**Содержание**

Введение 5

1 Физический уровень OSI. Модели каналов связи 7

1.1 Линия передачи 7

1.2 Модель длинной линии 9

1.2 Переходные процессы 9

1.3 Физическое кодирование 9

1.4 Постановка задачи 9

2 Алгоримты моделирования цифровой связи 10

2.1 Математическое моделирование длинной линии связи 11

2.2 Задача коши и численный метод решения 11

2.2.1 Выбор численного метода решения 9

2.3 Выводы по главе 2 11

3 Программная реализация стенда для исследования цифровых каналов связи 10

3.1 Алгоритмы физического кодирования 11

3.2 Пользовательский сценарий взаимодействия с ПС 11

3.3 Объектно-ориентированное проектирование ПС 11

3.4 Выбор и обзор инструментов 11

3.4.1 Выбор языка программирования, библиотек, среды разработки 9

3.4.2 Опыт использования инструментов 9

3.5 Выводы по третьей главе 11

Заключение 20

Перечень использованных информационных ресурсов 21

Приложение А Техническое задание 22

Приложение Б Листинг программы 29

Введение

Важнейшим фактором работы предприятий телекоммуникаций, отрасли информационных технологий, а также повседневной жизни обычного человека являются физические каналы или линии связи, использующиеся для обмена радиочастотными электрическими сигналами.

Возникающие при передаче данных падения напряжения на конце кабеля и переходные процессы [5, с. 306] требуют тщательного изучения, так как если не учитывать нюансы передачи сигналов по проводникам, это может привести к искажению, зашумлению и полной потере передаваемой информации.

Также физический уровень передачи информации важен для разработки и тестирования эффективных и надежных АЦП и ЦАП, проектирования алгоритмов канального кодирования. В этих и многих других случаях возникает необходимость оценить их работу в моделируемой ЛП.

Обычно трудности возникают при работе с высокочастотными и импульсными сигналами, которые часто используются в современной технике.

Именно поэтому построение моделей процессов, происходящих при передаче данных на физическом уровне, важно для изучения свойств физического кодирования.

Исследование переходных процессов и поведения напряжения на другом конце линии связи может быть произведено двумя путями:

* осуществление измерений на эксплуатируемых кабелях при помощи осциллографа;
* математическое моделирование и численное приближение процессов, протекающих в электрических кабелях, при помощи средств вычислительных устройств;

Проведение измерений на линиях связи позволяет получить реальное представление о происходящем, однако требует дорогостоящего оборудования и навыков работы с ним, а также, как правило, повреждения кабеля, что в некоторых случаях недопустимо.

По этим причинам лучше проводить исследования не на реальных объектах, а на физических моделях или с помощью компьютерных программ. Применение физических моделей позволяет получить необходимые данные в исследуемой области, но имеет ряд недостатков, таких как большие временные и финансовые затраты. Также физическое моделирование сложно осуществить при большом числе возможных вариантов исследуемой длинной линии, так как для каждого варианта необходимо изготовить физическую модель и провести на ней серию опытов.

Компьютерное моделирование и численное исследование позволяет отказаться от изготовления физических моделей, сократить временные затраты на проведение экспериментов и предоставляет возможность быстро и многократно изменять параметры исследуемого объекта.

С развитием вычислительной техники и существенным увеличением производительности компьютеров стало возможным проведение компьютерного моделирования сложных процессов, к которым и относятся процессы, происходящие в физическом канале при передаче данных.

# 1 Физический уровень OSI. Модель канала связи

В данном разделе приведены теоретические сведения, относящиеся к аспектам процесса передачи информации на физическом уровне.

## 1.1 Линия передачи

Линия передачи (ЛП) – устройство для направленной передачи электромагнитной энергии или электрических сигналов от источника электрических импульсов к нагрузке. Как правило, представляет собой систему проводов, либо в виде кабелей. ЛП используются в различных устройствах на частотах от нуля до соответствующих оптическому диапазону.

Конструкция ЛП во многом определяется областью её применения. Простейшая ЛП состоит из двух проводников, каждый из которых обладает определенным сопротивлением и вносит некоторое затухание.

При длинах волн , значительно превышающих длину линии, электрические процессы в ЛП описываются законом Ома. С уменьшением (практически с ), появляется запаздывание сигнала вдоль линии.

