Национальный исследовательский университет «Московский институт электронной техники»

Направление: Технические дисциплины

Выпускная квалификационная работа

**Разработка программного модуля визуализации конфигурирования микросхем**

**Работу выполнил:**

Зотов Владислав Дмитриевич

Студент 4 курса

Национальный исследовательский университет «Московский институт электронной техники»

**Научный руководитель:**

Федотова Елена Леонидовна

Д.п.н., доцент

Национальный исследовательский университет «Московский институт электронной техники»

Москва, 2021

Содержание

[Список используемых сокращений 4](#_Toc73918343)

[ВВЕДЕНИЕ 5](#_Toc73918344)

[1. ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ 7](#_Toc73918345)

[1.1 Анализ предметной области 7](#_Toc73918346)

[1.2 Актуальность изучаемой проблемы 16](#_Toc73918347)

[1.3 Текущее решение 16](#_Toc73918348)

[1.4 Аналогичные решения 17](#_Toc73918349)

[1.5 Сравнение решений 19](#_Toc73918350)

[1.6 Постановка разработки 21](#_Toc73918351)

[1.7 Концептуальная модель предметной области 21](#_Toc73918352)

[Вывод по исследовательскому разделу 22](#_Toc73918353)

[2. КОНСТРУКТОРСКИЙ РАЗДЕЛ 23](#_Toc73918354)

[2.1 Выбор программных средств для разработки ПМ ВКМ 23](#_Toc73918355)

[2.2 Архитектура ПМ ВКМ 30](#_Toc73918356)

[2.3 Проектирование ПМ ВКМ 33](#_Toc73918357)

[2.4 Разработка пользовательского интерфейса 44](#_Toc73918358)

[Вывод по конструкторскому разделу 45](#_Toc73918359)

[3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 46](#_Toc73918360)

[3.1 Отладка ПМ ВКМ 46](#_Toc73918361)

[3.2 Журналирование ПМ ВКМ 51](#_Toc73918362)

[3.3 Анализ методов и средств тестирования 53](#_Toc73918363)

[3.4 Тестирование ПМ ВКМ 56](#_Toc73918364)

[Вывод по технологическому разделу 62](#_Toc73918365)

[ЗАКЛЮЧЕНИЕ 63](#_Toc73918366)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 64](#_Toc73918367)

Приложение 1 - Руководство оператора

Приложение 2 - Техническое задание (ТЗ)

# Список используемых сокращений

ИМС – интегральная микросхема

ОС – операционная система

ПМ – программный модуль

ПМ ВКМ – программный модуль визуализации конфигурирования микросхем

ПО – программное обеспечение

ПС – программное средство

СП – среда программирования

ЭВМ – электронно-вычислительная машина

ЯП – язык программирования

IDE (Integrated Development Environment) – система программных средств, используемая программистами для разработки программного обеспечения

JSON (JavaScript Object Notation) – формат обмена текстовыми данными, основанный на JavaScript

JVM (Java Virtual Machine) – виртуальная машина языка программирования Java

# ВВЕДЕНИЕ

Современный мир с каждым днем все сложнее и сложнее представить без технических устройств, они заполоняют нашу жизнь практически во всех сферах нашей жизни. Каждое такое устройство разрабатывается под решение определенных задач.

В их составе имеется одна или более микросхема, каждая из которых состоит из большого числа различных элементов, которыми она и должна управлять, чтобы решить свою задачу.

Существует огромное множество различных микросхем, и все они нуждаются хотя бы в первоначальном конфигурировании. Микросхемы производит не одно предприятие, поэтому существует большое число отличающихся друг от друга типов микросхем, процесс конфигурирования которых может очень сильно различаться.

Процесс конфигурирования любой микросхемы – всегда нелегкий и высокозатратный процесс, поэтому изготовители микросхем ищут способы для повышения эффективности конфигурирования своей микросхемы.

На предприятии НПК "Технологический центр" разработана микросхема. Сразу же возник вопрос, каким образом пользователь будет осуществлять процесс ее конфигурирования. Длинная цепочка действия для осуществления этого процесса не устраивала компанию, так как эффективность конфигурирования при этом падала. Принято решение создать новую программную оболочку для конфигурирования этой микросхемы.

Тема выпускной квалификационной работы «Разработка программного модуля визуализации конфигурирования микросхем». Программному модулю присваивается шифр «ПМ ВКМ».

Выполнение ВКР проходило на предприятии НПК "Технологический центр".

Актуальность разработки:

Разрабатываемый ПМ ВКМ имеет большую значимость для решения задачи конфигурирования микросхемы, разработанной на предприятии.

Цель разработки:

Уменьшение времени процесса конфигурирования микросхем.

Задачи разработки:

* исследование предметной области;
* сравнение существующих аналогов;
* выбор языка и среды программирования;
* разработка схемы данных ПМ ВКМ;
* разработка схемы алгоритма ПМ ВКМ;
* разработка программы ПМ ВКМ;
* отладка и тестирование ПМ ВКМ;
* разработка руководства оператора.

Исследовательская работа состоит из введения, исследовательского, конструкторского и технологического разделов, заключения, списка литературы и приложений.

В исследовательском разделе изучается предметная область задачи, проводится обзор и сравнение текущего и аналогичных решений, а также делается вывод по данному разделу.

В конструкторском разделе проводится сравнение и выбор языка программирования и среды разработки, разрабатывается схема данных, схема алгоритма, функциональная схема ПМ ВКМ, рассматривается применение паттернов проектирования для разработки ПМ ВКМ, а также делается вывод по данному разделу.

В технологическом разделе различными методами производится отладка и тестирование ПМ ВКМ.

Приложение 1 включает в себя руководство оператора.

Приложение 2 включает в себя техническое задание (ТЗ).

# 1. ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

## 1.1 Анализ предметной области

Анализ предметной области конфигурирования микросхемы нужно начать с того, чтобы понять, что такое микросхема, из чего она состоит и каким образом можно провести процесс ее конфигурирования.

Микросхема, или чип/интегральная схема (ИМС) – это электронное устройство, состоящее из большого количества электрических компонентов и кристалла, соединенных по схеме, спроектированной изготовителем, а также произведенное на полупроводниковой пластине и помещенное в корпус. [22]

В современном мире существует огромное число технических устройств, содержащих в своем составе микросхемы. Все они выполняют различные функции, и нужны для решения определенных задач, возложенных на них. Но тем не менее, все микросхемы можно каким-либо образом сгруппировать по общим признакам.

Классификации ИМС

ИМС классифицируются по количеству компонентов на следующие виды:

* малая интегральная схема (МИС) — до 100 электрических компонентов, расположенных на схеме;
* средняя интегральная схема (СИС) — до 1000 электрических компонентов, расположенных на схеме;
* большая интегральная схема (БИС) — до 10 тыс. электрических компонентов, расположенных на схеме;
* сверхбольшая интегральная схема (СБИС) — более 10 тыс. электрических компонентов, расположенных на схеме.

По технологии изготовления все множество микросхем можно поделить на четыре группы [24]:

* пленочные;
* гибридные;
* полупроводниковые;
* совмещенные.

В пленочных ИМС все компоненты представляют собой пленки, которые нанесены на диэлектрическое основание (пассивную подложку). В зависимости от толщины пленок различают тонкопленочные (толщина пленок не превышает 1мкм) и толстопленочные (толщина пленок находится в пределах от 15 до 45 мкм) ИМС [23]. Так как на данный момент еще не получилось создать активные элементы (например, транзисторы) в виде пленок, то пленочные ИМС могут содержать в своем составе только пассивные компоненты (например, резисторы, конденсаторы, катушки индуктивности и тому подобное), на рис. 1.1 можно видеть, как пассивные элементы располагаются на пленочной ИМС.

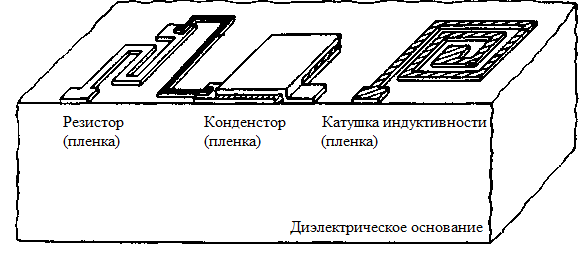


Рис. 1.1 Расположение пассивных элементов в пленочной ИМС

В следствие такого скудного состава в пленочных ИМС функции, которые они могут выполнять, крайне ограничены.

В гибридных ИМС на диэлектрической подложке располагается комбинация из пассивных элементов в виде пленок и активных компонентов, которые называются навесными, что подчеркивает их обособленность от пленочных компонентов. Это позволяет увеличить количество функция, для которых может использоваться микросхема такого типа. На рис. 1.2 можно видеть, как пассивные и активные элементы располагаются на пленочной ИМС.



Рис. 1.2 Расположение элементов в гибридной ИМС

Так же, как и пленочные ИМС, гибридные делятся на тонкопленочные (толщина слоев находится в пределах от 0.02 до 10 мкм) и толстопленочные (толщина слоев превышает 15 мкм) микросхемы [23].

Но, несмотря на расширенное количество выполняемых функций по сравнению с пленочными, гибридные, так же, как и пленочные ИМС не составляют основу современной микроэлектроники [24].

Наконец, перейдем к самым популярным на данный момент микросхемам, а именно к полупроводниковым ИМС.

В полупроводниковых ИМС все компоненты и их соединения сделаны в одном полупроводниковом кристалле в виде неразъёмных связанных соединений p-n переходов, поэтому располагаются не только на поверхности, но и в приповерхностном слое. На рис. 1.3 можно видеть, как элементы располагаются на полупроводниковой ИМС.

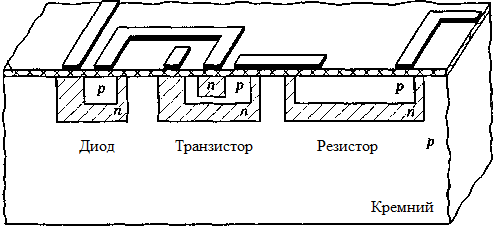


Рис. 1.3 Расположение элементов в полупроводниковой ИМС

Такая подложка в полупроводниковых микросхемах называется активной подложкой. Практически всегда ее делают из полупроводников, таких как кремний или арсенид галлия, но в ряде случаев используют и диэлектрический материал.

На данный момент существует несколько видов полупроводниковых ИМС, а именно:

* биполярные (ТТЛ, ТТЛШ);
* униполярные (КМОП).

Развитие биполярных полупроводниковых ИМС прошло через несколько серий логики в следующей последовательности: РТЛ (резисторно-транзисторная логика), РЕТЛ (резистивно-емкостная транзисторная логика), ДТЛ (диодно-транзисторная логика), ТТЛ (транзисторно-транзисторная логика), ЭСЛ (эмиттерно-связанная логика), ТТЛШ (транзисторно-транзисторная логика с диодами Шотки), И2Л (интегральная инжекционная логика).

Униполярные микросхемы – это ИМС, которые строятся на основе комплементарных МОП транзисторов с n- и p- каналами, называемые также КМОП-микросхемы.

Наибольшее распространение в современном мире получили биполярные полупроводниковые ИМС на основе ТТЛ и ТТЛШ и униполярные полупроводниковые ИМС на КМОП.

Существуют также и совмещенные ИМС, у которых активные компоненты располагаются в приповерхностном слое, так, как в полупроводниковых ИМС, а пассивные компоненты нанесены на поверхность в виде пленок, так, как в пленочных ИМС. На рис. 1.4 можно видеть, как элементы располагаются на совмещенной ИМС.

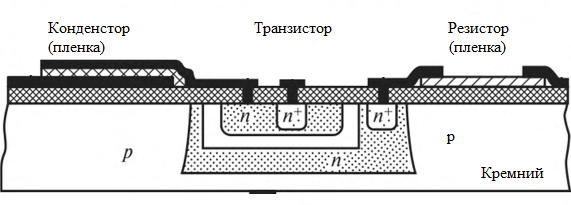


Рис. 1.4 Расположение элементов в совмещенной ИМС

Практически все современные ИМС выполняются по полупроводниковой технологии [23]. Микросхемы других типов используют только в некоторых специализированных случаях.

Изучив все типы ИМС по технологии изготовления, получаем на рис. 1.4 их классификацию по этому признаку.



Рис. 1.4 Классификация интегральных схем по технологии изготовления

По конструктивному оформлению различают следующие виды ИМС:

* безкорпусные;
* корпусные.

Безкорпусные ИМС – ИМС, предназначенные для непосредственного монтажа на печатную плату, покрытые специальным защитным лаком или компаундом, которые помогают снизить влияние на ИМС негативных факторов окружающей среды.

Для еще большего уменьшения таких факторов придумали помещать ИМС в различные корпуса, такие ИМС называются корпусными – это ИМС, помещенные в специальный пластиковый или металлический защитный корпус, имеющий специальные контактные разъемы (пины), которые позволяют производить монтаж при помощи пайки.

И, наконец, две самые главные разновидности микросхем – по функциональному назначению, то есть по виду обрабатываемого сигнала.

ИМС по этому признаку подразделяются на следующие виды [21]:

* аналоговые;
* цифровые.

Аналоговые ИМС работают с аналоговыми сигналами: усиливают, обрабатывают и преобразуют электрические сигналы. Аналоговая схемотехника имеет малое потребление энергии, максимальное быстродействие, но обладает малой стабильностью параметров системы [25]. Аналоговые ИМС делятся на две группы, универсального назначения – к ним относятся, например, матрицы согласованных диодов или транзисторов, а также операционный усилитель, и специализированного назначения – к ним относятся устройства, выполняющие фильтрацию, перемножение сигналов, например, генератор сигналов определенной формы, сумматор и тому подобное. Используются они в измерительных устройствах, радиолокации, аппаратуре связи и телевизионной аппаратуре. Аналоговые ИМС могут применять в сочетании с цифровыми микросхемами.

Цифровые ИМС работают с цифровыми сигналами, состоящими из нулей и/или единиц.

Самые распространенные цифровые микросхемы, так же, как и в полупроводниковых ИМС – произведенные по технологии ТТЛ (транзисторно-транзисторная логика) и по технологии КМОП (комплиментарный металл-оксид-полупроводник) [20].

Помимо физических отличий, существуют отличия и в уровнях нуля и единицы.

У микросхем технологии ТТЛ уровень нуля равен 0,4В, а уровень единицы 2,4В. У микросхем технологии КМОП уровень нуля почти равен нулю, а уровень единицы — равен почти напряжению питания микросхемы. Нулевое напряжение у микросхемы КМОП получается путем подключения соответствующего вывода к общему проводу, а напряжение высокого уровня — подключением к шине питания.

Поэтому, при проектировке схемы электронного устройства нужно использовать микросхемы одного типа логики, иначе может возникнуть несоответствие в верхнем (уровень единицы) и нижнем (уровень нуля) уровнях цифровых сигналов.

По мере развития цифровых микросхем их быстродействие становилось все больше и больше, и на данный момент достигло впечатляющих результатов, наиболее быстрые обладают скоростью переключения порядка 3.5 нс (серия 74ALS) [25].

Тем не менее, за такое быстродействие приходится платить, а именно, такие микросхемы потребляют значительный ток, а также приходится использовать много микросхем, что влияет на стоимость и габариты микросхемы не в лучшую сторону [25].

