

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Инструментальные средства крупногранулярных реконфигурируемых вычислителей
для системной динамики

Обучающийся / Student Гогиян Артур Гаикович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group P42171
Направление подготовки/ Subject area 09.04.04 Программная инженерия
Образовательная программа / Educational program Технологии промышленного программирования 2020
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Магистр
Руководитель ВКР/ Thesis supervisor Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

Документ подписан	
Гогиян Артур Гаикович	
31.05.2022	

(эл. подпись/ signature)

Гогиян Артур
Гаикович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
30.05.2022	

(эл. подпись/ signature)

Пенской
Александр
Владимирович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Гогиян Артур Гаикович

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42171

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Технологии промышленного программирования 2020

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Тема ВКР/ Thesis topic Инструментальные средства крупногранулярных реконфигурируемых вычислителей для системной динамики

Руководитель ВКР/ Thesis supervisor Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Создание средства проектирования специализированных вычислителей для расчета моделей системной динамики.


Задачи, решаемые в ВКР / Research tasks

1. Анализ возможности генерации графа потока данных для моделей системной динамики.
2. Сравнительный анализ популярных в сообществе системной динамики форматов моделей системной динамики.
3. Составление этапов поддержки выбранного формата моделей системной динамики в САПР NITTA.
4. Проектирование и реализация модуля анализа и трансляции моделей системной динамики для выбранного формата моделей системной динамики.
5. Тестирование разработанного модуля.

Краткая характеристика полученных результатов / Short summary of results/findings

Спроектирована и реализована архитектура модуля генерации графа потока данных САПР NITTA на основе моделей системной динамики, заданных в формате XMILE. Проведена верификация разработанного модуля, а также приведены и проанализированы результаты симуляции вычислителей, спроектированных системой NITTA на основе моделей системной динамики. Построен план дальнейших работ САПР NITTA для полной поддержки стандарта XMILE.

Обучающийся/Student


Документ подписан	
Гогиян Артур Гаикович	
31.05.2022	

(эл. подпись/ signature)

Гогиян Артур
Гаикович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
30.05.2022	

(эл. подпись/ signature)

Пенской
Александр
Владимирович

(Фамилия И.О./ name
and surname)

ОГЛАВЛЕНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	10
ВВЕДЕНИЕ.....	11
1.ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ.....	15
1.1.Системная динамика.....	15
1.1.1.Формат DYNAMO.....	19
1.1.2.Формат Vensim.....	19
1.1.3.Формат XMI LE.....	20
1.1.4.Сравнение форматов моделей системной динамики.....	22
1.2.Специализированные вычислители и их разновидности.....	25
1.2.1.Специализированные интегральные схемы.....	27
1.2.2.ПЛИС.....	29
1.3.Крупногранулярные реконфигурируемые архитектуры.....	31
1.3.1.Архитектура CGRA.....	31
1.3.2.Граф потока данных.....	33
1.4.Заключение.....	35
2.ПРИМЕНЕНИЕ САПР ПИТА ДЛЯ ЗАДАЧ СИСТЕМНОЙ ДИНАМИКИ.....	36

2.1.Описание САПР NITTA	37
2.2.Формирование графа потока данных в САПР NITTA	38
2.3.Соответствие САПР NITTA требованиям для расчетов моделей системной динамики.....	41
2.4.Заключение	43
3.РАЗРАБОТКА ПЛАНА ПОДДЕРЖКИ МОДЕЛЕЙ СИСТЕМНОЙ ДИНАМИКИ В СИСТЕМЕ NITTA	44
3.1.Выбор формата моделей системной динамики.....	44
3.2.Разработка плана поддержки формата XMI LE в САПР NITTA.	46
3.3.Заключение	47
4.ПРОЕКТИРОВАНИЕ МОДУЛЯ ПОДДЕРЖКИ СИСТЕМНОЙ ДИНАМИКИ В САПР NITTA	48
4.1.Этап 1 - сбор и анализ требований	48
4.2.Этап 2 – анализ точек расширения САПР NITTA	50
4.3.Этап 3 - проектирование архитектуры модуля	51
4.4.Этап 4 - реализация исполняемой базовой архитектуры	54
4.4.1.Тестирование	59
4.4.2.Результаты	60
4.5.Этап 5 - реализация всей функциональности, заложенной в формат XMI LE и поддерживаемой системой NITTA.....	61

4.6.Этап 6 - составление плана по расширению функциональности системы NITTA.....	65
4.7.Заключение	68
ЗАКЛЮЧЕНИЕ.....	69
СПИСОК ЛИТЕРАТУРЫ.....	71

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

CGRA - Coarse-Grained Reconfigurable Arrays

FPGA - Field-Programmable Gate Arrays

CPU - Central Processing Unit

GPU - Graphics Processing Unit

ASIC - Application-Specific Integrated Circuit

XMILE - XML Interchange Language for System Dynamics

XML - eXtensible Markup Language

САПР - Система Автоматизированного Проектирования

ПО - Программное Обеспечение

LUT – LookUp Table

SRAM – Static Random Access Memory

ПЛИС – Программируемая Логическая Интегральная Схема

ВВЕДЕНИЕ

Микропроцессоры используются для решения задач по обработке данных уже более 60 лет. Все это время их производительность непрерывно росла - увеличивалось количество транзисторов на кристалле процессора, размер этих транзисторов уменьшался, скорость обработки данных увеличивалась за счет внедрения конвейерной обработки, параллельной обработки независимых данных разного типа (суперскалярные архитектуры [21]). В 1965 году Гордон Мур, один из основателей компании Intel, выявил эмпирическую закономерность, согласно которой количество транзисторов, которое возможно уместить на интегральной схеме, удваивается каждые 18 месяцев. Согласно данной закономерности, производительность микропроцессоров увеличивалась экспоненциально с течением времени. Однако, в начале нулевых данная эмпирическая закономерность перестала отображать действительность. Размеры транзисторов стали настолько маленькими, что дальнейшее их уменьшение ограничивается законами физики. Дальнейший прирост производительности обеспечивался в основном за счет параллелизации обработки данных [22]. Однако, увеличение числа вычислителей не дает линейного прироста производительности. В соответствии с законом Амдала, в случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента. Соответственно, даже имея бесконечное количество вычислителей, общее время вычисления в общем случае ограничено снизу.

Решением описанных выше проблем могут стать специализированные вычислители. В сравнении со специализированными вычислителями, процессоры общего назначения обладают рядом недостатков. Рассмотрим основные из них.

- Ограниченный набор команд. Микроархитектура вычислителей общего назначения позволяет выполнять ограниченный, заранее определенный спектр операций. Комбинирование этих операций определенным образом позволяет решить произвольную поставленную задачу. Минусом такого подхода является необходимость разбивать сложные задачи на более простые, поддерживаемые процессором, и выполнять их последовательно. Специализированные вычислители, напротив, могут выполнять операции высокой сложности атомарно за счет реализации их на аппаратном уровне, что существенно увеличивает производительность вычислителя.
- Высокое энергопотребление. Современные вычислители общего назначения обладают большим количеством различных логических блоков, часть которых может вообще не использоваться для решения конкретной задачи. Специализированные вычислители, напротив, позволяют минимизировать количество используемых логических блоков.
- Накладные расходы на реализацию архитектуры набора команд. Большинство процессоров общего назначения реализуют один из распространенных наборов команд, что позволяет разработчикам программного обеспечения абстрагироваться от его внутреннего устройства. Однако, это создает лишний уровень абстракции, поддержка которого влечет за собой дополнительные накладные расходы.

- Дороговизна производства. Простота микроархитектуры специализированного вычислителя (в сравнении с вычислителями общего назначения) позволяет уменьшить себестоимость производства одного такого вычислителя, при значительных количествах производимых копий.

Однако, специализированные вычислители, создаваемые для решения конкретной, заранее известной задачи, также обладают недостатками, в первую очередь - невозможностью переиспользования для решения других задач. Компромиссным решением могут быть программируемые логические интегральные схемы, такие, как FPGA и CGRA. Использование таких архитектур вычислителей позволяет найти необходимый баланс между энергопотреблением, быстродействием, простотой программирования и возможностью переиспользования для различных задач.

Целью исследования является создание средства проектирования специализированных вычислителей для расчета моделей системной динамики.

Объектом исследования в данной работе является система автоматизированного проектирования NITTA.

Предметом исследования является процесс генерации графа, описывающего потоки данных, для алгоритма высокого уровня.

В ходе данной работы планируется решение следующих задач.

- Проведение анализа возможности генерации графа потока данных для моделей системной динамики.

- Проведение сравнительного анализа популярных в сообществе системной динамики форматов хранения и распространения моделей системной динамики. Выбор формата для реализации поддержки в САПР NITTA.
- Составление плана этапов поддержки выбранного формата в САПР NITTA.
- Проектирование и реализация модуля анализа и трансляции модели системной динамики для выбранного формата.
- Тестирование разработанного модуля.

В качестве метода разработки программного обеспечения для реализации поставленных задач будет использована инкрементная модель разработки ПО.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Системная динамика

Системная динамика — область моделирования систем, обладающих сложным внутренним состоянием, исследующая поведение таких систем во времени [3]. Целями системной динамики является изучение причинно-следственных связей, задержек реакции изменения одних элементов системы на изменения других, петель обратных связей [25].

Модель системной динамики состоит из следующих составляющих.

- Запасы (Stocks) — элементы, описывающие одну из метрик, влияющих на систему. С течением времени количественная оценка метрики растет или убывает в соответствии со входящими в нее потоками, увеличивающими ее значение, и исходящими потоками, уменьшающими его.
- Потоки (Flows) — описывают изменение метрики запасов. Задаются уравнением, вычисляемым на каждом шаге моделирования.
- Уравнения — описывают конкретный процесс, происходящий внутри системы. Задаются в виде комбинации математических операторов с заданными приоритетами.

Несмотря на то что системы, описываемые системной динамикой, обычно непрерывны во времени, моделируются они обычно дискретными методами. Происходит это в виду сложности моделирования системы как непрерывного процесса.

Непрерывность на практике заменяется дискретизацией с заданной, необходимой частотой [24]. Это позволяет рассматривать непрерывный процесс как последовательность снимков состояния системы, сделанных через равные промежутки времени. На каждом снимке происходит вычисление всех запасов системы, вычитая из них исходящие потоки и прибавляя входящие. Анализ такой последовательности дает возможность не только рассчитать значение запасов системы через определенный промежуток времени, но и изучить причины роста или уменьшения конкретного запаса в течении времени.

Часто системы, для моделирования которых используется системная динамика, имеют циклические зависимости: значение конкретного запаса системы часто влияет на значение этого же запаса в будущем. Подобное покадровое моделирование позволяет изучить структуру таких циклов, рассчитать задержку реакции в цикле, степень влияния запаса на свое значение в будущем.

В качестве примера модели системной динамики рассмотрим модель, отражающую поведение стакана горячего чая, оставленного в комнате (Схема 1).

Единственным запасом, существующим в данной системе, является температура стакана. Входных потоков для него не определено, что означает что увеличиваться данный запас с течением времени не может. Однако, определен исходящий поток, что означает что температура стакана будет уменьшаться с течением времени.

