инструменты и концепции для понимания основных принципов функционирования вычислительных устройств, программных систем и языков программирования [1].

Теория автоматов занимается изучением абстрактных вычислительных устройств и машин. В 1930-е годы, задолго до появления компьютеров, Тьюринг исследовал абстрактную машину, которая, по крайней мере в области вычислений, обладала всеми возможностями современных вычислительных машин. Целью Тьюринга было точно описать границу между тем, что вычислительная машина может делать, и тем, чего она не может. Полученные им результаты применимы не только к абстрактным машинам Тьюринга, но и к реальным современным компьютерам [2].

Написание курсового проекта для лабораторной работы по теории автоматов используют ручную проверку функциональной схемы, хотя очевидно, что это неэффективно. Поскольку и преподаватель тратит лишние силы для проверки всех работ студентов, так и студенты тратят много времени, чтобы написать курсовой проект. К тому же, преподаватель может проверить только одно или два состояния и не найти ошибки в курсовой работе, благодаря разрабатываемому мной приложению, можно будет проверить все возможные состояния, что поможет обнаружить ошибку в функциональной схеме и облегчит нелегкую работу преподавателя, а также исправит влияние человеческого фактора при проверке или написании работы.

1 АНАЛИЗ И ОБЗОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕАЛИЗАЦИИ ПРОВЕРКИ ФУНКЦИОНАЛЬНОЙ СХЕМЫ ЦИФРОВЫХ АВТОМАТОВ

Перед составлением целей для ВКР, следует разобраться, какие задачи нужно выполнить в курсовом проекте по теории автоматов. Студенту требуется:

– построить структурный цифровой автомат на основе заданной таблицы переходов и выходов;

– составить систему уравнений;

– минимизировать уравнения;

– построить функциональную схему на основе составленных минимизированных уравнений;

– провести проверку функциональной схемы по таблицам переходов и выходов.

Для автоматизации процесса проверки работоспособности функциональных схем в курсовых проектах по теории автоматов пользователю необходимо самому составить минимизированные уравнения. Благодаря минимизированным уравнениям можно построить и проверить функциональную схему.

Для анализа предметной области было рассмотрено множество программ, которые каким-либо образом связаны с конечными автоматами и минимизированными уравнениями, но ни одно из них не проверяет функциональную схему с поддержкой графического интерфейса, поэтому для анализа были выбраны приложения, наиболее подходящие под мои условия для ВКР.

Дипломная работа Щеголева А.В. на тему «Разработка симулятора для дистанционного выполнения лабораторных работ по теории автоматов»

В данной работе студент сделал симулятор, который предоставляет теоретические материалы, симулятор машины Тьюринга, симулятор машины

Поста, симулятор автомата Мура, симулятор автомата Мили, симулятор синтез автомата и минимизацию карт Карно. Но у него нет проверки функциональных схем по заданным значениям. На основе этих данных можно сделать вывод о достоинствах и недостатках данного приложения.

Достоинства:

- симуляция множества машин и автоматов;
- теоретическая информация для обучения.

Недостатки:

- отсутствие проверки функциональной схемы.

Конструктор конечных автоматов за авторством Эвана Уоллеса — это онлайн конструктор предназначен для разработки конечного автомата[3]. Тут есть возможность проектирования конечного автомата, но отсутствует функционал для его взаимодействия. Исходя из данных, полученных путем изучения данной программы, можно сформулировать вывод о преимуществах и недостатках этого приложения.

Достоинства:

- проектирование конечного автомата.

Недостатки:

- отсутствие поддержки русского языка;
- ограниченный функционал приложения;
- невозможность проверки функциональных схем.

Logisim — инструмент, позволяющий разрабатывать и моделировать цифровые электрические схемы, используя графический интерфейс пользователя. Logisim — свободное программное обеспечение, выпущенное под GNU GPL; может запускаться на Microsoft Windows, Mac OS X, и Linux. Код полностью написан на Java с использованием библиотеки Swing для графического интерфейса пользователя. Основной разработчик, Carl Burch,

работает над Logisim с его появления в 2001 году[4]. Путем изучения и просмотром обучающих роликов по этой программе, можно сделать вывод о том, какие преимущества и недостатки есть в этом приложении.

Достоинства:

- поддержка русского языка и полная документация;
- применение графического интерфейса для более легкого понимания обучающимся;
- возможность редактировать схемы в процессе моделирования;
- возможность пользоваться при отсутствии интернет-соединения.

Недостатки:

- отсутствие проверки функциональной схемы.

На основе всех этих данных составлю сравнительную таблицу для аналогов приложений по следующим критериям:

- поддержка русского языка;
- проверка функциональной схемы;
- применение графического интерфейса для интуитивно-понятного использования программы.

Таблица 1 – Сравнительный анализ рассмотренных программ

Название аналога	Поддержка русского языка	Проверка функциональной схемы	Применение графического интерфейса
Дипломная работа Щеголева А.В	+	-	+

Продолжение таблицы 1

Конструктор конечных автоматов за авторством Эвана Уоллеса	-	-	+
Logisim	+	-	+

Вывод по главе 1

Исходя из вышеперечисленных результатов можно сделать вывод, что ни одна из программ не подходит для проверки функциональной схемы по теории автоматов. Следовательно, эти данные подтверждают актуальность моей темы.

2 ВЫБОР СРЕДЫ РАЗРАБОТКИ ДЛЯ РЕАЛИЗАЦИИ ПОСТАВЛЕННОЙ ЗАДАЧИ

2.1. Выбор языка программирования

В современном мире программирования выбор языка программирования становится ключевым аспектом для успешного развития и реализации проектов. При выборе языка необходимо учитывать специфику задачи, требования к проекту, а также уровень удобства и эффективности в разработке. В данном проекте, для реализации его было принято решение использовать язык программирования Java.

Выбор Java обусловлен рядом причин, которые сделали этот язык оптимальным для реализации поставленной задачи. Во-первых, Java предоставляет высокоуровневые инструменты для разработки приложений, что делает процесс программирования более прозрачным и доступным. Во-вторых, Java является объектно-ориентированным языком программирования, что позволяет более эффективно организовывать код и повышает его модульность. В-третьих, язык Java имеет большое количество графических библиотек, что является важным фактором при выборе языка, поскольку это требуется для визуализации функциональной схемы.

Таким образом, выбор языка программирования Java для данного проекта оправдан его функциональностью, удобством разработки и наличием необходимых инструментов для успешной реализации поставленной задачи.

2.2. Выбор фреймворка с поддержкой графического интерфейса

Для реализации проекта требуется поддержка графического интерфейса, потому что это облегчит понимание работы приложения, а также упростит работу с проверкой функциональной схемой. После определения языка программирования, следует выбрать подходящий фреймворк с поддержкой графического интерфейса, из всех существующих были рассмотрены следующие:

- Swing;

– SWT;
– JavaFX.

Swing

Swing — это набор компонентов графического пользовательского интерфейса (GUI), которые являются частью платформы Java. Компоненты Swing построены поверх Abstract Window Toolkit (AWT), но они предоставляют ряд преимуществ по сравнению с компонентами AWT. Например, компоненты Swing более легкие и эффективные, и они не зависят от платформы [5].

Swing предоставляет собой более гибкие интерфейс компоненты, чем библиотека AWT. Компоненты Swing созданы для одинаковой кроссплатформенной работы. AWT, в свою очередь, повторяет интерфейс исполняемой платформы, используя стандартные элементы ОС для отображения. То есть, для каждого элемента создается отдельный объект ОС (окно), из-за чего AWT не позволяет создавать элементы произвольной формы (возможно только использование прямоугольных компонентов), элементы управления на основе библиотеки всегда отображаются поверх Swing-элементов (так как все Swing-компоненты отображаются на поверхности контейнера) [6].

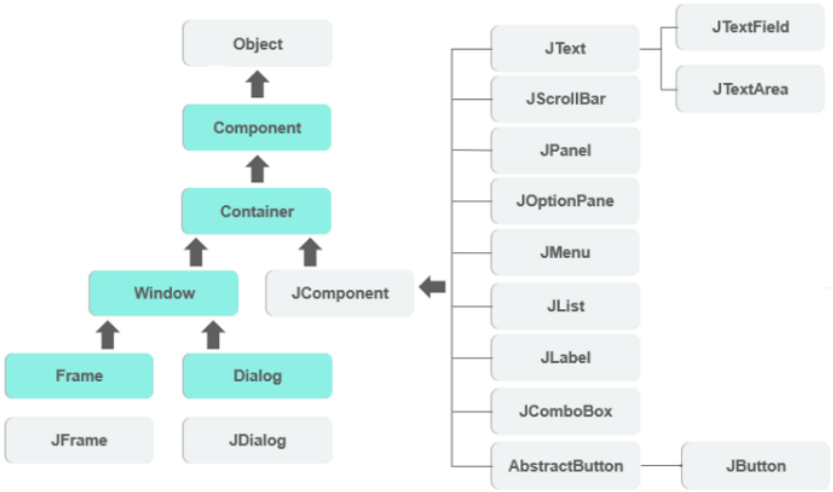

Рисунок 1 – Иерархия классов

Все компоненты в Swing, такие как JButton, JComboBox, JList, JLabel, унаследованы от класса JComponent, который можно добавить в классы контейнера.

Преимущества:

- совместим со всеми платформами и поддерживается множеством плагинов;
- один из самых популярных наборов инструментов с графическим интерфейсом, с большим сообществом и обширной документацией;
- Swing является частью библиотеки Java, значит нет необходимости в дополнительных встроенных библиотеках[7].

Недостатки:

- Swing напрямую подключен к JVM, что означает невозможность кроссплатформенного переноса Java-приложения с рабочего стола на мобильное устройство;
- ограниченная графическая модернизация. Создание современного и стильного интерфейса требует значительно больше усилий и времени.

SWT

SWT — библиотека представляет собой кросс-платформенную оболочку для графических библиотек конкретных операционных систем. SWT разработана с использованием стандартного языка Java и получает доступ к специфичным библиотекам различных ОС через JNI (Java Native Interface), который используется для доступа к родным визуальным компонентам операционной системы [8].

