

## Оглавление

Введение	5
1 Инструменты и средства системной динамики	6
1.1 Область системной динамики.....	6
1.2 Обзор средств моделирования.....	9
1.3 Задача моделирования в реальном времени.....	11
Выводы по главе 1.....	14
2 Анализ аппаратной реализации расчетов системной динамики	15
2.1 Анализ вычислительных платформ .....	15
2.2 Выбор целевой вычислительной платформы.....	18
2.3 Выбор управляющей системы .....	21
2.4 Выбор протоколов физического уровня.....	25
Выводы по главе 2.....	31
3 Проектирование и разработка системы управления	32
3.1 Особенности разработки на платформе Electric Imp.....	32
3.2 Проектирование и реализация интерфейса imp – sdCloud .....	34
3.3 Проектирование интерфейса Imp – NITTA.....	36
3.4 Разработка интерфейса Imp – NITTA .....	38
Выводы по главе 3.....	43
4 Испытание прототипа системы управления	44
4.1 Сборка и тестирование прототипа .....	44
4.2 Оценка эффективности.....	45
Выводы по главе 4.....	48
Заключение	49
Список используемых источников	50

## Введение

В настоящее время практически во всех системах используются процессоры общего назначения, разработанные прежде всего для обычных компьютеров, эти процессоры умеют решать практически любую вычислительную задачу. Из-за огромного рынка компьютеров, смартфонов и планшетов это самые популярные виды процессоров. Ими пользуются даже когда конечная цель не нуждается в таком обилии возможностей, благодаря широкой доступности, большому количеству специалистов и гарантии работы решения.

Для узких задач, где требуется большое количество расчетов по заранее заданной схеме, часто применяют ПЛИС. Это программируемая логическая интегральная схема, конфигурация которой, может быть загружена после включения питания. Такая схема может обеспечивать большой прирост скорости и потреблять меньше энергии, по сравнению с процессором общего назначения.

На базе ИТМО ведется разработка сервиса облачного моделирования системной динамики sdCloud. Этот сервис считает модели на процессорах общего назначения. Для ускорения моделирования уменьшения затрат на электроэнергию было решено применить ПЛИС для задач системной динамики.

Так как сам ПЛИС по своей сути обеспечивает только вычисления, для того чтобы осуществлять коммуникацию с сервером sdCloud, ему нужно управляющее устройство в виде небольшого компьютера, который обеспечивал бы соединение с облаком и передачу модельных данных. В силу таких особенностей как низкое энергопотребление и встроенные решения для работы на физическом уровне был выбран компьютер от компании Electric Imp, которая специализируется на Интернете вещей. Подробнее об этом выборе будет рассказано в главе 2. Разработка управляющей системы на устройстве от Electric Imp и будет подробно раскрываться в представленной работе.

# 1 Инструменты и средства системной динамики

## 1.1 Область системной динамики

Сложная система – это система из нескольких компонент, которые могут взаимодействовать с друг другом. По сути своей под это понятие подходит любое явление в нашей жизни. Так или иначе любая естественная наука занимается исследованием сложных систем. Физика ищет модели, по которым взаимодействуют объекты, а экономика занимается моделированием системы производства, распределения, обмена и потребления.

Системная динамика (System Dynamics) – это направление моделирования, представляющее собой исследование нелинейного поведения сложных систем в течение длительного времени [1].

Модели системной динамики описывают зависимости между состояниями и потоками изучаемой системы. Изменения на входе такой системы приводят к изменениям на выходе. Основными компонентами моделей системной динамики являются:

- состояния, соответствующие значениям переменных, которые будут подвергаться изменениям;
- потоки между состояниями, соответствующие функции изменения;
- обратные связи между различными состояниями, использование которых позволяет осуществлять действия, основанные на результатах предыдущих действий.

Из этих компонентов мы получаем инструмент, который может использоваться как для описания физических процессов, так и для описания сложных экономических зависимостей.

Область системной динамики появилась в эпоху автоматизации вычислений, в середине 1950-х годов. Модели, сложные для вычисления, но описанные в рамках несложных зависимостей подходят для компьютерных вычислений. Практический

результат любой модели системной динамики — это табличка с выходными данными моделирования, по которой уже можно сделать какие-то выводы о результатах или продолжить работу с этими данными.

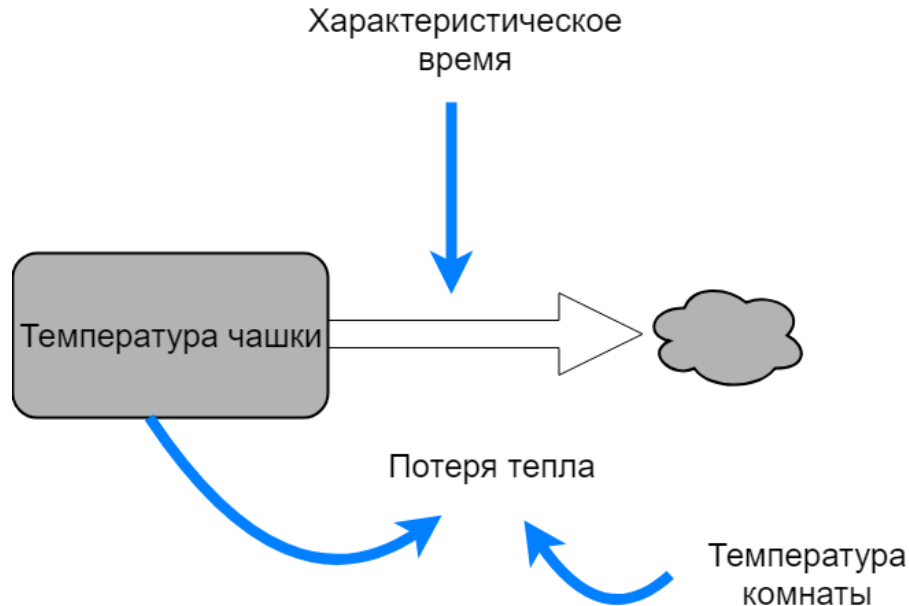


Рисунок 1 – Пример простейшей модели

Рассмотрим подробно модель остывающей чашки в комнате. У нас есть одно состояние, это температура кружки, которая постепенно уменьшается. Ее уменьшение описывается законом Ньютона. Мы можем определить остывание кружки аналитически, взяв первую производную по времени. Но в данном случае у нас всего одно уравнение вместо нескольких, и для более сложной системы часто найти решение аналитически не представляется возможным. Поэтому в данном примере каждый небольшой промежуток времени мы пересчитываем температуру кружки с учетом влияния внешней среды. Последовательные результаты расчета модели для температуры комнаты 21 градусов и температуры кружки 90 градусов в таблице 1.

Таблица 1 – Результат моделирования остывающей чашки

Время, сек.	Потери тепла, Дж.	Температура кружки, С	Температура комнаты, С
0,000	6,90000	90,000	21,000000
0,125	6,81375	89,137	21,000000
0,250	6,72857	88,285	21,000000
0,375	6,56141	87,444	21,000000
0,500	6,47939	86,614	21,000000

Чем точнее хочется описать какой-то процесс, тем больше состояний нужно включить в модель. К тому же, так как состояния используют данные от других состояний, моделирование происходит точнее при уменьшении шага модели. Например, для физического процесса нагревания чашки данные которые рассчитывались каждую минуту получатся менее точными, чем те, что вычислялись каждую секунду. Это все подводит нас к высокой вычислительной сложности моделей системной динамики. Чем сложнее процесс и чем больше в нем шагов, тем дольше идут вычисления для моделирования. Так как процессы могут быть неограниченной сложности, то чем быстрее мы умеем их считать, тем лучше [2]. Кроме того, считая точные модели в реальном времени мы можем предсказывать поведение моделируемой системы, что в конечном счете и есть задача системной динамики. Например, моделируя фондовый рынок с отставанием мы никогда не получим полезный результат, потому что актуальные данные из реальной жизни будут приходить быстрее, чем модель успеет их спрогнозировать. Но если некая точная модель справится с этой задачей в реальном времени, то мы сможем заключить выгодную сделку в нужный момент. Поэтому есть интерес в ускорении моделирования.

## 1.2 Обзор средств моделирования

Модели системной динамики были придуманы в 50-ых годах XX века Джейм Форрестером из МТИ. Он же и придумал первый формат для их компьютерного описания SIMPLE (Simulation of Industrial Management Problems with Lots of Equations, или Моделирование Проблем Промышленного Менеджмента Совокупностью Уравнений). С того времени системная динамика шагнула вперед и форматов для моделирования стало гораздо больше. Рассмотрим самые популярные из них:

VenSim – формат описания моделей системной динамики, который поддерживается компанией Ventana Systems [3]. Этот формат поддерживается в закрытом программном обеспечении от этой компании для исполнения моделей, и во многих других системах.

AnyLogic – формат описания моделей от компании AnyLogic [4], и тоже поддерживается в закрытом программном обеспечении.

Оба решения для исполнения моделей от этих компаний обладают значительными минусами. Прежде всего эти решения работают только на одном ядре процессора, и не исполняются параллельно. Также важной деталью является то, что на выходе мы получаем только результат подсчета и такой продукт тяжело интегрировать в свое приложение. На практике это означает, что мы не можем создать какую-то программу, в которой системно-динамическая модель была бы лишь компонентом. Например, нельзя создать модель города, в которой симуляция дорожного движения была бы построена на модели от VenSim или AnyLogic. Стоит упомянуть также активно развивающийся формат XMI LE [5], который является попыткой объединить многие текущие форматы в один, которым можно было бы удобно делиться и который был бы понятен для всех.

В частности, для решения задач интеграции в приложения и для более прозрачного моделирования были созданы свободные библиотеки PySD,

SDEverywere и sd.js, которые занимаются транслированием моделей системной динамики в исполняемый код на языках Python, C, и JavaScript соответственно. Это решает проблему интеграции в приложения и позволяет оптимизировать вычисление модели, так как алгоритм вычисления становится полностью доступен пользователю. Эти свободные библиотеки ориентированы прежде всего на инженеров-программистов, так как требуют знания конечных языков и умением работать с библиотеками, в отличие от закрытых решений, которые обсуждались ранее.

В связи с ограниченностью некоммерческих версий закрытых решений, и требований особых навыков для использования свободных библиотек, в некоторых учебных заведениях обходили практическое применение системной динамики стороной.

sdCloud – первое облачное решение для расчета и исполнения моделей системной динамики. Оно объединяет в себе плюсы от закрытых решений, используя при этом открытые библиотеки внутри.