Главными характеристиками ЛП являются диапазон рабочих частот, максимально допустимая средняя и импульсная мощность, волновое сопротивление.   
 Рабочий диапазон частот ЛП обычно выбирается так, чтобы импульсы в ней распространялись с малым коэффициентом затухания и небольшой дисперсией на один тип волны.

Максимально допустимая мощность определяется тепловыми потерями и эффективностью теплоотвода при передаче импульсных напряжений.

ЛП хорошо изучены и для них существует физико-математическая модель, которая называется “модель длинной линии”.

## 1.2 Модель длинной линии

Длинная линия связи — модель линии передачи, продольный размер (длина) которой превышает длину волны, распространяющейся в ней (либо сравнима с длиной волны), а поперечные размеры (например, расстояние между проводниками, образующими линию) значительно меньше длины волны.

Математическая модель длинной линии связи описывается Телеграфными уравнениями [6, с. 249] - парой дифференциальных уравнений, представленных в формуле 1.1, со следующими погонными параметрами:

* погонное сопротивление металла проводников R Ом/м;
* погонная индуктивность L Гн/м;
* погонная ёмкость C Ф/м;
* паразитная проводимость изоляции двух проводников G См/м.

(1.1)

Важной характеристикой длинной линии является режим её работы. Как правило, их разделяют на:

* режим бегущей волны, определяемый наличием только падающей волны, распространяющейся от генератора к нагрузке;
* режим стоячей волны, характеризующийся тем, что амплитуда отраженной волны равна амплитуде падающей;
* режим смешанных волн, в котором часть мощности падающей волны теряется в нагрузке, а остальная часть в виде отраженной волны возвращается обратно в генератор.

Далее будет рассматриваться только режим бегущей волны.

Данная модель учитывает основные эффекты при передаче импульсов по проводникам, в числе которых падение напряжение и переходные процессы.

### 1.2.1 Переходные процессы

В процессе передачи информации на физическом уровне, протекают так называемые переходные процессы - явления в электрических цепях, возникающие при воздействиях, переводящих их новое стационарное состояние, описываемых единичными функциями [5. с. 351]. То есть, — при действии различной коммутационной аппаратуры, например, ключей, переключателей для включения или отключения источника, или приёмника энергии, при обрывах в цепи, при коротких замыканиях отдельных участков цепи и т.д.

При теоретическом анализе переходных процессов в электрических цепях для однозначного определения понятия “резкий”, закон изменения напряжения U выбирают по аналогии с формулой 1.2.

(1.2)

Таким образом, в момент времени напряжение мгновенно изменяется от до , которая далее остается неизменной сколь угодно долго.

Важным свойством электрических цепей является то, что из-за наличия так называемых паразитных эффектов: индуктивностей и ёмкостей, - в реальных электрических цепях получение мгновенного скачка напряжения или тока физически невозможно, так как требует применения схем формирования тока бесконечно большой мощности.

Обращаясь к импульсным сигналам, можно более наглядно раскрыть свойства ЛП и повысить оперативность исследования их характеристик.

Как правило, двоичные данные при передаче в физическом канале, подвергаются предобработке по некоторому алгоритму физического кодирование.

Такое преобразование способствует решению проблем синхронизации, обеспечения приемлемой скорости передачи данных, управления частотами передаваемых импульсов.

## 1.3 Физическое кодирование

Физическое кодирование дискретных данных осуществляется с использованием потенциальных или импульсных кодов. Для представления двоичных нулей и единиц в потенциальных кодах используются разные значения потенциала сигнала, а в импульсных кодах – импульсы разной полярности или перепады потенциала.

С точки зрения физического кодирования цифровой сигнал может иметь два, три, четыре, пять и т. д. уровней амплитуды напряжения, амплитуды тока.

Качество передачи данных, а именно: надежность и достоверность доставки, возможность обнаружения и исправления возникающих ошибок, стоимость реализации, – существенно зависит от выбранного метода цифрового кодирования, который, в свою очередь, в значительной мере определяет пропускную способность среды передачи.

Поэтому, для обеспечения качества передачи данных к методам кодирования предъявляются следующие требования:

* уменьшение спектра сигнала при одинаковой битовой скорости;
* поддержка синхронизации между передатчиком и приемником за счёт наличия в сигналах признаков, на основе которых реализуется самосинхронизация;
* отсутствие постоянной составляющей в сигнале;
* низкая стоимость реализации метода.