Плюсы SWT:

- использует собственные элементы, когда это возможно, поэтому всегда использует собственное поведение;
- поддерживается eclipse, редактором графического интерфейса VEP;
- имеет встроенный мост awt / swt, позволяющий использовать компоненты awt и swing.

Минусы SWT:

- требуются собственные библиотеки для каждой поддерживаемой системы;
- может не поддерживать любое поведение во всех системах из-за используемых собственных ресурсов (варианты подсказок).

JavaFX

JavaFX — платформа на основе Java для создания приложений с насыщенным графическим интерфейсом. Может использоваться как для создания настольных приложений, запускаемых непосредственно из-под операционных систем, так и для интернет-приложений (RIA), работающих в браузерах, и для приложений на мобильных устройствах. JavaFX призвана заменить использовавшуюся ранее библиотеку Swing[9].

Преимущества:

- JavaFX поддерживает все основные операционные системы, такие как Windows, macOS, Linux и даже мобильные платформы, например, Android и iOS;
- JavaFX предоставляет множество встроенных элементов управления, таких как кнопки, текстовые поля, таблицы, списки, слайдеры и многое другое;
- JavaFX позволяет использовать CSS для стилизации элементов интерфейса, что делает процесс дизайна легким и гибким;
- FXML — это XML-подобный язык разметки, который упрощает процесс создания и модификации графического интерфейса пользователя.

Недостатки:

На основе всех эти данных, можно составить сравнительную таблицу, по следующим критериям:

- актуальность;
- для каких целей используется;
- поддержка MVC;
- поддержка сообщества и регулярные обновления;
- поддержка CSS;
- декларативная работа;

Таблица 2 – Сравнительный анализ рассматриваемых фреймворков

Название фреймворка	Актуальность	Для чего используется	Сообщество и обновления	Css	Декларативная работа
Swing	В программном коде приложения могут встречаться устаревшие модули.	Используется для работы с обычным GUI	Новые опции в Swing не добавляются.	Не работает с CSS – стили задаются через программный код. Для многопоточности необходимо устанавливать дополнительные API.	Не поддерживает декларативную работу с макетами
SWT	Используется в Eclipse IDE	Используется в Eclipse IDE и других настольных приложениях	Активное сообщество, обновления в рамках Eclipse IDE	Не работает с CSS	Не поддерживает декларативную работу с макетами
JavaFX	Исходный код имеет высокую читабельность. Он является чистым.	Применяется для создания программного обеспечения с насыщенным пользовательским интерфейсом.	Поддерживается дружелюбным сообществом. Регулярно обновляется и совершенствуется.	Поддерживает CSS и встроенный API для многопоточности.	Оснащен FXML для декларативного создания макетов

На основе этих данных, решено было использовать JavaFX, поскольку он более легкий в применении и более современный, чем другие рассмотренные фреймворки.

2.3. Выбор среды разработки

В современном мире разработки программного обеспечения, где каждый день появляются новые технологии, выбор подходящей среды разработки становится ключевым фактором для повышения производительности и качества кода. Существует несколько сред разработки, которые можно использовать с языком Java. Будут рассмотрены самые популярные из них.

Eclipse

Eclipse — это интегрированная среда разработки (IDE) с широким функционалом для разработчиков программного обеспечения [10].

Преимущества Eclipse IDE:

- бесплатная с открытым исходным кодом;
- большое количество плагинов[11];
- доступная и огромная база документации.

Недостатки Eclipse IDE:

- частые проблемы с управлением версиями, в частности с Git[11];
- требует много системных ресурсов;
- отсутствует встроенное приложение Scene Builder.

NetBeans

Apache NetBeans – это бесплатная интегрированная среда разработки с открытым исходным кодом. Ее применяют для кодинга, тестирования и отладки приложений. NetBeans поддерживает множество языков программирования, среди которых есть Java, C, C++, JavaScript и PHP. Данная IDE поддерживается фондом Apache Foundation [12].

Преимущества NetBeans:

- открытый исходный код;
- доступная и огромная база документации;
- интеграция с популярными фреймворками и платформами [13].

Недостатки NetBeans:

- ограниченное сообщество;
- медленные обновления;
- отсутствует встроенное приложение Scene Builder.

Intellij IDEA

IntelliJ IDEA — это IDE, интегрированная среда разработки для Java, JavaScript, Python и других языков программирования от компании JetBrains. Отличается обширным набором инструментов для рефакторинга и оптимизации кода[14].

Преимущества Intellij IDEA:

- обширные интеграции с различными системами и сервисами;
- возможности рефакторинга;
- встроенное и поддерживаемое приложение Scene Builder.

Недостатки Intellij IDEA:

- потребление большого количества ресурсов[15];
- платная.

Анализ сред разработки

На основе проведенного обзора различных сред разработки, были выбраны следующие критерии для анализа IDE:

- поддержка Scene Builder;
- поддерживаемая версия Java;
- удобство интерфейса;

– мой личный опыт использования.

Таблица 3 – Сравнительный анализ сред разработки

Название IDE	Eclispe	IntelliJ IDEA	NetBeans
Поддерживаемая версия Java	Java 8 и выше	Java 6 и выше	Java 8 и выше
Встроенный Scene Builder	-	+	-
Удобство пользовательского интерфейса	-	+	+
Опыт использования	+	+	-

На основе таблицы, можно сделать вывод, что самой подходящей средой разработки для проекта является Intellij IDEA, поскольку встроенная и поддерживаемая программа Scene Builder, сильно облегчит работу с интерфейсом приложения, а также у меня уже есть опыт использования этой среды разработки.

Вывод по главе 2

После всех проведенных сравнительных анализов для проекта были выбраны следующие средства разработки:

- язык программирования Java;
- фреймворк JavaFX;
- среда разработки IntellijIDEA.

3 АРХИТЕКТУРНОЕ ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

Приложение должно не только проверять функциональную схему автомата, но также еще и строить эту схему. Для удобства пользователей следует добавить функцию сохранения и выгрузки входных данных из файла формата .txt.

3.1. Функциональные требования

Приложение должно выполнять следующие функции.

1. Проверку функциональной схемы через заданные минимизированные уравнения.
2. Выгрузку и сохранение файлов формата .txt, для быстрого заполнения входных данных.
3. Построение функциональной схемы.

3.2. Нефункциональные требования

К нефункциональным требованиям относятся:

1. Соответствовать минимальным системным требованиям:
   - ОС: Windows 7 и выше;
   - оперативная память: 2 Гб;
   - место на диске: не более 1 Гб.
2. Разработанное приложение должно быть написано на языке программирования Java, с использованием фреймоворк JavaFX.

3.3. Графический план прецедентов

Для описания функциональности и поведения приложения на этапе проектирования было решено создать диаграмму прецедентов. Диаграмма прецедентов — диаграмма, отражающая отношения между акторами и прецедентами. В нашем случае актором является пользователь, а прецедентами являются взаимодействия с программой. Пользователь, запуская приложение, может открыть справку, сохранить входные данные в файл, открыть файл с входными данными, редактировать входные данные и

построить функциональную схему. После построения функциональной схемы открывается новое окно, в котором пользователь может редактировать входные данные для проверки схемы и проверить функциональную схему. Данная диаграмма прецедентов предоставлена на рисунке 2.

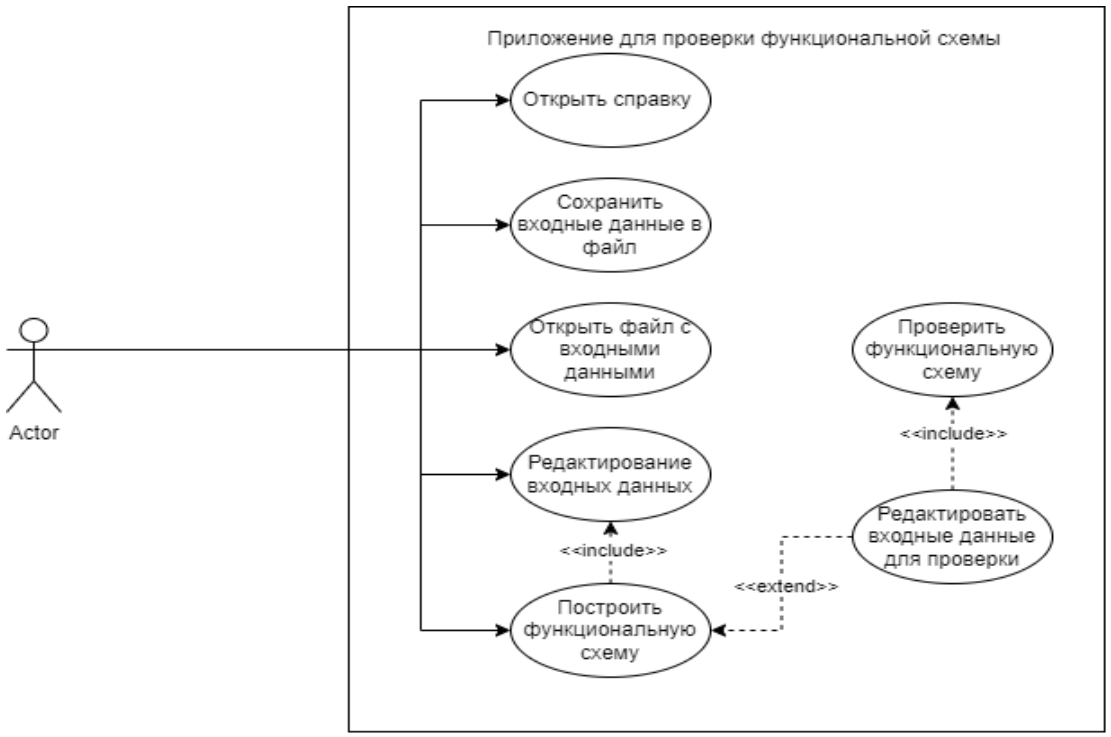

Рисунок 2 – Диаграмма прецедентов

3.4. Проектирование структуры интерфейса

Интерфейс окна «Построение функциональной схемы» должен состоять из:

1) кнопки «файл», при нажатии которой, пользователь сможет сохранить записанные входные данные в файл, открыть файл с входными данными и очистить все входные в приложении;

2) кнопка «помощь», при нажатии которой, пользователь может открыть справку, в которой есть вся необходимая информация о том, как пользоваться приложением;

3) поле ввода, где пользователь записывает количество входных сигналов;

4) раскрывающийся список, в которой пользователь выбирает тип триггера для своей функциональной схемы;

5) 4 колонки, в которые пользователь записывает минимизированные уравнения из своей курсовой работы;

6) кнопка «построить функциональную схему», при нажатии которой, открывается другое окно, где уже производится проверка функциональной схемы;

7) элемент выбора, при нажатии которого, будет выбрано строить функциональную схему с нумерацией или без.