Чтобы сделать процесс моделирования более простым и доступным, систему моделирования можно представить, как облачное решение, в которое моделисты будут иметь возможность загружать модели и планировать их исполнение. Такой подход решает сразу множество проблем: нехватку ресурсов, высокий уровень вхождения, требование умения программировать и необходимость локальной настройки и установки системы моделирования.

Помимо этого, облачная система моделирования даёт дополнительные возможности, такие, как например возможность быстро и просто делиться результатами своего моделирования или самими моделями между специалистами. Также облачное решение даёт моделистам большую свободу. Им больше не требуется следить за процессом моделирования на локальной рабочей станции,

процесс можно отслеживать с любого компьютера или мобильного устройства откуда и когда угодно.

В связи с этим sdCloud разрабатывает высокоэффективные алгоритмы, методы и подходы, которые позволяют оптимизировать используемые ресурсы сервиса.

Первая оптимизация касается электроэнергии. Любое снижение потребления снижает стоимость содержания серверов, что положительно сказывается на проекте. Современные датацентры тратят много энергии, а само электричество не дешевеет. В связи с этим любые оптимизации в этой области могут в теории сократить объем потребления.

Вторая оптимизация — это ускорение расчета модели. Любое ускорение дает нам большее количество моделей, которое мы можем посчитать за определенный промежуток времени, тем самым улучшив доступность сервиса. Также скорость важна для больших моделей, потому что прирост в скорости на 20 процентов, означает, что модель, которая считается 3 часа в обычных условиях, посчитается на полчаса быстрее, что важно для конечного пользователя.

### **1.3 Задача моделирования в реальном времени**

Стоит упомянуть еще одну оптимизацию - это возможность считать модели в реальном времени. Реальное время означает, что мы тратим на очередной расчет время, равное одному шагу моделирования. Для примера с чашкой этот шаг был равен 0.125 секунд, соответственно если бы мы получали новый результат каждые 0.125 секунд, то это было бы моделированием в реальном времени.

Большое применение эта оптимизация находит в области, которая называется программно-аппаратное моделирование. Это метод, который используется для разработки и тестирования сложных систем реального времени. Рассмотрим пример – есть какая-то встроенная система, пусть это система подачи топлива в двигатель. Для оптимизации параметров и тестирования такой системы нужно



реальное окружение, в данном примере машина с двигателем. Чтобы ее протестировать нужна трасса, команда инженеров, и много часов работы по управлению этим автомобилем в самых разных ситуациях. Как можно заметить, проведение большого количества таких тестов трудоемкая и дорогая задача. А возможность поломки машины при аппаратной ошибке в тестируемой системе может поставить под угрозу жизнь водителя. Чтобы избежать затрат и упростить тестирование есть возможность смоделировать окружение для определенной системы. Специальные модели могут помочь производить симуляцию каких-либо частей реального мира, в частности, тех, которые возможно моделировать с помощью системной динамики.

В этом месте требуется решать некоторые задачи в реальном времени, так как симулировать приходится реальное окружение и задержки или наоборот ускорения, могут заставить работать тестируемое оборудование абсолютно неправильно. Отсутствие дополнительных накладных расходов на такое моделирование, во-первых, упрощает разработку модели, во-вторых увеличивает её предельную сложность.

Программно-аппаратное моделирование используется во многих отраслях, таких как: робототехника, энергосистемы, силовая электроника, автомобильные системы и другие [6]. Она может применяться для тестирования и разработки практически любого устройства, которое так или иначе получает информацию от внешнего мира.

Удобнее всего подменить на модельное окружение именно аналого-цифровой преобразователь. Это устройство, которое преобразует аналоговый сигнал, например, температуру, в цифровой дискретный, понятный компьютеру и другим системам, например, в число с фиксированной точкой, которое означает температуру в градусах Цельсия. Условная схема приведена на рисунке 2:

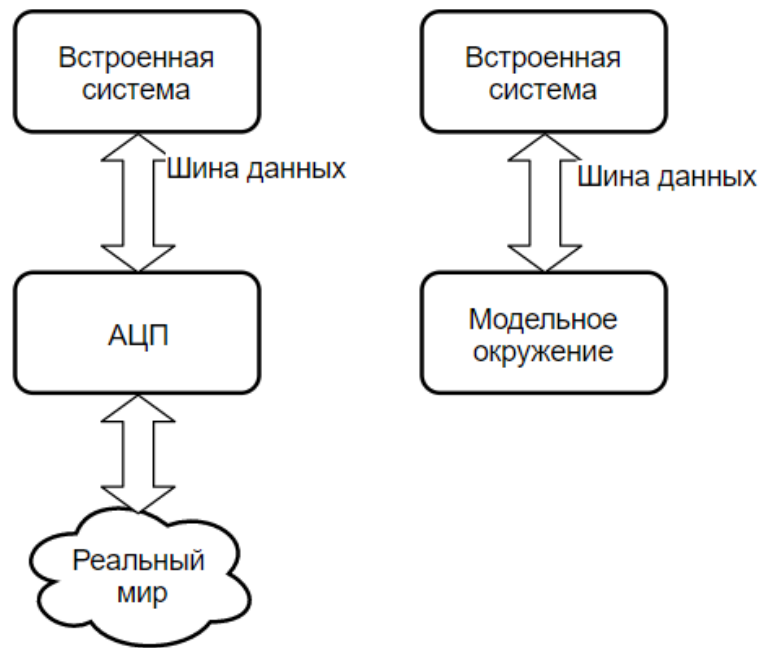


Рисунок 2 – Схема программно-аппаратного моделирования

В устройствах АЦП в основном используются простейшие протоколы, так как сам АЦП часто несложное с технической точки зрения устройство. Обычно они основаны на нескольких контактах и передают данные в двоичном коде с определенной частотой.

Для решения задач аппаратного ускорения и программно-аппаратного моделирования кафедра ИПМ совместно с sdCloud разрабатывают проект, который позволит запускать модели на программируемой логической интегральной схеме с архитектурой NITTA. Про это низкоуровневое устройство, которое позволяет ускорить расчет моделей будет подробно рассказано в главе номер 2. Для управления таким устройством нужен дополнительный модуль, который бы обеспечивал связь с сервером и обработку моделей. Разработка этого модуля и есть финальная задача этой работы. Сформулируем цель:

Разработать систему управления расчетами моделей системной динамики на аппаратном вычислителе реального времени для задач повышения эффективности процесса моделирования и НП.

В эту цель будут входить задачи:

1. Анализ отличительных черт процессов моделирования и анализ способов аппаратной реализации моделирования.
2. Проектирование системы управления.
3. Разработка и проектирование протоколов взаимодействия.
4. Испытание прототипа системы управления.

### **Выводы по главе 1**

В данной главе был проведен обзор области системной динамики, был приведён пример простейшей системно-динамической модели, исследованы средства моделирования. Была сформулирована задача моделирования в реальном времени. Поставлены цель и задачи работы.

## **2 Анализ аппаратной реализации расчетов системной динамики**

### **2.1 Анализ вычислительных платформ**

Моделирование в системной динамике — это итеративный процесс. Мы не можем пропустить какое-то состояние модели или считать состояния параллельно, потому что нам нужно использовать результаты предыдущего вычисления.

Центральный процессор (central processing unit, CPU) – устройство, исполняющее машинные инструкции. Такой процессор создан для того, чтобы решать любые задачи без сильного падения производительности. Центральный процессор исполняет цепочки из инструкций с самой быстрой доступной скоростью. Каждая инструкция в цепочке зависит от предыдущей, поэтому параллельность таких исполнений не определяется процессором напрямую. Но для быстрого исполнения современные процессоры содержат множество оптимизаций. Конвейерная архитектура позволяет ускорить множественное исполнение команд, занимающих несколько вычислительных модулей. Спекулятивное исполнение предугадывает команды, которые будут исполняться, чтобы начать их исполнение раньше. Суперскалярная архитектура позволяет исполнять несколько инструкций за один такт процессора, увеличивая количество исполнительных устройств. Несколько ядер позволяют выполнять несколько независимых цепочек одновременно, но требуется поддержка на уровне кода.

Как же можно ускорить вычисления? Центральный процессор умеет выполнять не более одной цепочки на одном виртуальном ядре, это значит, что для современных процессоров, независимо от количества цепочек команд мы не можем получить ускорение больше чем в  $n$  раз, где  $n$  это количество виртуальных ядер. Существуют также накладные расходы на управление потоками и реальное ускорение, получаемое от параллелизма меньше заявленного  $n$ . Для более узкоспециализованных задач были разработаны другие виды вычислителей,

которые имеют свои преимущества перед центральным процессором. Они будут раскрыты ниже.

Одним из таких вычислителей является графический процессор (graphics processing unit, GPU) – устройство, предназначенное для графической отрисовки трехмерной графики и визуальных эффектов. Он предполагает много однотипных операций с матрицами. Поэтому, в отличие от центрального процессора ориентирован не на быстрое выполнение одной задачи, а на параллельное исполнение одного кода на разных входных данных. Графический процессор состоит из большого количества исполнительных блоков. В современной видеокарте содержится порядка двух тысяч ядер, в то время как в центральном процессоре порядка восьми. В то же время графический процессор отличается большим объемом внутренней памяти, но скорость доступа к этой памяти ниже в сравнении с центральным процессором.

Для более узких задач, требующих самой быстрой скорости работы существует интегральная схема специального назначения (application-specific integrated circuit, ASIC). Это схема с жестко определенными функциями, которые нужны только в заданном продукте. По сути это алгоритм, реализованный на плате. Разработка шаблона для такой платы стоит больших денег и не имеет смысла без массового производства. Само производство при этом одно из самых дешевых, так как используется небольшой необходимый набор дорожек и блоков. Применяется в частности для аппаратного кодирования аудио и видео сигналов.

Еще один вариант технологии разработки узкоспециализированного вычислителя — это быстроразвивающееся направление программируемых пользователем вентильных матриц. Это полупроводниковые устройства на базе программируемых логических интегральных схем или сокращенно ПЛИС (field-programmable gate array, FPGA), которые могут быть сконфигурированы после производства, во время эксплуатации. Такая программируемая матрица работает

медленнее чем интегральная схема специального назначения, но подходит для штучного производства. ПЛИС состоит из регистров и множества логических блоков, которые реализуют базовые операции, такие как ИЛИ, НЕ, И или блоков цифрового сигнального процессора, например встроенный умножитель. В современных процессорах функции блоков не могут изменяться, соответственно и все вычисления работают примерно одинаково. В ПЛИС же с помощью специальных сигналов, логические блоки и соединения между ними могут изменяться. Есть и минусы — это значительное увеличение транзисторов на микросхеме, а значит сложнее и дороже производство, и более существенное ограничение на сложность схемы реализации алгоритма, вызванное предельным количеством блоков.