Первую проблему решают микросхемы на основе технологии КМОП, но в этом случае потребляемый ток зависит от скорости переключения логических вентилей в микросхеме. Вторую проблему решают двумя способам.

Разрабатываются специализированные БИС (ASIC), они уменьшают габариты устройства, но проблема со стоимостью будет решена только при крупносерийном производстве, такие ИМС используются, например, в сотовых телефонах, телевизорах и подобных технических устройствах.

Также разрабатываются программируемые логические схемы, называемые ПЛИС. В настоящее время это направление очень активно развивается, они очень удобны для производителей и потребителей, так как такие микросхемы можно каждый раз переконфигурировать (программировать) под решения все новой и новой задачи без физического вмешательства в микросхему, то есть изменения ее компонентов, состава и междуэлементными связями внутри ИМС.

Выделяют несколько видов ПЛИС [26][27]:

* постоянные запоминающие устройства (ПЗУ, ROM);
* программируемые логические матрицы (ПЛМ, PLA);
* сложно программируемая логическая ИМС (CPLD – complex PLD);
* программируемая пользователем вентильная матрица (ППВМ, FPGA).

Постоянные запоминающие устройства (ПЗУ, ROM) нужны для хранения информации, которая не изменяется в процессе работы с самим техническим устройством. Поэтому такие устройства для запоминания информации можно построить на мультиплексорах. Они строятся на основе матрицы, в каждом узле которой находится ячейка памяти, представляющая из себя МОП транзистор, если в его плавающем затворе нет заряда, то эта ячейка представляет из себя цифру 0, но если в нем содержится заряд, то эта ячейка представляет из себя цифру 1. Стирание производится ультрафиолетом или электрическим стиранием.

Основой программируемых логических матриц (ПЛМ, PLA) является реализация логической функции, представленной в совершенной дизъюнктивной нормальной форме (СДНФ), по технической реализации программирования контакта различают прожигаемые и перепрограммируемые ПЛИС в зависимости от технической реализации конфигурирования (программирования) контакта. Такой вид ПЛИС к настоящему времени устарел.

В сложных цифровых схемах используются ПЛИС, которые являются сложными программируемыми логическими устройствами (CPLD – complex PLD). В своем составе имеют строенную энергозависимую память, хранящая в себе свою прошивку, которая работает даже при отключении питания и имеют небольшое количество элементов, из-за чего дешевле и проще в применении, так как легко и быстро запускаются [28]. Такие ИМС используются, например, как контроллер VGA, USB и тому подобное [26].

Программируемые пользователем вентильные матрицы (ППВМ, FPGA) содержит в себе больше логических элементов и более гибкую архитектуру, чем CPLD. Широко используются для обработки сигналов. Сконфигурированная для данного типа ПЛИС программа не сохраняется при исчезании питания микросхемы, если она хранится в распределенной памяти на основе ОЗУ (оперативном запоминающем устройстве) и сохраняется, если она хранится в памяти на основе flash-памяти или перемычек antifuse [29]. Каждый логический блок имеет небольшое количество входов и выходов, что позволяет лучше использовать ресурсы микросхемы, что сказывается на быстродействии процесса решения задачи в микросхеме.

Самые актуальные виды ПЛИС на сегодня это CPLD и FPGA.

На рис. 1.5 можно увидеть полную классификацию ИМС по всем признакам.



Рис. 1.5 Классификация интегральных схем

После изучения классификации микросхем, нужно понять, как происходит процесс конфигурирования.

Конфигурирование микросхемы

Любая микросхема работает по строго заложенной в нее программе, для этого ее особым способом конфигурируют (программируют).

В аналоговых микросхемах такая “программа” является совокупностью расположения электрических элементов микросхемы и их физических межкомпонентных соединений, то есть производитель самостоятельно при разработке своей микросхемы создает такую “программу” внутри нее, которая будет выполнять определенную задачу, а именно усиление, обрабатывание или преобразование электрических сигналов, подающихся ей на вход [23]. Можно понять, что если нужно будет реализовать другой алгоритм действий в аналоговой микросхеме, то ее придется переделывать с нуля, создавая новую микросхему, для решения уже новой задачи.

Проведя классифицирование ИМС, можно заметить, что такую проблему решают цифровые программируемые логические схемы (ПЛИС). Такие микросхемы стоят дороже аналоговых, но зато позволяют менять свою конфигурацию для решения новой задачи.

Наибольшее распространение на сегодняшний день получили ПЛИС вида CPLD и FPGA. Они имеют на своем борту как и энергонезависимую в CPLD, что позволяет хранить программу в микросхеме даже после отключения ее питания, так и сочетание ее с энергозависимой памятью в FPGA, что помимо преимущества, как у CPLD позволяет выполнять программу быстрее за счет ее расположения в оперативной памяти, которая работает только при включенном питании [28][29]. Также такие ИМС не лишены возможности изменения программы при новом конфигурировании.

Чтобы провести процесс конфигурирования ИМС ПЛИС, нужно последовательно пройти через следующие фазы:

* создание схемы конфигурации;
* верификация созданной схемы конфигурации;
* трансляция схемы конфигурации в битовый поток;
* загрузка битового потока в ИМС.

Существует несколько схем конфигурирования [31]:

* конфигурационная микросхема – конфигурирование при помощи определенных типов микросхем;
* пассивная последовательная (PS – Passive Serial) – конфигурирования при помощи микропроцессора с последовательным синхронным интерфейсом, или конфигурационного устройства MasterBlaster или MasterBlasterMV;
* синхронная пассивная параллельная (PPS – Passive Parallel Synchronous) – конфигурирование при помощи микропроцессора с последовательным синхронным интерфейсом;
* асинхронная пассивная параллельная (PPA – Passive Parallel Asynchronous) – конфигурирование при помощи микропроцессора с последовательным синхронным интерфейсом, микропроцессор работает с конфигурируемой микросхемой как с внешней памятью;
* конфигурирование по JTAG (JTAG – Joint Test Action Group) – конфигурирование при помощи интерфейса JTAG.

## 1.2 Актуальность изучаемой проблемы

На предприятии НПК "Технологический центр" была разработана микросхема и поставлена следующая задача: разработать оболочку, в которой конечный пользователь мог бы сам создавать схему конфигурирования этой микросхемы с помощью кода, либо графической схемой, после чего совершать процесс ее конфигурирования. Поэтому, сразу же возник вопрос, каким образом пользователь будет осуществлять процесс ее конфигурирования. Таким образом, решение изучаемой проблемы имеет большую значимость для решения задачи конфигурирования микросхемы, разработанной на предприятии.

## 1.3 Текущее решение

На предприятии изначально использовалось бесплатное программное решение под названием Verilator.

Эта программа работает только в ОС Linux, чтобы установить данную программу, необходимо установить с помощью командной строки, дополнительные библиотеки а именно: perl, python3, g++, libfl2, libfl-dev, zlibc, zlib1g, zlib1g-dev, после чего правильно их сконфигурировать для корректной работы программы Verilator, после чего можно было приступить к работе в ней [3]. Verilator принимает код, написанный на языке описания аппаратуры – verilog (verilog HDL), используемый для моделирования электронных схем, выполняет его проверку, после чего компилируется в код на языке программирования C++. Его же нужно скомпилировать соответствующим компилятором C++, а именно g++, и наконец, совершить процесс конфигурирования микросхемы с помощью программатора. Также, нужно отметить, что такое конфигурирование происходит без пользовательского интерфейса, а также в трех разных программах – verilator, g++, программатор.

Так как конечный пользователь должен видеть свою схему в виде графической схемы и совершать свою работу в пределах одного приложения, было принято решение, найти аналогичные решения и проанализировать их на предмет решения данной проблемы.

## 1.4 Аналогичные решения

Для начала, были найдены и рассмотрены следующие решения, аналогичные текущему решению на предприятии:

* GHDL;
* САПР Quartus II;
* САПР "КОВЧЕГ";
* ModelSim;
* ISim.

Программное решение GHDL

Первое решение – GHDL, является бесплатным продуктом, так как имеет открытый исходный код, а значит им может пользоваться любой желающий.

Установка GHDL, так же, как и в текущем решении Verilator, происходит с помощью командной строки, так же позволяет компилировать код на языке описания аппаратуры Verilog и так же не имеет графического интерфейса для создания и изменения схем конфигурирования микросхем, а значит не подходит для решения поставленной задачи, но тем не менее, данное решение имеет и достоинства по сравнению с текущим решение, так как, в отличие от него является кроссплатформенным, так как работает в ОС Windows, Linux и Apple OS X [32].

Программное решение САПР Quartus II

Следующее решение, которое было рассмотрено – САПР Quartus II.

Данное программное решение позволяет создавать схемы конфигурирования не только с помощью кода на языке описания архитектуры Verilog, как представлено на рис.1.6, но и с помощью элементов на поле в графической оболочке, как представлено на рис.1.7.

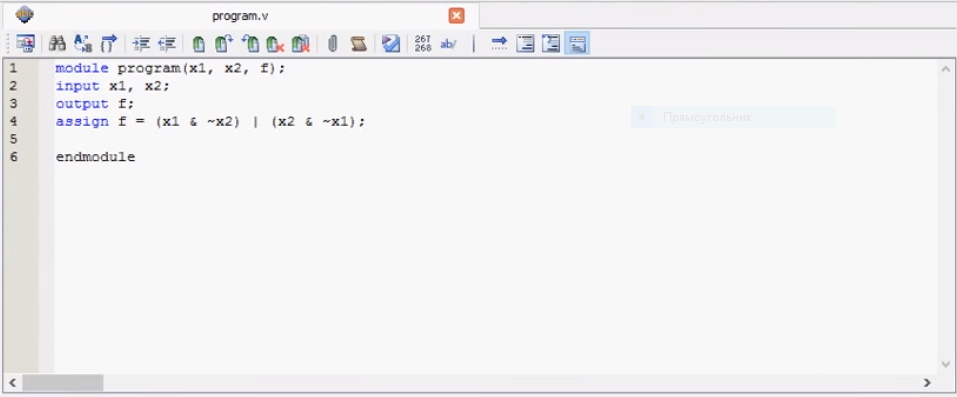


Рис. 1.6 Создание элемента схема на Verilog

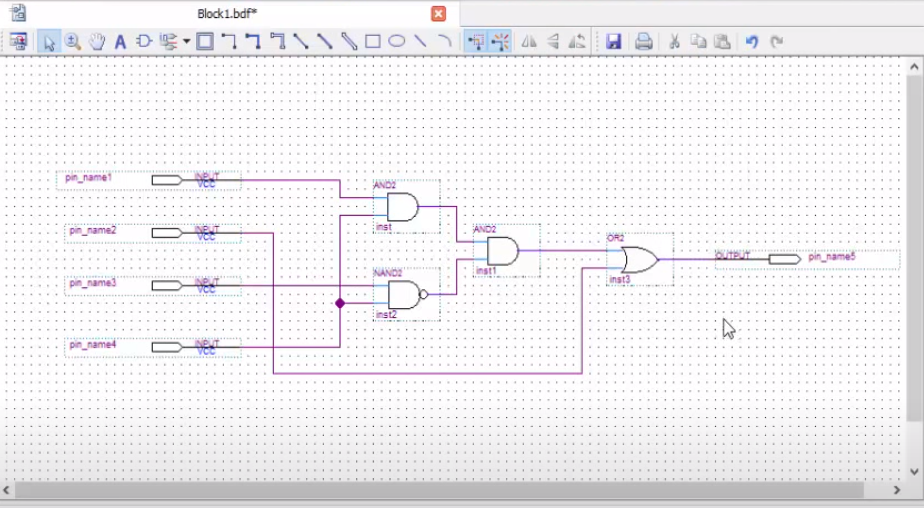


Рис. 1.7 Создание схемы конфигурирования на поле в графической оболочке

САПР Quartus II загружается на устройство пользователя при помощи своего установщика, а также имеет три вида версий, которые работают в ОС Windows и Linux: две платных (PRO, STANDARD) и одну бесплатную (LITE), платные версии обеспечены полным функционалом с мизерными отличиями и, в отличии от бесплатной версии, позволяют проводить процесс конфигурирования по схеме, созданной пользователем [4][5].

Программное решение САПР "КОВЧЕГ"

Программное решение САПР "КОВЧЕГ" было разработано на предприятии НПК "Технологический центр" для разработки КМОП ИМС на основе базовых матричных кристаллов.

Данное решение функционирует в ОС Windows и позволяет, так же, как и в предыдущем решении создавать схемы конфигурирования при помощи элементов на поле его графической оболочки и при помощи кода на Verilog [2].

Программное решение ModelSim

Программное обеспечение ModelSim поддерживает моделирование поведения и моделирования на уровне логических элементов схемы, включая стенды на VHDL или Verilog для всех устройств Intel® FPGA [33].

Работает в ОС Windows и Linux и имеет свой установщик.

Имеет две версии: платную (Pro), которая в отличие от бесплатной (Standard), позволяет проводить моделирование [33].

Программное решение ISim

ISim – cимулятор ISE, предоставляет полный, полнофункциональный симулятор HDL, включая Verilog. Загружается на устройство пользователя при помощи своего установщика, работает в ОС Windows и Linux [35].

Позволяет создавать схемы для моделирования HDL с помощью элементов на поле и кодом, но не для конфигурирования ИМС [34].

## 1.5 Сравнение решений

После проведения обзора текущего и аналогичных ему решений, стало понятно, что ни одно из них не удовлетворяет всем требованиям, которые находятся в табл. 1.1, для решения поставленной задачи.

Было принято решение создавать новый программный модуль.

Табл. 1.1

Сравнение программных решений.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Существующие  решения  Требования | Verilator (текущее решение) [3] | GHDL [32] | САПР Quartus II (LITE) [4][5] | САПР "КОВЧЕГ" [2] | ModelSim (Standard) [33] | ISim [34][35] |
| Кроссплатформен-ность | Нет | Да | Да | Нет | Да | Да |
| Создание схем конфигурирования | Да | Да | Да | Да | Нет | Нет |
| Установщик программного решения | Нет | Нет | Да | Да | Да | Да |
| Графическая оболочка для создания схемы конфигурирования | Нет | Нет | Да | Да | Нет | Нет |
| Процесс конфигурирования микросхемы | Нет | Нет | Нет | Нет | Нет | Нет |
| Изменение характеристик каждого элемента схемы | Да | Да | Да | Да | Да | Да |
| Подключение библиотек элементов для построения схемы | Да | Да | Да | Да | Да | Да |
| Стоимость продукта | Бесплатно | Бесплатно | Бесплатно | Бесплатно | Бесплатно | Платно |

## 1.6 Постановка разработки

Была выбрана тема «Разработка программного модуля визуализации конфигурирования микросхем». Программному модулю присваивается шифр «ПМ ВКМ».

Цель разработки ПМ ВКМ:

Уменьшение времени процесса конфигурирования микросхем.

Задачи разработки ПМ ВКМ:

* исследование предметной области;
* сравнение существующих аналогов;
* выбор языка и среды программирования;
* разработка схемы данных ПМ ВКМ;
* разработка схемы алгоритма ПМ ВКМ;
* разработка программы ПМ ВКМ;
* отладка и тестирование ПМ ВКМ;
* разработка руководства оператора.

## 1.7 Концептуальная модель предметной области

Концептуальная модель предметной области состоит в проектировании единого ПМ, которое позволило бы пользователю создавать схемы конфигурирования из доступных функциональных компонентов библиотек элементов, а также запускать процесс программирования микросхемы, разработанной на предприятии.