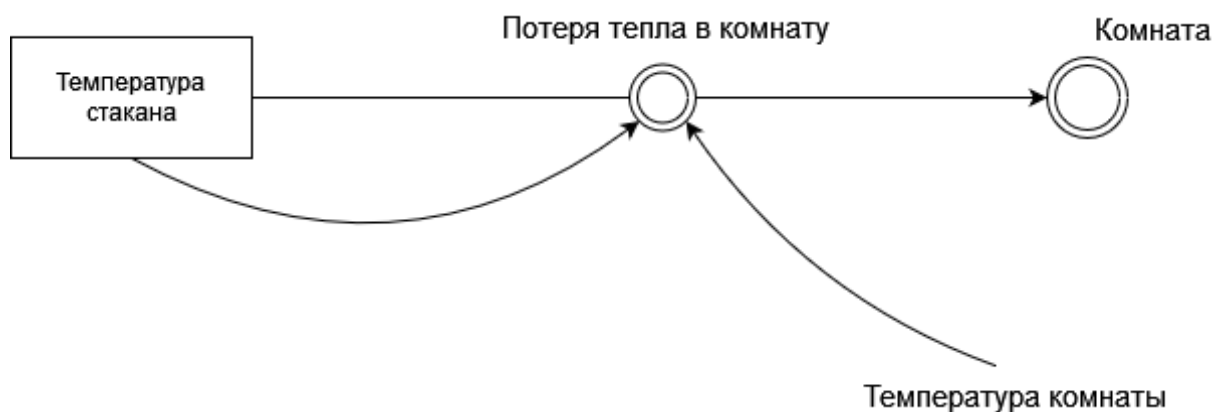


Схема 1 - модель охлаждения стакана чая в комнате

Также, в системе определен один поток, описывающий потери тепла стакана в комнату. Стрелки зависимостей показывают, что размер потока зависит от 2-х параметров — температуры стакана и температуры комнаты, в которой данный стакан находится. Допущением в данной модели является константная температура в комнате: с точки зрения законов термодинамики при охлаждении стакана энергия, потерянная им, должна рассеяться в комнате, увеличив тем самым температуру в ней. Однако, в данном случае изменение температуры комнаты будет незначительным, а учет такого изменения сильно усложнит модель. Поэтому, температура комнаты в данной модели рассматривается как константа.

График 1 отражает результат моделирования описанной выше модели.

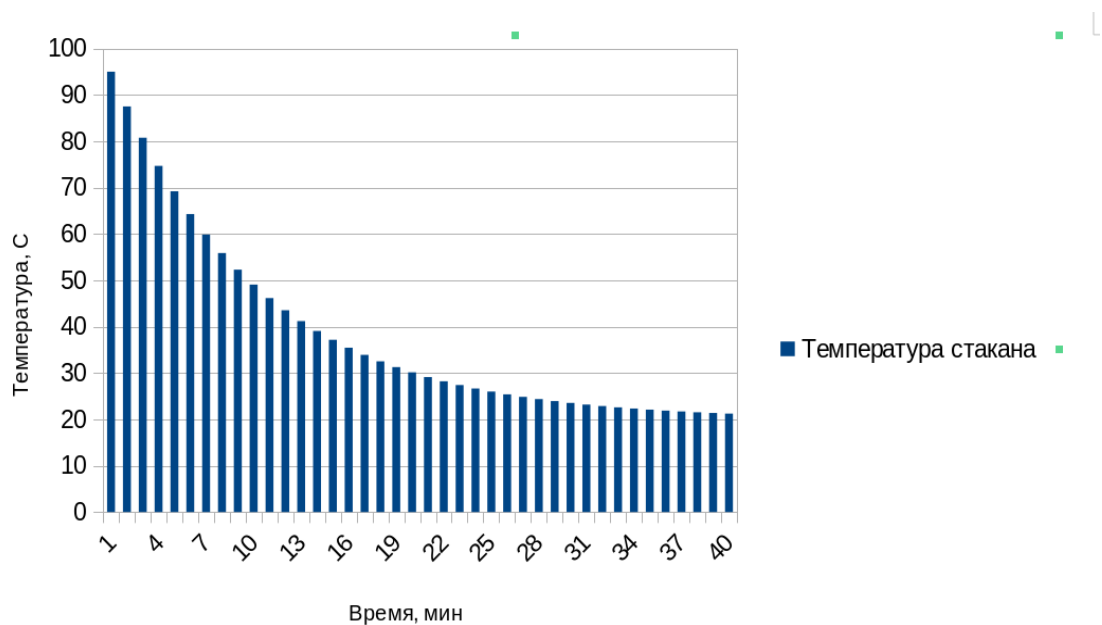


График 1 - изменение температуры стакана во времени

Как видно из графика 1, изменение температуры стакана происходит по зависимости, близкой к обратной экспоненциальной — охлаждение замедляется при приближении температуры стакана к температуре комнаты.

Существуют различные форматы хранения и распространения моделей системной динамики. В разделах 1.1.1 – 1.1.3 рассматриваются форматы DYNAMO, Vensim и XMILE, как наиболее популярные в сообществе.

1.1.1. Формат DYNAMO

Язык программирования DYNAMO представляет собой инструмент описания и симуляции моделей системной динамики. Язык DYNAMO обладает следующими преимуществами:

- простота использования для сообщества специалистов по моделированию промышленной динамики;
- немедленное выполнение скомпилированной модели без создания промежуточного объектного файла;
- обеспечение графического вывода с помощью принтера.

DYNAMO разрабатывался в 50-х годах 20 века. На данный момент данный формат распространения системной динамики является устаревшим и менее популярен в сообществе.

1.1.2. Формат Vensim

Vensim — это инструмент визуального моделирования, позволяющий документировать, моделировать, анализировать и оптимизировать модели системной динамики. Основным форматом хранения и распространения моделей, созданных в vensim, является формат mdl.

Формат mdl является иерархическим и хранит данные в текстовом виде. В общем случае формат представляет из себя набор пар из ключевых слов, являющихся идентификаторами свойства системы, и блока данных, хранящего параметры и их значения для данного свойства.

Модели, созданные инструментом Vensim и распространяемые в формате mdl хранят в себе следующие блоки.

- Уравнения. Задаются набором математических выражений. В файле располагаются в произвольном порядке.
- Макросы. Позволяют объединить блок уравнений в единую сущность для переиспользования в нескольких местах без дублирования уравнений.
- Настройки отображения. Визуальные составляющие отображения модели в графическом интерфейсе Vensim.
- Группировки. Позволяют группировать параметры модели по произвольным критериям, что упрощает навигацию по модели.

1.1.3. Формат XMILE

XMILE — формат описания, хранения и передачи моделей системной динамики. Формат XMILE определяет следующие ключевые элементы.

- Stocks. Описывает конкретный запас моделируемой системы. Содержит численное значение, на каждом шаге итерации увеличиваемое входящими потоками и уменьшаемое исходящими потоками, если таковые имеются.

- Flows. Описывает изменения запасов. Задается уравнением, описывающим силу потока на конкретной итерации.
- Auxiliaries. Синтаксическое упрощение, позволяющее именовать какое-либо уравнение и обращаться к нему по указанному имени.
- Графические функции. Описывают графики, сохраненные в модель системной динамики для обозначения зависимостей между различными элементами системы.
- Уравнения. Описывают конкретный процесс, происходящий внутри системы. Задаются в виде комбинации математических операторов с заданными приоритетами.
- Спецификация симуляции. Данные, необходимые для запуска моделирования. Обычно задают период времени, который нужно моделировать, и частоту дискретизации.

Формат XMILE разработан и поддерживается некоммерческой организацией OSASIS, одной из областей деятельности которой является стандартизация технологий и протоколов. Данный формат имеет большое распространение в сообществе. Согласно OASIS[5], целями разработки формата XMILE было следующее.

- Возможность переиспользования моделей системной динамики с наборами больших данных для демонстрации того, как разные политики приводят к разным результатам в сложных средах.

- Возможность хранения моделей в облачных библиотеках, совместного использования внутри и между организациями, а также использования для передачи различных результатов с помощью общего словаря.
- Возможность переиспользования компонентов модели, в том числе для подключения к другим симуляциям.
- Создание онлайн-репозитория, моделирующих различные общие бизнес-решения.
- Повышение признания и использование системной динамики как дисциплины.
- Помощь независимым поставщикам программного обеспечения в создании новых инструментов, упрощающая для компаний разработку и понимание моделей и симуляции.
- Возможность разработки приложений, основанных на стандартах для новых рынков, таких как мобильные устройства и социальные сети.

1.1.4. Сравнение форматов моделей системной динамики

При выборе формата, поддержку которого планируется реализовать в вычислительной системе, необходимо руководствоваться различными факторами. В контексте САПР NITTA (более подробно описанной в главе 2) были выделены следующие критерии.

- Популярность в сообществе. Сообщество системной динамики (как и большинство других сообществ) достаточно инертно. Поэтому, популярность формата на текущий момент времени стоит рассматривать как важный фактор, влияющий на востребованность формата в будущем.
- Открытость. Форматы, распространяемые под открытыми лицензиями, более привлекательны в виду возможности их свободного изучения, имплементации, модификации и распространения.
- Простота реализации. Трудоемкость реализации формата также является важным критерием, влияющим на время и ресурсы, затрачиваемые на реализацию его поддержки.

Сравним рассмотренные в разделах 1.1.1 – 1.1.3 форматы по данным критериям. Результаты сравнения приведены в таблице 1.

Таблица 1 – сравнение форматов хранения и распространения моделей системной динамики

Критерий	DYNAMO	Vensim	XMILE
Популярность	Практически не используется, в современных системах обычно реализуется для чтения старых моделей.	Используется инструментом симуляции Vensim, имеет довольно большое сообщество.	Рекомендован и стандартизирован консорциумом OASIS, широко распространен и реализован в различных инструментах.
Открытость	Проприетарный формат	Открытый формат, реализованный на базе формата mdl.	Открытый формат, реализованный на базе формата xml.
Простота реализации	В виду закрытости формата дать оценку сложности его реализации не представляется возможным.	Формат mdl, лежащий в основе формата Vensim, имеет простую для анализа декларативную структуру.	Формат xml, лежащий в основе формата xml, имеет простую для анализа декларативную структуру. Благодаря популярности формата xml существует множество свободно распространяемых готовых анализаторов, которые можно использовать для анализа формата XMILE.

1.2. Специализированные вычислители и их разновидности

Специализированный вычислитель – вычислитель, спроектированный для решения конкретной, заранее определенной задачи. Благодаря ограниченной области применения с одной стороны и ее предопределенности с другой, при проектировании таких вычислителей возможна оптимизация количества различных логических блоков, а также выбор как сложности этих блоков на этапе проектирования, так и архитектуры вычислителя в целом.

Архитектура вычислителя – набор ключевых свойств и качеств, присущих данному вычислителю. Понятие архитектуры важно по нескольким причинам. Во-первых, данное понятие упрощает обмен знаниями о системе между специалистами – возможность описать множество свойств системы одним выражением сильно упрощает коммуникацию. Во-вторых, реализация семейства вычислителей в рамках одной архитектуры позволяет сгладить разницу между ними и упростить переход с одного на другой.

Под сложностью логического блока подразумевается сложность операций, выполняемых таким блоком. В общем случае, вычислитель, решающий какую-либо вычислительную задачу, состоит из аппаратной и программной составляющих. При этом граница между аппаратурой и программным обеспечением может проходить на разных уровнях.

С одной стороны, вычислитель может представлять из себя реализацию машины Тьюринга [8] – конечного автомата с ограниченным набором команд и бесконечной в обе стороны ленты, представляющей из себя набор инструкций и память. Машина Тьюринга обладает вычислительной полнотой – с помощью нее можно вычислить всё, что можно вычислить с помощью любой другой вычислительной машины, выполняющей вычисления с помощью последовательного исполнения операций. Таким образом, вся архитектурная сложность вычислительной системы в случае с реализацией ее на базе машины Тьюринга, ложится на программную составляющую.

С другой стороны, данная граница может быть сдвинута в сторону реализации всей логики вычислительной системы в аппаратуре. В таком случае программной составляющей может не быть вообще.

Большинство архитектур вычислителей общего назначения подразумевают набор универсальных, но достаточно примитивных команд. Построение более сложной логики в вычислителях общего назначения происходит благодаря надстроенным над аппаратурой нескольким уровням абстракций, что перекладывает сложность с архитектуры вычислительной системы в программное обеспечение.