Есть три основных способа формирования прошивки ПЛИС [7]:

- 1) Register-transfer level (RTL) используя язык описания аппаратуры интегральных схем, например VHDL или Verilog. Требуется больших затрат в плане разработки, с привлечением специально обученного разработчика к проектированию каждой модели. К тому же синтез RTL не быстрый процесс;
- 2) Высокоуровневый синтез. Такой подход предполагает, что мы сможем взять язык высокого уровня, например, C, и перевести его в язык описания аппаратуры интегральных схем. На практике для хорошего результата, пользователю нужно глубоко понимать специфику инструмента высокоуровневого синтеза, так как синтез обычных программ может быть выполнен с большими накладными расходами. Также этот процесс занимает много времени;
- 3) Реконфигурируемый процессор и вычислительные платформы, которые разбивают процесс разработки на две части: аппаратная часть (обычно это автоматически сгенерированная часть RTL с заранее заданной

микроархитектурой) и программная часть, которая легко может быть изменена без проведения RTL синтеза;

В рассматриваемой задаче требуется быстрая подготовка для моделирования, так как модели бывают разных сложностей. Только последний вариант отвечает требованиям быстрого реконфигурирования. Он реализован в том числе на реконфигурируемой вычислительной платформе реального времени NITTA. Этот процессор использует различные типы вычислительных блоков и управляется микрокодом архитектуры MIMD (multiple instructions multiple data), в которой состав вычислительных блоков и их связи изменяются в зависимости от задачи. По сути это очень специфичный процессор с минимальными накладными расходами. Имея смешанную архитектуру VLIW, NISC, TTA, основная работа перекладывается на компилятор, а сама программа после реконфигурирования работает очень быстро. Так как в таких программах нет операций, с неопределенным временем исполнения, таких как чтения из памяти или с устройства ввода. Поэтому вычислительный цикл занимает фиксированное количество тактов, которые не меняются, пока не произойдет реконфигурирование.

## **2.2 Выбор целевой вычислительной платформы**

В прошлом пункте было рассмотрено несколько видов вычислительных платформ. В этом пункте подробно будет рассказано про применение этих платформ к задачам аппаратного ускорения. Мы не рассматриваем задачу программно-аппаратного моделирования в контексте этих вычислительных платформ, потому что построение такой системы на отличной от FPGA платформе представляет собой серьезную инженерную сложность и большие трудозатраты на производство. Например, достаточно сложно передать результаты моделирования на графическом процессоре напрямую в устройство, а соединение к вычислителю через облако в большинстве случаев окажется дольше чем требуемое реальное время.

В связи большой дороговизны первоначального производства, и как следствие не подходящим для прототипирования, в этой главе не рассматриваются интегральные схемы специального назначения. Таким образом выбор идет между графическим процессором и ПЛИС.

В настоящее время графические процессоры находят все большее применение в разных областях, таких как вычисление криптографических хэшей и обучение глубоких нейронных сетей. Но и в этих направлениях графические процессоры не являются специализированными устройствами, скорее это широко распространённые устройства, которые справлялись с задачей лучше, чем центральные процессоры. Чуть позже, когда рынок стал достаточно большим, для тех же целей появились ПЛИС, а потом и специальные интегральные схемы. Например, в недавнем сравнении NVIDIA Titan X Pascal, графического процессора последнего поколения и Intel Stratix 10 2800 FPGA, недавно разработанную ПЛИС, последняя оказалась практически настолько же быстрой на глубоких сверточных нейронных сетях на классическом тридцати двух битном представлении данных и более эффективной на восьми и двух битном представлении данных [8], которые постепенно становятся стандартом в индустрии [9]. И это при том, что при любой точности представления данных FPGA потребляет как минимум на 20 процентов меньше энергии.

Стоит подробнее рассмотреть область моделирования системной динамики, чтобы глубже понять системы, с которыми нам придется работать. Как уже было сказано системная динамика — это система дифференциальных уравнений, которая представлена в виде обычных уравнений с зависимостями от друг друга. Уравнение может быть представлено так:

$$y_i = \sum_{j=1}^n k_{ij} f(y_j)$$



Здесь  $i$  это число от 1 до  $n$ , где  $n$  это количество состояний,  $y_i$  это состояние, а уравнение внутри сигмы — это уравнение которое описывает связь между двумя состояниями. Функция  $f$  может быть любой. Матрицы могут быть эффективно вычислены на графическом процессоре, а эта структура представляется в виде квадратной матрицы размера  $n$ .

Такая система плохо ложится на параллельную структуру графического процессора по нескольким причинам. Первое, это то, что не все состояния связаны, и для матрицы в соответствующем состоянии будет ноль. Так как сложные процессы чаще всего содержат состояния, которые связаны с несколькими другими, но далеко не со всеми, то чаще всего в матричных преобразованиях будут проводиться арифметические операции с нулями, занимая лишнее время и затрачивая на это энергию. Второе это отсутствие какой-либо постоянной структуры в уравнениях. Это не позволяет распараллелить вычисления на графическом процессоре иначе. Итерации, как уже говорилось ранее, пропускать нельзя. Это делает графические процессоры не эффективным устройством для задач системной динамики.

Однако, стоит заметить, что в области системной динамики существует класс задач, которые отлично подходят для параллельных вычислений на графическом процессоре. Это так называемая задача оптимизации. Для некоторых моделей в системной динамике существует задача подбора начальных состояния так, чтобы в итоге система находилась в равновесии. Например, город хочет оптимизировать время работы светофоров. Тогда меняя начальные параметры времени длительности зеленого света на светофорах, можно производить моделирование, и смотреть за результатом, например, на наличие заторов в пиковые часы. Передавая разные начальные значения для модели, графический процессор сможет проводить моделирование сразу в несколько потоков одновременно, и хоть время выполнения одного моделирования будет дольше центрального процессора, большое

количество одновременных процессов позволит осуществить перебор и получить лучшие параметры намного быстрее.

В тоже время ПЛИС будет генерировать свой алгоритм более эффективно. Это будет происходить благодаря системе автоматизированного проектирования, которая по заданному алгоритму генерирует прошивку для ПЛИС. Такая система умеет оптимально использовать ресурсы вычислителя, и на выходе мы получаем параллельный и синхронизированный порядок действий. Стоит отметить что параллельность достигается проще, так как строго задано количество тактов для каждой операции. Соответственно программа будет исполняться параллельно. Также будут практически отсутствовать привычные для центрального процессора накладные расходы в виде синхронизации.

По причинам меньшего потребления энергии, параллельности без накладных расходов и более простой и естественной возможности решать задачу программно-аппаратного моделирования в качестве ускорителя была выбрана архитектура NITTA реализуемая на базе программируемой логической интегральной схемы.

### **2.3 Выбор управляющей системы**

Для обеих задач, как для аппаратного ускорения, так и для машины реального времени для программно-аппаратного моделирования необходим также дополнительный компьютер, который бы реализовывал управление ПЛИС с реализованной архитектурой NITTA.

Первый вариант, который фактически сейчас является стандартной технологией в индустрии это так называемая «система на кристалле» (System-on-a-Chip). Это электронные компоненты, выполняющие функции целого устройства и размещенные на одной интегральной схеме. В нашем случае это должен быть отдельный процессор с сопроцессором на базе NITTA. Соединение между ними должно быть по быстрой шине данных, например, PCI Express. Такая схема должна сама быть установлена как сервер, подключенный к сети по Ethernet или быть

подключенной к серверу напрямую. Этот подход отличается большими затратами на интегрирование и большими затратами на разработку управляющей системы как на PCie для NITTA, так и соответствующего софта для самой платформы. Требуется также дополнительные модули для того, чтобы решать задачу программно-аппаратного моделирования. Разработка такого модуля может занять много времени и быть достаточно ресурсоемкой, а первый прототип можно получить через большой промежуток времени, учитывая, что в реальности ПЛИС с реализованной архитектурой NITTA может оказаться не так хороша для решения задач системной динамики, как кажется в этой работе, по какой-либо независимой причине.

Второй вариант больше направлен на прототипирование, мы можем использовать ПЛИС как отдельный модуль, который будет использовать интерфейсы, популярные во встроенных системах, такие как SPI, I2C или RS485. Эти интерфейсы более простые в использовании и кроме того, могут использоваться в задаче программно-аппаратного моделирования для подключения вместо АЦП. Тогда для управления NITTA мы могли бы использовать высокоуровневые компьютеры.

Сформулируем основные требования к управляющему компьютеру:

- 1) Энергоэффективность. Управляющий компьютер должен потреблять меньше энергии чем обычный компьютер и потреблять не больше чем Nitta, так как это является одной из основных целей нашего проекта;
- 2) Возможность связи с сервером по интернет-каналу. Наличие TCP/IP технологий. Это нужно для передачи данных модели на NITTA и обратно на сервер;
- 3) Наличие базовых протоколов SPI, I2C, RS485 или возможность их реализации. Как было сказано выше для прототипа будет затруднительно

использовать более двух интерфейсов. Лишние интерфейсы потребуют дополнительной реализации на Nitta и усложнят работу;

- 4) Скорость работы не является критичным показателем, так как тяжелые расчеты происходят на Nitta;

Эти требования, характерные для многих встроенных систем, привели к дополнительным поискам такого устройства в области интернета вещей.

Интернет вещей — это концепция, которая заключается в идее, что повседневные физические объекты, такие как холодильник или градусник, подключены к интернету и способны взаимодействовать с остальными устройствами. Тогда повседневные устройства могут объединяться в нечто большее, взаимодействуя друг с другом и окружающим миром через интернет. Большое распространение эта область также получила в контексте понятия «умный дом», когда каждое электрическое устройство подключено к интернету вещей и взаимодействует между другими устройствами, обеспечивая удобные для жителя дома взаимодействия. Например, чайник, подключенный к умному дому может взаимодействовать с будильником. Тогда при пробуждении, будильник отправит сигнал чайнику, и тот автоматически нагреет воду.