Предъявляемые к методам цифрового кодирования требования являются противоречивыми. При этом каждый из методов цифрового кодирования по сравнению с другими обладает своими конкретными достоинствами и недостатками.

## 1.4 Постановка задачи

Итак, изучение этой области обусловлено важностью решения задач повышения надежности передачи данных в промышленной области, эффективности и стоимости построения проводных сетей телекоммуникаций.

Поэтому создание виртуального стенда для изучения переходных процессов и импульсно-частотных характеристик линии связи важно для обеспечения специалистов удобным средством моделирования реального физического уровня передачи информации.

Целью данной работы является создание программного стенда моделирования передачи информации на физическом уровне.

Таким образом, можно выделить перечень задач для создания программного стенда:

* знакомство с численным моделированием физико-технических объектов и систем распределёнными параметрами, методами цифрового кодирования;
* изучение применяемых математических моделей и алгоритмов;
* анализ посылок и допущений, определяющих адекватность модели явлению, вычислительных целей и алгоритмических способов их достижения;
* выбор необходимых и совместимых программных средств, инструментального ПО при реализации и комплексировании программной системы;
* разработка компьютерной модели и её учебно-исследовательского интерфейса.

Важным требованием к программному стенду является наличие графического интерфейса с формой ввода параметров кодера и линии, а также выводом результатов моделирования.

# 2 Алгоритмы моделирования цифровой связи

## 2.1 Математическое моделирование длинной линии связи

Для решения системы телеграфных уравнений в частных производных, представленной в формуле 1.1, с помощью программных средств, необходимо прежде всего сделать замену неким конечным, дискретным соотношением. Это осуществляется применением дискретной по пространству ячеечной модели. Схема ячеечной модели приведена на рисунке 2.1, где i – номер ячейки, n – количество ячеек, t – время.

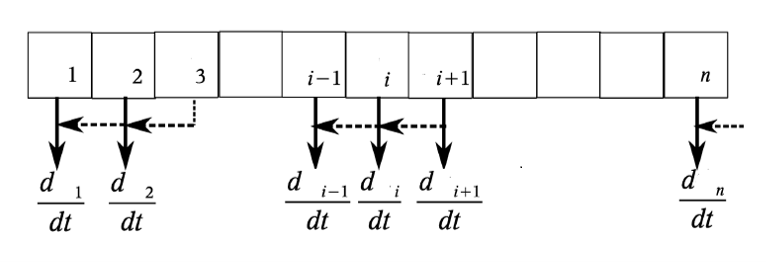


Рисунок 2.1 – Схема метода дискретизации

Длинную линию связи можно рассматривать как распределённую активную и реактивную нагрузку. Выделенный бесконечно короткий участок такой линии характеризуется постоянным активным, ёмкостным и индуктивным сопротивлением переменному току — R, L и C. Такой участок представляет собой классический линейный четырехполюсник.

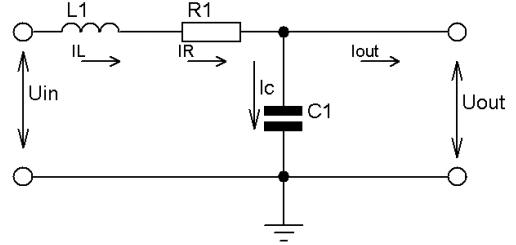


Рисунок 2.2 – Пассивный четырехполюсник

При допущении, что помимо эффектов ёмкости и самоиндукции и рассеяния энергии электричества в линии связи не наблюдается других физических эффектов, участок линии связи может быть приближённо смоделирован пассивным RLC четырёхполюсником, который изображен на рисунке 2,2.

Для такого четырехполюсника выполняются законы Кирхгофа [5, с. 42-33] из формул 2.1, 2.2, а также формулы 2.3, 2.4.

(2.1)

(2.2)

(2.3)

(2.4)

(2.5)

(2.6)

(2.7)

Необходимо отметить, что в формулах 2.1 – 2.7 все параметры I и U - функции времени t.

Из формул 2.1 – 2.4 можно вывести систему ОДУ, характеризующую пассивный RLC четырехполюсник. Система представлена в формуле 2.5, где – входное напряжение, – сила тока на выходе.

Систему из формулы 2.5 можно привести к задаче Коши, которая представлена в формуле 2.7, если найти начальные условия, решив СЛАУ при условии регулярности линии, которое представлено в формуле 2.6.