Использование специализированных вычислителей позволяет сдвинуть грань между аппаратным и программным обеспечением в сторону реализации большей части логики в аппаратуре. Это уменьшает гибкость системы, но позволяет увеличить производительность и уменьшить энергопотребление за счет оптимизации неиспользуемых логических блоков. Также важным преимуществом специализированных вычислителей является возможность достижения в теории неограниченной параллелизации за счет реализации их на основе потоков данных, а не на последовательном исполнении инструкций как в вычислителях общего назначения. Подробнее потоки данных рассматриваются в пункте 1.3.2.

Рассмотрим основные разновидности архитектур специализированных вычислителей. В виду того что данная работа посвящена архитектуре CGRA, обзор этой архитектуры вынесен в отдельный раздел 1.3, и в этом разделе рассматриваться не будет.

1.2.1. Специализированные интегральные схемы

Специализированные интегральные схемы (англ. ASIC) это интегральные схемы, спроектированные для решения заранее определенной задачи. Такой вычислитель не может быть перепрограммирован после создания.

Несмотря на то что создание и проектирование таких вычислителей для каждой отдельно взятой задачи трудоемко, ASIC обладает следующими преимуществами.

- Создание специализированных логических блоков под нужды конкретной задачи. Реализация таких блоков может существенно повысить производительность вычислителя, позволяя выполнять нетривиальные операции атомарно.
- Уменьшение физических размеров схемы. Элементы можно разместить оптимальным образом, а их количество минимизировать на этапе дизайна схемы.
- Уменьшение энергопотребления. Причиной энергоэффективности таких схем является минимизация логических блоков, включенных в схему.

Специализированные интегральные схемы часто используются в качестве аппаратных ускорителей, как часть процессоров общего назначения. Часто их применяют для обучения глубоких нейронных сетей и обработки сигналов [9].

Существенным недостатком таких схем, помимо ограниченной области применения, является сложность их разработки, следствием чего является необходимость в достаточно высокой квалификации разработчика, позволяющей ему работать на низких уровнях абстракции к аппаратуре. Кроме того, при малых партиях себестоимость одного такого вычислителя может быть существенно выше стоимости универсального аналога, из-за чего использование ASIC обычно обоснованно только при крупных сериях производства.

1.2.2. ПЛИС

В отличие от ASIC, логика работы ПЛИС (англ. FPGA) закладывается не на этапе изготовления микросхемы, а задается путем программирования [12]. Основными составляющими FPGA являются программируемые логические блоки и программируемые соединения между ними. Упрощенно структура FPGA изображена на рисунке 1.

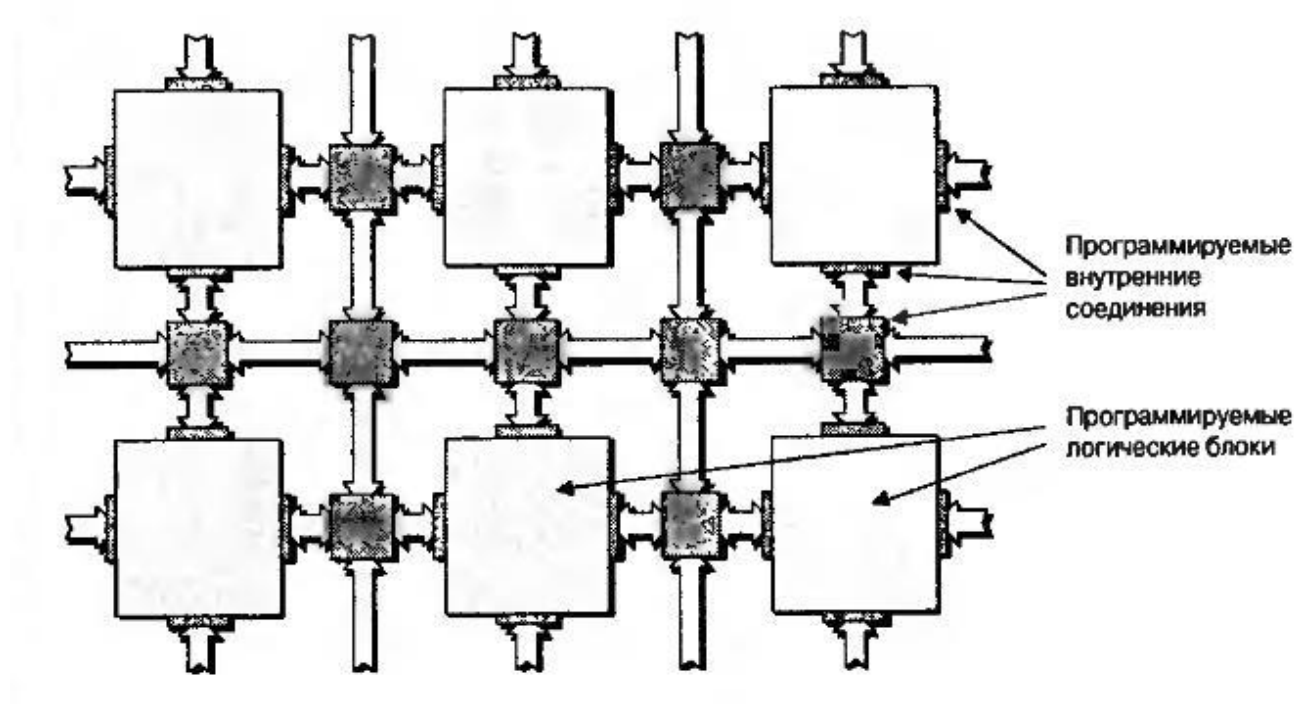


Рисунок 1 – Структура FPGA [10]

Ключевым компонентом программируемого логического блока является таблица соответствия (LUT). Такая таблица соответствия может быть представлена как ячейки SRAM, а значения, записанные в эти ячейки, являются значениями эмулируемой функции. Принцип работы LUT продемонстрирован на рисунке 2.

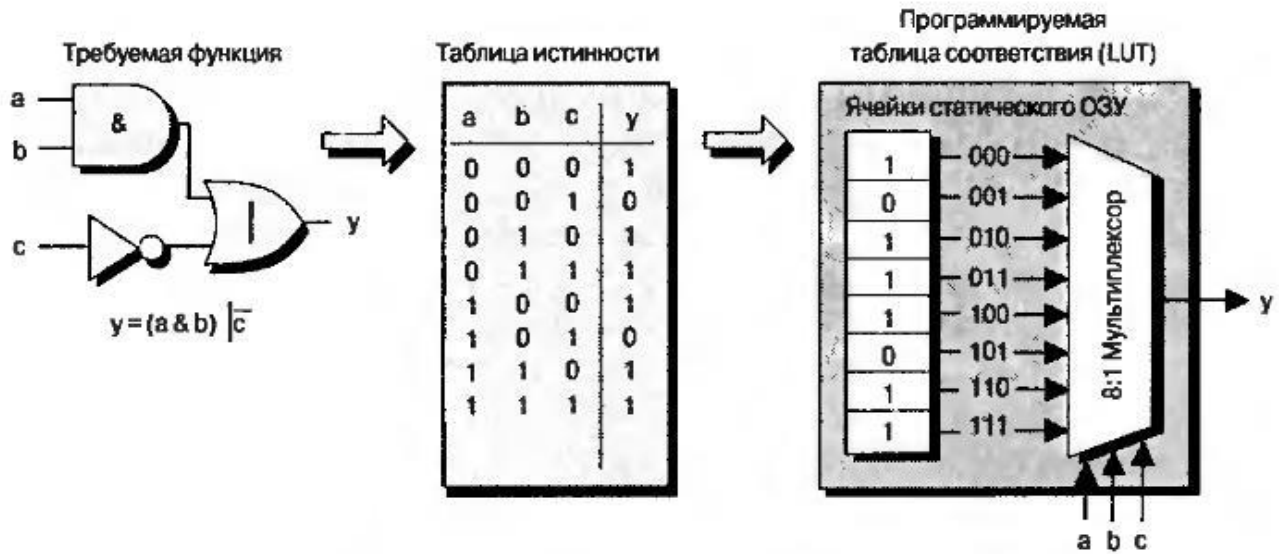


Рисунок 2 – соответствие между LUT и логической функцией [10]

Входные сигналы программируемого логического блока подаются на управляющие входы мультиплексора [13] и на выход подается соответствующее значение таблицы истинности.

FPGA часто применяются в областях, требующих высоких вычислительных мощностей, таких как обработка сигналов в реальном времени, машинное обучение, обработка сетевого трафика.

К минусам FPGA (относительно ASIC) можно отнести дороговизну, меньшую производительность и более высокое энергопотребление. К плюсам – большую гибкость – возможность перепрограммирования схемы после ее производства.

1.3. Крупногранулярные реконфигурируемые архитектуры

1.3.1. Архитектура CGRA

Крупногранулярные реконфигурируемые вычислительные архитектуры, известные под английской аббревиатурой CGRA, позволяют найти баланс между высокой производительностью и низким энергопотреблением интегральных схем специального назначения и гибкостью области применения вычислителей общего назначения. Принцип функционирования архитектуры CGRA [1] заключается в комбинировании логических блоков, часто представляющих собой примитивные арифметико-логические устройства. Структура логических блоков predetermined, однако их соединение может быть перенастроено, что позволяет изменять поведение вычислителя. Основное отличие от программируемых логических интегральных схем, или FPGA, состоит в сложности логического блока [26]. Соответственно, плюсом FPGA относительно CGRA является большая гибкость конфигурации, позволяющая рассматривать FPGA как вычислитель общего назначения, в то время как архитектура CGRA, за счет крупных логических блоков, позволяет уменьшить сложность и, как следствие, время, затрачиваемое на конфигурацию.

На рисунке 3 приведено сравнение вычислительных архитектур по эффективности энергопотребления, гибкости программирования и производительности.

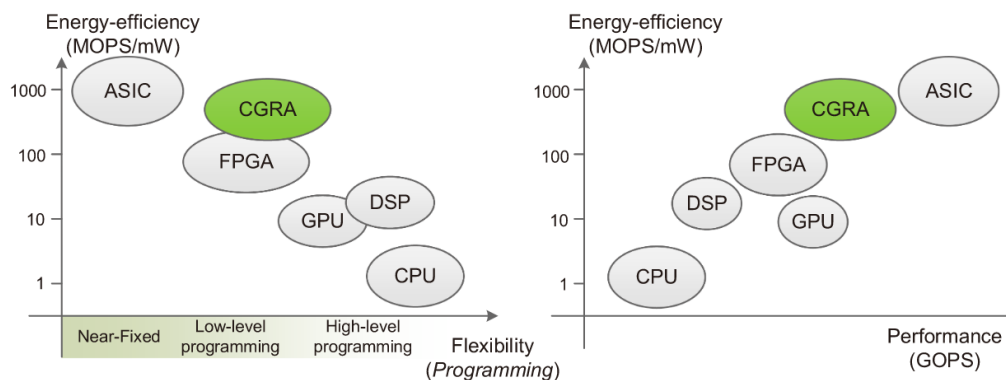


Рисунок 3 - сравнение вычислительных архитектур по эффективности энергопотребления, гибкости программирования и производительности [1].

Как видно на рисунке 3, архитектура CGRA позволяет добиться характеристик производительности и энергопотребления, сравнимых с ASIC, добавляя при это возможность ограниченной переконфигурации.

Тем не менее, данная архитектура имеет и ряд недостатков. Список основных из них приведен ниже.

- Ограничения по реконфигурации. Ввиду крупных размеров логических блоков, из которых собирается вычислитель, возможности по переконфигурации такого вычислителя ограничены. Таким образом, архитектура CGRA позволяет реализовывать специализированные вычислители, переконфигурация которых возможна лишь в узком диапазоне, допускаемом возможностями микроархитектуры. В отличии от архитектуры CGRA, архитектура FPGA, оперирующая более примитивными логическими блоками, имеет большие возможности по переконфигурации, позволяющие строить на FPGA вычислители общего назначения.