Предполагаемый алгоритм работы программы:

ПМ при старте считывает все элементы во всех заранее созданных библиотеках и создает из них список компонентов. Затем пользователь сам конфигурирует из них свою схему на предназначенном для этого поле, настраивая каждый нужный ему элемент и создавая межкомпонентные соединения между ними. После чего он запускает процесс конфигурирования микросхемы, который собирает всю созданную пользователем схему в единый файл, который транслируется в код, понятный для микросхемы, а после чего и в саму микросхему для завершения процесса ее программирования.

Таким образом, входными данными ПМ будут являться файлы библиотек элементов, хранящиеся в памяти пользовательского устройства.

ПМ в процессе своей работы будет работать с данными в формате JSON.

Файл, содержащий в себе код конечной схемы конфигурирования, будет являться выходными данными ПМ.

## Вывод по исследовательскому разделу

В исследовательском разделе изучена и проанализирована предметная область, рассмотрена актуальность изучаемой проблемы, затем проведен обзор и сравнение текущего и аналогичных программных решений.

После чего принято решение проектировать и разрабатывать новое программное решение, а именно ПМ ВКМ, а также описана концептуальная модель его предметной области.

# 2. КОНСТРУКТОРСКИЙ РАЗДЕЛ

Разработку программного модуля стоит начать с обзора и анализа существующих программных средств, а также выбора наиболее подходящих из них для решения поставленной задачи.

## 2.1 Выбор программных средств для разработки ПМ ВКМ

Выбор языка программирования для разработки ПМ ВКМ

Для начала было проведено сравнение языков программирования (ЯП), результаты которого можно видеть в табл. 2.1.

Табл. 2.1

Сравнение языков программирования для разработки ПМ ВКМ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ЯП  Критерий | C | C++ [9] | C# [10] | Python [11] | Ruby [36] | Java [8] |
| Кроссплатформенность | + | + | + | + | + | + |
| Поддержка ООП | - | + | + | + | + | + |
| Возможность подключения графического фреймворка | + | + | + | + | + | + |
| Наличие библиотек для работы с JSON данными | - | + | + | + | + | + |
| Опыт работы с ЯП | + | + | + | - | - | + |
| Используемый язык на предприятии НПК "Технологический центр" | - | + | - | - | - | + |
| Наличие встроенного монитора ресурсов | + | + | + | - | - | + |
| Возможность работы с файлами | + | + | + | + | + | + |
| Наличие документации | + | + | + | + | + | + |
| Обработка исключений | - | + | + | + | + | + |

Продолжение табл. 2.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Автоматическое удаление неиспользуемых объектов | - | - | - | + | + | + |

Язык программирования C был разработан в 1972 году Деннисом Ритчи, является процедурным компилируемым и статически типизированным ЯП, поддерживает процедурную парадигму программирования.

Был разработан для разработки ОС UNIX, но в дальнейшем стал поддерживаться и другими ОС, что делает этот ЯП кроссплатформенным. Конструкции этого ЯП близко сопоставляются машинным инструкциям, поэтому его активно используют в проектах, которым свойственен язык ассемблер. За счет этого, код компилируется, собирается и выполняется с очень высокой скоростью по сравнению с другими языками. Это, в свою очередь является и недостатком, так как за это приходится платить синтаксической и структурной бедностью языка, из-за чего приходится писать большой объем кода для реализации простой программы. Так же язык позволяет напрямую обращаться к памяти устройства без каких-либо средств автоматического управления памятью, что чревато ошибками и большим временем их обнаружения в случае их возникновения, так как в этом случае программа просто перестанет работать и не будет показано, где она возникла. Имеет на своем борту скудный графический режим, встроенный монитор ресурсов, а также позволяет работать с файлами.

C18 (2018 год) – самый последний на данный момент стандарт этого ЯП, в нем содержатся только исправления последнего самого значительного стандарта, а именно C11 (2011 год). ЯП C оказал влияние на развитие других языков программирования, таких как C++, C#, Java и Objective-C.

C++ появился в 1983 году благодаря Бьёрну Страуструпу, является компилируемым, статически типизированным языком программирования общего назначения.

В отличие от C поддерживает сразу несколько парадигм программирования, а именно процедурное, объектно-ориентированное и обобщенное программирование. Широко используется для разработки программного обеспечения (ПО). Сочетает в себе свойства низкоуровневых и высокоуровневых языков. Синтаксис унаследован от языка C, поэтому совместим, но не включает его в себя, поэтому содержит в себе все его достоинства. Отличается высокой производительностью, множеством разнообразных конструкций и библиотек (например, графических фреймворков), а также кроссплатформенностью, а также имеет документацию. [9]

Используется на предприятии НПК "Технологический центр" для программирования микросхем.

На данный момент стандарты языка разрабатываются международной организацией по стандартизации (ISO). Новые стандарты языка появляются раз в три года, последний был опубликован в декабре 2020 году под названием C++20.

C# (C Sharp) был разработан компанией Microsoft в 1998-2001 годах под руководством Андерса Хейлсберга и Скотта Витльтамота для разработки приложений на платформе Microsoft .NET Framework.

Является мультипарадигмальным, поддерживает объектно-ориентированную, обобщённую, процедурную, функциональную, событийную, рефлективную парадигмы программирования. Является C-подобным языком, имеет статическую типизацию. Изначально разрабатывался для ОС Windows, но благодаря .NET Core может использоваться в качестве кроссплатформенного инструмента. Имеет большое множество библиотек для повышения функционала. [10]

Последняя спецификация языка на данный момент была разработана ISO/IEC совместно с ECMA в декабре 2017 года в версии C# 5.0, при этом последняя версия на данный момент – C# 9.0 вышла в сентябре 2020 года.

Python появился 20 февраля 1991 года благодаря компании Python Software Foundation и программисту Гвидо ван Россуму. Является высокоуровневым ЯП общего назначения с динамической строгой типизацией с автоматическим управлением памятью.

Python является мультипарадигмальным ЯП: поддерживает императивное, процедурное, структурное, объектно-ориентированное, мета- и функциональное программирование. Имеет большое множество различных библиотек, позволяющих расширить функционал. Портирован почти на всех известных платформах – кроссплатформенный, а также имеет циклический сборщик мусора для менеджмента памяти. [11]

Разработка ЯП ведется через предложения по расширению языка. На текущий момент поддерживает версия Python 3.

Ruby – динамический, рефлективный, интерпретируемый, высокоуровневый ЯП, появился благодаря разработчику Юкихиро Мацумото в 1995 году.

Поддерживает процедурный, объектно-ориентированный, функциональный стили программирования. Все чаще используется в веб-разработке. Позволяет работать с файлами, создавать приложения с использованием графического фреймворка, имеет на своем борту сборщик мусора и является кроссплатформенным ЯП. [36]

Последний рабочий релиз Ruby 3.0.1 вышел 5 апреля 2021.

Язык программирования Java был разработан компанией Sun Microsystems в 1995 году, но в 2009 году владельцем стала компания Oracle, которая и является владельцем на данный момент.

Java – строго типизированный объектно-ориентированный язык программирования общего назначения. Написанные приложения транслируются в байт-код, после чего происходит его выполнения с помощью виртуальной Java-машины (JVM), благодаря чему это позволяет производить JIT-компиляцию, а также запускать такие приложения везде, где есть JVM. является кроссплатформенным ЯП. Важной особенностью языка также является безопасность работы с памятью и информативный отклик о работе программы. Имеет большое количество библиотек, позволяющих расширить функционал. [8]

Используется на предприятии НПК "Технологический центр".

Существует большое количество версия языка, которые постоянно улучшаются и дорабатываются. Java Version 8 Update 281 является последней и актулальной на данный момент.

В итоге сравнительного анализа был выбран язык программирования Java, как наиболее подходящий по требованиям предприятия НПК "Технологический центр".

Выбор среды программирования для разработки ПМ ВКМ

После выбора ЯП, необходимо выбрать среду программирования (СП) на языке Java 8 для разработки ПМ ВКМ.

Табл. 2.2

Сравнение сред программирования для разработки ПМ ВКМ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СП  Критерий | BlueJ [37] | IntelliJ IDEA (Ultimate) [13] | NetBeans [14] | Xcode [15] | Eclipse [12] |
| Поддержка ЯП Java | + | + | + | + | + |
| Возможность подключения фреймворков | - | + | + | + | + |
| Поддержка системы контроля версий Git | - | + | + | + | + |
| Поддержка работы в ОС Linux Ubuntu | + | + | + | - | + |
| Опыт работы со СП | - | + | - | - | + |
| Используемая среда на предприятии НПК "Технологический центр" | - | - | - | - | + |
| Наличие функции тестирования | + | + | + | + | + |
| Наличие функции автодополнения кода | - | + | - | + | + |
| Встроенная JVM | + | - | - | + | + |
| Создание документации кода | + | + | + | + | + |
| Стоимость СП | Бесплатно | Платно (499$) | Бесплатно | Бесплатно | Бесплатно |

Начнем со среды программирования под названием BlueJ. Это интегрированная среда разработки (IDE) для языка программирования Java, которая разработана специалистами британского Университета Кента, австралийского Университета Монаша и компании Sun Microsystems.

BlueJ является бесплатным продуктом, написана на Java и создана в основном для обучения программированию, а также подходит также и для разработки небольших программ, имеет простой и понятный интерфейс. Но, несмотря на это, имеет много различных стандартных и специфичных инструментов.

Разработка проекта осуществляется в виде графического представления интерактивной схемы проекта (на UML-подобной диаграмме), где находятся все ее объекты и связи между ними, а также среда позволяет изменять код каждого такого объекта для более детальной проработки своего проекта. BlueJ позволяет проводить тестирование в проекте, а также читать и, что не менее важно, создавать свою собственную документацию на основе созданного кода. Среда работает в ОС Windows, Mac OS X, Ubuntu и других, что делает ее кроссплатформенной. [37]

Также имеет и недостатки, а именно – не имеет функции автодополнения вводимого кода и не имеет поддержку подключения фреймворков, кроме написанных классов на ЯП Java.

Актуальная версия BlueJ 5.0.0 была выпущена 28 января 2021 года.

IntelliJ IDEA (Ultimate) – интегрированная среда разработки ПО для большого числа ЯП, в том числе и Java. Первая версия появилась в январе 2001 года. Разработчиком является компания JetBrains.

Является платным продуктом (499$), но является ведущей средой разработки на языке Java. Содержит в себе множество высокотехнологичных инструментов, которые позволяют избавить разработчика от рутинной работы, делая ее автоматически за него. Работает в ОС Windows, Linux и macOS, но для ее работы нужно самостоятельно установить JVM на свое устройство. IntelliJ IDEA поддерживает все фреймворки и платформы, которые только могут понадобиться разработчику, имеет функцию умного автодополнения, а также анализа кода, который позволяет выявлять ошибки и делает проект только качественнее. Пользователи могут самостоятельно настраивать сборку своего проекта. Позволяет проводить тестирование кода, а также создавать документацию написанного кода (JavaDoc файлы), а также поддерживает работу системы контроля версий Git. [13]

Актуальной версией на данный момент является IntelliJ IDEA 2021.1, который была создана 7 апреля 2021 года.

Apache NetBeans IDE – свободная бесплатная интегрированная среда разработки на таких ЯП, как Java, Python, C/C++ и ряда других. Данная среда была разработана Apache Software Foundation, Oracle и Sun Microsystems в 1997 году.

Позволяет разрабатывать различные приложения для различных устройств в ОС Microsoft Windows, Linux, macOS и Solaris. Имеет возможность подключения фреймворков, для повышения доступного функционала. Присутствует возможность использования системы контроля версий Git, проводить отладку проекта и тестирование, а также создавать документацию своего кода. [14]

Также стоит сказать и про встроенный инструмент анализа кода, который в интеграции с плагином FindBug позволяет выявить и устранить сложности в коде. Но, к сожалению, среда не имеет функции автодополнения кода, а также для работы в среде необходимо самостоятельно установить JVM. [14]

Apache NetBeans IDE 12.3 на данный момент является самой актуальной версией.

Xcode — интегрированная среда разработки ПО для платформ macOS, iOS, watchOS и tvOS. Разработана корпорацией Apple Inc. Является бесплатным продуктом.

Поддерживает языки C/C++, Objective-C, Objective-C++, Swift, Java, AppleScript, Python и Ruby. Установка происходит из приложения App Store пользовательского устройства Mac без загрузок дополнительных файлов, среда является бесплатной.

Пользовательский интерфейс (UI) располагается только в одном окне, где располагаются все доступные инструменты и элементы управления. Данная среда позволяет пройти от самого начала разработки приложения доего завершения, а именно содержит инструменты создания, тестирования, оптимизации, и даже представления приложения в App Store. [15]

Присутствует функция интеллектуального автодополнения кода, а также создания JavaDoc файлов, что позволяет грамотно документировать своей проект. Имеет встроенную систему контроля версий Git, может подгружать в проект различные библиотеки и фреймворки. [15]

Выпущенная 24 марта 2020 года Xcode 11.4 (11E146) – самая актуальная версия данной среды разработки.

Свободная интегрированная среда разработки Eclipse была спроектирована некоммерческой организацией Eclipse Foundation, является бесплатным продуктом, предназначенным для разработки модульных кроссплатформенных приложений. [12]

Eclipse написан на ЯП Java, поэтому является кроссплатформенной средой и работает в ОС GNU/Linux, macOS, Windows и Solaris. При установке может автоматически установить нужную версию JVM. Является гибким инструментом, так как является расширяемой средой – любой пользователь может расширить свою среду на свое усмотрение, в том числе при помощи подключения различных библиотек и фреймворков. [12]

В Eclipse встроена система контроля версий Git, присутствует возможность тестирования и документирования написанного кода, а также его автодополнения. [12]

Eclipse 4.19, вышедший 17 марта 2021, является самой последней и актуальной разработкой данной среды.

Из всех рассмотренных сред программирования Eclipse лучше всего подходит для реализации поставленной задачи, а также удовлетворяет всем требованиям и критериям предприятия НПК "Технологический центр".

## 2.2 Архитектура ПМ ВКМ

Согласно ГОСТу Р 57100-2016/ISO/IEC/IEEE 42010:2011 архитектура (системы) (architecture) – это основные понятия или свойства системы в окружающей среде, воплощенной в ее элементах, отношениях и конкретных принципах ее проекта и развития.

Так как ПМ при старте должен считывать все файлы, относящиеся к конфигурированию схема, то его входными данными будут являться файлы уже созданных схем конфигурирования и библиотек всевозможных элементов, нужных для их построения. Все файлы хранятся в памяти пользовательского устройства.

Такие конфиг-файлы библиотек содержат в себе информацию в виде структур набора пар ключей – значений, и/или упорядоченного набора значений. Такое решение было продиктовано требованиями предприятия по разработке данного модуля. Было принято решение, что ПМ в процессе своей работы будет работать с данными в формате JSON, так такой формат как нельзя кстати подходит для работы с такой информацией.

ПМ ВКМ будет создавать файлы, содержащие в себе код конечной схемы конфигурирования, а значит они и будут являться выходными данными ПМ.

Перед разработкой самого ПМ ВКМ было принято решение сначала спроектировать схему данных, алгоритм работы и функциональную схему данного ПМ.