При предельном переходе система уравнений из формул 2.1 – 2.4 должна быть эквивалентна системе телеграфных уравнений для линии электрической связи. Таким образом, на метров кабеля придётся , , , а на входе приложено напряжение , правее конденсатора в горизонтальной ветви течёт выходной ток .

После такого преобразования получим цепь из последовательно соединенных четырехполюсников, изображенная рисунке 2.2.

Данная модель является хорошим приближением к реальной линии передачи.

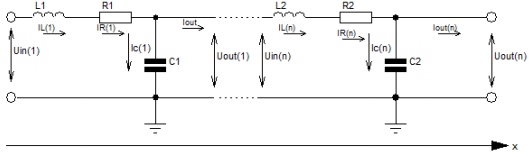


Рисунок 2.3 – Схема модели длинной линии

Математическая модель, характеризующая явления, происходящие, в такой цепи, представляет собой систему дифференциальных уравнений, которая представлена в формуле 2.8.

(2.8)

Эти уравнения построены при допущении, что помимо эффектов ёмкости и самоиндукции и рассеяния энергии электричества в линии связи не наблюдается других физических эффектов и являются неким конечным дискретным соотношением для телеграфных уравнений.

Система ОДУ из формулы 2.8 и будет использоваться в качестве математической модели физического уровня передачи информации.

Полученная задача Коши может быть решена двумя способами: аналитическим и численным, имеющим программно-алгоритмическое воплощение.

## 2.2 Задача коши и численный метод решения

В ходе выполнения данной работы, возникла необходимость решения задачи Коши, общая форма которой представлена в формуле 2.9.

(2.9)

Ввиду того, что в данной работе импульсы представлены сложной функцией от времени, рационально использовать численный метод решения.

Главная идея этих методов заключается в следующем: рассматривается последовательность точек , (0 возможно с переменным шагом .

В каждой точке решение аппроксимируется значением , которое вычисляется по предыдущим значениям . Такие методы называются k-шаговыми.

## 2.2.1 Выбор метода численного интегрирования

Для численного решения задачи Коши существует много различных методов, которые в общем случае делятся на 2 класса:

* дискретные методы, позволяющие найти решение только в узлах сетки. В этих методах каждая вновь вычисленная точка решения уравнения определяется через одну или несколько предшествующих точек. При уменьшении шага сетки последовательность точек аппроксимирует (приближает) решение;
* непрерывно-дискретные методы, основанные на использовании дискретных методов и сплайн-функций для восполнения численных результатов.

В этой работе достаточно дискретных методов, поэтому далее рассмотрены только они.

* метод Эйлера. Существенным недостатком простого метода Эйлера является слишком большая погрешность, при этом погрешность имеет тенденцию накапливаться, что не подходит при моделировании на больших промежутках времени;
* усовершенствованный метод Эйлера. Показывает лучшие результаты в сравнении с обыкновенным методом Эйлера, но при этом недостаточно точен для данной задачи;
* классический метод Рунге-Кутты 4-го порядка. Обеспечивает высокую точность решения, которая оправдывает увеличение объема вычислений. Более высокая точность метода Рунге – Кутта часто позволяет увеличить шаг интегрирования, что необходимо в программном стенде.

Так как методы Рунге-Кутты высокого порядка, в общем случае, имеют большую точность [7], для решения задачи был выбран метод Рунге-Кутты 4-го порядка.

Рассмотрим алгоритм метода Рунге-Кутта 4-го порядка [8, с. 134] для функции f с переменной x и величиной шага h для каждого номера шага по времени i. По этому алгоритму необходимо выполнить:

1. вычислить коэффициенты k:

* ;
* ;
* ;
* ;

1. найти поправку: (;
2. вычислить очередное значение: ;

В нашей модели используется Классический метод Рунге-Кутта четвертого порядка с постоянным шагом, так как:

* этот метод достаточно устойчив и может применяться как для постоянного шага, так и для переменного;
* для вычисления каждой точки решения дифференциального уравнения необходимо знать только одну предшествующую точку, т.е. для решения дифференциального уравнения достаточно иметь начальные условия интегрирования;
* используемый в этом методе четвертый порядок точности является наиболее разумным;
* благодаря относительно простым формулам интегрирования при достаточной точности решения, он удобен для программирования.

Процесс решения уравнений математической модели численным методом при переменных граничных условиях является основой имитационного моделирования – создания программного стенда.