- Сложность программирования. Специализированные вычислители, обладающие специфической микроархитектурой, нацеленной для решения конкретной задачи, тяжелее поддаются программированию в виду невозможности полноценного использования языков высокого уровня. Данное ограничение связано с тем что не каждую конструкцию языка высокого уровня возможно исполнить на конкретной микроархитектуре специализированного вычислителя.

Данные особенности позволяют использовать потенциал CGRA в областях, задачи которых имеют большие вычислительные сложности, имея при этом типовую структуру алгоритма, изменения которого от задачи к задаче не высоки. Примерами таких областей могут быть аппаратные ускорители для задач машинного обучения, а также вычислители моделей системной динамики.

Важным аспектом при проектировании CGRA является представление исходного алгоритма в виде графа потока данных.

1.3.2. Граф потока данных

Процессоры общего назначения, архитектура которых подразумевает последовательное выполнение инструкций, имеют ограничения, большая часть которых возникает из-за конструктивных особенностей императивных языков программирования, предполагающих выполнение инструкций строго последовательно.

Различные оптимизации, такие как конвейеризация и суперскалярные архитектуры, позволяют частично обойти эти ограничения. Однако, полная параллелизация выполнения независимых друг от друга инструкций трудно достижима в виду как конструктивных особенностей микроархитектуры процессоров общего назначения, так и сложностей эксплуатации возможностей по параллелизации компиляторами исходного кода.

В отличие от процессоров общего назначения, в основе архитектуры CGRA лежит принцип потоков данных [1]. Это означает что вместо счетчика команд, указывающего, какую команду необходимо выполнить следующей, возможность и необходимость выполнения той или иной инструкции определяется доступностью ее входных аргументов. В общем случае порядок выполнения инструкций во времени для подобной архитектуры не детерминирован. Такой подход позволяет снять ограничения на параллельное исполнение инструкций, заложенное в императивном подходе к программированию, ограничив параллелизм лишь спецификой исполняемого алгоритма и возможностями микроархитектуры вычислителя.

Для конфигурирования вычислителя на основе потока данных необходима следующая информация.

- Информация о микроархитектуре этого вычислителя. Так как возможности параллельного исполнения инструкций ограничены физическим количеством логических блоков на схеме, для оптимальной конфигурации схемы необходимо иметь информацию о наличии и количестве тех или иных логических блоков.

- Граф потока данных. Извлекается из алгоритма высокого уровня, для вычисления которого конфигурируется вычислитель. Отображает ограничения на параллелизацию, заложенные в данном алгоритме, задавая последовательность исполнения связанных инструкций.

1.4. Заключение

В данной главе были даны определения основных терминов, понимание которых необходимо для дальнейшего чтения данной работы. В частности, были изложены основы системной динамики, рассмотрены несколько основных форматов хранения и распространения моделей системной динамики. Также были рассмотрены основные подходы к проектированию специализированных вычислителей. Особое внимание было уделено семейству архитектур CGRA и понятию графа потока данных.

2. ПРИМЕНЕНИЕ САПР NITTA ДЛЯ ЗАДАЧ СИСТЕМНОЙ ДИНАМИКИ

Для создания САПР, способной автоматизированно спроектировать специализированный вычислитель, который можно было бы использовать для расчета модели системной динамики, необходимо следующее.

- САПР должна поддерживать задание входного алгоритма высокого уровня в виде одного из стандартов описания моделей системной динамики.
- САПР должна поддерживать достаточное количество различных вычислительных блоков, комбинация которых позволит симулировать процесс, описанный моделью системной динамики.
- САПР должна поддерживать возможность создания итеративных алгоритмов. Такое требование возникает из-за итеративной сущности моделей системной динамики – моделирование происходит путем создания множества снимков состояния модели с равными промежутками времени между ними.

Рассмотрим возможность выполнения данных требований на основе САПР NITTA.

2.1. Описание САПР NITTA

САПР NITTA – система автоматизированного проектирования, областями применения которой являются [11]:

- Разработка киберфизических систем, основанных на алгоритмах адаптивного управления и искусственного интеллекта с высокими требованиями к задержкам и вычислительному объему, мощности и потребляемой площади.
- Разработка аппаратных программируемых ускорителей и сопроцессоров.
- Разработка проблемно-ориентированных программируемых ASIC.
- Разработка динамически реконфигурируемого IP-ядра и софт-ядра для FPGA.

Ключевыми особенностями САПР NITTA являются:

- Ориентация на модельно-ориентированную инженерию, а не на разработку программного обеспечения.
- Автоматизация большинства этапов разработки, включая проектирование алгоритмов, моделей, функциональное моделирование, создание прототипов, комплексную верификацию, комплексную автоматизацию межуровневого тестирования и синтеза, а также оптимизацию целевой системы.

- Глубокая реконфигурация вычислительной платформы на аппаратном, программном и инструментальном уровнях. Прозрачность рабочего процесса САПР.
- Блочный (крупногранулярный) подход к проектированию аппаратуры. Вместо проектирования вычислительных блоков, исполняющих конкретные задачи, описанные заданным алгоритмом, система NITTA проектирует модель вычислителя как композицию predetermined вычислительных блоков. Это немного ухудшает характеристики проектируемой системы, но сильно упрощает процесс синтеза и уменьшает время, затрачиваемое на него.

2.2. Формирование графа потока данных в САПР NITTA

Для синтеза вычислительной системы САПР NITTA транслирует алгоритм высокого уровня, на базе которого требуется построить вычислительную систему, во внутреннее представление такого алгоритма в виде графа потока данных.

Граф потока данных САПР NITTA состоит из узлов, представленных функциями, и передаваемых значений, представленных его ребрами. Каждое передаваемое значение может быть инициализировано и считано лишь по одному разу. Таким образом, если выходное значение функции является входным значением для нескольких других функций, каждое такое использование должно быть описано как отдельное ребро в графе.

Рассмотрим пример трансляции алгоритма высокого уровня на языке Lua в граф потока данных САПР NITTA. Для примера возьмем следующий простой алгоритм, вычисляющий изменение температуры стакана с горячей водой в комнате с течением времени.

```
function teacup(time, temp_cup)
  local temp_ch = 10
  local temp_room = 70
  local time_step = 0.125

  send(time)
  send(temp_cup)

  time = time + time_step
  local acc = temp_room - temp_cup
  local temp_loss, _ = acc / temp_ch

  local delta = temp_loss * time_step
  temp_cup = temp_cup + delta

  teacup(time, temp_cup)
end
teacup(0, 180)
```

Листинг 1 – модель, описывающая охлаждение стакана с чаем, описанная на языке Lua

Данный алгоритм описывает простейшую модель системной динамики, изображенную на схеме 1. На основе данного входного алгоритма может быть сгенерирован граф потока данных, изображенный на схеме 2.

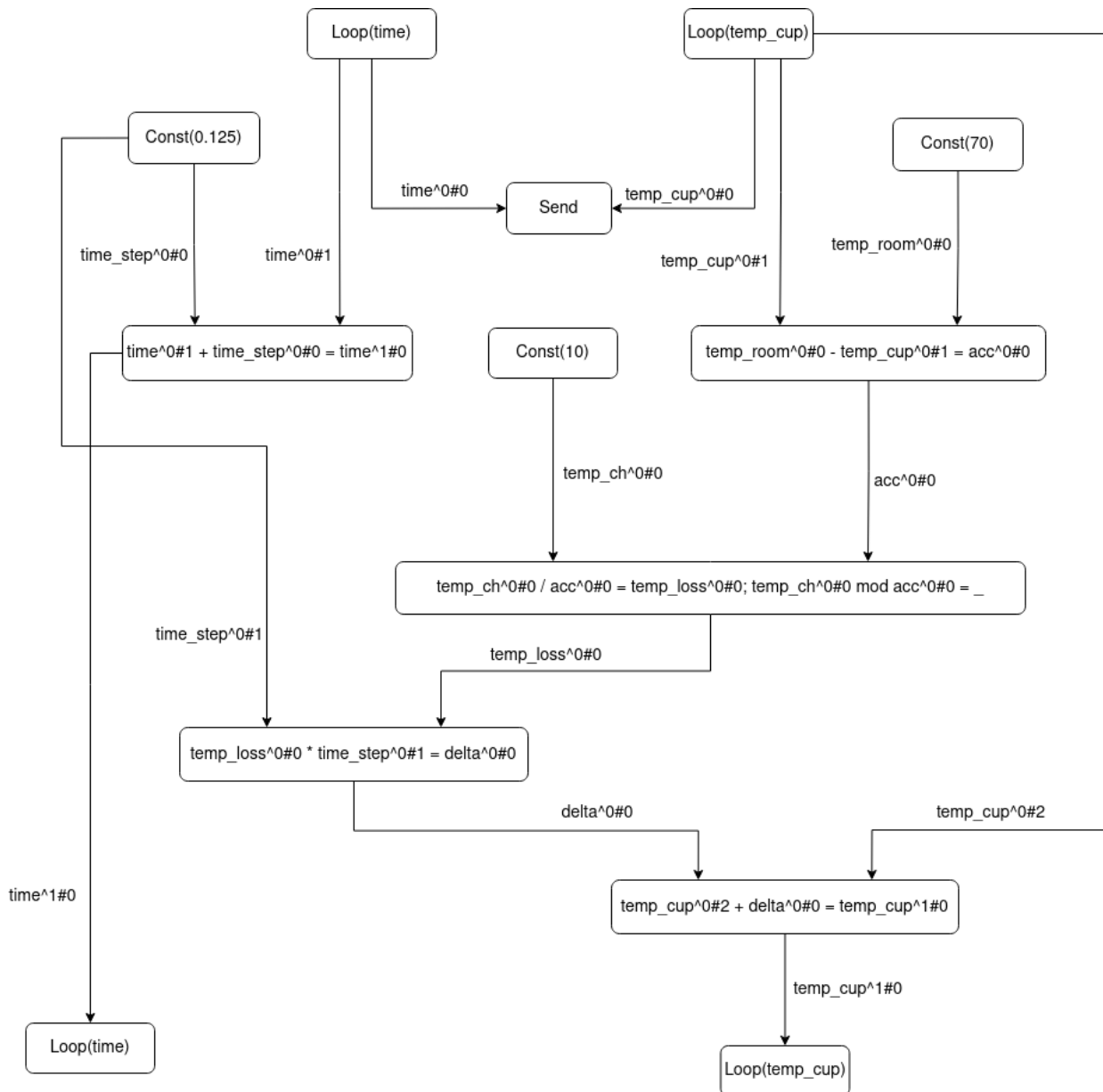


Схема 2 - граф потока данных для модели температуры стакана

Возможность выполнения каждой из операций, описанной в узлах графа, обуславливается доступностью аргументов, описываемых направленными ребрами. Разложение императивного алгоритма в подобный граф позволяет максимизировать параллелизацию вычислений. На дальнейших этапах синтеза для каждого узла подбирается логический блок, способный вычислить описываемую узлом операцию.

Как видно из листинга 1, исходный код представленного алгоритма имеет рекурсивную структуру – функция `teasir` вызывает сама себя. Логику передачи значения переменных между итерациями вычисления функции в графе потока данных берет на себя функция `Loop`, позволяющая передать значение на следующую итерацию алгоритма.

На схеме 2 можно увидеть, что узлы функции `Loop` на графе дублируются. В противном случае граф был бы циклическим – исполнение начиналось и заканчивалось бы на одном и том же узле, что делало бы такой граф в общем случае невозможным для вычисления. Разделение функции `Loop` на 2 части позволяет явно разделить конец предыдущей итерации с началом следующей.

2.3. Соответствие САПР NITTA требованиям для расчетов моделей системной динамики

Рассмотрим выполняемость выведенных ранее требований в САПР NITTA.