Область интернета вещей также имеет обширное коммерческое применение. Связано это в первую очередь с автоматизацией производства и контролем безопасности. Температурные датчики, которые среагируют на высокую температуру, включают тревогу и вызовут пожарных скоро станут стандартом на любом крупном производстве.

Также есть компании, которые обслуживают эту инфраструктуру. Один из первых участников этого рынка это компания Electric Imp, которая предлагает промежуточное решение в области интернета вещей. Компания производит платы со встроенным процессором и Wi-Fi модулем, который позволяет устанавливать соединение с интернетом. Они применяются для создания Internet of Things из

обычных вещей. На платах расположены физические контакты, которые можно соединить с электрическим устройством используя базовые протоколы. Сама плата по интернету соединяется с облаком Electric Imp, которое может использоваться как сервер для общения с остальным интернетом или для централизации нескольких устройств. Сами платы программируются на C-подобном языке Squirrel, который имеет свою виртуальную машину внутри устройства. У этих плат есть весомые преимущества:

- 1) Внутренняя реализация многих базовых протоколов. Плата от Electric Imp содержит реализацию протоколов I2C, SPI и многих других, поддержка которых обеспечивается на уровне библиотек;
- 2) Наличие модуля для интернет-соединения, с использованием собственного протокол поверх TCP. Это упрощает первоначальную настройку и развертывание, а также разработку программ, позволяя сконцентрироваться на логике приложения, а не писать соединения с интернетом;
- 3) Высокая отказоустойчивость. Коммерческое использование продукта, совместно с его закрытой системой позволяет всегда иметь устойчивое к ошибкам приложение. Плата перезагружается в случае падения и может сообщать о проблемах пользователю;
- 4) Платы от Electric Imp энергоэффективны, многие из них работают на пальчиковых батарейках, что позволяет судить о высокой оптимизации энергопотребления;

Таким образом платы полностью соответствуют списку требований, который мы выделили. Из нескольких моделей было выбрано самое мощное устройство imp 005 [10] с следующими техническими характеристиками, написанными в таблице 2:

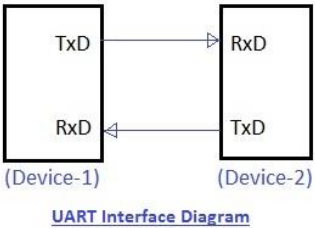
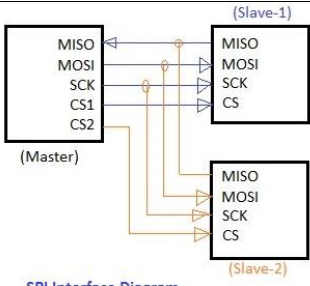
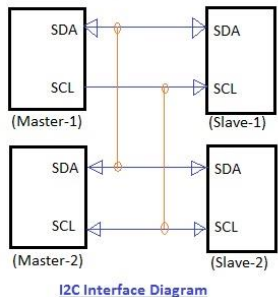
Таблица 2 – Технические характеристики imp 005

Название	imp005
Количество контактов	34
Протоколы	I2C, SPI, UART
Wi-Fi	802.11abgn 2.4/5GHz
Ethernet	10/100Mbps
CPU	ARM Cortex R4, 320MHz
RAM доступная для приложения	1 Mb

#### 2.4 Выбор протоколов физического уровня

Прежде чем будет осуществлен переход непосредственно к проектированию и разработке программного обеспечения, стоит сказать о протоколах физического уровня, которые будут использоваться. В устройстве imp 005 реализовано много протоколов. Как было сказано до этого, для связи с НИТТА нам нужен один протокол, который при этом мог бы использоваться и для программно-аппаратного моделирования. Сложные параллельные протоколы потребуют решения множества дополнительных проблем реализации и синхронизации. Для нашей задачи мы остановились на трех простых и распространённых последовательных интерфейсах: I2C, SPI, UART. В сравнении используются термины ведущий (master) и ведомый (slave), где ведущий это то устройство, которое начинает и заканчивает передачу данных. В этой работе ведущий это устройство от Electric Imp, а ведомый это реализация архитектуры НИТТА на базе ПЛИС. Рассмотрим их преимущества и недостатки в таблице 3 [11][12][13]:

Таблица 3 – Сравнительная характеристика протоколов

Критерий	UART	SPI	I2C
Схема интерфейса	 <p>UART Interface Diagram</p>	 <p>SPI Interface Diagram</p>	 <p>I2C Interface Diagram</p>
Название контактов	<p>TxD: Transmit Data RxD: Receive Data</p>	<p>SCLK: Serial Clock MOSI: Master Output, Slave Input MISO: Master Input, Slave Output SS: Slave Select</p>	<p>SDA: Serial Data SCL: Serial Clock</p>
Пропускная способность	Зависит от устройств, должна быть одинаковой у обоих, 230-460 Кбит/сек	Не определен максимум, обычно 10-20 Мбит/сек	Несколько режимов: 100, 400 или 3500 Кбит/сек
Число ведущих	Нет ведущих, два устройства максимум	Один	Один или больше
Часы	Не используются часы для синхронизации, каждое устройство использует свои часы	Общий сигнал часов между ведущим и ведомыми	Общий сигнал часов между ведущими и ведомыми
Сложность аппаратной части	Низкая	Средняя	Высокая

Адресация	Два устройства, не требуется	Ведомый выбирается установкой соответствующего контакта	Все ведущие могут контактировать со всеми ведомыми
-----------	------------------------------	---	--

#### Преимущества UART:

1. Самый простой в использовании и реализации.
2. Популярный.

#### Недостатки UART:

1. Соединяет только два устройства.
2. Нужно согласование скорости передачи, иначе данные передать не получится.
3. Низкая максимальная скорость.

#### Преимущества SPI:

1. Простой протокол и не требует дополнительных накладных расходов в передаче, как следствие меньше энергозатрат чем в I2C.
2. В любой момент времени может одновременно передавать и принимать данные.
3. Легкое масштабирование на несколько ведомых.
4. Высокая пропускная способность.

#### Недостатки SPI:

1. Требуется большее количество контактов, и требуется на один больше за каждое ведомое устройство.
2. Отношение ведущий/ведомый не может быть изменено, как иногда это делается в I2C.

#### Преимущества I2C:



1. Может быть несколько ведущих и ведомых, они могут меняться ролями.
2. Только два физических контакта.
3. Адресация на уровне программы позволяет проще добавлять новых ведомых.

Недостатки I2C:

1. Увеличивается сложность при увеличении ведущих и ведомых.
2. Одновременно можно либо отдавать данные, либо принимать.
3. Требуется более сложных программ для управления, увеличивает нагрузку на микропроцессор.

Какие требования предъявляются к интерфейсу передачи? Во-первых, требуется скорость, так как результаты моделирования — это большой поток данных, особенно для больших моделей. Во-вторых, нужна энергоэффективность, так как ее повышение это одна из целей. В-третьих, возможность подключения нескольких ведомых устройств к одному ведущему может являться плюсом, так как для расчета нескольких сложных моделей в таком случае мы сможем использовать одно управляющее устройство. В устройстве от Electric Imp есть достаточное количество контактов, и минусы SPI связанные с этим мы не рассматриваем. SPI к тому же, отвечает всем требованиям к интерфейсу. Кроме того, нам не нужно иметь несколько ведущих и ведомых устройств в одном интерфейсе, поэтому этот плюс I2C будет скорее минусом, который перегружает интерфейс на стороне программного обеспечения.

Конечно, кроме задачи аппаратного ускорения, нужно иметь интерфейс, который находит применение в программно-аппаратном моделировании. Один из лидеров в этой области является компания Speedgoat, производящая машины реального времени и программное обеспечение для них. Устройство Speedgoat в частности предоставляет подключение по SPI [14]. Это означает, что наш выбор отвечает условию используемости в программно-аппаратном моделировании.

Стоит также подробнее рассмотреть протоколы для связи с сервером sdCloud. Здесь управляющее устройство imp 005 предлагает два независимых друг от друга варианта.

Первый вариант это Wi-Fi модуль, который включает в себя все современные стандарты 802.11a/b/g/n на частотах 2.4/5GHz. Особенности Wi-Fi модуля в устройствах Electric Imp таковы, что устройство может соединяться только с облачным сервером компании, на котором тоже есть возможность запускать какой-либо код. Сам же сервер, который внутри Electric Imp называется агент(agent), имеет свой выход в интернет, и может осуществлять стандартные действия в сети, например, посылать GET и POST запросы, или получать GET запрос самому. Такая двухуровневая структура, во-первых, позволяет облегчить логику самого устройства, оставляя для него только непосредственную работу, связанную с физическим миром, во-вторых, позволяет обращаться к устройству напрямую, без необходимости иметь собственный выделенный ip-адрес или домен. Это полезно, когда мы хотим иметь много одинаковых устройств для моделирования. Тогда каждое устройство сможет регистрироваться в системе самостоятельно, сообщая о том, что оно готово к моделированию. Это избавит sdCloud от необходимости вводить адреса и способы доступа к устройству вручную.

Второй вариант, доступный только в устройстве imp 005 в качестве прототипа, это Ethernet модуль, позволяющий создавать прямые соединения с каким-либо ip-адресом, в частности локальным. Ethernet отличается большей скоростью и возможностью не задействовать облачные сервера Electric Imp для передачи данных. Не использовать сервера Electric Imp может понадобиться для того, чтобы не быть зависимыми при моделировании от пропускной способности серверов Electric Imp и не нагружать их сеть постоянным трафиком. К тому же, с использованием Ethernet устройство для моделирования можно будет предоставлять в виде отдельного модуля, который после единичной загрузки

модели будет независимо работать, возвращая данные на любой локальный компьютер через Ethernet. Это может быть использовано в программно-аппаратном моделировании, когда результаты представляют из себя коммерческую тайну.

В итоге, было принято решение, что модели и дополнительные начальные данные передаются по Wi-Fi каналу в силу удобства, малого объема и готовых решений со стороны Electric Imp для соединения с сервером, а результаты моделирования будут передаваться через Ethernet, для возможности локальной работы и для независимости большого объема трафика от промежуточных серверов.

На рисунке 3 представлена финальная схема работы управляющей системы и аппаратного ускорителя. Пользователь загружает модель на сайт sdCloud.io, и выбирает расчет на программируемой логической интегральной схеме. Модель загружается в систему, которая по готовности агента управляющего устройства начинает передачу модели. Управляющее устройство, подключенное по Wi-Fi, получает модель от агента и загружает по SPI в устройство на базе NITTA. NITTA по завершению каждого цикла расчетов возвращает результат по SPI на управляющее устройство, а устройство через Ethernet передает этот результат на сервера sdCloud. Сайт sdCloud показывает пользователю результат моделирования. На рисунке 3 изображена схема работы управляющей системы и всего процесса системно-динамического моделирования.