Интерфейс окна «Проверка функциональной схемы» должен состоять из:

1) из построенной функциональной схемы, на основе данных, введенных в первом окне;

2) два поля для ввода, где пользователь вводит состояние входных сигналов и состояния автомата;

3) результат проверки.

Макеты графического интерфейса приложения представлены на рисунках 3-5.

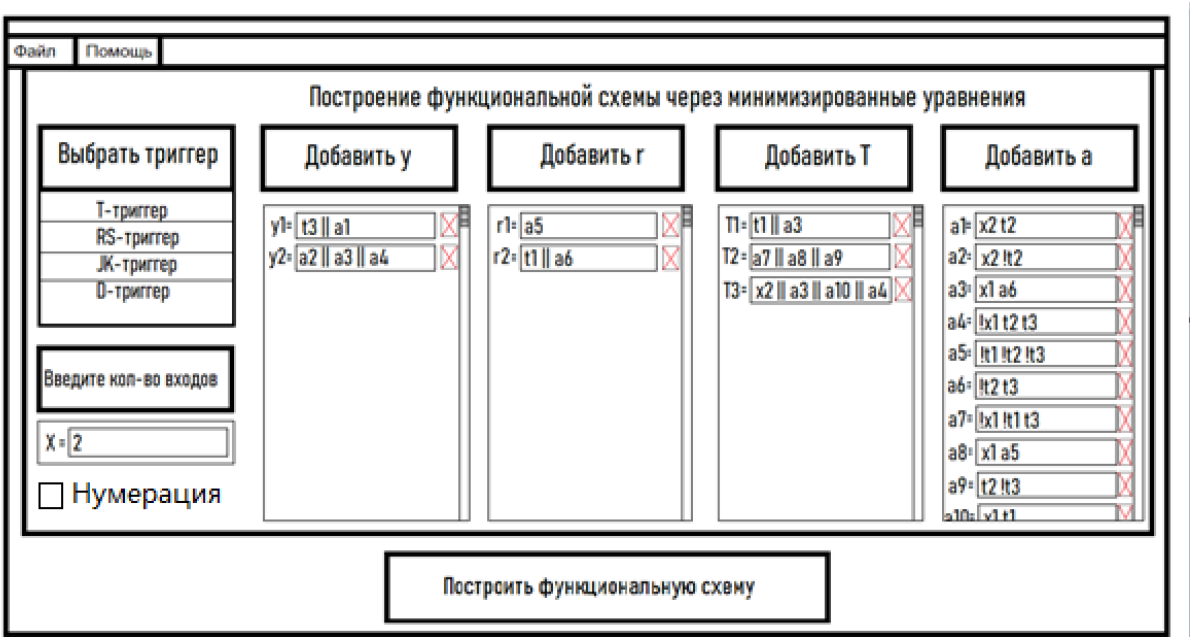

Рисунок 3 – Макет окна заполнения данными

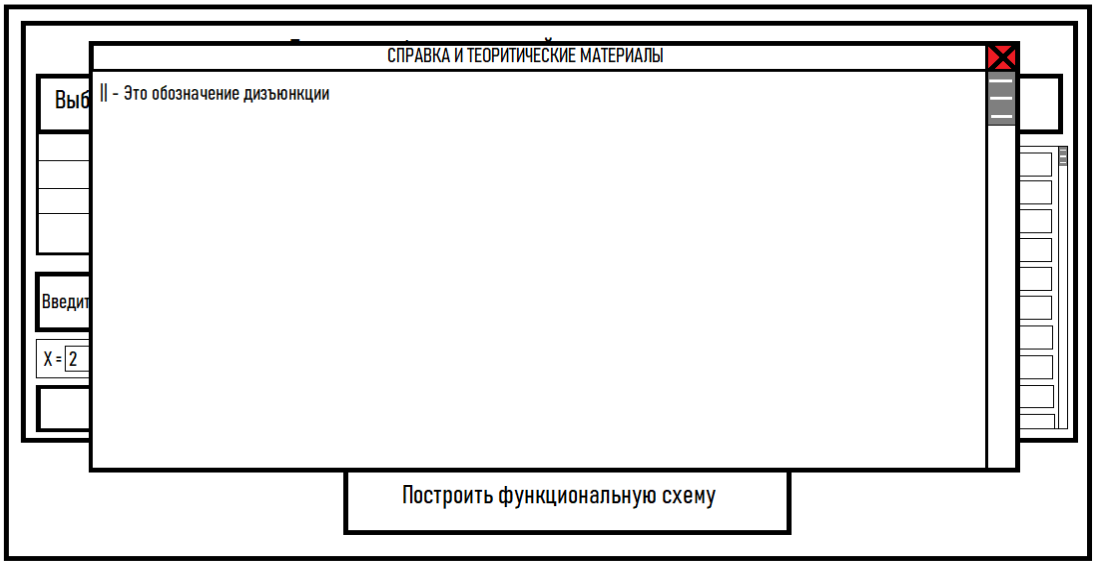

Рисунок 4 – Макет окна справки

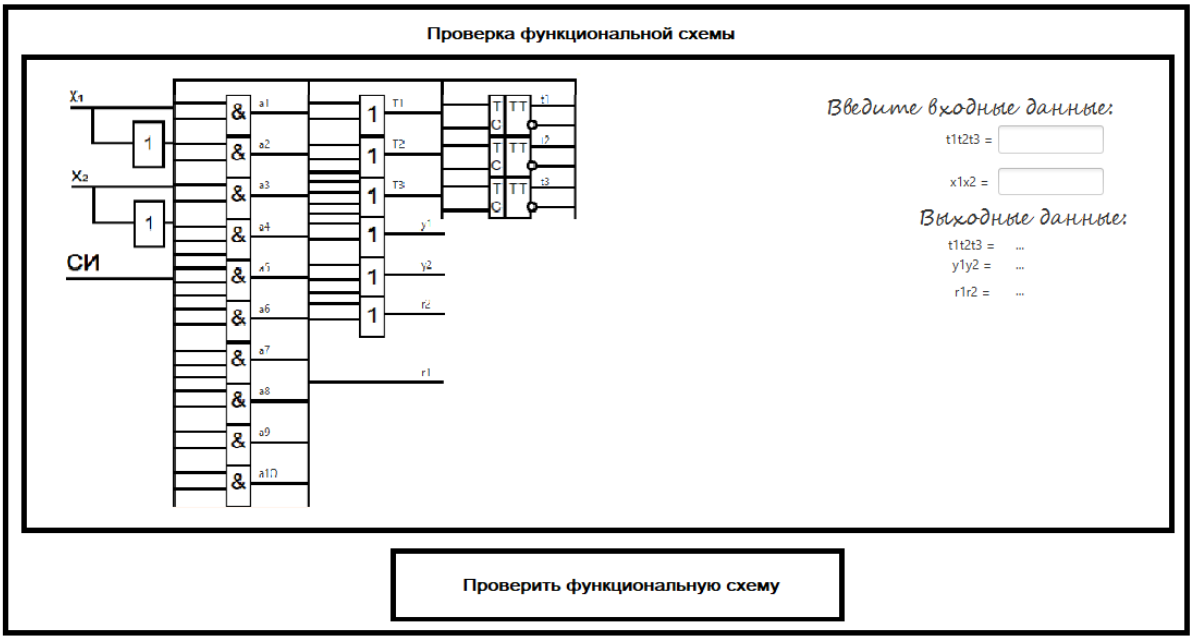

Рисунок 5 – Макет окна проверки функциональной схемы

Вывод по главе 3

Были выявлены функциональные и нефункциональные требования, сделаны макеты приложения и сформирована диаграмма прецедентов.

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Во время реализации проекта использовалась среда разработки IntelliJ IDEA 2023.1.3, язык программирования Java, фреймворк JavaFX.

4.1. Реализация интерфейсов

В ходе реализации были созданы следующие интерфейсы:

– интерфейс окна «Построение функциональной схемы»;

– интерфейс окна «Проверка функциональной схемы».

В интерфейсе окна «Построение функциональной схемы» реализовано:

– обработчики для кнопок сохранения входных данных в файл, открытие файла с входными данными, очистка всех списков в окне;

– обработчик кнопки «построить функциональную схему», при нажатии которой, открывается следующее окно проверки, на основе написанных минимизированных уравнений;

– поле ввода количества входных сигналов;

– раскрывающийся список, где пользователь выбирает триггер;

– 4 колонки, где пользователь заполняет минимизированные уравнения, которые он посчитал в своей курсовой работе.

Интерфейс окна «Построение функциональной схемы» предоставлен на рисунке 6.