- САПР NITTA не имеет возможности задания исходного алгоритма форматом, используемым для описания и распространения моделей системной динамики.
- САПР NITTA имеет predetermined набор вычислительных блоков, реализующих основные арифметические операции. Композиции таких блоков позволяют задавать приоритеты одних операций над другими.

Данные возможности покрывают большую часть функциональности, определенной в области системной динамики. Тем не менее, многие функции, к примеру, тригонометрические, не имеют predetermined вычислительных блоков, что уменьшает количество моделей системной динамики, реализация которых возможна на данном этапе.

САПР NITTA хорошо справляется с проектированием вычислителей для итеративных алгоритмов. Функциональность проектирования передачи значений, рассчитанных на предыдущей итерации в новую итерацию уже заложена и реализована в данной САПР.

Таким образом, для поддержки функциональности по проектированию вычислителей для расчета моделей системной динамики в САПР NITTA необходимо следующее.

- Добавление в САПР NITTA функциональности задания исходного алгоритма при помощи одного из стандартов описания моделей системной динамики.

- Добавление в САПР NITTA дополнительных predefined вычислительных блоков, реализующих функциональность тригонометрических функций.
- Выполнение этих дополнительных требований позволит использовать САПР NITTA для проектирования специализированных вычислителей, способных рассчитывать заранее определенную модель системной динамики.

2.4. Заключение

Проведенный анализ возможности проектирования специализированного вычислителя для расчета заранее определенной модели системной динамики, на примере САПР NITTA, показало, что теоретических ограничений на практическую реализацию подобного проекта нет.

САПР NITTA хорошо подходит для работы с моделями системной динамики, так как один вычислительный цикл графа потока данных NITTA можно представить как изменение состояния модели системной динамики на единицу времени [31]. Количество вычислений на каждой итерации детерминировано, что делает возможным рассматривать такой вычислитель как систему реального времени. Состояния запасов модели системной динамики можно передавать между итерациями с помощью функции Loop.

3. РАЗРАБОТКА ПЛАНА ПОДДЕРЖКИ МОДЕЛЕЙ СИСТЕМНОЙ ДИНАМИКИ В СИСТЕМЕ NITTA

Для возможности проектирования специализированных вычислителей на базе моделей системной динамики необходимо расширить функциональность САПР NITTA возможностями по построению графа потока данных по модели, описанной в одном из форматов хранения и распространения моделей системной динамики. Также необходимо разработать дорожную карту поддержки выбранного формата, позволяющую вводить новую функциональность поэтапно, постепенно наращивая поддерживаемую функциональность.

3.1. Выбор формата моделей системной динамики

В качестве формата хранения и распространения моделей системной динамики, поддержку которого планируется внедрить в систему NITTA, был выбран формат XMILE. Данный формат обладает следующими преимуществами.

- **Открытость.** Формат XMILE является открытым, что позволяет свободно распространять модели, описанные в данном формате, а также свободно реализовывать поддержку данного стандарта программными инструментами.

- **Читаемость.** XMILE, как и лежащий в его основе XML, являются форматами, удобными для чтения человеком. Это позволяет стороннему наблюдателю понять содержимое модели, описанной в формате XMILE, без использования специализированного ПО.
- **Простота.** XMILE реализован как расширение формата XML. Благодаря широкому распространению, в большинстве технологических стеков существуют готовые модули для обработки данных в формате XML. Как следствие, реализовать анализ модели, описанной в формате XMILE достаточно просто на любом технологическом стеке.
- **Иерархическая структура.** Структура хранения модели системной динамики в формате XMILE позволяет при необходимости абстрагироваться от определенных блоков. К примеру, части формата, отвечающие за математическую модель отделены от частей, отвечающих за графическое отображение. Такое разделение упрощает возможность поэтапной реализации всей вложенной в формат функциональности.

3.2. Разработка плана поддержки формата XMILE в САПР NITTA.

Одномоментная реализация всей функциональности, заложенной в формат XMILE является достаточно трудной и объемной задачей. Проектирование, в таком случае, может занять продолжительное время, а демонстрация результата будет возможна лишь после реализации всей функциональности. Вместо этого было решено вести работы по поддержке системой NITTA формата XMILE в поэтапном режиме - каждый этап должен расширять функциональность предыдущего этапа.

Этапы, их продолжительность и цели представлены в таблице 1.

Таблица 1 – план поддержки формата XMILE в САПР NITTA

Номер этапа	Цели
1	Подготовительный этап. Сбор и анализ требований.
2	Проектирование точки расширения поддерживаемых языков задания входных алгоритмов системы NITTA.
3	Проектирование архитектуры блока трансляции формата XMILE в граф потока данных системы NITTA.
4	Реализация минимальной исполняемой архитектуры.
5	Реализация всей функциональности, заложенной в формат XMILE и поддерживаемой системой NITTA.
6	Составление плана по расширению функциональности системы NITTA для поддержки всей функциональности формата XMILE.

Этап 6 нужен для формализации дальнейших доработок в системе NITTA для полноценной поддержки формата XMILE.

В следующей главе содержатся отчеты по каждому этапу.

3.3. Заключение

Выбор формата XMILE для поддержки в САПР NITTA потенциально позволит использовать ее для большинства как уже созданных моделей системной динамики, так и для моделей, создаваемых в будущем.

Разработанный на данной стадии план поддержки формата XMILE в системе NITTA позволит более точно спланировать время, необходимое на выпуск версии системы 0.0.0.1, а также заранее согласовать артефакты, являющиеся результатом каждого из описанных этапов.

4. ПРОЕКТИРОВАНИЕ МОДУЛЯ ПОДДЕРЖКИ СИСТЕМНОЙ ДИНАМИКИ В САПР NITTA

Целью всех описанных в данной главе этапов являлось проектирование модуля автоматизированной системы NITTA, добавляющего в нее функциональность построения специализированных вычислителей на базе произвольной модели системной динамики. В ходе работ по данному этапу необходимо было:

- собрать и проанализировать требования к разрабатываемому модулю;
- спроектировать архитектуру модуля;
- реализовать спроектированную архитектуру;
- верифицировать функциональность реализованной архитектуры.

4.1. Этап 1 - сбор и анализ требований

В ходе обсуждения функциональности модуля с научным руководителем были выявлены следующие функциональные и нефункциональные требования к разрабатываемому модулю:

- разрабатываемый модуль должен в качестве формата входных данных принимать модель системной динамики в формате XMILE;

- разрабатываемый модуль должен поддерживать арифметические операции сложения, вычитания, деления и умножения;
- разрабатываемый модуль должен поддерживать возможность задания приоритета арифметических операций;
- разрабатываемый модуль должен поддерживать следующие элементы формата XMILE: Stocks, Flows, Auxiliaries;
- разрабатываемый модуль должен поддерживать возможность именования сущностей модели в соответствии со стандартом XMILE;
- результатом работы разрабатываемого модуля должен стать граф потока данных — внутреннее представление алгоритма автоматизированной системой НИТТА;
- разрабатываемый модуль должен быть реализован на языке программирования Haskell [17];
- возможность использования нового языка описания алгоритма высоко уровня должна быть добавлена в интерфейс САПР НИТТА;
- должна быть реализована функциональность по автоматическому определению языка высокого уровня по входным данным;
- должна быть реализована возможность явного задания пользователем языка высокого уровня, переопределяющая логику автоматического определения.

4.2. Этап 2 – анализ точек расширения САПР NITTA

Целью данного этапа является анализ архитектуры системы NITTA с целью поиска места, подходящего для встраивания модуля анализа моделей системной динамики в формате XMI/E. Модуль должен быть интегрирован во внешний интерфейс системы NITTA таким образом, чтобы пользователь системы мог переключаться между языками описания исходного алгоритма удобным для себя образом. Кроме того, необходимо реализовать логику автоматического определения языка высокого уровня.

В качестве логики по автоматическому определению языка алгоритма был выбран анализ расширения файла, хранящего такой алгоритм. В виду того что файлы с исходным кодом чаще всего именуются с соответствующими расширениями, такая логика покрывает большее количество сценариев и при этом очень проста в реализации.

В качестве способа явного задания пользователем системы NITTA языка описания алгоритма высокого уровня был выбран дополнительный аргумент командной строки. При выставлении в аргумент командной строки значения, соответствующего тому или иному языку, логика по автоматическому определению языка игнорируется.

4.3. Этап 3 - проектирование архитектуры модуля

Система NITTA уже поддерживает возможность генерации графа потока данных на базе исходного кода Lua. Поэтому, при проектировании модуля генерации графа потока данных из формата XMILE было рассмотрено 2 подхода.

- Генерация исходного кода на языке Lua, соответствующего логике симуляции модели системной динамики с последующей генерацией графа потока данных на базе исходного кода Lua.
- Генерация графа потока данных на базе формата XMILE.

Второй подход обладает рядом преимуществ, описанных ниже.

- Меньшее число преобразований данных из одного формата в другой уменьшает вероятность возникновения ошибки.
- Неполноценная реализация генератора графа потока данных из исходного кода Lua – не все конструкции языка поддерживаются существующим в САПР NITTA генератором.
- Сложность реализации генератора исходного кода языка Lua не ниже сложности реализации генератора графа потока данных напрямую.

В виду описанных значительных недостатков подхода с генерацией графа потока данных через язык Lua, было решено реализовать генератор графа потока данных из формата XMILE напрямую.

Ниже представлена схема архитектуры разрабатываемого модуля в виде нескольких UML-диаграмм. Графический язык UML был выбран из-за его наглядности, простоты проектирования и наличия свободно распространяемых инструментов для проектирования, поддерживающих его.