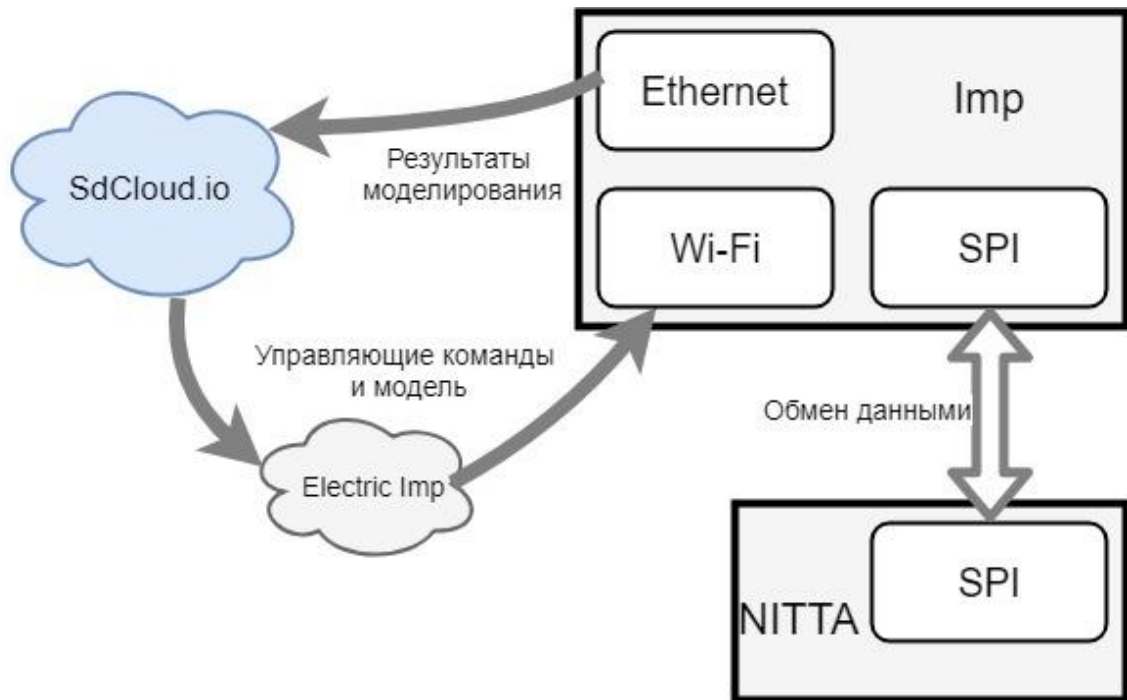


Рисунок 3 – Схема работы управляющей системы

### Выводы по главе 2

Во второй главе был проведен анализ характеристик вычислительных платформ. В качестве целевой платформы была выбрана ПЛИС с реализованной архитектурой NITTA. После анализа было выбрано также и управляющее устройство – imp 005 от компании Electric Imp. Была спроектирована система. В качестве интерфейса общения устройств был выбран SPI, а соединение между сервером и imp будет происходить по Wi-Fi и Ethernet.

### **3 Проектирование и разработка системы управления**

#### **3.1 Особенности разработки на платформе Electric Imp**

Платформа для разработки от компании Electric Imp предлагает много удобных инструментов, таких как сервер с агентом, о котором было упомянуто в предыдущей главе. Такое разделение облегчает работу для людей без опыта промышленного программирования. На сайте Electric Imp находится подробная документация и большое множество примеров. Но есть и свои минусы. Платформа закрытая, исполнять код, написанный на языке Squirrel – единственном поддерживаемом языке, можно только онлайн в облаке компании и на устройстве. Загрузить код напрямую в устройство, не используя облако, не представляется возможным. В устройстве используется виртуальная машина, которая может исполнять код. Файловая система не доступна для пользователя. Энергонезависимая память имеется, но в очень малом объеме.

Несмотря на все эти недостатки, программирование на платформе не представляет сложности для новичков и профессионалов. Во-первых, виртуальная машина позволяет не задумываться об управлении памятью. Во-вторых, этот подход более устойчив к возможным критическим ошибкам, устройство не потеряет связь с сервером и не потребует физических действий для перезагрузки. В-третьих, такой подход облачного программирования позволяет писать программы в браузере и исполнять их в облаке, даже без физического доступа к устройству. Однако, для более привычного программирования есть утилита, которая позволяет разворачивать локальный проект на облачном сервере и получать информацию о процессе исполнения. Внешне это ничем не отличается от локального интерпретатора, нужно лишь подключение к интернету.

Программа для платформы Electric Imp состоит из двух частей. Первая часть — это код для агента, а вторая – код для устройства. На устройстве отсутствуют встроенные библиотеки для работы с интернетом, а на агенте нет встроенных

библиотек для физических интерфейсов. В остальном код полностью совместим и может быть запущен как на агенте, так и на устройстве. Дополнительные библиотеки можно загрузить автоматически, если они есть на серверах Electric Imp. Все библиотеки открытые и распространяются по лицензии MIT. Это позволяет видеть, как внутри платформы реализованы популярные инструменты. Наследуя лучшие решения языка JavaScript для веб-разработки, Squirrel взял модель событийно-ориентированного программирования. На практике это означает, что запросы в интернет выполняются асинхронно и не блокируют основной поток исполнения.

Существует также и встроенный механизм общения устройства и агента, основанный на событиях. В агенте и устройстве есть объект `device` и `agent` соответственно. У обоих объектов есть методы `on()` и `send()`. Метод `send()` принимает строку темы и объект. Метод `on()` принимает строку темы и функцию, которая исполнится, когда придет сообщение с соответствующей темой. Простейший пример такой коммуникации в таблице 4. Код агента находится в колонке `agent`, код устройства находится в колонке `device`:

Таблица 4 – Пример простой коммуникации

Agent	Device
<pre>device.on(   "clock.switch.mode",   false );</pre>	<pre>function switchMode(passedValue) {   if (!passedValue &amp;&amp; hr24Flag) {     hr24Flag = false;     server.log("Clock set to AM/PM");   }   else if (passedValue &amp;&amp; !hr24Flag) {     hr24Flag = true;     server.log("Clock set 24-hour mode");   }    updateDisplay(); }</pre>

Этот код является примером простого обмена сообщением между агентом и устройством. Он переключает отображение времени с двадцатичетырехчасовой формат на двенадцатичасовой, используя тему сообщения «clock.switch.mode».

### 3.2 Проектирование и реализация интерфейса imp – sdCloud

Прежде всего стоит упомянуть об внутреннем устройстве sdCloud. У этого сервиса есть несколько независимых машин, которые выполняют пользовательские модели. Вся серверная часть написана на языке программирования C# с использованием базы данных MongoDB. Задачи распределяет один компьютер с помощью современной библиотеки, которая выполняет функцию брокера сообщений RabbitMQ.

RabbitMQ запускается как отдельный сервис [15]. Общение с этим сервисом происходит по протоколу AMQP (Advanced Message Queuing Protocol) – это открытый протокол, созданный для того, чтобы разные приложения, написанные на разных языках, могли общаться между собой. Есть три основных понятия, которые используются в RabbitMQ:

- 1) Поставщик – это программа, отправляющая сообщения. В нашем случае это сервер, который принимает пользовательские модели;
- 2) Очередь – это структура данных, которая хранит в себе сообщения, в нашем случае это задачи моделирования;
- 3) Подписчик – программа, которая принимает сообщения. В нашем случае это серверы, которые производят моделирование;

Сам RabbitMQ поддерживает большие сообщения, подтверждение выполнения моделей и легко масштабируется. Для того, чтобы добавить дополнительный сервер в такой архитектуре нужно всего лишь стать одним из подписчиков и принимать задачи из очереди. Такой подход позволяет балансировать нагрузку и удобно сохранять модели в очереди, а библиотека берет на себя обязанность сохранности данных.

SdCloud является облачным сервисом и он построен по современным стандартам web-разработки. Прежде всего он поддерживает архитектурный стиль REST [16]. Это значит, что сервер выполняет несколько ограничительных требований. Первое требование – модель клиент-сервер. Это ограничение разделяет код, который исполняется на клиенте и на сервере, с целью сделать серверный код более масштабируемым, а клиентский код более переносимым. Второе ограничение – отсутствие состояния. В архитектуре без состояния вся нужная серверу информация передается вместе с запросом, и состояние как таковое отсутствует. Третье ограничение — это кэширование ответов. Такое требование позволяет ускорить систему, запоминая ответы на частые запросы и не обрабатывая их заново. Четвертое ограничение — это единообразие интерфейса. Все интерфейсы должны быть приведены к общему виду, для удобной работы с API. Последнее ограничение — это наличие слоев для балансировки нагрузки.

Наличие REST архитектуры в sdCloud позволяет управляющему устройству от Electric Imp обращаться к серверу с помощью обычных HTTP-запросов, таких



как POST и GET. Для масштабируемости было решено, что управляющее устройство будет регистрироваться в облаке sdCloud, посредством отправки своего постоянного адреса. Это значит, что sdCloud будет сам решать, когда аппаратный ускоритель с управляющим устройством должен посчитать модель. После принятия решения, сервер sdCloud отправит ссылку на бинарный файл модели на агент. Агент скачает бинарный файл и передаст его на управляющее устройство. Отправка модели с управляющего устройства на аппаратный ускоритель будет описана в следующем пункте главы.

Остаётся только передавать результаты моделирования с устройства на сервер. Так как у серверов sdCloud нет постоянного ip адреса, в текущей реализации устройство и сервер должны находиться в одной сети. Управляющее устройство сможет связаться с сервером по Ethernet. В будущем для этих задач целесообразно выделить отдельный сервер со статическим ip адресом, который собирал бы данные с нескольких ускорителей. Это нужно для того, чтобы не быть привязанным к физическим серверам и иметь возможность перенести структуру в датацентры.

### **3.3 Проектирование интерфейса Imp – NITTA**

Несмотря на то, что в прошлой главе мы уже выбрали интерфейс SPI его спецификация, не является достаточной для нашей задачи. Это значит, что нам нужен какой-то протокол для самих данных, передаваемых между управляющим устройством и ускорителем. Для его разработки сначала разберемся с ограничениями SPI для используемых устройств и составим требования для протокола.