## 2.3 Выводы по второй главе

В данной главе была произведена полудискретизация телеграфных уравнений, решение сведено к задаче Коши, был выбран оптимальный метод численного интегрирования.

# 3 Программная реализация стенда для исследования цифровых каналов связи

В данном разделе описаны используемые в стенде алгоритмы физического кодирования, ожидаемый сценарий взаимодействия с программным стендом, выбор инструментов для реализации, а также разделена функциональность приложения на классы в объектно-ориентированном стиле.

## 3.1 Алгоритмы физического кодирования

Для лучшего изучения процессов, происходящих при передаче данных на физическом уровне, требуется взять методы кодирования с различной максимальной частоты несущей.

Поэтому в этой работе будут рассмотрены следующие трехуровневые коды:

* RZ;
* MLT-3;
* Manchester IEEE 802.3.

В RZ формирование сигнала происходит по следующему правилу: возврат к нулевому уровню происходит после передачи значащего интервала. Информационный переход осуществляется в начале значащего интервала, возврат к нулевому уровню — в середине значащего интервала, далее уровень является нулевым до конца значащего интервала [9, с. 34].

В MLT-3 формирование сигнала происходит по следующему правилу: последовательно переключаются уровни напряжения -1, 0, +1, 0. Он переходит в следующее состояние для передачи 1 бита и остается в том же состоянии для передачи 0 бита [10].

В Manchester IEEE 802.3 логическому нулю соответствует переход на верхний уровень в центре битового интервала, логической единице - переход на нижний уровень. Логика кодирования хорошо видна на примере передачи последовательности единиц или нулей. При передаче чередующихся битов частота следования импульсов уменьшается в два раза.

Информационные переходы в средине бита остаются, а граничные (на границе битовых интервалов) - при чередовании единиц и нулей отсутствуют. Это выполняется с помощью последовательности запрещающих импульсов. Эти импульсы синхронизируются с информационными и обеспечивают запрет нежелательных граничных переходов.

Изменение сигнала в центре каждого бита позволяет легко выделить синхросигнал. Самосинхронизация дает возможность передачи больших пакетов информацию без потерь из-за различий тактовой частоты передатчика и приемника [10].

## 3.2 Пользовательский сценарий взаимодействия с ПС.

Сценарий взаимодействия с программным средством представляет собой простую последовательность шагов:

Шаг 1. Ввод параметров моделирования.

Шаг 2. Запуск моделирования.

Шаг 3. Ожидание результата или остановка и переход к шагу 1.

Шаг 4. Отображение результата в виде графиков.

Блок-схема шагов сценария представлена на рисунке 3.1.

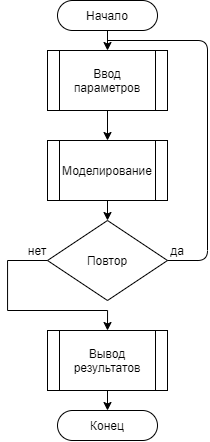


Рисунок 3.1 – Главный пользовательский сценарий

### 3.3 Объектно-ориентированное проектирование ПС

Весь функционал, требуемый от данного ПС можно условно разбить на несколько классов:

От класса “MainWindow” требуется обеспечение ввода параметров, запуска и остановки моделирования, должен содержать следующие методы:

* метод init должен инициализировать поля параметрами по умолчанию и устанавливающий ограничения;
* метод start должен запускать процесс моделирования;
* метод stop должен останавливать моделирование;
* метод draw должен выводить выходное напряжение в виде графиков;
* метод clear должен графики;
* поле stopped должно отражать состояние – моделирование или ввод параметров.

Интерфейс “Encoder”, представляющий схему кодирования данных, содержащий следующие методы:

* метод encode - это главный метод, который должен отвечать за кодирование. Принимает на вход данные, возвращает последовательность напряжений для моментов времени ;
* поле zero – уровень напряжения, взятый за ноль;
* поле step – шаг в уровнях напряжения кодера.

Классы “RZEncoder”, “ManchesterEncoder”, “MLT3Encoder”, должны реализовать интерфейс “Encoder” согласно алгоритму цифрового кодирования.

Главный класс программы “Machine”, должен отвечать за численное моделирование со следующими методами:

* поле opt – параметры проводника;
* поле N – число четырехполюсников;
* метод initState должен принимать значение входного напряжение и силы тока на выходе в момент и возвращать начальные условия;
* поле payload хранящее нагрузку в виде массивов значений силы тока и напряжения в моменты времени , и шага h;
* метод processNextPayload должен возвращать последовательность напряжений для моментов времени .