Схема данных

При старте ПМ считывает все конфиг-файлы уже созданных компонентов, а именно схем конфигурирования и библиотек их элементов. После чего пользователь проводит свою работу со схемами – создание или редактирование, каждое изменение будет сохраняться в соответствующем файле. После завершения работы со схемой или схемами создается единый итоговый файлы схемы, содержащий в себе все элементы, которые были расположены на ней. Далее этот итоговый файл конвертируется в понятный микросхеме бинарный код, который сохраняется в файле. После чего происходит его трансляция в микросхему.

На рис. 2.1 можно видеть полученную схему данных ПМ ВКМ.



Рис. 2.1 Схема данных ПМ ВКМ

Схема алгоритма

Алгоритм работы ПМ ВКМ начинается со считывания конфиг-файлов схем конфигурирования и конфиг-файлов библиотек элементов, из которых составляется графический список элементов, которые пользователь сможет перемещать на поле схемы конфигурирования, создавать межкомпонентные связи, и изменять или заменять данные каждого такого элемента. После чего пользователь запускает процесс конфигурации микросхемы, ПМ создает итоговый файл текущей схемы, из которого образуется файл с бинарным кодом итогового файла, и происходит его транслирование в микросхему.

На рис. 2.2 можно видеть полученную схему алгоритма ПМ ВКМ.



Рис. 2.2 Схема алгоритма ПМ ВКМ

Функциональная схема

Пользователь, используя периферию своего устройства (монитор, мышь, клавиатура), при запуске ПМ ВКМ может либо создать новую схему конфигурирования, либо открыть уже созданную. После чего может отредактировать ее, эти изменения запишутся в соответствующие файлы. При запуске процесса конфигурирования микросхемы ПМ соберет все элементы текущей схемы в один итоговый файл, сконвертирует его в бинарный код, и проведет его транслирование в микросхему, которая в случае положительного результата верификации будет запрограммирована и вернет результат транслирования пользователю.

На рис. 2.3 можно видеть полученную функциональную схему ПМ ВКМ.



Рис. 2.3 Функциональная схема ПМ ВКМ

## 2.3 Проектирование ПМ ВКМ

После выбора языка и среды программирования, проектирования схемы данных, алгоритма работы и функциональной схемы настало время разрабатывать ПМ ВКМ.

Разрабатывать код программы трудоемкая задача. Для оптимизации этого процесса было принято решение продуманно и обоснованно использовать паттерны проектирования, это позволило бы повысить качество модуля, структурировать код и сделать его понятнее как для себя, так и для других.

Паттерн MVC

MVC (Model-View-Controller) – паттерн проектирования, которые разделяет решение на три блока:

* Модель (Model) – заключает в себе объектную модель предметной области;
* Представление (View) – отвечает за визуальное отображение данных;
* Контроллер (Controller) – служит связующим звеном между системой и пользователем.

Каждый такой блок разрабатывается независимо друг от друга, но между ними существует связь (рис. 2.4): когда пользователь совершает какое-либо действие, контроллер обновляет модель, которая совершает запрос изменения данных в представлении, что и отображается пользователю в ответ на его действие.



Рис. 2.4 Паттерн MVC

В ПМ ВКМ было создано три отдельных проекта, как три соответствующих блока паттерна MVC:

Модель (ru.tcen.pak.workarea.model), содержащая в себе класс MindMapNode, который отражает модель Node:

package ru.tcen.pak.workarea.model;

public class MindMapNode extends AbstractMindMapItem implements Serializable {

private static final long serialVersionUID = 8875579454539897410L;

public static final String userDir = System.getProperty("user.dir");

public static final String PROP\_COLOR = "color";

public static final String PROP\_BOUNDS = "bounds";

public static final String PROP\_INCOMING\_CONNECTIONS = "incomingConnections";

public static final String PROP\_OUTGOGING\_CONNECTIONS = "outgoingConnections";

public static final String PROP\_IMAGE = "image";

public static final String PROP\_FILE = "file";

public static final String PROP\_QUANTITYANCHORS = "quantityAnchors";

public static final String PROP\_ANCHORS = "anchors";

public static final String PROP\_NAME = "name"; // for functional node

public static final String PROP\_DESCRIPTION = "description"; // for functional node

public static final String PROP\_NUMBER\_OF\_HEX\_PARAMETERS = "number\_of\_hex\_parameters"; // for functional node

public static final String PROP\_HEX\_PARAMETERS = "hex\_parameters"; // for functional node

public static final String PROP\_PARAMETERS = "parameters"; // for functional node

public static final String PROP\_FUNCTION\_HEX\_FIELD = "function\_hex\_field"; // for functional node

private String nodeCustomJSON = userDir + File.separator + "files" + File.separator + "nodes" + File.separator;

private String nodeAllJSON = userDir + File.separator + "files" + File.separator;

private String nodeCode = userDir + File.separator + "files" + File.separator + "nodes" + File.separator;

private String nodeDirectory = userDir + File.separator + "files" + File.separator + "nodes" + File.separator;

private Set<String> titlesIncomingConnection = new HashSet<>();

private Set<String> titlesOutgoingConnection = new HashSet<>();

private String name; // for functional node

private String description; // for functional node

private String number\_of\_hex\_parameters; // for functional node

private String function\_hex\_field; // for functional node

private ArrayList<ArrayList<HashMap<String, String>>> hex\_parameters; // for functional node

private ArrayList<ArrayList<HashMap<String, String>>> parameters; // for functional node

}

}

Представление (ru.tcen.pak.workarea.visuals), содержащее в себе класс MindMapNodeVisual, который представляет класс MindMapNodeVisual:

package ru.tcen.pak.workarea.visuals;

public class MindMapNodeVisual extends Region {

private static final double HORIZONTAL\_PADDING = 20d;

private static final double VERTICAL\_PADDING = 10d;

private static final double VERTICAL\_SPACING = 5d;

private static final int SIZERECTANGLEBOX = 10;

private double NODE\_WIDTH = 170;

private double NODE\_HEIGH = 170;

private TextFlow descriptionFlow;

private Text descriptionText;

private Text nameText;

private GeometryNode<RoundedRectangle> shape;

private VBox labelVBox;

private Image descriptionImage;

private String urlImage = "null.png";

private int quantityImage = 0;

private List<Point> points = new ArrayList<>();

private List<Rectangle> pointsBox = new ArrayList<>();

private Color color;

private boolean isStatic;

private boolean connectionOnlyRight;

private int quantityRectangleConnection;

public List<Point> pointConnection = new ArrayList<>();

private boolean startPainting = true;

}

Контроллер (ru.tcen.pak.workarea), нужный для запуска и координирования работы ПМ ВКМ, содержащий в себе класс SimpleMindMapApplication

package ru.tcen.pak.workarea;

/\*\*

\* Entry point for our Simple Mind Map Editor, creating and rendering a JavaFX

\* Window.

\*

\*/

public class SimpleMindMapApplication extends Application {

public static InfiniteCanvas paletteRootNode;

public static IViewer viewer;

public static void main(String[] args) {

Application.launch(args);

}

private Stage primaryStage;

private HistoricizingDomain domain;

/\*\*

\* Creates the undo/redo buttons

\*

\* @return

\*/

private Node createButtonBar() {

Button undoButton = new Button("Undo");

undoButton.setDisable(true);

undoButton.setOnAction((e) -> {

undo();

});

Button redoButton = new Button("Redo");

redoButton.setDisable(true);

redoButton.setOnAction((e) -> {

redo();

});

// add listener to the operation history of our domain

// to enable/disable undo/redo buttons as needed

domain.getOperationHistory().addOperationHistoryListener((e) -> {

IUndoContext ctx = domain.getUndoContext();

undoButton.setDisable(!e.getHistory().canUndo(ctx));

redoButton.setDisable(!e.getHistory().canRedo(ctx));

});

return new HBox(10, undoButton, redoButton);

}

@Override

public void start(Stage primaryStage) throws Exception {

SimpleMindMapModule module = new SimpleMindMapModule();

this.primaryStage = primaryStage;

// create domain using guice

this.domain = (HistoricizingDomain) Guice.createInjector(module).getInstance(IDomain.class);

// update, refresh source and data

updateSource();

// create viewers

hookViewers();

// activate domain

domain.activate();

// load contents

populateViewerContents();

defineHotKeys();

// set-up stage

primaryStage.setResizable(true);

primaryStage.setTitle("GEF Simple Mindmap");

primaryStage.sizeToScene();

primaryStage.show();

}

}

Паттерн декоратор

Паттерн декоратор (Decorator) необходим для расширения функционала класса MindMapNodeVisual с помощью класса MindMapNodePart. Он добавляет ему динамическое обновление позиций и размеров элементов в зависимости от его наполнения:

package ru.tcen.pak.workarea.parts;

/\*\*

\* the {@link MindMapNodePart} is responsible to create and update the

\* {@link MindMapNodeVisual} for a instance of the {@link MindMapNode}.

\*

\*/

public class MindMapNodePart extends AbstractContentPart<MindMapNodeVisual> implements

ITransformableContentPart<MindMapNodeVisual>, IResizableContentPart<MindMapNodeVisual>, PropertyChangeListener {

public static Set<MindMapNode> mindMapNode = new HashSet<>();

public static Set<MindMapNodeVisual> mindMapNodeVisual = new HashSet<>();

private static Color greenNode = Color.GREENYELLOW; // the color of the node if it is in a chain

private static Color redNode = Color.PALEVIOLETRED; // the color of the node if it is not in a chain

public int quantityRectangleConnection;

public boolean connectionOnlyRight;

public boolean isStatic;

@Override

protected void doActivate() {

super.doActivate();

getContent().addPropertyChangeListener(this);

}

@Override

protected MindMapNodeVisual doCreateVisual() {

propertySet();

MindMapNodeVisual visual = new MindMapNodeVisual(quantityRectangleConnection, connectionOnlyRight, isStatic);

visual.rePaintingRectangleConnection();

return visual;

}

@Override

protected void doDeactivate() {

getContent().removePropertyChangeListener(this);

super.doDeactivate();

}

@Override

public void doRefreshVisual(MindMapNodeVisual visual) {

MindMapNode node = getContent();

visual.setName(node.getName());

visual.setDescription(node.getDescription());

if (node.getImage() != null) {

visual.setImage(node.getImage());

}

visual.setColor(node.getColor());

visual.rePaintingRectangleConnection();

// use the IResizableContentPart API to resize the visual

setVisualSize(getContentSize());

// use the ITransformableContentPart API to position the visual

setVisualTransform(getContentTransform());

// added each node in collection for update each node

MindMapNodePart.mindMapNode.add(node);

// added each node visual in collection for update each node visual

MindMapNodePart.mindMapNodeVisual.add(visual);

}

@Override

public MindMapNode getContent() {

return (MindMapNode) super.getContent();

}

@Override

public Dimension getContentSize() {

for (MindMapNodeVisual mindMapNodeVisual : mindMapNodeVisual) {

if (mindMapNodeVisual.getNameText().getText().equals(getContent().getName())) {

mindMapNodeVisual.rePaintingRectangleConnection();

}

}

return getContent().getBounds().getSize();

}

@Override

public Affine getContentTransform() {

Rectangle bounds = new Rectangle();

bounds = getContent().getBounds();

return new Affine(new Translate(bounds.getX(), bounds.getY()));

}

@Override

public void propertyChange(PropertyChangeEvent event) {

String prop = event.getPropertyName();

if (MindMapNode.PROP\_COLOR.equals(prop) || MindMapNode.PROP\_DESCRIPTION.equals(prop)

|| MindMapNode.PROP\_IMAGE.equals(prop) || MindMapNode.PROP\_NAME.equals(prop)

|| MindMapNode.PROP\_BOUNDS.equals(prop)) {

refreshVisual();

}

}

@Override

public void setContentSize(Dimension totalSize) {

// storing the new size

getContent().getBounds().setSize(totalSize);

}

@Override

public void setContentTransform(Affine totalTransform) {

Rectangle bounds = getContent().getBounds().getCopy();

totalTransform.setTx(bounds.getX());

totalTransform.setTy(bounds.getY());

ITransformableContentPart.super.setVisualTransform(totalTransform);

}

@Override

public void setVisualSize(Dimension totalSize) {

IResizableContentPart.super.setVisualSize(totalSize);

// perform layout pass to apply size

getVisual().getParent().layout();

}

}

Паттерн наблюдатель

Паттерн наблюдатель (Observer) – паттерн проектирования, нужный для создания механизма автоматического оповещения подписчиков об изменениях.

Использование это паттерна в ПМ ВКМ необходимо, так как пользователь постоянно совершает различные действия, которые приводят к изменению данных.

Класс AbstractMindMapItem, являющийся наблюдателем:

package ru.tcen.pak.workarea.model;

/\*\*

\* This class provides the {@link PropertyChangeSupport} for the MindMapitems

\*

\*/

public class AbstractMindMapItem implements Serializable {

/\*\*

\* Generated UUID

\*/

private static final long *serialVersionUID* = -2558628513984118991L;

protected PropertyChangeSupport pcs = new PropertyChangeSupport(this);

public void addPropertyChangeListener(PropertyChangeListener listener) {

pcs.addPropertyChangeListener(listener);

}

public void removePropertyChangeListener(PropertyChangeListener listener) {

pcs.removePropertyChangeListener(listener);

}

}

И, например, класс MindMapNode, являющийся подписчиком, который содержит данные методы:

public void setBounds(Rectangle bounds) {

pcs.firePropertyChange(PROP\_BOUNDS, this.bounds, (this.bounds = bounds.getCopy()));

}

public void setColor(Color color) {

pcs.firePropertyChange(PROP\_COLOR, this.color, (this.color = color));

}

public void setDescription(String description) {

pcs.firePropertyChange(PROP\_DESCRIPTION, this.description, (this.description = description));

}

public void setFile(File file) {

pcs.firePropertyChange(PROP\_FILE, this.file, (this.file = file));

}

public void setFunctionHexField(String function\_hex\_field) {

pcs.firePropertyChange(PROP\_FUNCTION\_HEX\_FIELD, this.function\_hex\_field,

(this.function\_hex\_field = function\_hex\_field));

}

public void setHexParameters(ArrayList<ArrayList<HashMap<String, String>>> hex\_parameters) {

pcs.firePropertyChange(PROP\_HEX\_PARAMETERS, this.hex\_parameters, (this.hex\_parameters = hex\_parameters));

}

public void setImage(Image image) {

pcs.firePropertyChange(PROP\_IMAGE, this.image, (this.image = image));

}

public void setName(String name) {

titlesIncomingConnection.add(name);

titlesOutgoingConnection.add(name);

pcs.firePropertyChange(PROP\_NAME, this.name, (this.name = name));

}

public void setNodeAllJSON(String nodeAllJSON) {

this.nodeAllJSON = nodeAllJSON;

}

public void setNodeCode(String nodeCode) {

this.nodeCode = nodeCode;

}

public void setNodeCustomJSON(String nodeCustomJSON) {

this.nodeCustomJSON = nodeCustomJSON;

}

public void setNodeDirectory(String nodeDirectory) {

this.nodeDirectory = nodeDirectory;

}

public void setNumberOfHexParameters(String number\_of\_hex\_parameters) {

pcs.firePropertyChange(PROP\_NUMBER\_OF\_HEX\_PARAMETERS, this.number\_of\_hex\_parameters,

(this.number\_of\_hex\_parameters = number\_of\_hex\_parameters));

}

public void setParameters(ArrayList<ArrayList<HashMap<String, String>>> parameters) {

pcs.firePropertyChange(PROP\_PARAMETERS, this.parameters, (this.parameters = parameters));

}

public void setStatic(boolean isStatic) {

this.isStatic = isStatic;

}

## 2.4 Разработка пользовательского интерфейса

Разработку пользовательского интерфейса было предложено создавать на основе JavaFX, а именно плагина для Eclipse – Eclipse Graphical Editing Framework (GEF).