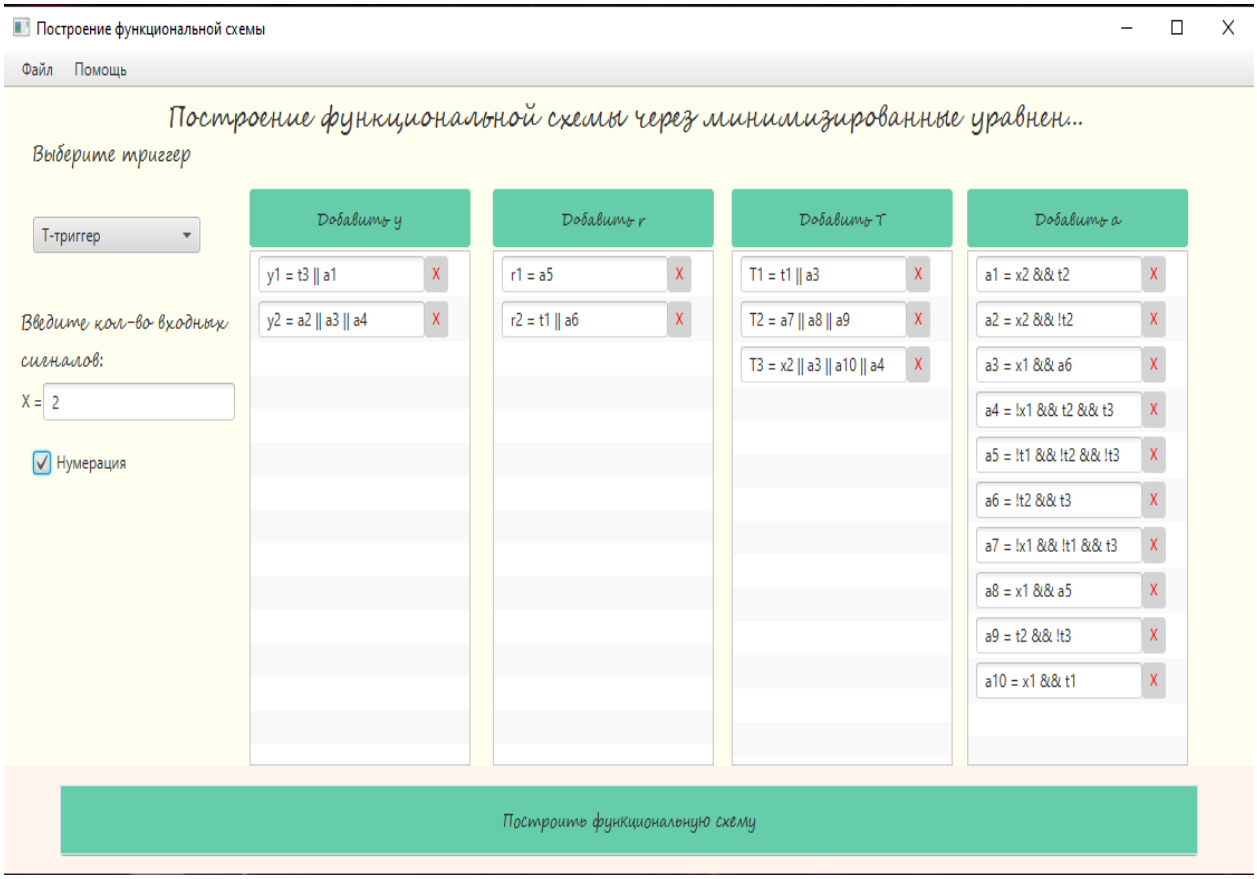

Рисунок 6 – Интерфейс окна «Построение функциональной схемы»

В интерфейсе окна «Проверка функциональной схемы»:

– два поля ввода данных, где пользователь вводит состояние автомата, и состояния входных сигналов;

– построенная функциональная схема;

– результаты проверки, а именно измененные состояния автомата, выходные сигналы 1 рода и выходные сигналы 2 рода.

Интерфейс окна «Проверка функциональной схемы» предоставлен на рисунке 7.

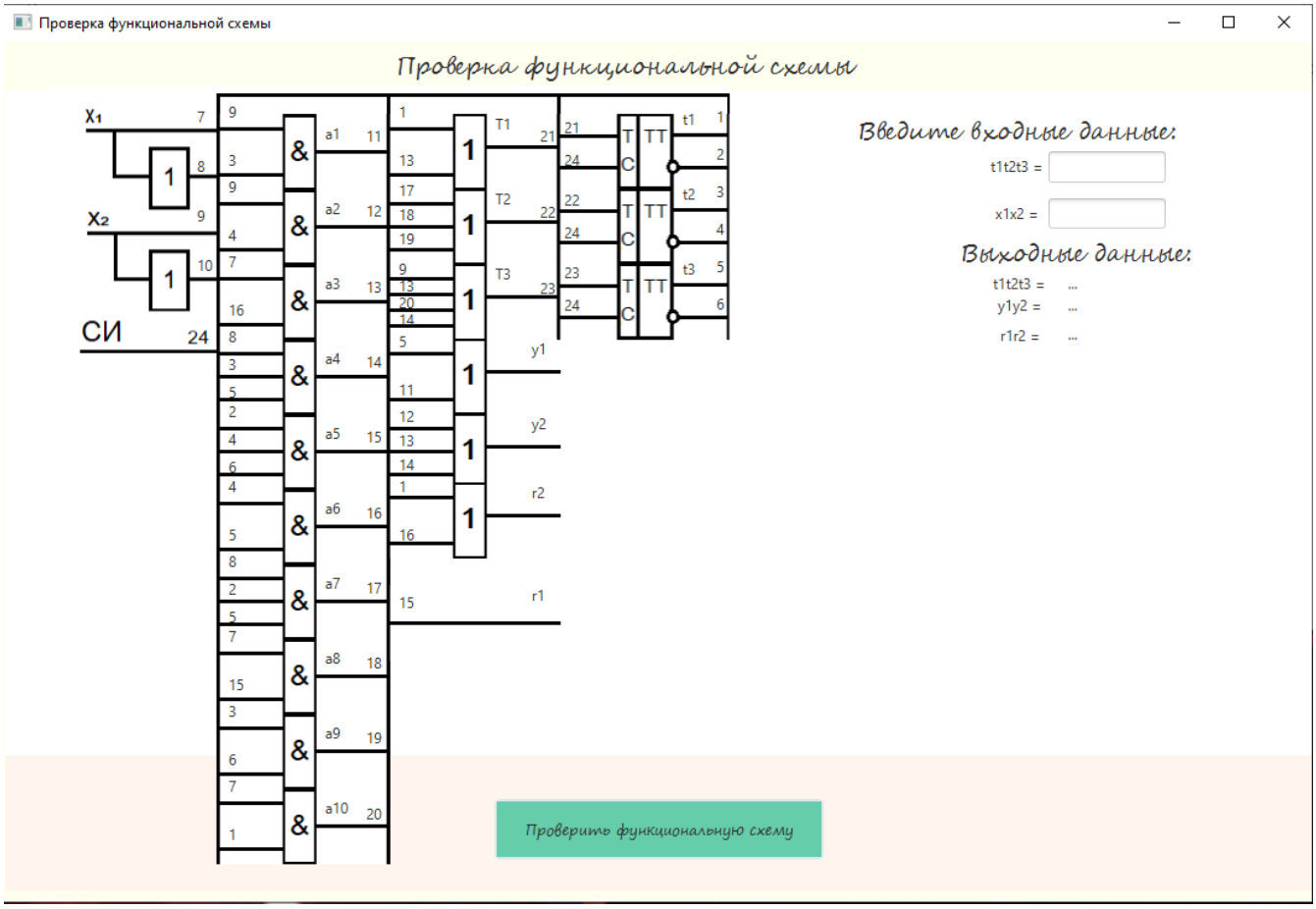

Рисунок 7 – Интерфейс окна «Проверка функциональной схемы»

4.2. Реализация классов

В процессе реализации приложения, было написано 11 классов на языке Java.

1. JarHelper — этот класс нужен для создания jar-файла.
2. Launcher — отвечает за запуск приложения и настройку окна «Построение функциональной схемы».
3. MainController — отвечает за работу с окном «Построение функциональной схемы», все события с кнопками описаны в этом методе.
4. MapSymbolCounter — считает количество элементов в минимизированных уравнениях, для того, чтобы на основе этих данных, приложение построило функциональную схему.

5. ResultComponent — отвечает за работу окна «Проверка функциональной схемы», все элементы и события, которые связаны с этим окном, написаны в этом классе.

6. TriggerStateHandler — в этом классе реализовано управление состояниями различных типов триггеров, а также обновление соответствующих меток в пользовательском интерфейсе.

7. FileUtil — отвечает за работу с файлами, обеспечивая сохранение данных из приложения в файл и загрузку данных из файла в приложение.

8. ErrorNotification — отвечает за обработку и отображение ошибок и уведомлений в пользовательском интерфейсе. В этом классе предоставлены методы для проверки допустимости введенных данных, отображение предупреждений и сообщений об ошибках.

9. CheckFunDiagram — отвечает за работу с логическими диаграммами и обновлением значений триггеров на основе входных данных. Этот класс включает в себя методы для обработки текстовых полей, обновление логических карт и вычисление логических значений.

10. FunctionalDiagramBuilder — этот класс отвечает за построение функциональной схемы, а именно добавляет те картинки, которые требуются, исходя из заполненных данных пользователем.

11. NumberingSignals — в этом классе реализовано заполнение карт сигналов, а именно присваиваются порядковые номера сигналам.