Как уже было сказано, управляющее устройство является ведущим, и оно выбирает, когда начнется передача. Для того чтобы начать передачу нужно сделать сигнал CS активным. Длина одного фрейма кратна 8 битам, и может быть произвольной. Пакет данных готовится перед началом работы с шиной, и

операция передачи данных выполняется синхронно и в дуплексном режиме. Это означает что данные одновременно будут идти с ПЛИС с NITTA на *imr* и наоборот.

На ПЛИС за интерфейс SPI отвечает отдельный блок. Его реализация позволяет выполнять передачу и работу с интерфейсом параллельно с основным циклом моделирования. Это ускоряет моделирование, но создает проблемы синхронизации. На ПЛИС разработано два режима работы. В первом режиме процесс моделирования входит в цикл передачи данных по интерфейсу. Это значит, что следующая итерация не начнется, пока не будут переданы данные. Второй режим нужен для программно-аппаратного моделирования, он подразумевает, что модель считается непрерывно, а данные кладутся в буфер. Из буфера *imr* по готовности забирает данные, для того, чтобы передать их на сервер. С буфером также может работать не управляющее устройство, а устройство для которого выполняется программно-аппаратное моделирование.

Системно-динамическое моделирование предполагает, что формат начальных данных и результата не изменяется. Поэтому, для упрощения протокола можно запоминать формат данных на сервере, управляющем устройстве и закладывать его в модель. Это позволяет оперировать данными между управляющим устройством и ПЛИС без дополнительной обертки, такой как JSON или XML. Этот протокол имеет небольшие дополнительные расходы в сообщении, что напрямую влияет на скорость обмена данными. Системная динамика предполагает, что операции идут только на численных данных, то есть кроме целых и нецелых чисел со знаком никакие другие данные, например, строки, не будут являться результатом моделирования. Все сообщения передаются в двоичном формате, синхронизация передачи происходит с помощью сигнала CLK.

Скорость управляющего устройства в разы ниже, поэтому в любой момент *imr* сможет забирать данные, не опасаясь, что они еще не готовы для передачи с ускорителя.

Сервис sdCloud поддерживает функции останова модели и перезапуска. Поэтому в протоколе должны быть управляющие команды. Кроме того, должна быть возможность отдать команду на ускоритель о том, что управляющее устройство будет загружать новую модель.

Так как собирается прототип, и ожидается, что он будет работать в специальных условиях, в разработке протокола нет задачи обеспечения помехоустойчивости. Кодирование усложнило бы схему ПЛИС и замедлило скорость работы управляющего устройства.

### **3.4 Разработка интерфейса Imp – NITTA**

Теперь рассмотрим непосредственно вопрос разработки и более детально разберем протокол. С самого начала, надо определиться с типами данных. NITTA поддерживает только целочисленную арифметику, а для работы с числами с плавающей запятой требовался дополнительный модуль операций с плавающей запятой. Моделирование физических моделей потребовало бы несколько таких модулей, иначе весь алгоритм упирается в его использование. Вместо чисел с плавающей запятой использовались числа с фиксированной запятой. Их арифметика не меняется на операциях сложения и вычитания и немногим сложнее на остальных операциях, по сравнению с целыми числами. В контроллере такие операции отсутствовали, так что был создан класс `fixedPoint` на базе целых знаковых чисел, который помимо значения числа хранил и номер разряда, после которого стоит запятая. Для удобства поиска ошибок были созданы также методы перевода из чисел с плавающей запятой в фиксированную, и обратно.

Последовательность данных, нужных для корректного считывания и отправки хранит сервер, и присылает ее в виде строки, где каждая буква это определенный тридцати двух битный тип данных, на данном этапе это `i` – integer, целочисленный тип данных и `f` – fixed, тип данных с фиксированной запятой.

Библиотека в `impr` умеет передавать и принимать два типа данных по SPI, это строка в стандартном формате и массив двоичных данных. Для передачи вполне достаточно второй возможности, но управляющее устройство само будет делить на данные, согласно преданной с сервера строки. Управляющее устройство могло бы не разбираться с данными, оперируя двоичными числами, передавая их с сервера на устройство и обратно. Но тогда бы пропала возможность работы устройства без сервера после первичной загрузки модели. Кроме того, для программно-аппаратного моделирования могут потребоваться простейшие преобразования данных в удобный формат, который для NITTA будет труднореализуем, потому что, либо придется перепрошивать ускоритель без помощи `sdCloud`, либо каким-то образом добавлять такие преобразования в системно-динамическую модель, что не всегда просто сделать. Тогда на помощь может прийти `impr`, который с помощью несложных модификаций кода позволит выполнить преобразования и отправить преобразованные данные по назначению. Например, на вход устройства, для которого проводится моделирование.

Так как общение по Ethernet протоколу долгая операция, мы будем складывать несколько полученных по SPI данных в буфер, и отправлять на сервер сразу несколько результатов. Это позволит немного сократить издержки на передачу по Ethernet протоколу. Данные будут передаваться в JSON формате по TCP. Этот протокол гарантирует доставку и правильный порядок сообщений.

Так как было решено работать с данными без какой-либо обертки, возник вопрос о разделении команд и данных. Первый вариант — это отправка специфических данных, которые ПЛИС однозначно понял бы как команду. Например, отправка на один байт больше или меньше, чем предполагает протокол. Но такой подход неизбежно приведет к проблемам при очень больших или очень маленьких объемах данных, для которых добавить или убавить данные означает сломать что-нибудь или не послать ничего. Так как управляющее устройство не так

часто шлет сообщения ускорителю было решено добавить один байт перед каждым сообщением, который означал бы команду. Таким образом NITTA всегда сможет проверять первый байт и определять, есть ли команда в этом сообщении и если есть, то исполнять ее, не пытаясь прочитать все данные. Например, при команде остановки моделирования, `imp` не будет добавлять какие-либо данные, что упрощает наш протокол за маленькую цену в один байт, по сравнению даже с одним числовым значением из четырех байт. Пустой первый байт означает отсутствие команды и тогда обрабатывать входные данные не требуется.

В аппаратном ускорителе случаются ошибки, например, ошибки переполнения. Это значит, что в протоколе должна быть возможность сказать, что посчитанное значение не является правильным. Для этого в протокол будет добавлен байт после каждых семи значений с маской корректности. Это также избавит протокол от проблемы синхронизации, когда вычислительный цикл больше цикла передачи. Если результат еще не готов, то `imp` получит нули, но сможет их отличить от результата из нулей. Последний бит – является контрольным битом, и получается с помощью XOR остальных битов. Этот бит не может гарантировать хорошую защиту от ошибок, но его наличие позволит понять об искажениях в канале, если они появятся. Если в битовой маске один из битов равен нулю, то мы больше не работаем с этим значением. Если не сходится контрольный бит, то мы помечаем все последние семь значений как неправильные. Сервер получает сообщение об ошибке.

В таблице 5 описано назначение всех команд:

Таблица 5 – Команды моделирования

Команда с сервера	Выполняемое действие	Байт
NEW_MODEL	Загрузить новую модель.	11000000
STOP	Остановить вычисление текущей модели.	00110000
NEW_DATA	Загрузить новые начальные данные для текущей модели.	00001100
Нет команды	Не считывать данные с MOSI	00000000

Несмотря на то, что мы выбрали самый быстрый интерфейс его пропускная способность все равно может являться узким местом, так как она напрямую зависит от тактовой частоты процессора управляющего устройства. Процессор *imr* не является сильно производительным, что позволяет нам экономить энергию. Для увеличения пропускной способности возникла идея кодировать данные на основе ряда динамики (*time series*). Такое кодирование позволяет не пересылать все значение целиком, а отправлять только изменение относительно предыдущего шага. Так как обычно в системной динамике шаг моделей небольшой, это позволит переслать в разы меньше данных.

Возникает проблема: как разделять двоичные данные нефиксированной длины? Эту проблему решит сервер, который будет выполнять моделирование на нескольких шагах, чтобы создать протокол с размерами данных ряда динамики. Кроме того, теперь управляющему устройству нужна будет информация о начальных пакетах в протоколе. Информация о каждом значении будет разделяться точкой с запятой, а информация внутри разделяться двоеточием. В информации о значении может быть и дополнительная информация, заранее определенная видом значения. Например, для числа с фиксированной запятой, кроме количества бит на новое значение и начального значения также будет содержаться и бит, после

которого в числе стоит точка. В таблице 6 написан формат протокола для двух типов данных:

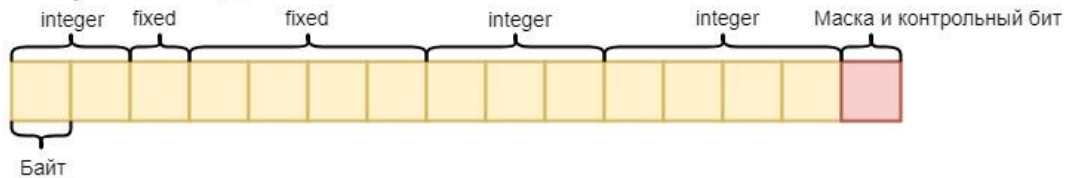
Таблица 6 – Формат протокола

Тип данных	Формат	Пример
integer	i:<кол-во бит>:<нач. значение>;	i:8:1000;
fixed	f:<кол-во бит>:<бит точки>:<нач. значение>;	f:4:4:4.75;

На рисунке 4 изображена структура пакета исходящих и входящих пакетов.

Протокол: **i:16:170;f:8:100;5;f:32:100;i:24:1;i:32:200;**

Полученные данные:



Протокол: **iffi**

Отправленные данные:

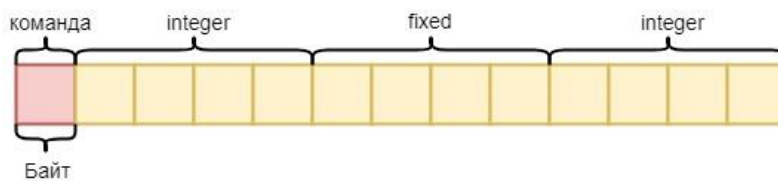


Рисунок 4 – Пример исходящего и входящего пакета данных.

Стоит заметить, что для задач реального времени, в частности для программно-аппаратного моделирования такой формат данных не применим. Это связано с тем, что управляющее устройство может пропустить несколько значений, что является допустимым в реальном времени, и все данные станут некорректными.