При запуске ПМ ВКМ открывается окно, как видно на рис. 2.5, которое делится на две части: левую, которая содержит сверху всевозможный инструментарий и список компонентов, прочитанных из конфиг-файлов библиотек элементов, которые можно использовать в правой части приложения – на поле, предназначенном для создания схемы конфигурирования микросхемы.

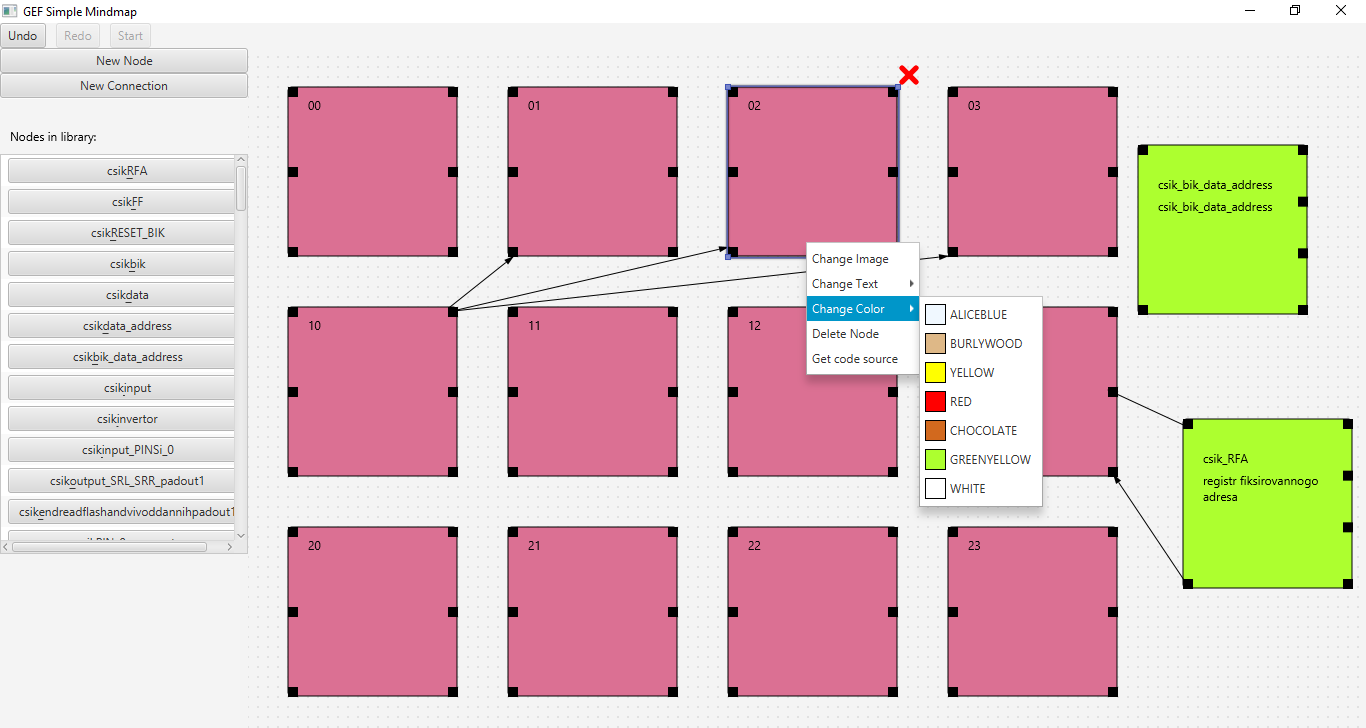


Рис. 2.5 Пользовательский интерфейс

## Вывод по конструкторскому разделу

В конструкторском разделе определен язык программирования и выбран среда программирования для разработки ПМ ВКМ.

Описана архитектура ПМ, основанная на разработке схемы данных, схемы алгоритма и функциональной схемы. Спроектирован ПМ ВКМ с использованием паттернов проектирования, а также разработан пользовательский интерфейс.

# 3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

Помимо реализации самого программного продукта, не менее важными этапами его разработки является отладка и тестирование написанного программного кода.

## 3.1 Отладка ПМ ВКМ

Отладка – это процесс, который позволяет обнаружить и исправить ошибки в программном коде.

Существует две технологии, с помощью которых можно осуществлять процесс отладки:

* использование отладчика;
* журналирование.

Отладчик – программа, которая позволяет в автоматическом режиме совершать поиск ошибок в программном коде, а также предоставлять дополнительные мощные инструменты контроля его выполнения, а именно:

* пошаговое выполнение кода;
* выполнение кода до определенных, поставленными на свое усмотрение программистом точек останова, называемых breakpoint-ами;
* отслеживание значений любых инициализированных на данный момент переменных.

В ЯП java присутствует встроенный инструмент отладчик называемый jdb (Java Debugger). В СП Eclipse так же есть встроенный отладчик, называемый Eclipse Debugger.

Отладка Java Debugger

Отладка написанного программного кода при помощи jdb (Java Debugger) осуществляется при помощи встроенных инструментов ЯП Java, поэтому весь процесс работы с отладчиком будет происходить непосредственно в командной строке операционной системы разработчика, например ОС Windows.

Поэтому для этой ОС необходимо установить системные переменные среды для javac и jdb, чтобы команды, запускаемые в командной строке, стали глобальными и доступными из любого расположения, либо перейти в расположение ЯП Java и там же вызвать командную строку, с которой далее будет происходить работа по отладке кода.

Следующее что нужно сделать, это подготовить написанный код, создав байт-код этого класса, с которым нужно провести процесс отладки, делается это в уже запущенной командной строке при помощи команды:

javac <путь до файла><имя файла>.java

Если произойдет ошибка, то об этом можно будет узнать из соответствующего сообщения в командной строке, нужно будет ее исправить и сделать так, чтобы не было ни одной ошибки, так как иначе байт-код не создастся. Если никакого сообщения не было выведено, так же, как и на рис. 3.1, значит байт-код этого класса был создан и в этом же расположения должен появиться файл с тем же названием и расширением “.class”.

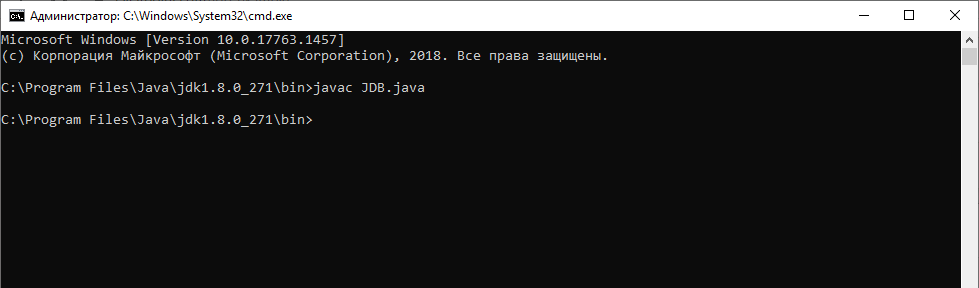


Рис. 3.1 Положительный результат работы javac

После успешного создания байт-кода нужно запустить сам отладчик Java Debugger на нем с помощью команды:

jdb <путь до файла><название файла>

Успешный запуск показан на рис. 3.2, где было выведено, что отладчик проинициализирован и готов к работе.

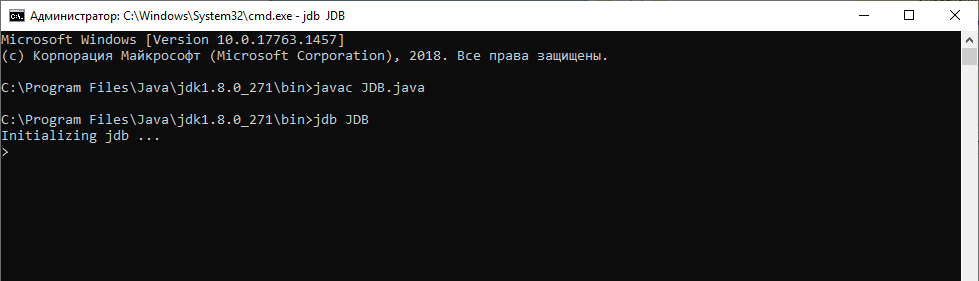


Рис. 3.2 Успешный запуск отладчика Java Debugger

После этого необходимо написать команду в самом отладчике:

run

После чего отладчик в автоматическом режиме проведет проверку кода, и покажет ошибки, если они присутствуют в коде, и выполнит саму программу, как на рис.3.3, если ошибок нет.

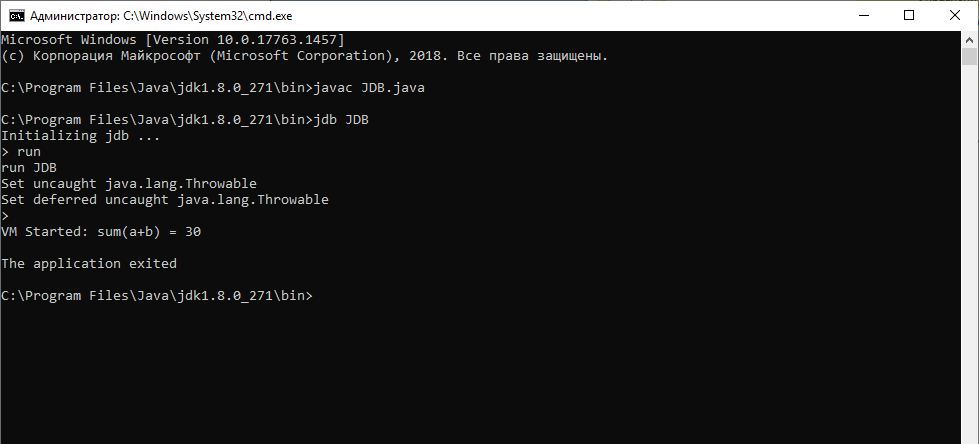


Рис. 3.3 Успешная отладка написанного программного кода

Отладчик Java Debugger выполняет роль отладчика, но при этом имеет ряд существенных недостатков:

* процесс отладки проходит в консоли, без графического интерфейса;
* отсутствует пошаговое выполнения кода, так же, как и до точек останова;
* невозможно отследить значение проинициализированных переменных

Отладка Eclipse Debugger

Eclipse Debugger – инструмент для отладки, который встроен в СП Eclipse, благодаря чему разработка проекта может совмещаться с его отладкой без длительной подготовки, как с отладчиком jdb. Отладчик имеет графический интерфейс, что позволяет удобно использовать все его возможности.

Чтобы запустить этот отладчик, достаточно открыть контекстное меню выбранного проекта, и запустить его инструментом “Java Application” из вкладки “Debug as”, как показано на рис. 3.4, после чего произойдет автоматическая замена перспективы (вида) проекта на Debug.

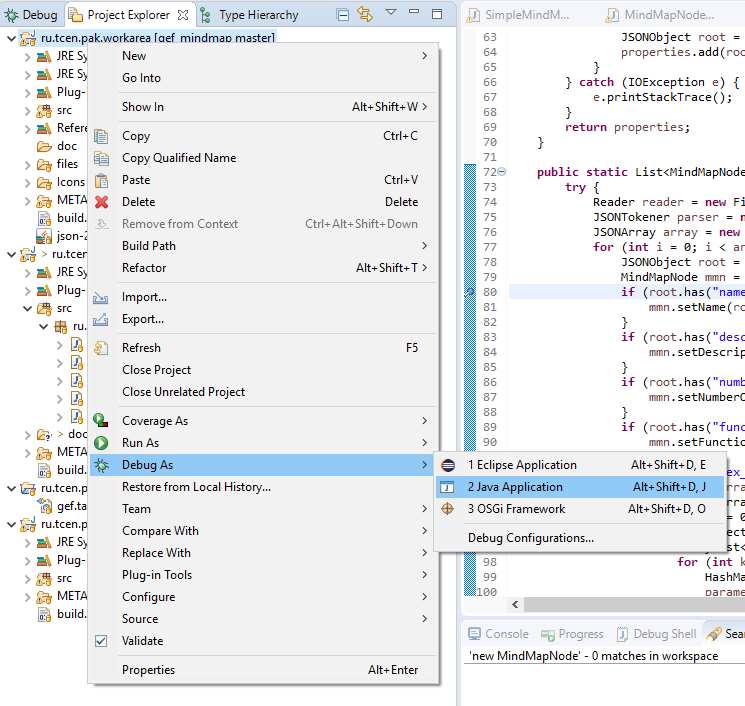


Рис. 3.4 Запуск отладчика по выбранному проекту

Помимо главной цели любого отладчика, а именно проверки программного кода и обнаружения в нем ошибок, в Debug перспективе проекта присутствует множество дополнительных мощных инструментов, которые помогают не только проводить процесс отладки, но и активный процесс разработки программного продукта.

В java-проекте Eclipse можно расставлять точки останова на любых номерах линий кода, которые называются breakpoint-ами, на рис. 3.5 показано каким образом они отображаются во вкладке Breakpoints СП Eclipse.

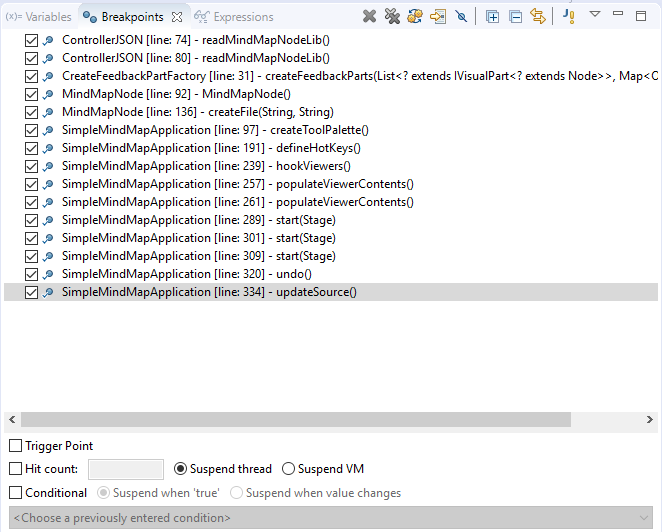


Рис. 3.5 Расположение точек останова во вкладке Breakpoints

Такие точки останова позволяют отладчику выполнять код до определенного момента, то есть до линии, где расположена точка останова. В этот момент выполнение программы останавливается, и отладчик дает программисту возможность отслеживания значений любых проинициализированных переменных на текущий момент времени.

Eclipse Debugger позволяет отслеживать значения переменных во вкладке Expression, как показано на рис. 3.6, в поле Name нужно указать название переменной, после чего в поле Value будет указанное значение.