Диаграмма классов представлена на рисунке 8

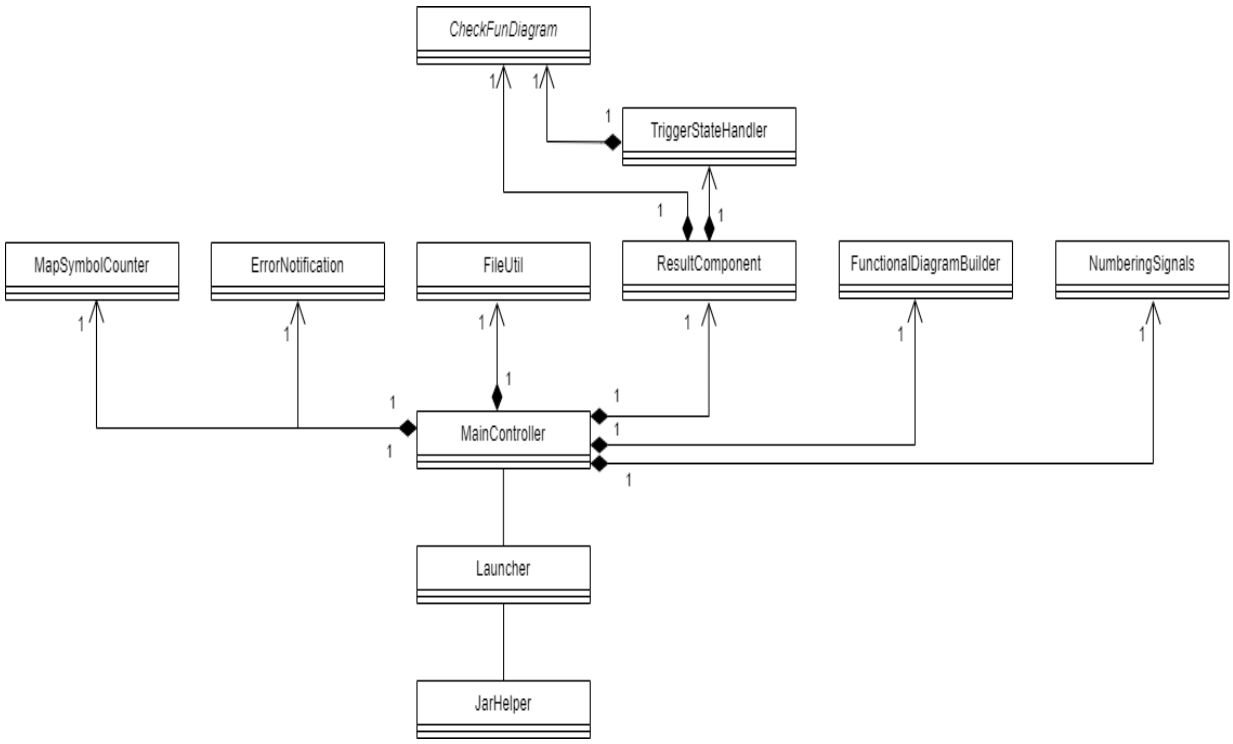

Рисунок 8 – Диаграмма классов

4.3. Описание работы программы

Приложение работает следующим образом. Пользователь выбирает тип триггера из раскрывающегося списка, это значение сохраняется в классе MainController в переменной SelectIndexTrigger, и используется при построении и проверки функциональной схемы. Затем пользователь либо вводит вручную минимизированные уравнения в соответствующие колонки, например, в колонку а «a1 = t1 && !t2», и указывает количество входных сигналов, либо вносит данные из текстового файла. На рисунке 9 показано, как выглядит файл с записанными минимизированными уравнениями. Эти данные записываются в методах addAClick, addTClick, addRSClick и так далее класса MainController и хранятся в картах, а именно в aMap, rMap, tMap, TMap, RSMap, JKMap и DMap. В эти карты значения записываются в формате ключ и значение. Например, ключ «a1», значение «t1 && !t2»

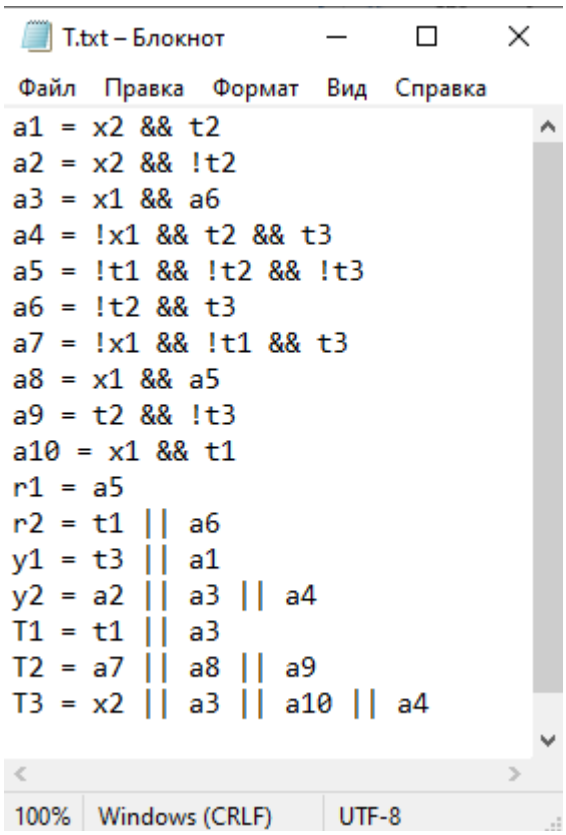

Рисунок 9 – Текстовый файл, в котором записаны минимизированные уравнения для Т-триггера

При нажатии на кнопку «Построить функциональную схему» класс MainController обращается к классу ErrorNotification для проверки корректности введённых данных. Как правильно записывать минимизированные уравнения, пользователь может посмотреть, нажав кнопку «Помощь». На рисунке 10 предоставлена информация из справки. Если обнаружены ошибки, пользователю показывается соответствующее сообщение, где именно пользователь неправильно ввел входные данные. Пример сообщения предоставлен на рисунке 11. В случае корректного ввода, программа в классе NumberingSignals в методе filltxMapsSignals и в методе fillATMapsSignals присваивает нумерацию сигналам по ключам из карт, записанных ранее в классе MainController, например для «t1» нумерация будет «1», а для «!t2» нумерация будет «2». Все эти значения хранятся в картах tMapSignals, xMapSignals, aMapSignals и TMapSignals.

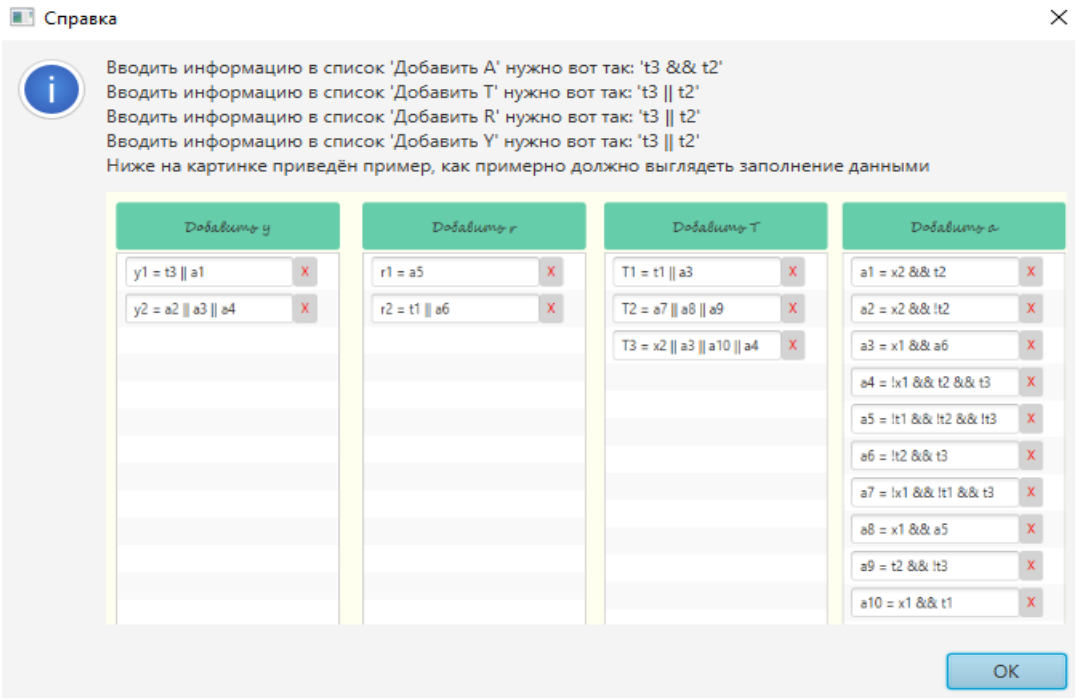

Рисунок 10 – Информация для пользователя из справки

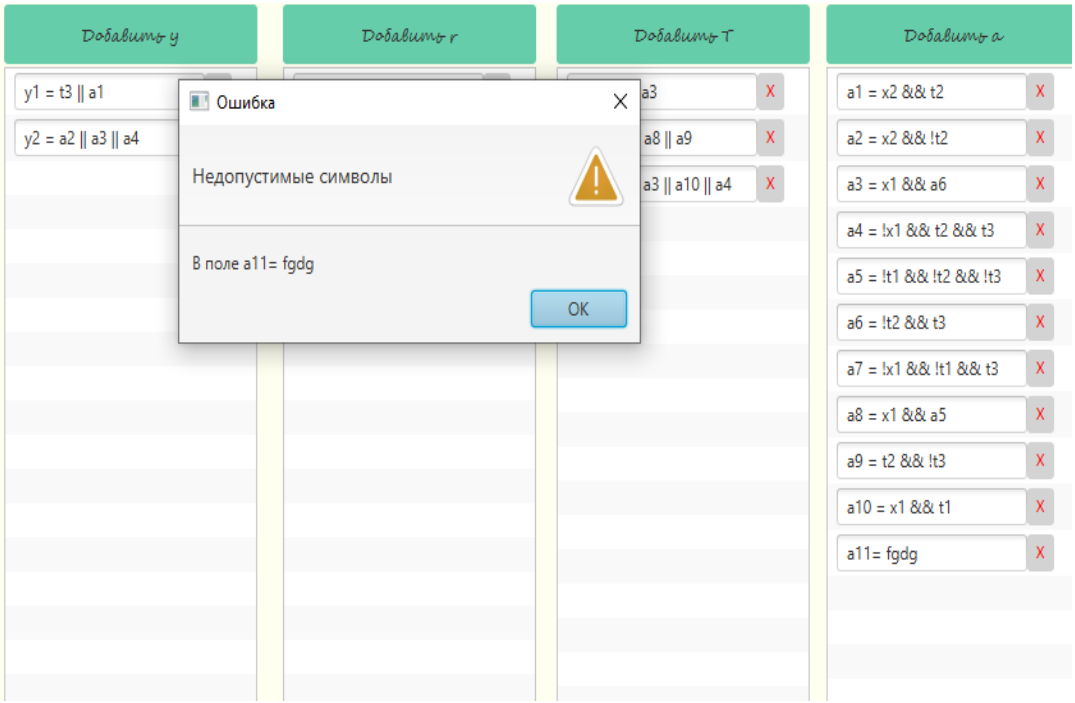

Рисунок 11 – Ошибка при неправильном заполнении списков

Дальше в классе MapSymbolCounter в методе SearchValue подсчитывает количество сигналов для каждого уравнения и записывает их в новые карты, например, для «a1 = t1 && !t2» записывается в aMapCounter, что для ключа a1 есть два сигнала. Также в этом методе записывается

нумерация для минимизированных уравнений в новые карты MapUpdate. Эти значения используются для построения функциональной схемы.