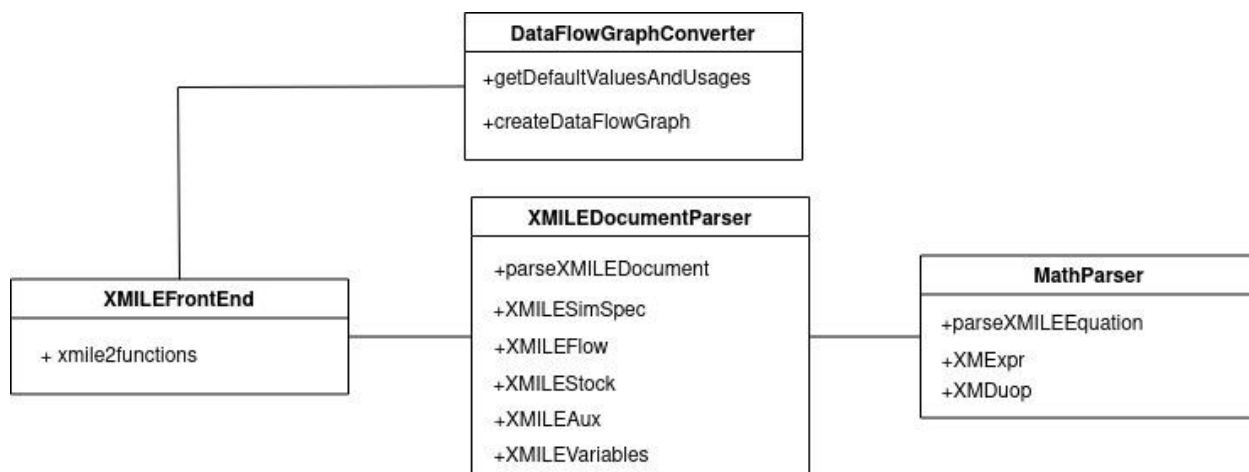


Схема 3 - диаграмма компонентов модуля обработки моделей системной динамики

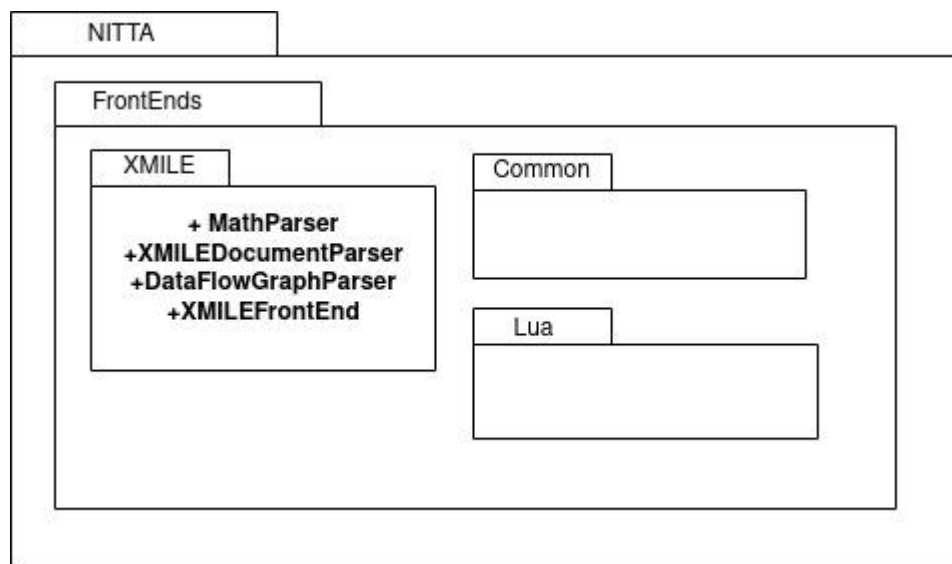


Схема 4 - диаграмма пакетов модуля обработки моделей системной динамики

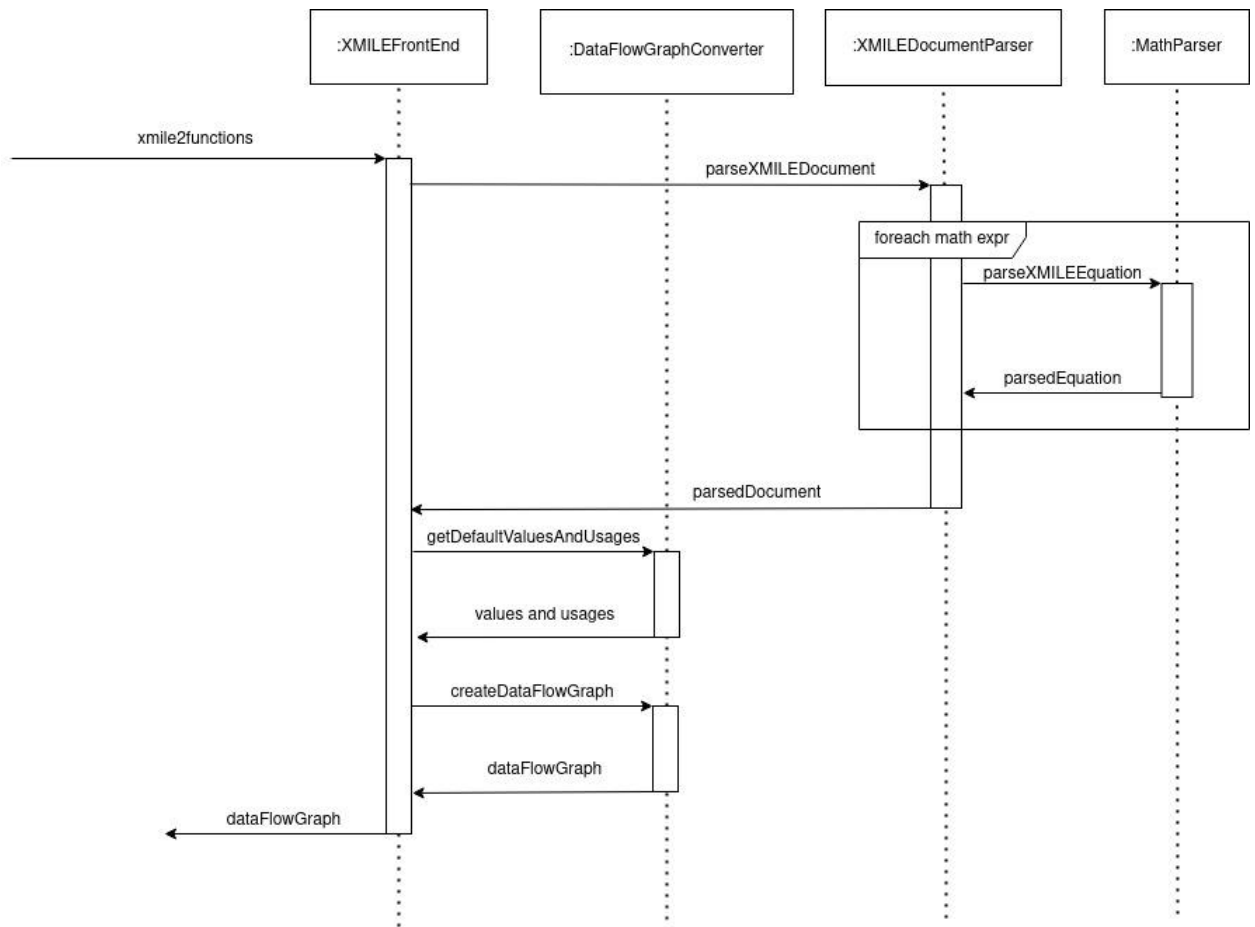


Схема 5 - диаграмма последовательностей модуля обработки моделей системной динамики

Диаграмма компонентов [14] (схема 3) описывает компоненты проектируемого модуля, а также функциональность, предоставляемую каждым из компонентов. Диаграмма пакетов (схема 4) описывает размещение пространств имен проектируемых компонентов в пространстве имен системы NITTA. В данном случае, иерархия пространств имен дублирует физическое размещение файлов с исходным кодом компонентов в иерархии файлов с исходным кодом системы NITTA. Диаграмма последовательностей (схема 5) демонстрирует обработку запроса на анализ модели системной динамики разрабатываемым модулем. На данной схеме можно увидеть последовательность взаимодействий компонентов модуля в процессе обработки запроса.

Данные схемы описывают архитектуру проектируемого модуля обработки моделей системной динамики, описанных в формате XMILE.

4.4. Этап 4 - реализация исполняемой базовой архитектуры

На данном этапе было необходимо разработать базовую исполняемую архитектуру – ключевую функциональность модуля – для демонстрации работоспособности модуля научному руководителю и исправления или обновления исходных требований к модулю. Разрабатываемый модуль состоит из следующих компонентов.

- `XMILEDocumentParser` — реализует функциональность анализа документа в формате XMILE, выделение из структуры документа важных для системы элементов и параметров (исходные значения, количество итераций, уравнения элементов системы) и передача извлеченных данных в вызывающий компонент. В качестве низкоуровневой библиотеки для обработки документа в формате XMILE была выбрана библиотека `hxt` [16]. Причина выбора библиотеки `hxt`, предназначенной для обработки документов в формате XML, является построение формата XMILE на базе формата XML. Кроме того, `hxt` предлагает богатую функциональность по поиску и обработке тегов внутри документа, а также обладает высокой производительностью.
- `MathParser` — реализует обработку математических выражений, хранящихся в исходном документе в формате XMILE, в древовидную структуру, хранящую имена переменных, операции над ними и приоритет этих операций. В качестве низкоуровневой библиотеки для обработки математических выражений в текстовом формате была выбрана библиотека `parsec`. `Parsec` [27] — библиотека, содержащая большое количество predefined низкоуровневых парсеров с возможностью их произвольной комбинации. Данная библиотека проста в использовании и обладает достаточной гибкостью для реализации парсера высокого уровня, решающего задачу, поставленную данному компоненту.
- `DataFlowGraphConverter` — реализует логику обработки последовательностей математических выражений, поступающих на вход компонента во внутренний граф данных автоматизированной системы НИТТА.

- XMILEFrontEnd — компонент, объединяющий функциональность описанных выше модулей. Преобразует исходный файл в формате XMILE, поданный на вход модулю, во внутренний граф потока данных автоматизированной системы NITTA.

Pull request с исходным кодом модуля доступен по ссылке [15].

Рассмотрим пример модели системной динамики, продемонстрированный в листинге 2. Данная модель описана в формате XMILE.

Граф потока данных, сгенерированный для данной модели, изображен на схеме 6. Данный граф удовлетворяет всем ограничениям, выставленным ранее для графа потока данных САПР NITTA.

Данный этап подразумевает реализацию базовой исполняемой архитектуры - ключевой функциональности модуля. В базовую исполняемую архитектуру было решено включить все элементы формата XMILE, входящие в простейшую модель системной динамики, описывающую изменение температуры стакана с горячим чаем, продемонстрированную на рисунке 1. Таким образом, базовая исполняемая архитектура должна реализовать следующую функциональность:

- элементы формата XMILE: Stock, Flow и Aux;
- конфигурацию модели, задаваемую элементом `sim_specs`;
- задание входных и выходных потоков в виде математических выражений;
- задание исходных значений запасов в виде констант.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmile version="1.0" xmlns="http://docs.oasis-open.org/xmile/ns/XMILE/v1.0">
  <header>
    <options>
      <uses_outputs/>
    </options>
    <product version="1.0">Hand Coded XMILE</product>
  </header>
  <sim_specs>
    <stop>30.0</stop>
    <start>0.0</start>
    <dt>0.125</dt>
  </sim_specs>
  <model>
    <variables>
      <flow name="Heat Loss to Room">
        <doc>Heat Loss to Room</doc>
        <eqn>("Teacup Temperature"- "Room Temperature")/"Characteristic
Time"</eqn>
      </flow>
      <aux name="Room Temperature">
        <doc>Ambient Room Temperature</doc>
        <eqn>70</eqn>
      </aux>
      <stock name="Teacup Temperature">
        <doc>The average temperature of the tea and the cup</doc>
        <outflow>"Heat Loss to Room"</outflow>
        <eqn>180</eqn>
      </stock>
      <aux name="Characteristic Time">
        <eqn>10</eqn>
      </aux>
    </variables>
  </model>
</xmile>

```

Листинг 2 - модель системной динамики, описывающая изменение температуры стакана, описанная в формате XMILE

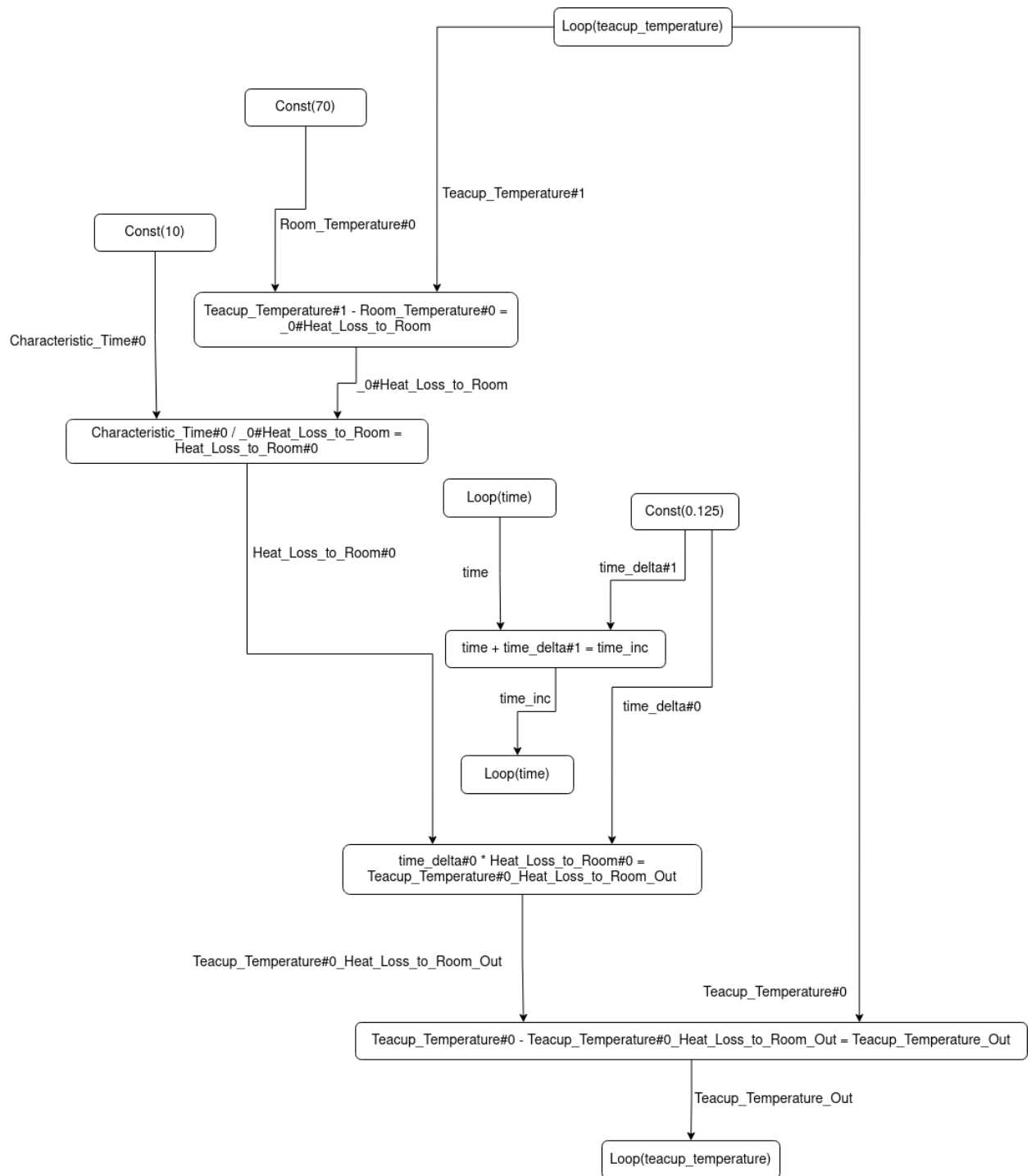


Схема 6 - граф потока данных для модели температуры стакана для формата XMILE

При сравнении графа потока данных, сгенерированного на основе модели системной динамики в формате XMILE (схема 6), с графом потока данных, сгенерированным на основе исходного кода Lua (схема 2) с аналогичной логикой, можно увидеть, что графы практически идентичны, с разницей в именовании передающихся между функциями значений. Идентичность графов потоков данных говорит об одинаковых результатах процесса синтеза для алгоритмов, при задании их как в формате Lua, так и в формате XMILE.

4.4.1. Тестирование

Перед вводом программного обеспечения в эксплуатацию необходимо убедиться в его работоспособности и верифицировать его на соответствие поставленным требованиям. Для такой верификации разработанного модуля были проведены несколько этапов тестирования.

- Аудит исходного кода. Позволяет выявить уязвимости исходного кода, ошибки в функциональности, дублирование и неконсистентность кодовой базы до запуска программы. Аудит исходного кода проводился с использованием системы контроля версий git [20] и платформы Github. Исходный код проверялся научным руководителем данной работы.
- Функциональное тестирование. Позволяет верифицировать интеграцию разработанного модуля в целевую систему.
- Модульное тестирование. Позволяет верифицировать функционал конкретной функции в отрыве от системы в целом.

Для разработанного модуля было создано тестовое покрытие, покрывающее основные элементы логики, а именно:

- анализ исходного документа в формате XMILE;
- построение дерева математических операторов на основе математического выражения, заданного в виде строки;
- вычисление значения построенного дерева;
- построение графа потока данных по полученным из исходного файла данным.

Тесты были написаны на основе тестовых фреймворков Hunit [18] и Tasty [19], предоставляющих широкую функциональность по верификации функций, написанных на языке Haskell. Данные фреймворки широко используются в тестировании других компонентов системы НИТГА.

4.4.2. Результаты

Результаты функциональной симуляции работы вычислителя, основанного на базе модели системной динамики, изображенной на схеме 1, приведены на рисунке 4. Для наглядности количество итераций ограничено 10.

```
base) [artur@artur-pc nitta]$ stack exec nitta -- examples/teacup.xmile --fsim --format=md -t=fx24.32
Cycle | Teacup_Temperature | time |
:-----|:-----|:-----|
1 | 180.000 | 0.000 |
2 | 178.625 | 0.125 |
3 | 177.375 | 0.250 |
4 | 176.125 | 0.375 |
5 | 174.875 | 0.500 |
6 | 173.625 | 0.625 |
7 | 172.375 | 0.750 |
8 | 171.125 | 0.875 |
9 | 169.875 | 1.000 |
10 | 168.750 | 1.125 |
```

Рисунок 4 - результат симуляции вычислителя, спроектированного на основе модели системной динамики

Как видно из результатов симуляции, температура стакана с горячей водой постепенно уменьшается с течением времени. При этом разность температур между равными промежутками времени уменьшается. Так, за первую итерацию температура изменилась на 1,375 градуса по Фаренгейту, в то время как на последней, десятой, итерации, изменение составило лишь 1,125 градуса. Таким образом, вычисление модели системной динамики, рассмотренной выше, показывает, что температура тела, помещенного в среду с другой температурой, стремится к температуре окружающей среды с обратной экспоненциальной зависимостью.

4.5. Этап 5 – доработка функциональности, заложенной в формат XMILE и поддерживаемой системой NITTA

Целью данного этапа является доработка функциональности, заложенной в базовую исполняемую архитектуру на предыдущем этапе всеми возможностями, заложенными в САПР NITTA на момент проведения работ. К таким доработкам относится следующая функциональность:

- возможность задания исходного значения аккумулятора в виде математического выражения;
- поддержка нескольких входных и выходных потоков;
- оптимизация потоков, дублирующих другой поток.

В архитектурном плане, изложенном на схемах 3,4 и 5, данные доработки ничего не меняют, изменяется и расширяется лишь функциональность реализованных на предыдущем этапе блоков.

В качестве демонстрации инкремента системы, созданного на данном этапе работы, рассмотрим граф потока данных, сгенерированный разработанным модулем для более сложной модели системной динамики.

Рассматриваемая модель в формате XMILE продемонстрирована в листинге 3.

Модель, изображенная на схеме 7, симулирует численность популяции зайцев. На популяцию влияют такие параметры как площадь, на которой данная популяция обитает, рождаемость зайцев, их смертность, а также популяция лисиц на этой территории. Кроме того, можно увидеть, что дополнительная формула `'hares_killed_per_lynx'`, определенная через равенство потоку `'hare_density'`, была оптимизирована. Во всех местах ее использования используется непосредственно `'hare_density'`.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmile version="1.0" xmlns="http://docs.oasis-open.org/xmile/ns/XMILE/v1.0">
  <sim_specs>
    <stop>30.0</stop>
    <start>0.0</start>
    <dt>0.125</dt>
  </sim_specs>
  <model name="Hares">
    <variables>
      <stock name="Hares">
        <eqn>5E4</eqn>
        <inflow>hare_births</inflow>
        <outflow>hare_deaths</outflow>
      </stock>
      <flow name="hare_births">
        <eqn>Hares*hare_birth_fraction</eqn>
      </flow>
      <flow name="hare_deaths">
        <eqn>Lynx*hares_killed_per_lynx</eqn>
      </flow>
      <stock name="Lynx">
        <eqn>1250</eqn>
      </stock>
      <aux name="hare_birth_fraction">
        <eqn>1.25</eqn>
      </aux>
      <aux name="hare_density">
        <eqn>Hares/area</eqn>
      </aux>
      <aux name="area">
        <eqn>1E3</eqn>
      </aux>
      <aux name="hares_killed_per_lynx">
        <eqn>hare_density</eqn>
      </aux>
    </variables>
  </model>
</xmile>

```

Листинг 3 - модель системной динамики, моделирующая популяцию кроликов, описанная в формате XMILE

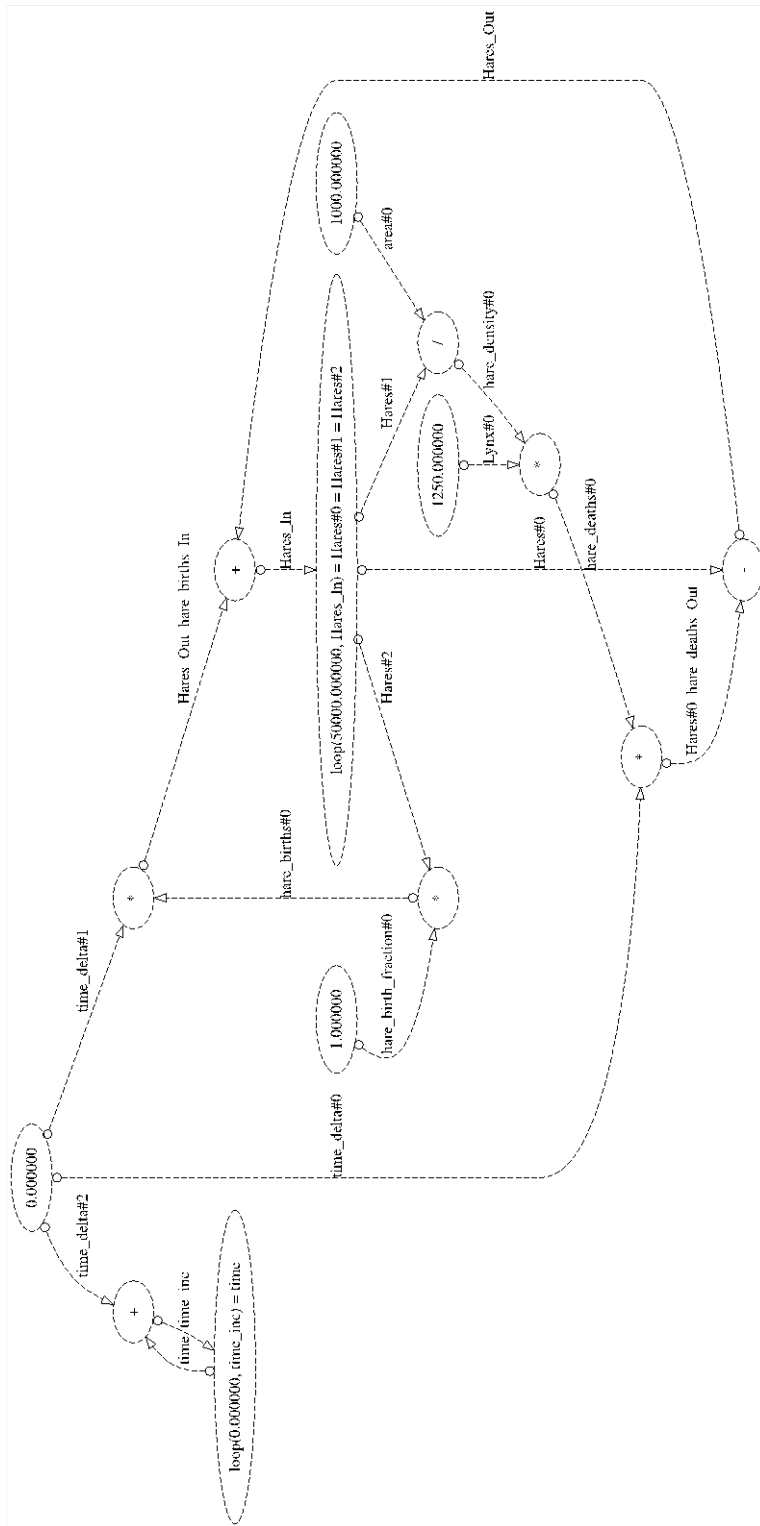


Схема 7- граф потока данных для модели популяции зайцев

Относительно модели, симулирующей изменение температуры стакана с чаем, в данной модели увеличено количество запасов, присутствуют как входящие, так и исходящие потоки. Также в данной модели используется запись констант в экспоненциальном виде.