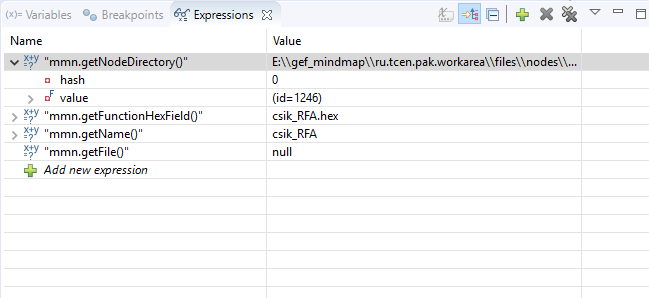


Рис. 3.6 Отслеживание значения переменных во вкладке Expression

Также этот отладчик позволяет при наведении курсора мыши на переменную, показать ее значение, как на рис. 3.7.

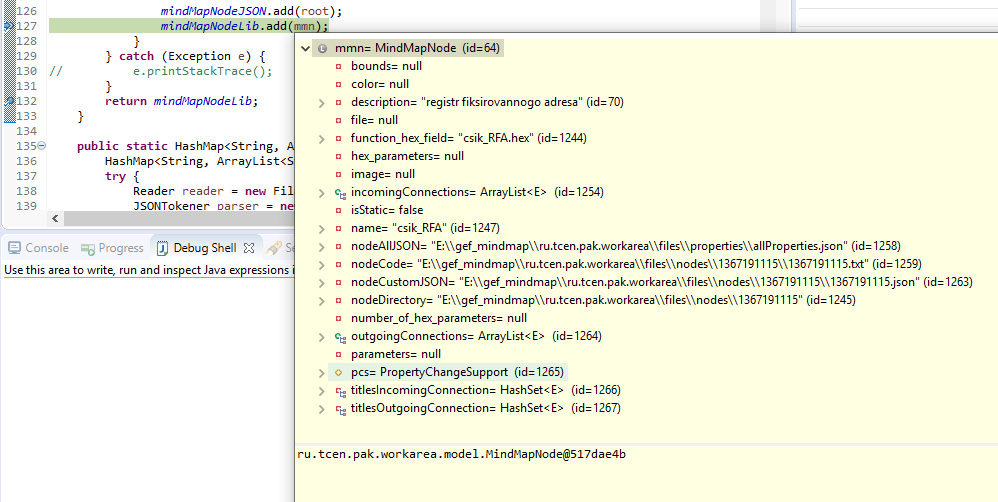


Рис. 3.7 Отслеживание значения переменных при наведении курсора мыши

Отладчик Eclipse Debugger был использован при разработке ПМ ВКМ в качестве обнаружения ошибок и отслеживания различных значений множеств переменных в процессе работы ПМ ВКМ.

## 3.2 Журналирование ПМ ВКМ

При разработке программного продукта нередко могут возникнуть ситуации, когда происходит неизвестная ошибка, которая весьма проблематично отследить, в том числе и при помощи отладчика.

Решение такой проблемы возможно найти в использовании журналировании (логировании) программного продукта, то есть записи информации о выполнении действий, произошедших ошибках, а также другую необходимую информацию в отдельный файл или файлы. Такие файлы называются файлами-дампами или файлами-журналами.

Конечно, для быстрого получения сообщений или логов об ошибках можно просто-напросто использовать встроенный в ЯП java инструмент вывода в консоль или файл, где можно будет указывать свое сообщение, но такое решение не является предпочтительным, так как имеет малозначительный функционал и увеличивает количество повторяемого кода.

Но существует значительно лучший и продуктивный инструмент журналирования, который называется Log4j, который самостоятельно записывает все сообщения в файл. Каждый такой лог содержит полную дату и время, когда произошла запись, название метода, в котором был запущен этот лог, указанный уровень логирования и само сообщение.

Log4j имеет несколько уровней логирования, для упрощения различия и понимания сообщений, каждый уровень имеет свое значение и смысл, которые указаны в табл. 3.1, чем меньше значение уровня, тем сообщение должно быть хуже и фатальнее, а значит на него нужно обратить внимание, разобраться в чем проблема и исправить ее.

Табл. 3.1

Уровни журналирования/логирования

|  |  |  |
| --- | --- | --- |
| Название уровня логирования | Описание уровня логирования | Значение уровня логирования |
| OFF | Предназначен для отключения логирования | 0 |
| FATAL | Ошибка, после которой приложение будет остановлено | 100 |
| ERROR | Ошибка, которая не останавливает приложение, но которую необходимо решить | 200 |

Продолжение табл. 3.1

|  |  |  |
| --- | --- | --- |
| WARN | Действие будет выполнено, но возникнет предупреждение, говорящее о неожиданном результате | 300 |
| INFO | Информация, в которой отображаются обычные действия программы | 400 |
| DEBUG | Логи для отладки приложения | 500 |
| TRACE | Менее приоритетные логи для отладки | 600 |
| ALL | Под этим логом будут указаны все логи, создаваемые системы, в которых не указаны любые другие логи | MAX\_VALUE |

Во время работы ПМ ВКМ осуществляется журналирование/логирование с помощью инструмента Log4j. На рис. 3.8 можно видеть пример логирования ПМ ВКМ при его запуске.

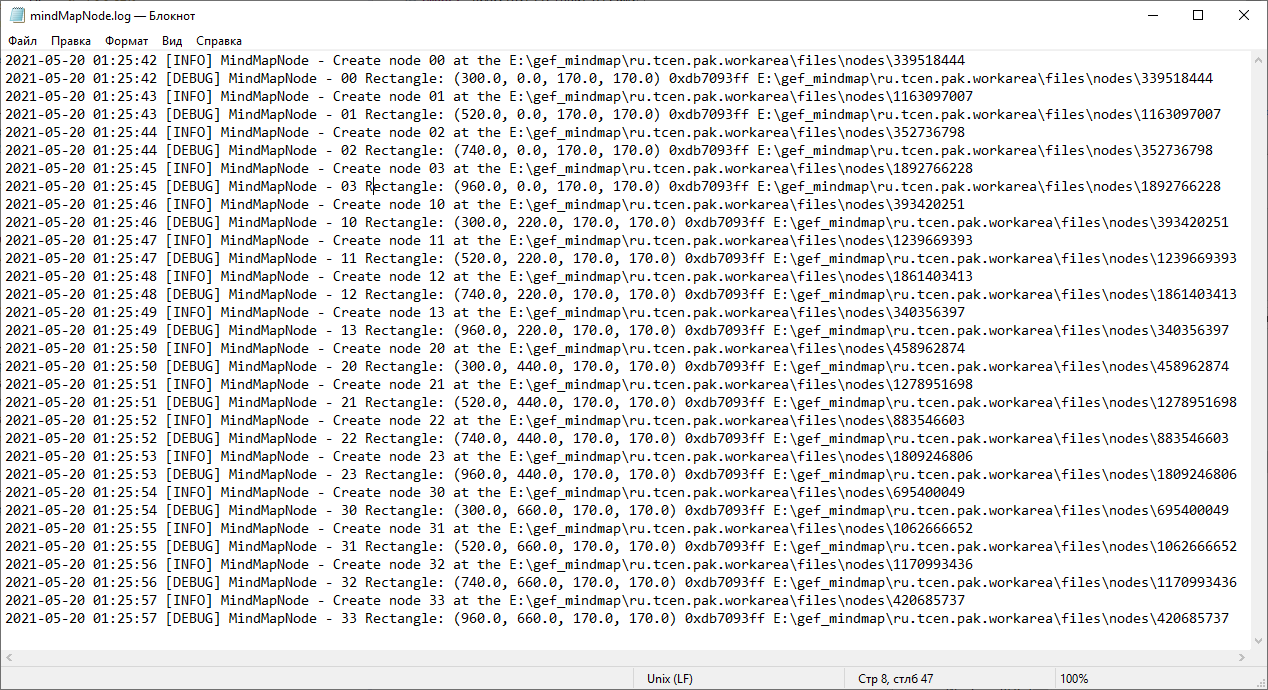


Рис. 3.7 Пример логирования ПМ ВКМ при запуске

## 3.3 Анализ методов и средств тестирования

Процесс тестирования очень сильно связан с процессом отладки, так результаты тестирования могут дать понять, что допущена ошибка, которую проще и быстрее всего будет найти и исправить при помощи отладчика.

Тестирование является важнейшим этапом разработки программного решения после его написания, так как тестирования программного кода могут выявить множество ошибок, которые были допущены либо по незнанию, либо вследствие человеческого фактора.

Для начала нужно понять, какие существуют методы и средства тестирования, провести их анализ и сравнение, выбрать необходимые и провести само тестирование.

Тестирование – процесс выполнения программного продукта с целью выявления ошибок.

Существует три вида тестирования:

* тестирование белым ящиком (White Box)
* тестирование черным ящиком (Black Box)
* тестирование серым ящиком (Grey Box)

Суть белого ящика сводится к тому, что тестировщику известно все устройство тестируемой системы. Тестировщик внимательно изучает весь код программы, и на основе предусмотренных и не предусмотренных внутренних конструкций сравнивает ожидаемый и полученный результат [39].

Таким методом можно покрыть большую часть приложения, но для самого тестирования нужно разбираться во всей его структуре.

В тестирование белым ящиком входит в основном модульное (юнит-тестирование) и интеграционное тестирование.

Тестирование черным ящиком основано на поведении программы, тестирование использует исключительно внешние интерфейсы тестируемой системы, то есть главное здесь, то, что приложение делает, а не то, каким образом это делает [39]. Тестировщик не имеет знаний по тому, как устроена тестируемая система.

Такое тестирование может выявить непредусмотренные ошибки или противоречия системы, но протестировать получится ограниченное число путей выполнения такой системы, у таких тестов нет четкого описания и спецификации.

В тестирование черным ящиком входит в основном системное и приемочное тестирование.

Тестирование серым ящиком совмещает в себе тестирования белым и черным ящиками, внутренняя структура тестируемой системы частично известна, но само тестирование производится с позиции черного ящика, то есть конечного пользователя, а не разработчика, как в белом ящике.

На рис. 3.8 можно понять все вышеизложенные различия между черным, серым и белым тестированием.

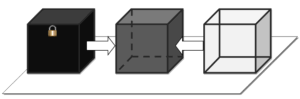


Рис. 3.8 Тестирование черным, серым и белым ящиком

Различают следующие средства (методики) тестирования:

* модульное тестирование;

Модульное тестирование (Unit-тестирование), как понятно из названия, тестирование по определенным модулям – отдельным программным компонентам, например отдельным классам или даже функциям, то есть тестирование проходит на объектном уровне.

Для этого создаются тест-коды, осуществляющие проверку на соответствие ожидаемого и полученного результата.

* интеграционное тестирование;

Интеграционное тестирование проводят тестирование совмещенных между собой отдельных модулей, которые прошли модульное тестирование.

* системное тестирование;

Системное тестирование позволяет проверить всю систему на наличие ошибок. То есть все программные и аппаратные составляющие системы соединяются между собой в единую программу, которая проходит тестирование черным ящиком.

* приемочные испытания.

Приемочные испытания, как понятно из названия, это самые последние тесты, проводящиеся перед передачей готового продукта заказчику, нужен для обеспечения гарантий работоспособности этого продукта.

## 3.4 Тестирование ПМ ВКМ

При разработке ПМ ВКМ для тестирования был использован метод белого ящика, а также стратегия модульного тестирования (Unit-тестирования). Такое тестирование позволило на всех этапах разработки следить за правильностью выполнения кода

Были написаны множество тест-кейсов, используя инструмент JUnit5. На рисунке 3.9 можно видеть результат тестирования некоторых из таких сценариев, название каждого такого теста содержит в себе описание сценария.

Как видно из рисунка 3.9 они все были пройдены.

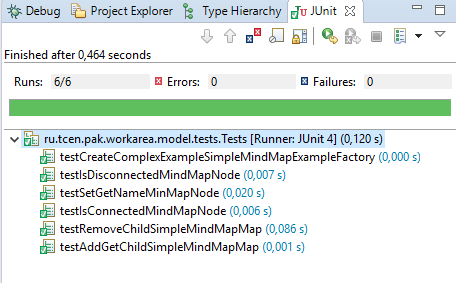


Рис. 3.9 Положительно пройденные Unit-тесты

Описание пройденных тестов:

* testAddGetChildSimpleMindMapMap – тестирование добавления элемента MindMapNode в репозиторий хранения и его получение;
* testCreateComplexExampleSimpleMindMapExampleFactory – тестирование создания комплексного хранилища, содержащего в себе несколько элементов MindMapNode;
* testIsConnectedMindMapNode – тестирование соединения двух элементов MindMapNode;
* testIsDisconnectedMindMapNode – тестирование разъединения двух элементов MindMapNode;
* testRemoveChildSimpleMindMapMap – тестирование удаления элемента MindMapNode из его хранилища;
* testSetGetNameMinMapNode – тестирование установки имени элемента MindMapNode и его получение.

Программный код тест-кейсов этих сценариев представлен в классе Tests:

public class Tests {

private MindMapNode mindMapNode = new MindMapNode();

private MindMapNode mindMapNode1 = new MindMapNode();

private MindMapConnection mindMapConnection = new MindMapConnection();

private SimpleMindMap simpleMindMap = new SimpleMindMap();

private SimpleMindMapExampleFactory simpleMindMapExampleFactory = new SimpleMindMapExampleFactory();

@Test

public void testAddGetChildSimpleMindMapMap() {

simpleMindMap.addChildElement(mindMapNode);

simpleMindMap.addChildElement(mindMapNode1);

simpleMindMap.addChildElement(mindMapConnection);

assertEquals("Add and get child is working", simpleMindMap.getChildElements().get(0), mindMapNode);

}

@Test

public void testCreateComplexExampleSimpleMindMapExampleFactory() {

SimpleMindMap simpleMindMap1 = simpleMindMapExampleFactory.createComplexExample();

mindMapNode.setName("name");

int i = 0, j = 0, WIDTH = 170, HEIGHT = 170, DELTA = 50;

assertEquals("Create complex example is working",

((MindMapNode) simpleMindMap1.getChildElements().get(0)).getBounds(),

new Rectangle(300 + (j \* WIDTH) + (j \* DELTA), 0 + (i \* HEIGHT) + (i \* DELTA), WIDTH, HEIGHT));

}

@Test

public void testIsConnectedMindMapNode() {

mindMapNode.setName("mindMapNode");

mindMapNode1.setName("mindMapNode1");

mindMapConnection.connect(mindMapNode, mindMapNode1, null);

assertEquals("Connected is working", mindMapConnection.getSource().getName(), "mindMapNode");

}

@Test

public void testIsDisconnectedMindMapNode() {

mindMapNode.setName("mindMapNode");

mindMapNode1.setName("mindMapNode1");

mindMapConnection.connect(mindMapNode, mindMapNode1, null);

mindMapConnection.disconnect();

assertEquals("Disconnected is working", mindMapConnection.isConnected(), false);

}

@Test

public void testRemoveChildSimpleMindMapMap() {

simpleMindMap.addChildElement(mindMapNode);

simpleMindMap.addChildElement(mindMapNode1);

simpleMindMap.addChildElement(mindMapConnection);

simpleMindMap.removeChildElement(mindMapNode);

assertEquals("Remove child is working", simpleMindMap.getChildElements().get(0), mindMapNode1);

}

@Test

public void testSetGetNameMinMapNode() {

mindMapNode.setName("MindMapNode");

assertEquals("Set and get name is working", mindMapNode.getName(), "MindMapNode");

}

}

Проведение модульного тестирования не гарантирует отсутствие ошибок на высшем уровне – когда такие модули соединяются и работают сообща, как единая программа могут появиться ошибки различного рода.