В проекте есть папка «assets», в которой хранятся различные рисунки, которые требуются для построения функциональной схемы. Пример рисунок находящихся в этой папке показаны на рисунках 12-13. Функциональная схема строится в классе FunctionalDiagramBuilder. На основе количества сигналов для каждого уравнения из карт MapCounter и количества входных сигналов схема строится, используя изображения из папки «assets». Например, если указано два входных сигнала, программа выбирает соответствующее изображение из папки и вставляет его в интерфейс пользователя.

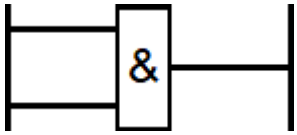

Рисунок 12 – Пример картинки конъюнктора из папки «assets» для двух сигналов

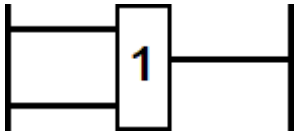

Рисунок 13 – Пример картинки дизъюнктора из папки «assets» для двух сигналов

После нажатия на кнопку «Построить функциональную схему» открывается окно «Проверка функциональной схемы» с построенной схемой, с нумерацией или без, в зависимости от выбора пользователя. Для проверки схемы используются классы:

- ResultComponent;
- CheckFunDiagram;
- TriggerStateHandler.

Класс ResultComponent управляет работой второго окна «Проверка функциональной схемы». После ввода данных для проверки и нажатия кнопки «Проверить функциональную схему» программа в классе CheckFunDiagram подсчитывает какие сигналы получатся для каждого ключа, то есть для конъюнкторов, если хоть один сигнал равен 0, то для ключа итоговое значение будет равно 0, для дизъюнкторов, если хоть один сигнал равен 1, то для ключа итоговое значение будет равно 1 и записывает их в отдельную карту MapValue. Например, «a1 = t1 && !t2», при условии, что t1 = 0, а !t2 = 1, в карту aMapValue будет записано «a1 = 0», поскольку все t1 = 0. Затем в классе TriggerStateHandler на основе выбранного типа триггера вычисляются конечные сигналы по таблицам состояний. Код проверки функциональной схемы представлен в приложении.

4.4. Функциональное тестирование

Для тестирования приложения использовалось функциональное тестирование. Функциональное тестирование — это процесс необходимый для проверки реализуемости функциональных требований, согласно спецификации тестируемого программного обеспечения. Функциональное тестирование проводится для оценки соответствия системы или компонента заданным функциональным требованиям[16].

Проведенное тестирование показано в таблице 4.

Таблица 4 - Функциональное тестирование

Название	Действия	Ожидаемый результат	Полученный результат	Итог
Проверка на работу с файлами. Сохранение и открытие файлов в формате .txt	Сохранить данные в файл и открыть файл	Результат работы с файлами правильно выполняется	Результат работы с файлами правильно выполнился	Пройден

Продолжение таблицы 4

Проверка на очищение всех данных с помощью кнопки	Очистить заполненные данные с помощью кнопки	Очиститься все заполненные данные	Очистились все заполненные данные	Пройден
Проверка на правильное заполнение данными и построение функциональной схемы	Ввести входные сигналы, выходные сигналы 1 рода, выходные сигналы 2 рода, состояние автомата, минимизированные уравнения и нажать кнопку «Построить функциональную схему»	Результат работы построения функциональной схемы и заполнение данными отобразиться корректно	Результат работы построения функциональной схемы и заполнение данными отобразился корректно	Пройден
Проверка на правильность проверки Т-триггера	Заполнить входные данные, выбрать Т-триггер, ввести значения для проверки функциональной схемы	Результат работы проверки функциональной схемы для Т-триггера сработает корректно	Результат работы проверки функциональной схемы для Т-триггера работает корректно	Пройден
Проверка на правильность проверки D-триггера	Заполнить входные данные, выбрать D-триггер, ввести значения для проверки функциональной схемы	Результат работы проверки функциональной схемы для D-триггера сработает корректно	Результат работы проверки функциональной схемы для D-триггера работает корректно	Пройден

Продолжение таблицы 4

Проверка на правильность проверки JK-триггера	Заполнить входные данные, выбрать JK-триггер, ввести значения для проверки функциональной схемы	Результат работы проверки функциональной схемы для JK-триггера сработает корректно	Результат работы проверки функциональной схемы для JK-триггера работает корректно	Пройден
Проверка на правильность проверки RS-триггера	Заполнить входные данные, выбрать RS-триггер, ввести значения для проверки функциональной схемы	Результат работы проверки функциональной схемы для RS-триггера сработает корректно	Результат работы проверки функциональной схемы для RS-триггера работает корректно	Пройден

Тестирование показывает, что приложение соответствует требуемому функционалу.

4.5 Тестирование на примере курсовой работы

За пример курсовой для теста был взят вариант 142 с Т-триггером. Таблица переходов представлена на рисунке 14.

Вариант 142

	u_3	u_2	u_1	u_1	u_2
	a_1	a_2	a_3	a_4	a_5
z_1	—	a_4 / w_3	a_1 / w_2	a_3 / w_4	a_1 / w_2
z_2	a_2 / w_1	—	a_2 / w_3	—	—
z_3	a_3 / w_2	a_5 / w_4	—	a_4 / w_3	a_2 / w_2

Т-триггер

Рисунок 14 – Пример таблицы переходов для теста разработанной программы

После кодирования входных–выходных сигналов была составлена закодированная таблица выходных сигналов, представленная на рисунке 15.

$r_1\ r_2$	1 0	0 1	0 0	0 0	0 1
$\tau_1\ \tau_2\ \tau_3$ / $x_1\ x_2$	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0
0 0	–	011/1 0	000/0 0	010/1 1	000/0 0
0 1	001/0 1	–	001/1 0	–	–
1 0	010/0 0	100/1 1	–	011/1 0	001/0 0
	$y_1\ y_2$	$y_1\ y_2$	$y_1\ y_2$	$y_1\ y_2$	$y_1\ y_2$

Рисунок 15 – Закодированная таблица выходных сигналов

После ручным способом были составлены минимизированные уравнения с помощью карт Карно и были записаны в программу, запись этих данных представлена на рисунке 16.

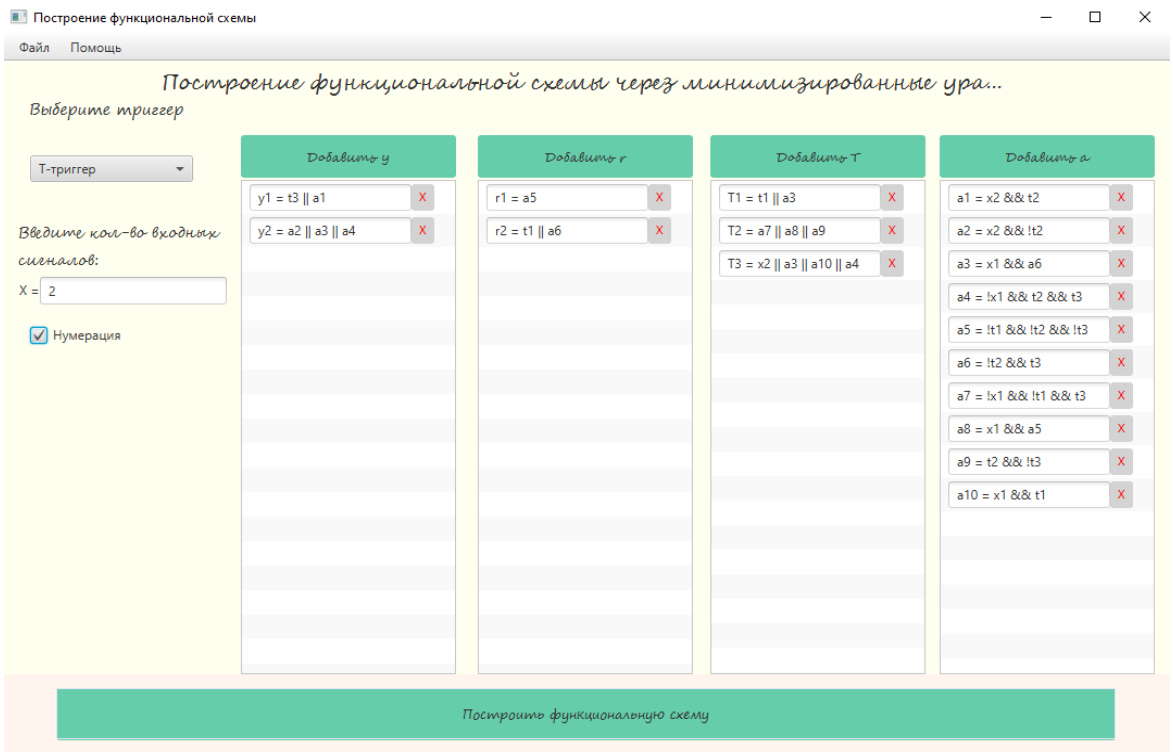

Рисунок 16 – Запись минимизированных уравнений

После прохождения проверки всех сигналов, было сравнение с закодированной таблицей выходных сигналов. Все результаты совпали, что означает, что разработанная программа правильно проверяет функциональную схему в курсовых проектах по теории автоматов. На рисунке 17 приведен один пример проверки сигналов.