Граф потока данных, изображенный на схеме 7, получен из графического интерфейса САПР NITTA.

4.6.Этап 6 - составление плана по расширению функциональности системы NITTA

Цель данного этапа – составить план доработок САПР NITTA для полноценной поддержки формата XMILE.

Формат XMILE обладает широкими возможностями по хранению данных о различных графических функциях, таких как графики и стили отображения. Графики позволяют визуально отобразить изменение аккумуляторов модели с течением времени в графическом интерфейсе. Стили отображения позволяют настроить параметры графического отображения, такие как цвета, шрифты, размеры шрифтов, границы и т.д. Реализация данного функционала выходит за рамки текущей работы, так как она посвящена непосредственно проектированию вычислителей для расчета моделей системной динамики.

Кроме того, формат XMILE описывает некоторые операции, недоступные в системе NITTA на данный момент. К наиболее значимым из таких операций можно отнести математические (ABS, ARCCOS, ARCSIN, ARCTAN, COS, EXP, INF, INT, LN, LOG10, MAX, MIN, PI, SIN, SQRT, TAN) и статистические (EXPRND, LOGNORMAL, NORMAL, POISSON, RANDOM). Реализация в системе NITTA логических блоков, способных вычислять такие функции позволит существенно увеличить покрытие системой NITTA множества моделей системной динамики, для которых возможно создать специализированный вычислитель.

Более подробно данные операции, их описание и приоритет реализации представлены в таблице 2.

Таблица 2 — описание функций и их приоритетов по реализации в системе NITTA

Название функции	Описание	Приоритет
ABS	Модуль числа	Высокий
COS	Косинус числа	Высокий
EXP	Возведение числа в заданную степень	Высокий
LN	Натуральный логарифм	Высокий
MAX	Наибольшее из 2-х чисел	Высокий
MIN	Наименьшее из 2-х чисел	Высокий
SIN	Синус числа	Высокий
SQRT	Квадратный корень положительного числа	Высокий
INT	Наибольшее целое число, меньшее или равное заданному	Средний
ARCTAN	Арктангенс числа	Средний

Название функции	Описание	Приоритет
LOG10	Логарифм с основанием 10	Средний
ARCCOS	Арккосинус числа	Средний
ARCSIN	Арксинус числа	Средний
INF	Бесконечность	Средний
RANDOM	Случайное число, полученное с помощью равномерного распределения	Средний
TAN	Тангенс числа	Средний
EXPRND	Случайное число, полученное с помощью экспоненциального распределения	Средний
LOGNORMAL	Случайное число, полученное с помощью логнормального распределения	Средний
NORMAL	Случайное число, полученное с помощью нормального распределения	Средний
POISSON	Случайное число, полученное с помощью распределения Пуассона	Средний
PI	Константа — число Пи	Низкий

Данные, представленные в таблице 2, можно использовать для приоритезации работ по дальнейшему совершенствованию функциональности системы НИТТА.

Кроме того, формат XMIŁE подразумевает хранение всех численных констант, а также результатов выполнения всех математических операций в стандарте IEEE 754 [28], описывающем представление чисел с плавающей точкой. САПР NITTA не поддерживает математические операции с числами с плавающей точкой. Реализация поддержки чисел в формате IEEE 754 также является высокоприоритетной задачей.

Наконец, важным аспектом в формате XMIŁE является поддержка различных методов интегрирования. Помимо метода Эйлера [29], подразумеваемого по умолчанию, формат поддерживает интегрирование методами Рунге-Кутты 2 и 4 порядка [29], а также численный метод Гира [30].

4.7. Заключение

В ходе данного этапа работ был спроектирован, реализован и протестирован модуль САПР NITTA для поддержки моделей системной динамики в формате XMIŁE. Несмотря на то что на данный момент функциональности системы NITTA недостаточно для полноценной поддержки формата, была реализована поддержка основной функциональности, включая запасы, потоки, спецификации симуляции, а также обработку элементарных математических операций. Реализация данной функциональности позволяет использовать САПР NITTA для проектирования вычислителей, моделирующих простейшие модели системной динамики.

ЗАКЛЮЧЕНИЕ

В ходе представленной работы были решены следующие задачи.

- Проведен анализ возможности генерации графа потока данных для моделей системной динамики.
- Проведен сравнительный анализ трёх наиболее популярных в сообществе форматов хранения и распространения моделей системной динамики. В качестве реализуемого в САПР NITTA формата был выбран формат XMILE.
- Составлен план поддержки формата XMILE в САПР NITTA.
- Спроектирована и реализована архитектура модуля трансляции модели системной динамики в формате XMILE.
- Проведено тестирование разработанного модуля, приведены результаты симуляции системой NITTA моделей, загруженных из формата XMILE.

Результатом данной работы является практическая демонстрация возможности проектирования специализированных вычислителей на базе архитектуры CGRA для произвольных моделей системной динамики. Возможность проектирования таких вычислителей позволит существенно увеличить производительность симуляции таких моделей, а также уменьшить энергозатраты их симуляции. Кроме того, ограниченных возможностей по переконфигурации CGRA должно быть достаточно для варьирования входных параметров модели. Такое варьирование параметров при анализе моделей системной динамики позволяет отследить влияние конкретного параметра на поведение системы с течением времени.

Также результатом проделанной работы является расширение функциональности САПР NITTA возможностями по построению и симуляции специализированных вычислителей на базе моделей системной динамики, заданных в формате XMLE. Такое расширение функциональности позволяет увеличить спектр задач, решаемых с помощью данной системы.

СПИСОК ЛИТЕРАТУРЫ

1. Liu L, Zhu J, Li Z, Lu Y A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications // ACM Computing Surveys. - 2020. - №52. - С. 1-39.
2. SMILE and XMILE: A Common Language and Interchange Format for System Dynamics // citeseerx.ist.psu.edu URL: citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.517.5975&rep=rep1&type=pdf (дата обращения: 01.02.2022).
3. Колесов, Ю.Б. Моделирование систем. Динамические и гибридные системы / Ю.Б. Колесов. – Санкт-Петербург : БХВ-Петербург, 2006г. – 224 с.
4. XML Interchange Language for System Dynamics (XMILE) Version 1.0 // docs.oasis-open.org URL: docs.oasis-open.org/xmile/xmile/v1.0/cos01/xmile-v1.0-cos01.html (дата обращения: 06.03.2022).
5. OASIS XML Interchange Language (XMILE) for System Dynamics Technical Committee // www.oasis-open.org URL: www.oasis-open.org/committees/xmile charter.php (дата обращения: 18.03.2022).
6. Ventana Simulation Environment. User's Guide // www.cs.vsu.ru URL: www.cs.vsu.ru/~svv/swe/VensimUsersGuide.pdf (дата обращения: 21.12.2021).

7. T. Nowatzki, V. Gangadhan, K. Sankaralingam and G. Wright, "Pushing the limits of accelerator efficiency while retaining programmability," *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 27-39, doi: 10.1109/HPCA.2016.7446051.
8. Хопкрофт Д, Мотвани Р, Ульман Д Введение в теорию автоматов, языков и вычислений. - 2-е изд. - СПб: Вильямс, 2008. - 528 с.
9. Rapid ASIC Design for Digital Signal Processors // www2.eecs.berkeley.edu URL: www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-32.pdf (дата обращения: 23.04.2022).
10. История развития интегральных схем (от ASIC до FPGA). Примеры современного использования FPGA. // hpc-education.ru URL: hpc-education.ru/files/lectures/2011/rumyantsev/rumyantsev_2011_lectures01.pdf (дата обращения: 29.04.2022).
11. Synthesis Method for CGRA Processors based on Imitation Model // www.researchgate.net URL: www.researchgate.net/publication/350871215_Synthesis_Method_for_CGRA_Processors_based_on_Imitation_Model (дата обращения: 11.12.2021).
12. Пенской А.В., Платунов А.Е., Ключев А.О., Горбачев Я.Г., Яналов Р.И. Система высокоуровневого синтеза на основе гибридной реконфигурируемой микроархитектуры // Научно-технический вестник информационных технологий, механики и оптики. 2019. Т. 19. No 2. С. 306–313. doi: 10.17586/2226-1494-2019-19-2-306-313

13. Хэррис Д., Хэррис С. Цифровая схемотехника и архитектура компьютера. Morgan Kaufman, 2013. 1627 с.
14. Арлоу, Д UML 2 и Унифицированный процесс / Д Арлоу, А Нейштадт. – Санкт-Петербург: Символ, 2008. – 624 с.
15. pull request с исходным кодом [Электронный ресурс]. – Режим доступа: github.com/ryukzak/nitta/pull/204. – Дата доступа: 13.04.2022.
16. hxt: A collection of tools for processing XML with Haskell. // [hackage.haskell.org](https://hackage.haskell.org/package/hxt) URL: <https://hackage.haskell.org/package/hxt> (дата обращения: 03.04.2022).
17. Уилл К. Програмируй на Haskell. - М.: ДМК Пресс, 2019. - 648 с. – ISBN 978-5-97060-694-0
18. HUnit: A unit testing framework for Haskell // [hackage.haskell.org](https://hackage.haskell.org/package/HUnit) URL: <https://hackage.haskell.org/package/HUnit> (дата обращения: 29.02.2022).
19. tasty: Modern and extensible testing framework // [hackage.haskell.org](https://hackage.haskell.org/package/tasty) URL: hackage.haskell.org/package/tasty (дата обращения: 29.02.2022).
20. Чакон С., Штрауб Б. Git для профессионального программиста. - 1-е изд. - СПб: Питер, 2022. - 496 с.
21. Э. Таненбаум. Архитектура компьютера. — 6 изд. — СПб: Питер, 2018. — 1150 с. — ISBN 978-5-4461-1103-9
22. М. Клеппман. Высоконагруженные приложения. Программирование масштабирование поддержка. — СПб: Питер, 2018. — 805 с. — ISBN 978-5-4461-0512-0.

23. Кон М. Agile: Оценка и планирование проектов. - 1-е изд. - Москва: Альпина Паблишер, 2022. - 424 с. – ISBN 978-5-9614-6947-9
24. Perl I., Mulyukin A., Kossovich T. Continuous execution of system dynamics models on input data stream // Proc. 20th Conference of Open Innovations Association (FRUCT). 2017. P. 371–376. doi: 10.23919/fruct.2017.8071336
25. J. W. Forrester, The Beginning of System Dynamics, Cambridge: MIT, 1989.
26. Architecture Exploration of Standard-Cell and FPGA-Overlay CGRAs Using the Open-Source CGRA-ME Framework // cgra-me.ece.utoronto.ca URL: cgra-me.ece.utoronto.ca/downloads/ispd003i-chinA.pdf (дата обращения: 06.02.2022).
27. parsec: Monadic parser combinators // hackage.haskell.org URL: <https://hackage.haskell.org/package/parsec> (дата обращения: 03.04.2022).
28. IEEE Standard for Floating-Point Arithmetic // standards.ieee.org URL: <https://standards.ieee.org/ieee/754/6210/> (дата обращения: 14.04.2022).
29. Зенков А.В. Численные методы. Учебное пособие. - 1-е изд. - Екатеринбург: Издательство Уральского университета, 2016. - 128 с.
30. Формулы дифференцирования назад и методы Гира. Представление Нордсика // intuit.ru URL: intuit.ru/studies/courses/1012/168/lecture/4606?page=8 (дата обращения: 04.05.2022).
31. Penskoï A., Perl I. Hardware Accelerator for System Dynamic Modeling // The 2020 Conference of the System Dynamics Society. - Bergen: 2020