Поэтому после разработки ПМ ВКМ и проведения модульного тестирования, было принято решение провести интеграционное тестирование, используя все тот же метод белого ящика.

Также были составлены тест-кейсы для этого тестирования, составлена таблица 3.2, содержащая в себе сценарии некоторых тест-кейсов по интеграционному тестированию, их описание, входные данные, ожидаемый и полученный результат, а также результат их прохождения. Ниже таблицы размещено более детальное описание этих сценариев.

Табл. 3.2

Сценарии тест-кейсов для интеграционного тестирования ПМ ВКМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание сценария | Действие | Ожидаемый результат | Полученный результат | Результат прохож-дения |
| 1. Создание блока элементов при запуске | Запуск приложения | Запуск приложения и отображение блока элементов | Приложение запущено, блок элементов отображен | Тест пройден |
| 1. Добавление на поле элемента из библиотеки | Нажать на нужный элемент в библиотеке и нажать на свободное место на поле | На поле появится выбранный элемент | На поле появился выбранный элемент | Тест пройден |
| 1. Создание нового элемента | Нажать на кнопку New Node в инструментах, после чего нажать на свободном месте на поле | На поле появится новый элемент | На поле появился новый элемент | Тест пройден |
| 1. Создание соединения между элементами | Нажать на кнопку New Connection в инструментах, после чего нажать на один элемент на поле и на другой | На поле между выбранными элементами появится соединение | На поле появилось соединение между элементами | Тест пройден |

Продолжение табл. 3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Откат на действие назад | Совершить любое действие в приложении, после чего нажать на кнопку Undo, либо сочетание клавиш ctrl+z в инструментах | Произойдет откат действия назад | Произошел откат действия назад | Тест пройден |
| 1. Откат на действие вперед | Совершить откат действия назад, после чего нажать на кнопку Redo в инструментах, либо сочетание клавиш ctrl+r | Произойдет откат действия вперед | Произошел откат действия вперед | Тест пройден |
| 1. Изменение названия элемента | Нажать ПКМ на элементе, заменить название | Произойдет замена названия элемента | Произошла замена названия элемента | Тест пройден |
| 1. Удаление элемента из поля | Навестись курсором мыши на элемент, после чего нажать на красный крестик | Элемент будет удален | Элемент был удален | Тест пройден |
| 1. Добавление фотографии на элемент | Нажать ПКМ на элементе, заменить фотографию | Произойдет замена фотографии элемента | Произошла замена фотографии элемента | Тест пройден |
| 1. Прерывание добавления фотографии на элемент | Нажать ПКМ на элементе, заменить фотографию, выйти из появившегося диалогового окна выбора фотографии | Вывод сообщения File not selected в диалоговом окне | Произошел вывод сообщения File not selected в диалоговом окне | Тест пройден |

Детальное описание сценариев:

1. тест необходим для проверки работоспособности запуска приложения со всеми модулями (модуль инструментов, модуль библиотек элементов, поле для создания схем конфигурирования) и отображение на поле тестового варианта схемы конфигурирования;
2. тест должен показать работоспособность функции создания нового элемента из библиотек элементов, который содержит ту же информацию, что и в самой библиотеке;
3. тест необходим для проверки работоспособности создания элемента не из библиотеки;
4. соединение между элементами одна из неотъемлемых частей создания различных схем, поэтому тест, проверяющий работоспособность этой функции является необходимым;
5. откат действия назад очень полезная функция при разработке чего-либо для конечного пользователь, поэтому необходимо протестировать, что этот инструмент работает верно;
6. откат действия вперед так же может помочь конечному пользователю в разработке;
7. каждый элемент должен иметь свое уникальное имя, поэтому необходимо провести тестирование функции замены названия элемента;
8. удаление элемента из поля так же является важной частью при разработке схемы конфигурирования, поэтому так же нуждается в тестировании;
9. корректно добавление фотографии так же является нужной функцией, поэтому так же нуждается в тестировании;
10. тест необходим для проверки обработчика замены фотографии, так как прерывание добавление фотографии на элемент может привезти к неожиданному результату от программы.

Интеграционное тестирование дало понять, что различные модули, прошедшие модульное тестирование, могут работать вместе без ошибок.

## Вывод по технологическому разделу

В технологическом разделе проведена отладка ПМ ВКМ при помощи двух инструментов, а именно отладчика Eclipse Debugger и журналирования Log4j.

Проведен анализ и выбор методов и средств тестирования, проведены модульное и интеграционное тестирование ПМ ВКМ, используя метод белого ящика, а также детально описаны их сценарии.

# ЗАКЛЮЧЕНИЕ

Во время ВКР изучена и проанализирована предметная область, проведен обзор и сравнение текущего и аналогичных существующих решений, выбраны язык и среда программирования, разработана схема данных и схема алгоритма ПМ ВКМ, спроектирован ПМ ВКМ и разработана экранная форма для него, а также проведены процессы отладки и тестирования ПМ ВКМ.

Результатом выпускной квалификационной работы является рабочая версия программного модуля визуализации конфигурирования микросхем.

ПМ ВКМ позволяет создавать различные схемы конфигурирования микросхем для их дальнейшего программирования, используя при этом графическую оболочку.

ПМ ВКМ является актуальным для предприятия НПК "Технологический центр", так как позволяет совершать процесс создания схем конфигурирования микросхем, разработанных на этом же предприятии.

В рамках выпускной квалификационной работы решены следующие задачи:

* исследована предметная область;
* выполнен сравнительный анализ существующих программных решений;
* выбраны язык и среда разработки;
* разработана схема данных ПМ ВКМ;
* разработана схема алгоритма ПМ ВКМ;
* осуществлена программная реализация ПМ ВКМ;
* разработан пользовательский интерфейс;
* проведена отладка и тестирование ПМ ВКМ;
* разработано руководство оператора ПМ ВКМ.

Работа завершена, ПМ ВКМ используется на предприятии НПК "Технологический центр" для конфигурирования микросхем, которые там же и разрабатываются.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Гагарина Л.Г., Касимов Р.А., Коваленко Д.Г., Федотова Е.Л, Чжо Зо Е, Черников Б.В. Методические указания по подготовке выпускной квалификационной работы по направлению подготовки бакалавров 09.03.04 «Программная инженерия»/ Под редакцией Б.В. Черникова; М., МИЭТ, 2016 г., 20 с.
2. НПК "Технологический центр" // НАЗНАЧЕНИЕ САПР "КОВЧЕГ" URL: http://www.asic.ru/index.php/sapr/naznach
3. Veripool // Verilator Intro URL:

https://www.veripool.org/projects/verilator/wiki/Intro

1. Акчурин А.Д., Юсупов К.М., Колчев А.А. ОСНОВЫ РАБОТЫ В СРЕДЕ QUARTUS II. – Казань: КФУ, 2017. – 49 с.
2. Intel® Quartus® Prime Design Software // Различия между вариантами Quartus URL:

https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/po/ss-quartus-comparison.pdf

1. ПЛИС Altera // Сборка схемы и загрузка конфигурации в Quartus II URL: https://robotclass.ru/articles/fpga-quartus-programmer/
2. Phyton // Программирование микроконтроллеров URL:

https://www.phyton.ru/programmers/microcontrollers-programming-for-beginners

1. Oracle // Язык программирования Java URL:

https://www.oracle.com/ru/java/

1. Isocpp // Язык программирования C++ URL:

https://isocpp.org/

1. Microsoft // Язык программирования C# URL:

https://docs.microsoft.com/ru-ru/dotnet/csharp/

1. Гвидо ван Россум // Язык программирования Python URL:

https://www.python.org/

1. Eclipse Foundation// Среда разработки Eclipse URL:

https://www.eclipse.org/

1. JetBrains // Среда разработки IntelliJ IDEA URL:

https://www.jetbrains.com/ru-ru/idea/

1. Apache // NetBeans URL:

https://netbeans.apache.org/

1. Apple Developer // Среда разработки Xcode 2019 URL:

https://developer.apple.com/xcode/

1. GitHub Inc. // Среда разработки Atom 2019 URL:

https://atom.io/

1. Дуглас Крокфорд // JSON URL:

https://www.json.org/json-ru.html

1. Indicator // Микросхема URL:

https://indicator.ru/label/mikroshema

1. URL:

https://radiostorage.net/538-chto-takoe-integralnaya-mikroskhema-ims.html

1. Electrik Info // Виды современных интегральных микросхем URL:

http://electrik.info/main/praktika/1598-vidy-sovremennyh-integralnyh-mikroshem-tipy-logiki-korpusa.html

1. Полная энциклопедия. Наука и техника // Аналоговые и цифровые микросхемы URL:

https://www.polnaja-jenciklopedija.ru/nauka-i-tehnika/analogovye-i-tsifrovye-mikroshemy.html

1. Большая энциклопедия Нефти и Газа // URL:

https://www.ngpedia.ru/id156558p1.html

1. Свистова Т.В. Основы микроэлектроники: учеб. пособие [Электронный ресурс].
2. Helpiks.org // Типы интегральных схем URL:

https://helpiks.org/7-5762.html

1. Микушин А.В. // Что такое цифровые микросхемы. Виды цифровых микросхем URL:

https://digteh.ru/digital/digital/

1. Программируемые логические интегральные схемы (ПЛИС) URL: https://portal.tpu.ru/SHARED/g/GREBENNIKOVVV/students/Tab4/Tab/11\_Electronics\_22\_PLD\_2\_2016.pdf
2. Микушин А.В. // Программируемые логические интегральные схемы (ПЛИС) URL:

https://digteh.ru/digital/PLD/

1. Parallel.ru // Технология устройств CPLD URL:

https://parallel.ru/fpga/cpld.html

1. FPGA Ignition // Обзор технологии FPGA URL:

https://fpgaignition.by/technology/fpga-technology-overview

1. Intel Glossary // Passive Parallel Asynchronous configuration scheme (PPA) Definition URL:

https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/reference/glossary/def\_parasynch.htm

1. Р.А. Мяльк // Конфигурирование ПЛИС Altera со статической памятью конфигурации URL:

https://serurvis.3dn.ru/\_Altera\_.pdf

1. GitHub // GHDL URL:

http://ghdl.free.fr/

1. Intel // ModelSim URL:

https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/model-sim.html

1. Xilinx // ISim URL:

https://www.xilinx.com/products/design-tools/isim.html

1. Xilinx // ISim User Guide URL:

https://www.xilinx.com/support/documentation/sw\_manuals/xilinx14\_1/plugin\_ism.pdf

1. ruby-lang.org // Язык программирования Ruby URL:

https://www.ruby-lang.org/ru/

1. bluej.org // BlueJ URL:

https://www.bluej.org/

1. Apache.org // Log Levels

https://logging.apache.org/log4j/2.x/manual/customloglevels.html

1. Степанченко И. В. МЕТОДЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ: Учеб. пособие / ВолгГТУ, Волгоград, 2006. – 74 с.
2. Святослав Куликов Тестирование программного обеспечения. Базовый курс. © EPAM Systems, 2015–2021

ПРИЛОЖЕНИЕ 1

Программный модуль визуализации

конфигурирования микросхемы (ПМ ВКМ)

Руководство оператора

Москва, 2021

АННОТАЦИЯ

В данном программном документе приведено руководство оператора по использованию ПМ ВКМ, предназначенного для конфигурирования микросхем, посредством создания схем конфигурирования.

В разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы.

В разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск и выполнение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузки и управляет выполнением программы, а также ответы программы на эти команды.

В разделе «Сообщения оператору» приведены тексты сообщений, выдаваемых в ходе выполнения программы, описание их содержания и соответствующие действия оператора.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77, ГОСТ 19.103-77, ГОСТ 19.104-78, ГОСТ 19.105-78, ГОСТ 19.106-78, ГОСТ 19.505-79).

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ 72](#_Toc75432692)

[1 НАЗНАЧЕНИЕ ПРОГРАММЫ 73](#_Toc75432693)

[1.1 Функциональное назначение 73](#_Toc75432694)

[1.2 Эксплуатационное назначение 73](#_Toc75432695)

[1.3. Основные функциональные модули 73](#_Toc75432696)

[2 УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ 74](#_Toc75432697)

[2.1 Требования к составу и параметрам технических средств 74](#_Toc75432698)

[3 ВЫПОЛНЕНИЕ ПРОГРАММЫ 74](#_Toc75432699)

[3.1 Установка и запуск программы 74](#_Toc75432700)

[3.2. Выполнение программы 75](#_Toc75432701)

[3.3. Экранные формы 75](#_Toc75432702)

[4 СООБЩЕНИЯ ОПЕРАТОРУ 77](#_Toc75432703)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

ИМС – интегральная микросхема

ОС – операционная система

ПМ – программный модуль

ПМ КВМ – программный модуль визуализации конфигурирования микросхем

ПО – программное обеспечение

ЭВМ – электронно-вычислительная машина

JVM (Java Virtual Machine) – виртуальная машинная языка программирования Java

1 НАЗНАЧЕНИЕ ПРОГРАММЫ

1.1 Функциональное назначение

Функциональным назначением ПМ ВКМ является конфигурирование микросхем с помощью создания и редактирования различных схем конфигурирований.

ПМ ВКМ обладает следующем функционалом:

* создание пользовательских и библиотечных элементов (Node) на поле, предназначенном для создания и редактирования схем конфигурирования;
* редактирование каждого из элементов на поле;
* соединение элементов на поле различными способами с помощью соединений (Connection);
* создание и редактирование схем конфигурирования;
* создание конфиг-файлов для каждого элемента на поле и для схемы в целом;
* откат на действие вперед и назад;
* считывание информации из конфиг-файлов элементов и схем;
* процесс конфигурирования.

1.2 Эксплуатационное назначение

ПМ ВКМ может быть использовано на промышленных предприятиях, где необходимо запрограммировать ИМС. Конечными пользователями могут являться сотрудники предприятия, которые имеют знания и навыки в конфигурировании микросхем, а также в ОС Windows или Linux.

1.3. Основные функциональные модули

Рассмотрим назначение основных функциональных модулей, на которые можно разбить ПМ ВКМ. ПМ можно разбить на два основных модуля:

* Модуль «Инструменты» – предоставляет из себя набор инструментов для отката на действие назад, отката на действие вперед и запуска процесса конфигурирования ИМС.
* Модуль «Поле» – предоставляет поле для создания и редактирования различных схем конфигурирования. Предоставляет пользователю переносить, изменять, соединять различные элемtнты на нем, называемые нодами (Nodes).

2 УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

2.1 Требования к составу и параметрам технических средств

В состав технических средств должны входить ЭВМ поду управлением ОС Windows или Linux.

Минимальный состав требований к ЭВМ:

* установленная и работающая JVM;
* оперативная память DDR 1 Гб;
* жесткий диск 1 Гб;
* графический адаптер;
* USB-вход;
* клавиатура, мышь;
* монитор с разрешением 800x600(для обеспечения комфортности восприятия).

Рекомендуемый состав требований к ЭВМ:

* установленная и работающая JVM;
* оперативная память DDR 2 Гб;
* жесткий диск 10 Гб;
* графический адаптер;
* USB-вход;
* клавиатура, мышь;
* конфигурируемая микросхема;
* монитор с разрешением 800x600(для обеспечения комфортности восприятия).