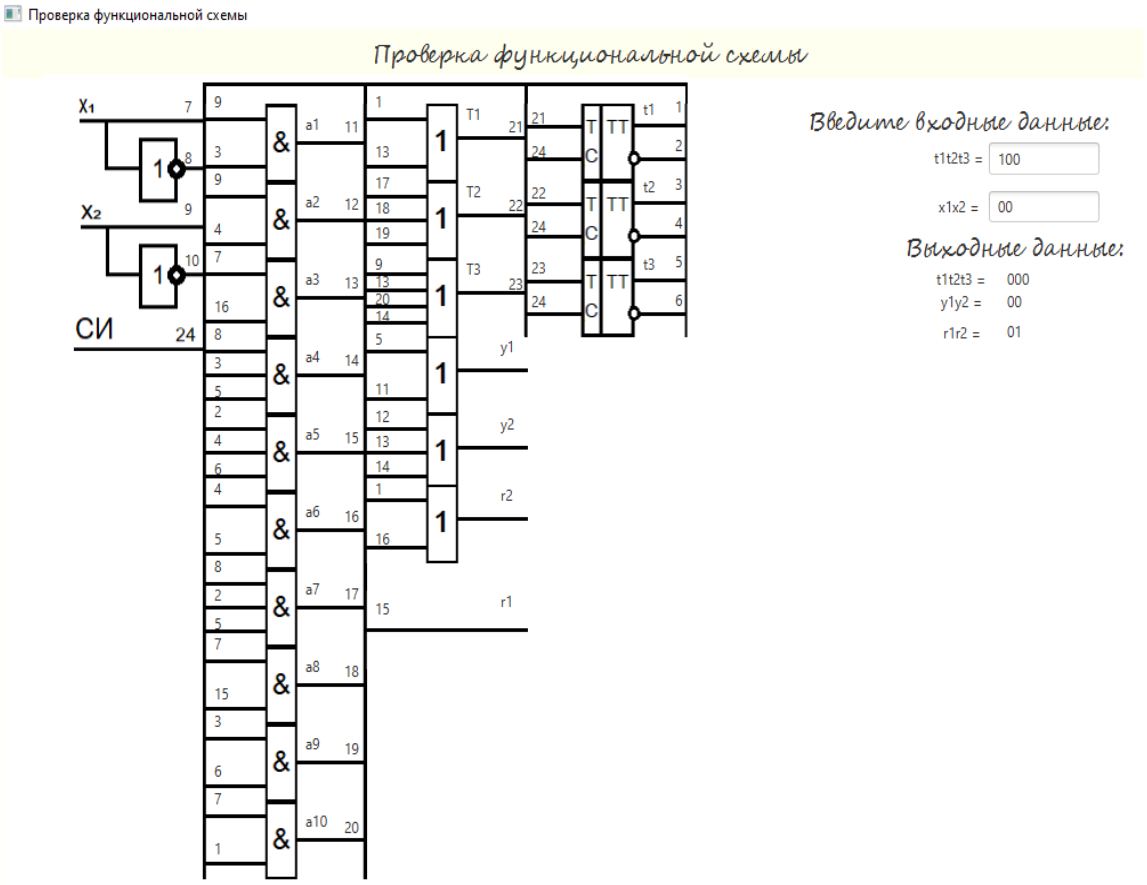

Рисунок 17 – Пример проверки функциональной схемы

Вывод по главе 4

Было разработано приложение, которое строит и проверяет функциональную схему в курсовых проектах по теории автоматов. Проведено функциональное тестирование и тестирование с примером из курсовой работы.

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы было разработано приложение для проверки функциональной схемы из курсового проекта по теории автоматов.

В ходе выполнения работы были сделаны следующие выводы, по выполненным задачам:

– проведен анализ и обзор аналогов разрабатываемого приложения. Было выявлено, что ни одно из предоставленных приложений не проверяет функциональную схему в курсовых проектах по теории автоматов;

– были выбраны средства разработки для реализации поставленной задачи. Язык программирования – Java, среда разработки – IntelliJ IDEA, фреймворк – JavaFX.

– были выявлены функциональные и нефункциональные требования, сделаны макеты приложения и сформирована диаграмма прецедентов;

– реализовано и протестировано разработанное приложение.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Учебное пособие / Л.И. Федосеева, Р.М. Адилов, М.Н. Шмокин. Основы теории конечных автоматов и формальных языков. Издательство Пенз. гос. технол. ун-та, 2013. – 136 с.

2. Мотвани Раджив, Хопкрофт Джон Э., Ульман Джеффри Д. Введение в теорию автоматов, языков и вычислений. Классическое издание: Перевод с английского О.И. Васылык, М. Саит-Аметова, канд.физ.-мат.наук А.Б. Ставровского / Издательский дом “Вильямс”, 2008. — 528 с.

3. Конструктор конечного автомата / [Электронный ресурс] // URL: https://madebyevan.com/fsm/ (дата обращения: 19.01.2024).

4. Статья Logisim на платформе Википедия / [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Logisim (дата обращения 19.01.2024).

5. Введение в программирование Java GUI с помощью Swing / [Электронный ресурс] // URL https://medium.com/@rolandmack63/introduction-to-java-gui-programming-with-swing-c9bedda86ee8 (Дата обращения 22.01.2024).

6. Статья о Swing на платформе Википедия/ [Электронный ресурс] // URL https://ru.wikipedia.org/wiki/Swing_(библиотека) (Дата обращения 22.01.2024).

7. Библиотека Swing / [Электронный ресурс] // URL https://java-online.ru/libs-swing.xhtml (Дата обращения 22.01.2024).

8. Библиотека SWT (Standart widget Toolkit) / [Электронный ресурс] // URL https://java-online.ru/libs-swt.xhtml (Дата обращения 22.01.2024).

9. Статья о JavaFX на платформе Википедия / [Электронный ресурс] // URL https://ru.wikipedia.org/wiki/JavaFXm (Дата обращения 23.01.2024).

10. Статья об Eclipse Java / [Электронный ресурс] // URL https://javarush.com/groups/posts/2359-obzor-eclipse-java-sreda-razrabotki-pod-sebja (Дата обращения 01.02.2024).

11. Статья об преимуществах и недостатках Eclipse IDE / [Электронный ресурс] // URL https://proglib.io/p/ide-eclipse-za-i-protiv-ot-vedushchih-programmistov-2019-11-02 (Дата обращения 01.02.2024).

12. Статья о NetBeans IDE / [Электронный ресурс] // URL https://workspace.ru/tools/environment/net-beans/ (Дата обращения 01.02.2024).

13. Статья о NetBeans IDE / [Электронный ресурс] // URL https://blog.skillfactory.ru/glossary/netbeans/ (Дата обращения 01.02.2024).

14. Статья об IntelliJ IDEA / [Электронный ресурс] // URL https://blog.skillfactory.ru/glossary/intellij-idea/ (Дата обращения 01.02.2024).

15. Статья об IntelliJ IDEA / [Электронный ресурс] // URL https://skillbox.ru/media/code/intellij-idea-chto-eto-za-sreda-razrabotki-i-kak-v-ney-rabotat/#stk-5 (Дата обращения 01.02.2024).

16. Статья о функциональном тестировании / [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Функциональное_тестирование (дата обращения: 12.05.2024).

ПРИЛОЖЕНИЕ

Код работы алгоритмов за логику проверки функциональной схемы.