После загрузки модели `imr` начинает забирать результаты моделирования из буфера ПЛИС и компоновать их в ответ для сервера `sdCloud`. Если один ответ превышает размер половины буфера, то `imr` отправляет данные на сервер без

сохранения в буфер. Если ответ ожидается слишком большой, то есть больше чем вместится в оперативную память `imp`, то устройство отправляет ошибку на сервер через агента. Мы не проверяем это на агенте, чтобы не менять код агента при изменении модели устройства, например, с большим количеством оперативной памяти. Сервер сам решает, когда моделирование должно прекратиться, если это не заложено в модели. Длительность не ограничена иначе и цикл моделирования может повторяться раз за разом, пока не закончится память на сервере.

После заполнения буфера работа с SPI приостанавливается и происходит отправка сообщений по Ethernet. В то же время в этот момент устройство может принять команду от агента, и после передачи данных устройство отправит команду на ПЛИС.

### **Выводы по главе 3**

В этой главе были спроектированы и разработаны протоколы для передачи данных по SPI, Ethernet и Wi-Fi. Были рассмотрены особенности разработки, основные требования к протоколам. Были реализованы интерфейсы `imp – sdCloud`, и `imp – NITTA`.



## 4 Испытание прототипа системы управления

### 4.1 Сборка и тестирование прототипа

На рисунке 5 изображены управляющее устройство imp 005 с аппаратным ускорителем на отладочной плате DE0-NANO. Соответствующие контакты SPI соединены друг с другом, также соединена земля. Ethernet подключен напрямую к серверу. На рисунке 6 проведена осциллограмма передачи битов между управляющим устройством и аппаратным ускорителем, с CLK сигналом, подключенным к осциллографу.

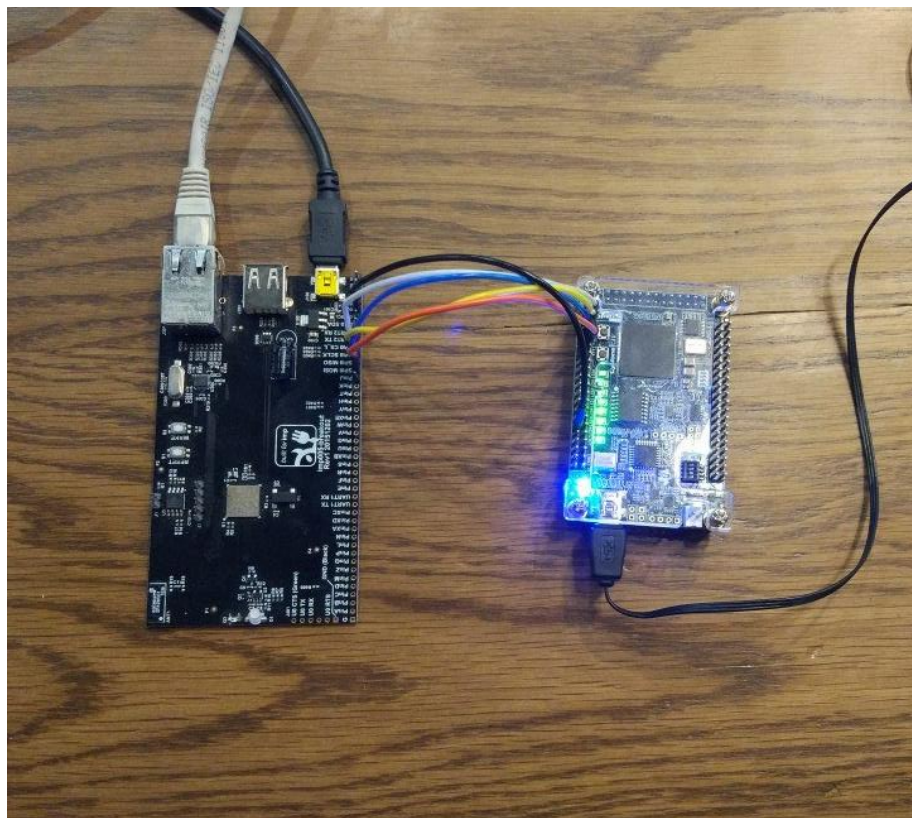


Рисунок 5 – Снимок установки

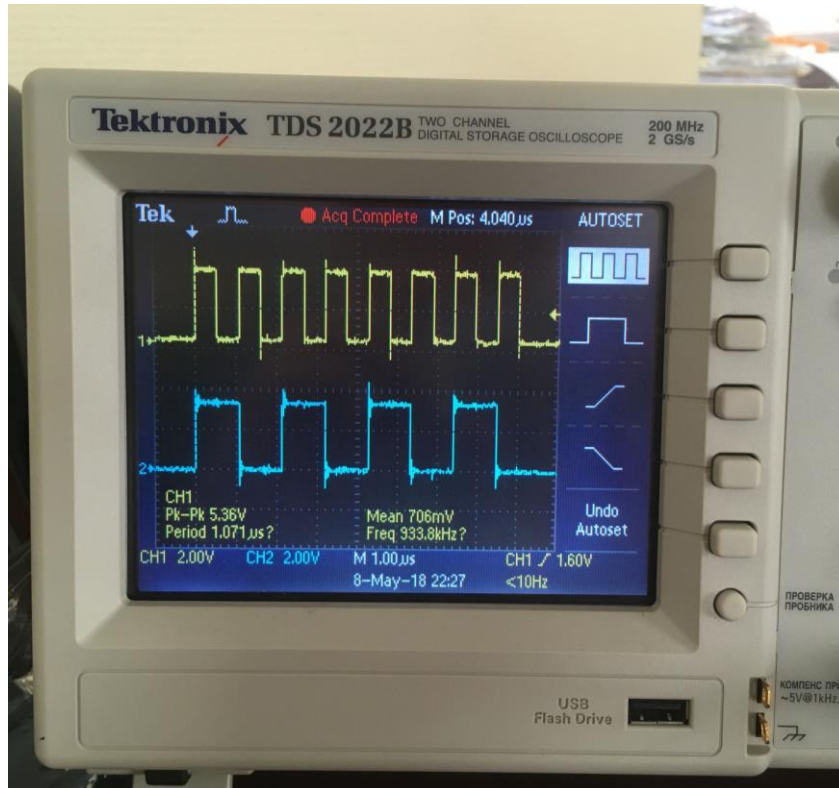


Рисунок 6 – Показания осциллографа

## 4.2 Оценка эффективности

Рассмотрим алгоритм сжатия данных для примера из таблицы 1 первой главы. В реализации без сжатия на один вычислительный цикл необходимо передать 4 значения, все в формате числа с фиксированной запятой, которые занимают 32 бита каждый. Плюс один байт в протоколе задействован как управляющий. Итого мы получаем 136 бит информации за одну итерацию. Для этой же чашки сервер после тестового запуска сгенерировал такой протокол приема: f:6:16:0;f:16:16:6.9;f:16:8:90;f:16:1:21;

Передано  $6 + 16 + 16 + 1$ , что равно 39 бит. Так как минимальное слово равно 8 битам, будет передано 40 бит информации. И один байт управляющий. В сумме получается 48 бит. Это почти в три раза меньше итерации без динамического ряда.

Оценим параметры потребления. Современные процессоры тратят от 40 до 400 Ватт в зависимости от комплектации. Серверные процессоры sdCloud тратят 50 Ватт в режиме ожидания и до 100 Ватт в пиковой нагрузке.

Оценим максимальные параметры энергопотребления. Ускоритель NITTA, согласно спецификации [17] тратит до 2,5 Ватт в пиковой нагрузке. Imp 005 тратит до 2 Ватт с использованием Ethernet. Расчеты показывают, что связка из NITTA и imp эффективнее в пиковой нагрузке, чем простой современный сервер. Это говорит о понижении энергопотребления в разы. Заметим, что есть затраты сервера на подготовку модели, но они занимают не более половины времени расчета модели.

Управляющее устройство тратит порядка двух секунд на загрузку небольшой модели размером 16 Кб, что позволяет нам судить о низкой степени накладных расходов времени на передачу модели.

Также была произведена оценка скорости NITTA на модели расчета чашки. Для 10000 итераций NITTA потратила 30 миллисекунд, в то время как программа для системно-динамического моделирования pySd используемая в sdCloud затратила 700 миллисекунд на исполнение задачи. Такой разный результат прежде всего вызван накладными расходами компьютера и использованием FPU. Сейчас DE-0 Nano рассчитана на 50 стоков из-за размеров платы. Увеличение времени обработки чашки на NITTA с увеличением общего числа чашек связано с ограниченным количеством делителей на плате. При этом решение может масштабироваться. На рисунке 7 изображена зависимость времени расчета модели в секундах от количества чашек в модели для ПЛИС, 10000 итераций.

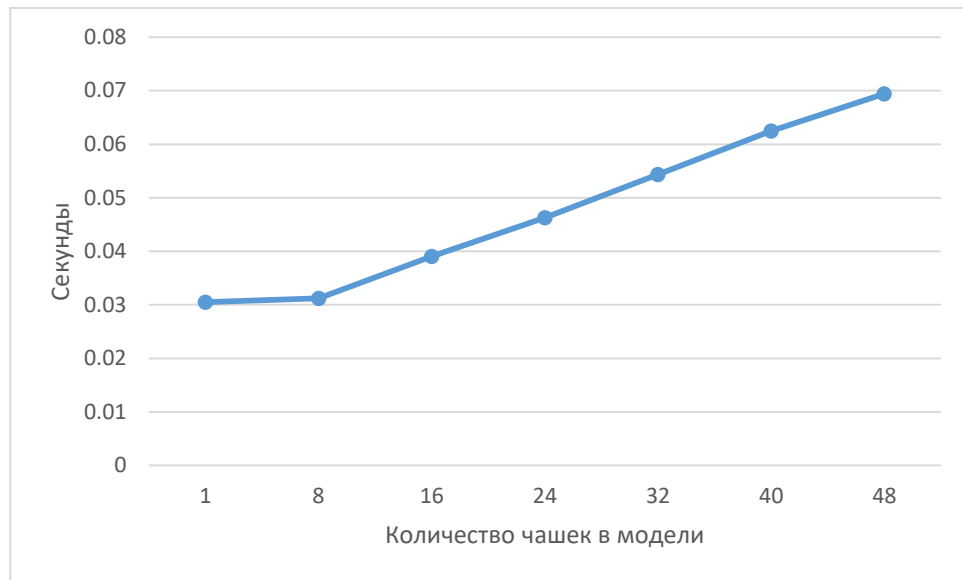


Рисунок 7 – Зависимость времени от количества чашек

На рисунке 8 изображена зависимость времени расчета модели в секундах от количества чашек для CPU на ruSd, 10000 итераций.