3 ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 Установка и запуск программы

Для установки ПМ ВКМ необходимо загрузить в память ЭВМ последнюю рабочую версию приложения.

Для запуска ПМ ВКМ необходимо найти исполняемый файл ПМ, либо найти иконку ПМ и открыть ее, двойным нажатием на мышь.

3.2. Выполнение программы

Выполнение работы с ПМ ВКМ осуществляется в несколько этапов:

1. необходимо загрузить файлы библиотек элементов в память ЭВМ;
2. открыть уже существующую, либо создать новую схему для конфигурирования ИМС;
3. перенести все необходимые элементы на поле;
4. соединить их так, как пользователь считает нужным;
5. подсоединить в USB-вход микросхему, которую необходимо запрограммировать;
6. начать процесс конфигурирования ИМС.

3.3. Экранные формы

На рис. 3.1 представлена экранная форма Главного окна ПМ ВКМ.

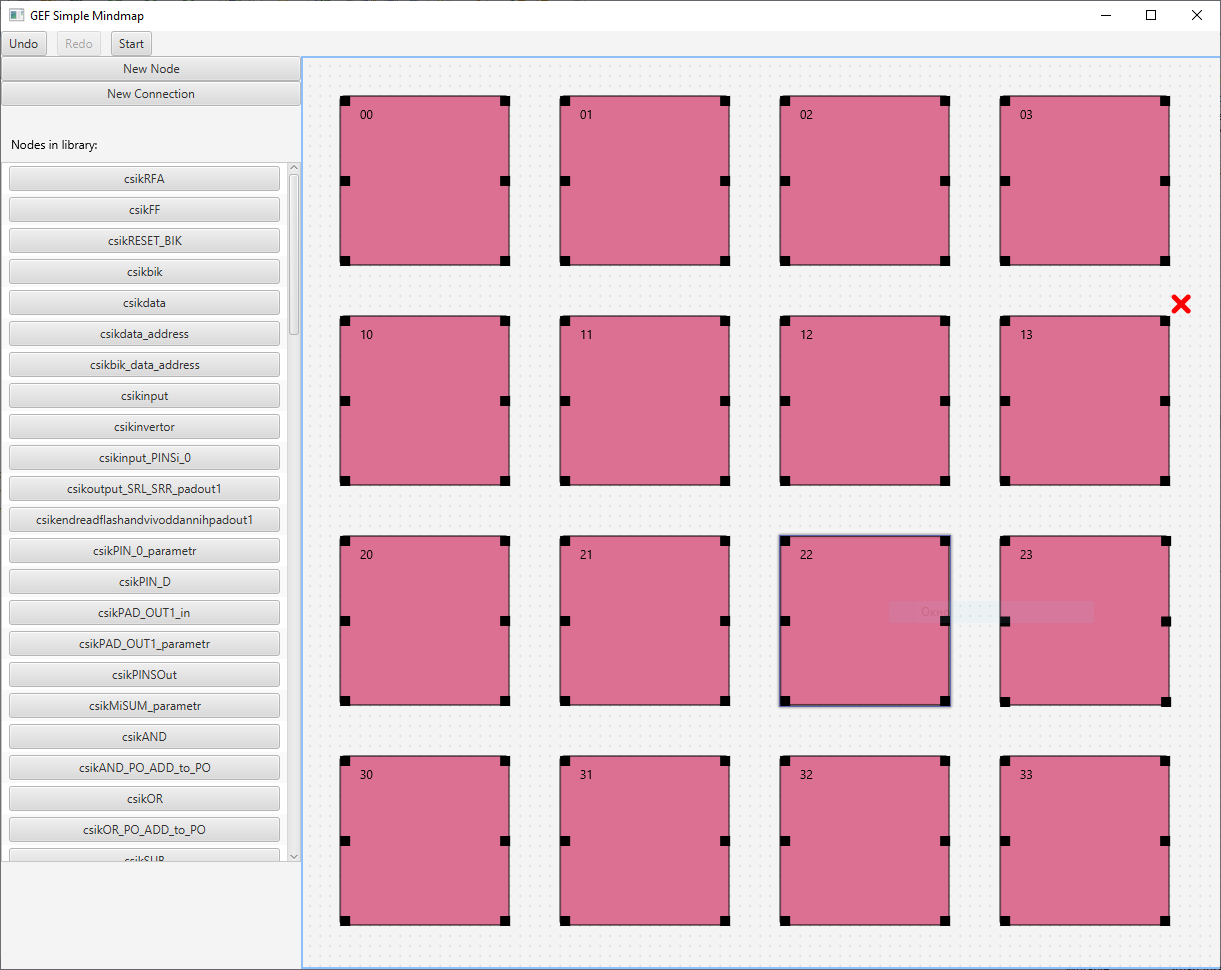


Рис. 3.1. Экранная форма Главного окна ПМ ВКМ

В правой части экранной формы располагается модуль с полем, предназначенным для создания и редактирования схемы конфигурирования с помощью создания, редактирования и перемещения элементов схемы, а также созданий соединения между ними.

В левой части экранной формы располагается модуль с панелью инструментов содержит в себе загруженные из библиотеки элементы, необходимые для схем конфигурирования. Также имеются кнопки для отката действий назад (Undo) и вперед (Redo), создание элемента (New Node) и соединения (New Connection), и кнопка (Start) для старта процесса конфигурирования ИМС. Откаты действий назад вперед и назад также может осуществляться нажатием сочетаний кнопок соответственно Ctrl+R и Crtl+Z, создание элемента – Ctrl+X, создание соединения – Ctrl+C.

4 СООБЩЕНИЯ ОПЕРАТОРУ

Информационное сообщение о загрузке элементов из библиотек «Libraries loaded (Библиотеки загружены)».

Данное сообщение информирует об успешной загрузке всех элементов из библиотек элементов. Информационное сообщение выводится в отдельном диалоговом окне, как показано на рис. 4.1.

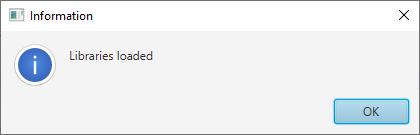


Рис. 4.1. Диалоговое окно информационного сообщения об успешной загрузке элементов из библиотек.

Информационное сообщение об ошибке загрузки элементов из библиотек «Libraries not loaded (Библиотеки не загружены)».

Данное сообщение информирует об ошибке, которая возникла при загрузке элементов из библиотек, элементы загружены не будут. Информационное сообщение выводится в отдельном диалоговом окне, как показано на рис. 4.2.

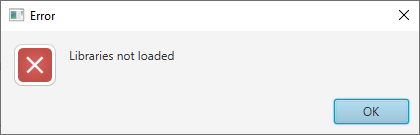


Рис. 4.2. Диалоговое окно информационного сообщения с ошибкой загрузки элементов из библиотек.

Информационное сообщение о пути, где находится конфиг-файл выбранного элемента «File this node locates in the folder: <path node> (Файл текущего элемента находится в папке по пути <путь до файла>)».

Данное сообщение появляется при выборе соответствующего пункта из контекстного меню выбранного элемента, а именно, Get code source. Сообщение показывает путь до файла с кодом выбранного элемента. Информационное сообщение выводится в отдельном диалоговом окне, как показано на рис. 4.3.

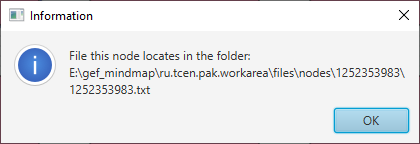


Рис. 4.3. Диалоговое окно с информацией о пути до файла с кодом выбранного элемента.

Информационное сообщение о загрузке элементов из библиотек «Libraries loaded (Библиотеки загружены)».

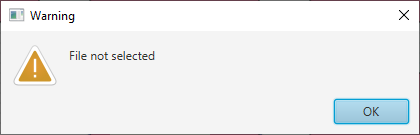
Для каждого элемента на поле можно заменить файл с фотографией, выбрав в контекстном меню пункт Change Image (заменить фотографию). Если файл по каким-либо причинам не будет выбран, то появится информационное сообщение с предупреждением, что файл не был выбран, как показано на рис. 4.4. 

Рис. 4.4. Диалоговое окно информационного сообщения с предупреждением, что файл не был выбран.

Информационное сообщение об ошибке подсоединения микросхемы к ЭВМ «The microcircuit is not connected (Микросхема не подсоединена)».

Данное сообщение с ошибкой возникнет при инициации процесса конфигурирования при неподключенной микросхеме. Информационное сообщение с ошибкой выводится в отдельном диалоговом окне, как показано на рис. 4.5.

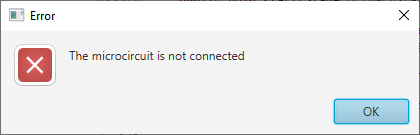


Рис. 4.5. Диалоговое окно информационного сообщения с ошибкой, что микросхема для конфигурирования не подсоединена к ЭВМ.

Информационное сообщение о программировании ИМС «The microcircuit has been programmed (Микросхема была запрограммирована)».

Данное сообщение информирует об успешном конфигурировании микросхемы схемой на поле ПМ ВКМ. Информационное сообщение выводится в отдельном диалоговом окне, как показано на рис. 4.6.

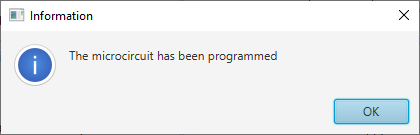


Рис. 4.6. Диалоговое окно информационного сообщения об успешном программировании ИМС.

ПРИЛОЖЕНИЕ 2

Программный модуль визуализации

конфигурирования микросхемы (ПМ ВКМ)

Техническое задание (ТЗ)

Москва, 2021

1. Введение

На предприятии НПК "Технологический центр" была разработана микросхема. Сразу же возник вопрос, каким образом пользователь будет осуществлять ее программирование. Для того чтобы пользователь смог самостоятельно провести конфигурирование микросхемы, он должен изучить и понимать все тонкости работы по ее настройке и программированию.

Проанализировав текущее решение, и сравнив существующие аналоги, пришли к тому, что все они не подходят и не удовлетворяют запросам компании по реализации этой задачи, поэтому было решено создавать новое программное средство.

Это ПС должно повысить эффективность конфигурирования микросхемы, а также позволить пользователю самому конфигурироваться микросхему так, как он захочет с помощью графической оболочки.

1. Основания для разработки
   1. Основание для разработки

* Задание на выпускную квалификационную работу;
* Решение директора института СПИНТех.
  1. Наименование разработки

«Разработка программного модуля визуализации конфигурирования микросхем». Программному модулю, разрабатываемому по настоящему ТЗ, присваивается шифр «ПМ ВКМ» (Далее в тексте - ПМ).

* 1. Исполнитель

Исполнителем является студент группы ПИН-41 НИУ МИЭТ Зотов Владислав Дмитриевич.

1. Назначение разработки

ПМ создается с целью уменьшения времени процесс конфигурирования микросхем.

1. Технические требования
   1. Требования к функциональным характеристикам
      1. Программный модуль должен обеспечивать возможность выполнения следующих функций:

* Считывание входных данных:
  + - Считывание данных из библиотек элементов для конфигурирования
    - Считывание данных схем конфигурирования
* Сохранение данных:
  + - Сохранение данных о каждом элементе схемы
    - Сохранение схем конфигурирования
* Изменение данных по каждому пункту элемента схемы
* Осуществление команды назад и вперед (Undo, Redo)
* Проверка на правильность составления схемы конфигурации
* Формирование байт-кода из схемы
* Загрузка (программирование) микросхемы
* Оповещение о результатах действий пользователя
  + 1. Организация входных и выходных данных

Входные данные:

В качестве входных данных для ПМ выступают данные из библиотек элементов схем и самих схем.

Выходные данные:

Выходными данными являются:

* Оповещение о результатах действий пользователя
  1. Требования к надежности

Работа ПМ не должна приводить к сбоям, ошибкам, либо зависаниям операционной системы, а также самой конфигурируемой микросхемы. ПМ должен работать с входными данными, предусмотренными техническими требованиями в соответствии с алгоритмом функционирования, обрабатывать и сообщать об ошибках при неверном задании входных данных и прочих нештатных ситуациях.

* 1. Условия эксплуатации

Пользователь, осуществляющий работу по конфигурированию микросхемы, должен быть ознакомлен с документацией компании НПК "Технологический центр. Также оператор должен обладать навыками работы в ОС Linux/Windows.

* 1. Требования к составу и параметрам технических средств

В состав технических средств должны входить, как минимум:

* ПК на базе ОС Linux/Windows с работающей JVM:
  + Оперативную память DDR не менее 1 Гб;
  + Жесткий диск 1 Гб;
  + Графический адаптер;
  + USB-вход;
  + Клавиатура, мышь.
* Конфигурируемая микросхема.
  1. Требования к информационной и программной совместимости

Базовый язык программирования – Java, среда разработки – Eclipse IDE. ПМ должен работать под ОС Windows и Linux.

* 1. Специальные требования

Специальных требований к характеристикам программы не предъявляется.

1. Требования к программной документации
   1. Требования к составу программной документации

В комплект документации должны входить: руководство пользователя.

* 1. Требования к оформлению документации

Программная документация должна быть разработана и оформлена в соответствии с ЕСПД.

1. Порядок контроля и приёмки

Контроль и приёмка разработки осуществляется на ЭВМ Заказчика на основе испытаний контрольно-отладочных примеров. При этом проверяется выполнение всех функций ПМ.

1. Стадии и этапы разработки

В течении периода с февраля 2021 года по июнь 2021 года должны быть проведены работы, указанные в Таблице 1.

Таблица 1

Стадии и этапы разработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *№* | *Наименование работы* | *Дата начала* | *Дата окончания* | *Форма отчетности* |
| *1* | *Постановка задачи* | *10.02.2021* | *15.02.2021* | *Слайд презентации* |
| *2* | *Изучение предметной области* | *15.02.2021* | *20.02.2021* | *Слайд презентации* |
| *3* | *Разработка ТЗ* | *20.02.2021* | *18.03.2021* | *Документ ТЗ* |
| *4* | *Выбор среды разработки и языка программирования* | *20.02.2021* | *25.02.2021* | *Текстовый документ* |
| *5* | *Разработка структуры данных* | *25.02.2021* | *01.03.2021* | *Схема данных* |
| *6* | *Разработка алгоритма для решения задачи* | *01.03.2021* | *07.03.2021* | *Схема алгоритма* |
| *7* | *Уточнение структуры данных и алгоритма решения задачи* | *07.03.2021* | *18.03.2021* | *Уточненная схема данных* |
| *8* | *Программирование структур данных* | *18.03.2021* | *30.03.2021* | *Модуль структуры данных* |
| *9* | *Программирование алгоритма* | *30.03.2021* | *20.04.2021* | *Тексты программ с комментариями* |
| *10* | *Тестирование и отладка* | *20.04.2021* | *01.05.2021* | *Тест кейсы* |
| *11* | *Проведение испытаний* | *01.05.2021* | *10.05.2021* | *Испытания* |
| *12* | *Разработка программных документов* | *10.05.2021* | *15.05.2021* | *Результаты испытаний* |
| *13* | *Корректировка программы и документов, подготовка программы для передачи* | *16.05.2021* | *20.05.2021* | *Программа и программные документы* |
| *15* | *Передача программы и документов* | *20.05.2021* | *31.05.2021* | *Документ о передачи программы и документов* |