Листинг 1 – CheckFunDiagram

```
package com.example.demo4;

import javafx.scene.control.TextField;

import java.util.*;

import static com.example.demo4.MapSymbolCounter.*;

public class CheckFunDiagram {

    static Boolean[] ArrayX = new Boolean[MapSymbolCounter.CountInput]; // Массив для
хранения значений входных сигналов

    static  Boolean[] ArrayT; // Массив для хранения значений состояний автомата

//Переменные для хранения значений в каждом уравнении

    static Map<String, Boolean> aMapValue = new TreeMap<>(new Comparator<String>() {

        @Override

        public int compare(String o1, String o2) {

            int index1 = Integer.parseInt(o1.substring(1));

            int index2 = Integer.parseInt(o2.substring(1));

            return Integer.compare(index1, index2);

        }

    });

    static Map<String, Boolean> TMapValue = new HashMap<>();

    static Map <String,Boolean> yMapValue = new HashMap<>();

    static Map <String,Boolean> rMapValue = new HashMap<>();

    static Map <String,Boolean> DMapValue = new HashMap<>();

    static Map <String,Boolean> JKMapValue = new HashMap<>();

    static Map <String,Boolean> RSMapValue = new LinkedHashMap<>();

//метод для заполнения размерности массива ArrayT

    public static void setArrayTsize(int selectedTriggerIndex){

        if (selectedTriggerIndex == 0){

            ArrayT = new Boolean[TMap.size()];

        }else if (selectedTriggerIndex == 1) {

            ArrayT = new Boolean[RSMap.size()/2];

        }else if (selectedTriggerIndex == 2) {

            ArrayT = new Boolean[JKMap.size()/2];

        }else if (selectedTriggerIndex == 3) {

            ArrayT = new Boolean[DMap.size()];

        }
```

```
}
//Метод для заполнения значений массива в интерфейс пользователя
    public static void  setValueToArray(TextField textField,Boolean[] Array) {
        String value = textField.getText().trim();
        int Index = 0;
        for (int i = 0; i < value.length() && Index < Array.length; i++) {
            char ch = value.charAt(i);
            if (ch == '0') {
                Array[Index] = false;
                Index++;
            } else if (ch == '1') {
                Array[Index] = true;
                Index++;
            } else {
                System.out.println("Некорректное значение для записи в массив Array: " +
ch);
            }
        }
        System.out.println("Array: " + Arrays.toString(Array));
    }
    private static int extractIndex(String str) {
        String indexStr = str.replaceAll("[^0-9]", "");
        if (!indexStr.isEmpty()) {
            return Integer.parseInt(indexStr) - 1;
        }
        return -1;
    }
//Метод который считает каждое значение для каждого ключа минимизированных уравнений в
aMap
    public static void UpdateAMap(Boolean[] arrayX, Boolean[] arrayT) {
        aMapValue.clear();
        boolean updatedOnce; // Флаг, чтобы отслеживать, было ли хотя бы одно обновление
значения
        do {
            updatedOnce = false;
            for (Map.Entry<String, String> entry : aMap.entrySet()) {
                String key = entry.getKey();
                String value = entry.getValue()
```

```
                if (aMapValue.containsKey(key)) {
                    continue;
                }
                // Разделяем строку по символу "&&"
                String[] parts = value.split("\\&\\&");
                // Обрабатываем каждую часть
                List<Boolean> listA = new ArrayList<>();
                boolean allDependenciesResolved = true;
                for (String part : parts) {
                    part = part.trim();
                    int index = extractIndex(part);
                    int indexA = extractIndex(part) + 1;
                    boolean negate = false;
                    if (part.startsWith("!")) {
                        negate = true;
                        part = part.substring(1);
                    }
                    if (part.contains("x")) {
                        if (index >= 0 && index < arrayX.length) {
                            listA.add(negate ? !arrayX[index] : arrayX[index]);
                        }
                    } else if (part.contains("t")) {
                        if (index >= 0 && index < arrayT.length) {
                            listA.add(negate ? !arrayT[index] : arrayT[index]);
                        }
                    } else if (part.contains("a")) {
                        String aMapKey = "a" + indexA;
                        if (!aMapValue.containsKey(aMapKey)) {
                            System.out.println("Пропускаем значение для ключа " + key +
". Требуется вычислить сначала " + aMapKey);
                            allDependenciesResolved = false;
                            break;
                        }
                        listA.add(negate ? !aMapValue.get(aMapKey) :
aMapValue.get(aMapKey));
                    }
                }
                if (allDependenciesResolved) {
```

```
                    Boolean result = multiplyListValues(listA);
                    aMapValue.put(key, result);
                    updatedOnce = true; // Устанавливаем флаг обновления значения
                }
            }
        } while (updatedOnce);
        // Выводим полученный aMapValue
        System.out.println("aMapValue: " + aMapValue);
    }
    private static Boolean multiplyListValues(List<Boolean> list) {
        Boolean result = true;
        for (Boolean value : list) {
            result &= value;
        }
        return result;
    }
//Метод который считает каждое значение для каждого ключа минимизированных уравнений для
всех остальных карт
    public static void UpdateMap(Boolean[] arrayX, Boolean[] arrayT, Map<String, Boolean>
MapValue, Map<String, String> Map) {
        MapValue.clear();
        for (Map.Entry<String, String> entry : Map.entrySet()) {
            String key = entry.getKey();
            String entryValue = String.valueOf(entry.getValue());
            entryValue = entryValue.trim();
            if (entryValue.length() == 1 && entryValue.matches("[01]")){
                MapValue.put(key, Integer.parseInt(entryValue) == 1);
                continue;
            }
            // Разделяем строку по символу "||"
            String[] parts = entryValue.split("\\|\\|");
            // Обрабатываем каждую часть
            List<Boolean> listT = new ArrayList<>();
            for (String part : parts) {
                // Удаляем лишние пробелы
                part = part.trim();
                int index = extractIndex(part);
```

```
                boolean negate = false; // Переменная для хранения флага отрицания
                if (part.startsWith("!")) {
                    negate = true; // Если перед символом стоит '!', устанавливаем флаг
отрицания
                    part = part.substring(1); // Удаляем '!' из строки
                }
                // Проверяем наличие символов 'x', 't' или 'a'
                if (part.contains("x")) {
                    // Получаем числовое значение после 'x'
                    if (index >= 0 && index < arrayX.length) {
                        if(negate) {
                            listT.add(!arrayX[index]);
                        }
                        else {
                            listT.add(arrayX[index]);
                        }
                    }
                } else if (part.contains("t")) {
                    // Получаем числовое значение после 't'
                    if (index >= 0 && index < arrayT.length) {
                        if(negate) {
                            listT.add(!arrayT[index]);
                        }
                        else {
                            listT.add(arrayT[index]);
                        }
                    }
                } else if (part.contains("a")) {
                    // Получаем числовое значение после 'a'
                    String aMapKey = "a" + (index + 1) ;
                    // Проверяем, есть ли такой ключ в aMapValue
                    if (!aMapValue.containsKey(aMapKey)) {
                        System.out.println("Пропускаем значение для ключа " + key + ".
Требуется вычислить сначала " + aMapKey);
                        continue;
                    }
                    if(negate) {
                        listT.add(!aMapValue.get(aMapKey));
                    }
```

Продолжение листинга

```
                    else {
                        listT.add(aMapValue.get(aMapKey));
                    }
                    // Получаем значение из aMapValue и добавляем в listA
                    listT.add(aMapValue.get(aMapKey));
                }
            }
            // Если хотя бы одно значение в listT равно true, устанавливаем результат в
true
            Boolean result = false;
            for (Boolean value : listT) {
                if (value) {
                    result = true;
                    break;
                }
            }
            MapValue.put(key, result);
        }
        System.out.println("MapValue: " + MapValue);
    }
}
```

Окончание листинга

Листинг 2 – TriggerStateHandler

```
package com.example.demo4;
import javafx.application.Platform;
import javafx.scene.control.Label;
import static com.example.demo4.MainController.selectedTriggerIndex;
public class TriggerStateHandler {
    private CheckFunDiagram checkFunDiagram = new CheckFunDiagram();
    private Label SolutionT; // Поле для отображения результата по T
    private Label SolutionR; // Поле для отображения результата по R
    private Label SolutionLabel; // Поле для отображения общего результата
    public TriggerStateHandler(Label SolutionT, Label SolutionR, Label SolutionLabel) {
        this.SolutionT = SolutionT;
        this.SolutionR = SolutionR;
        this.SolutionLabel = SolutionLabel;
```

```
    }
// Статический метод для формирования текста и установки его в label
   public static void printLabel(int count, String letter, Label label){
    StringBuilder labelBuilder = new StringBuilder();
    for (int i = 1; i <= count; i++) {
        labelBuilder.append(letter).append(i);
    }
    labelBuilder.append(" =");
    label.setText(labelBuilder.toString());
}
// Метод для обработки изменения состояния в зависимости от выбранного индекса
   void SolutionTrigger() {
        StringBuilder newText = new StringBuilder();
        StringBuilder newTextT = new StringBuilder();
        StringBuilder newTextR = new StringBuilder();
        if (selectedTriggerIndex == 0) {
// Логика для T-триггера
            for (int i = 0; i < checkFunDiagram.TMapValue.size(); i++) {
                Boolean value = checkFunDiagram.TMapValue.get("T" + (i + 1));
                if (value != null && i < checkFunDiagram.ArrayT.length) {
                    if (!value) {
                        newTextT.append(checkFunDiagram.ArrayT[i] ? "1" : "0");
                    } else {
                        checkFunDiagram.ArrayT[i] = !checkFunDiagram.ArrayT[i];
                        newTextT.append(checkFunDiagram.ArrayT[i] ? "1" : "0");
                    }
                }
            }
            Platform.runLater(() -> {
              SolutionT.setText(newTextT.toString());
            });
        } else if (selectedTriggerIndex == 1) {
// Логика для RS-триггера
            for (int i = 0; i < checkFunDiagram.RSMapValue.size(); i++) {
                Boolean R = checkFunDiagram.RSMapValue.get("R" + (i + 1));
                Boolean S = checkFunDiagram.RSMapValue.get("S" + (i + 1));
                if (R !=null && S !=null && i<checkFunDiagram.ArrayT.length) {
```

```
                    if (S && !R) {
                        checkFunDiagram.ArrayT[i] = true;
                        newTextT.append("1");
                    } else if(!S && R ) {
                        checkFunDiagram.ArrayT[i] = false;
                     newTextT.append("0");
                    }else if (!S && !R){
                        newTextT.append(checkFunDiagram.ArrayT[i] ? "1": "0");
                    } else {
                        ErrorNotification.showError("R и S не могут быть равны 1
одновременно!");
                    }
                }
            }
            Platform.runLater(() -> {
               SolutionT.setText(newTextT.toString());
            });
        } else if (selectedTriggerIndex == 2) {
// Логика для JK-триггера
            for (int i = 0; i < checkFunDiagram.JKMapValue.size(); i++) {
                Boolean J = checkFunDiagram.JKMapValue.get("J" + (i + 1));
                Boolean K = checkFunDiagram.JKMapValue.get("K" + (i + 1));
                if (J !=null && K !=null && i<checkFunDiagram.ArrayT.length) {
                    if (J && !K) {
                        checkFunDiagram.ArrayT[i] = true;
                        newTextT.append("1");
                    } else if(!J && K ) {
                        checkFunDiagram.ArrayT[i] = false;
                        newTextT.append("0");
                    }else if (J && K){
                        checkFunDiagram.ArrayT[i] = !checkFunDiagram.ArrayT[i];
                        newTextT.append(checkFunDiagram.ArrayT[i] ? "1": "0");
                    } else {
                        newTextT.append(checkFunDiagram.ArrayT[i] ? "1": "0");
                    }}}
            Platform.runLater(() -> {
              SolutionT.setText(newTextT.toString());
```

Продолжение листинга

```
            });
        } else if (selectedTriggerIndex == 3) {
// Логика для D-триггера
            for (int i = 0; i < checkFunDiagram.DMapValue.size(); i++) {
                Boolean value = checkFunDiagram.DMapValue.get("D" + (i + 1));
             if (value !=null && i<checkFunDiagram.ArrayT.length) {
                    if (!value) {
                        newTextT.append("0");
                    } else {
                        newTextT.append("1");
                        checkFunDiagram.ArrayT[i] = !checkFunDiagram.ArrayT[i];
                    }
                }
            }
            Platform.runLater(() -> {
            SolutionT.setText(newTextT.toString());
            });
        }
// Обработка yMapValue для формирования newText (для SolutionLabel)
        for (int i = 0; i < checkFunDiagram.yMapValue.size(); i++) {
            Boolean value = checkFunDiagram.yMapValue.get("y" + (i + 1));
            if (value != null) {
                newText.append(value ? "1" : "0");
            }
        }
// Обработка rMapValue для формирования newTextR (для SolutionR)
        for (int i = 0; i < checkFunDiagram.rMapValue.size(); i++) {
            Boolean value = checkFunDiagram.rMapValue.get("r" + (i + 1));
            if (value != null) {
                newTextR.append(value ? "1" : "0");
            }}
        Platform.runLater(() -> {
         SolutionLabel.setText(newText.toString());
         SolutionR.setText(newTextR.toString());
        });}}
```

Окончание листинга