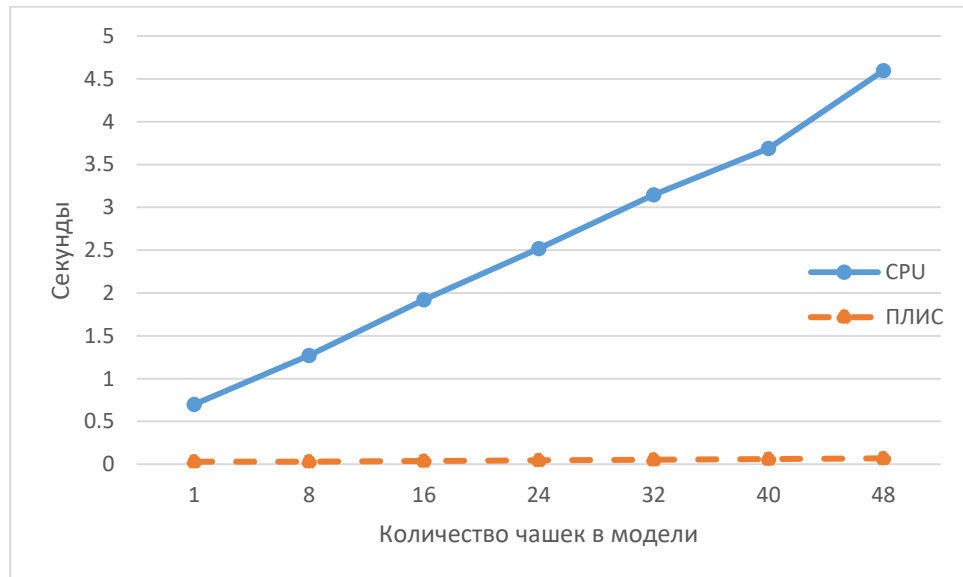


Рисунок 8 – Зависимость времени от количества чашек в модели

Из двух графиков видно, что время возрастает линейно от числа стоков, но ПЛИС проводит моделирование гораздо быстрее. Это значит, что ПЛИС на архитектуре NITTA эффективна для задач системно-динамического моделирования.

Также устройство было протестировано на задаче программно-аппаратного моделирования. На выход SPI после загрузки модели было подключено устройство, которое считывала данные модели из NITTA в реальном времени. Это позволило убедиться в возможности применения связки imp и NITTA в программно-аппаратном моделировании. Тестирование изображено на рисунке 9.

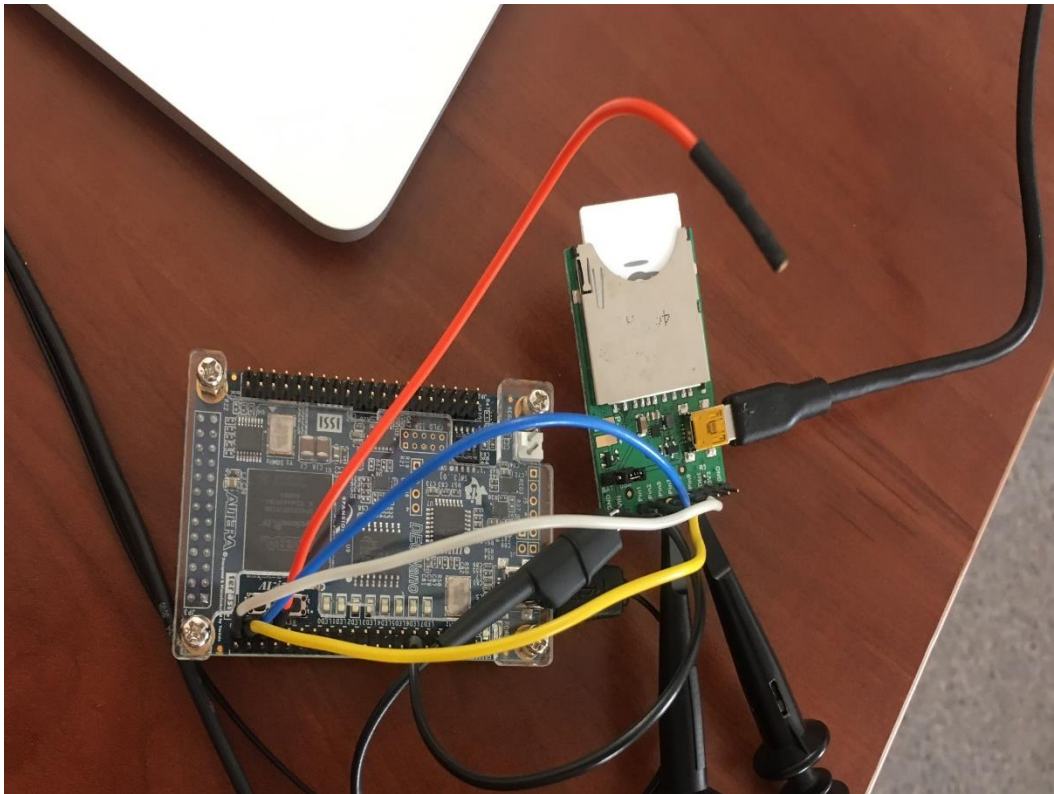


Рисунок 9 – Тестирование программно-аппаратного моделирования

#### **Выводы по главе 4**

В этой главе было изложено описание прототипа, в рамках которого было показано само устройство и разобраны детали его устройства.

Прототип позволил получить метрики производительности и энергоэффективности. Прделанная работа позволила понять, что платформа с архитектурой NITTA имеет перспективы для задач системной динамики. Также были проанализированы результаты применения сжатия данных. Было проведено сравнение исполнения моделей на ПЛИС с обычным серверным компьютером sdCloud, где разработанная система показала лучший результат.

## Заключение

В представленной работе разработана система управления расчетами моделей системной динамики на аппаратном вычислителе реального времени. Поставлены задачи, определена цель. Для достижения поставленной цели проведен подробный анализ процессов системно-динамического моделирования и особенностей их работы. Также рассмотрены варианты акселерации за счет применения различных вычислительных платформ, сделан выбор подходящих устройств. По итогам анализа плюсов и минусов выбраны интерфейсы передачи данных.

Для управляющего устройства разработано программное обеспечение, позволяющее соединять его с сервером по интерфейсам Ethernet и Wi-Fi, а также с ускорителем по интерфейсу SPI.

Разработан и собран прототип, произведено моделирование. Также воспроизведен процесс программно-аппаратного моделирования с использованием заранее загруженной модели.

По итогам проекта можно сделать вывод о том, что разработанное устройство энергоэффективней обычного сервера, а НИТТА производит моделирование быстрее, чем обычный сервер sdCloud. Это позволяет экономить электроэнергию при облачном моделировании и увеличивать скорость.

Все поставленные в работе задачи выполнены, цель достигнута.

## Список используемых источников

[1]. Джон Штерман. Business dynamics: systems thinking and modeling for a complex world [Текст] : Бизнес-процессы: Системное мышление и моделирование сложного мира [пер. с англ.] / Джон Штерман (Sterman John) – Кембридж, штат Массачусетс – 2012. –32 с.

[2]. Лапин А.А. Интерфейсы. Выбор и реализация. — М.: Техносфера, 2015. — 168 с.

[3] Vensim Software/ Vensim [Электронный ресурс] // URL: <http://vensim.com/vensim-software/> (Дата обращения 01.05.2018).

[4] AnyLogic: имитационное моделирование для бизнеса [Электронный ресурс] // URL: <https://www.anylogic.ru/> (Дата обращения 02.05.2018).

[5] XML Interchange Language for System Dynamics (XMILE) Version 1.0, 29 July 2015 [Электронный ресурс] // URL: <http://docs.oasis-open.org/xmile/xmile/v1.0/cos01/xmile-v1.0-cos01.html> (Дата обращения 03.05.2018).

[6] T. Hwang, J. Rohl, K. Park, J. Hwang, K. H. Lee, K. Lee, S.-J. Lee, and Y.-J. Kim, "Development of HIL Systems for active Brake Control Systems", SICE-ICASE International Joint Conference, 2006.

[7] N. Hemsoth and T. P. Morgan, FPGA Frontiers: New Applications in Reconfigurable Computing. Next Platform Press, 2017.

[8] Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Learning Can FPGAs beat GPUs in Accelerating Next-Generation Deep Neural Networks published in ACM Digital Library, February 2017. [Электронный ресурс] // URL: <https://www.nextplatform.com/2017/03/21/can-fpgas-beat-gpus-accelerating-next-generation-deep-learning/> (Дата обращения 04.05.2018).

[9] Imp Hardware Options | Dev Center [Электронный ресурс] // URL: <https://developer.electricimp.com/hardware/imp/modules> (Дата обращения 04.05.2018).

[10] Fixed Point Quantization [Электронный ресурс] // URL: <https://www.tensorflow.org/performance/quantization> (Дата обращения 05.05.2018).

[11] KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART) Literature Number: SPRUGP1 November 2010 [Электронный ресурс] // URL: <http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf> (Дата обращения 10.05.2018).

[12] KeyStone Architecture Serial Peripheral Interface (SPI) Literature Number: SPRUGP2A March 2012 [Электронный ресурс] // URL: <http://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf> (Дата обращения 11.05.2018).

[13] J. Valdez, J. Becker, Understanding the I2C Bus, Texas Instruments, June 2015 [Электронный ресурс] // URL: <http://www.ti.com/lit/an/slva704/slva704.pdf> (Дата обращения 12.05.2018).

[14] Speedgoat SPI Slave and sniffer support for Simulink on FPGAs [Электронный ресурс] // URL: <https://www.speedgoat.com/products-services/communications-protocols/spi> (Дата обращения 14.05.2018).

[15] RabbitMQ- Documentation: Table of Contents [Электронный ресурс] // URL: <https://www.rabbitmq.com/documentation.html> (Дата обращения 15.05.2018).

[16] What is REST [Электронный ресурс] // URL: <http://www.restapitutorial.com/lessons/whatisrest.html> (Дата обращения 16.05.2018).

[17] DE0-Nano User manual (Terasic/Altera) [Электронный ресурс] // URL: <http://www.ti.com/lit/ug/tidu737/tidu737.pdf> (Дата обращения 17.05.2018).