Связи между классами удобно изображать на основе стандарта языка UML, который используется для спецификации, визуализации, конструирования и документирования системы.

UML-диаграмма данных классов, представленная на рисунке 3.2, а также описание классов, представленное выше, отражает общую схему классов в ПС и не является строгим требованием.

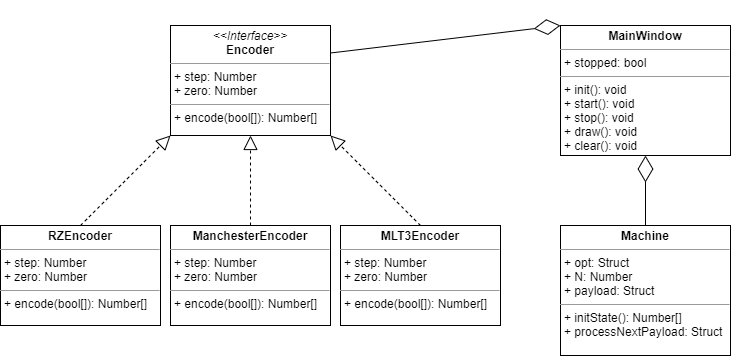


Рисунок 3.2 – UML-диаграмма классов ПС.

### 3.4 Выбор и обзор инструментов

Выбор программного обеспечения обуславливается спецификой языка программирования, удобством использования, возможностью запуска на различных платформах, содержанием инструмента для решения сложных математических задач, взаимодействия с пользователем и осуществлением ввода-вывода, а также доступностью.

### 3.4.1 Выбор языка программирования, библиотек, среды разработки

Главными требованиями к языку программирования является наличие библиотек для создания графического пользовательского интерфейса, библиотек компьютерной математики, средства построения графиков функций.

Рассмотрим наиболее популярные в настоящий момент языки программирования:

Язык Python часто используется в научных вычислениях, так как обладает относительно простым синтаксисом, а также для него написано много библиотек, предназначенных для выполнения научных и инженерных расчётов, однако, помимо всех положительных качеств, Python является интерпретируемым языком программирования, что отражается на производительности.

Язык C++ широко используется для разработки программного обеспечения с повышенными требованиями к производительности. Также, как и для Python, на C++ написано множество библиотек, предназначенных для научных расчетов.

Учитывая все преимущества и недостатки этих языков, был сделан выбор в пользу C++ ввиду его производительности, которая является самым важным требованием при большой разрешающей способности разностной схемы.

Помимо этого, C++ обладает следующими положительными качествами:

* модульность и раздельная компиляция;
* обработка исключений;
* абстракция данных;
* объявление типов (классов) объектов;
* наличие стандартной библиотеки с широкими возможностями;
* сочетание свойств высокоуровневых и низкоуровневых языков;
* компиляция программного кода непосредственно в машинный код.

Разработка осуществляется в интегрированной среде разработки (IDE) Qt Creator. Данная среда разработки является кроссплатформенной и свободно распространяемой, поддерживает возможность расширения функционала через установку дополнительны плагинов.

Для компиляции используется компилятор GCC, работающий под операционной системой Linux и также свободно распространяемый.

Для реализации численного решения математической модели, являющейся системой дифференциальных уравнений, необходимо найти решение системы линейных алгебраических уравнений, а также численно проинтегрировать систему дифференциальных уравнений.

Для первой задачи выбрана библиотека Eigen, которая содержит определения подходящих типов данных и определение кода подпрограмм (функций) для решения СЛАУ, является кроссплатформенной и свободно распространяемой. Она также имеет развернутую документацию.

Для решения задачи Коши выбрана библиотека boost::odeint, которая содержит реализации алгоритмов решения дифференциальных уравнений, в том числе классический метод Рунге-Кутты 4-го порядка. Является кроссплатформенной и свободно распространяемой, имеет отличную документацию.

Для реализации взаимодействия с пользователем, ввода им исходных данных и вывода результатов моделирования необходимы дополнительные библиотеки с возможностью вывода результатов в графической форме, выбрана библиотека (фреймворк) Qt, которая содержит компоненты графического интерфейса, функции ввода-вывода информации, является кроссплатформенной и распространяемой по универсальной общественной лицензии (GNU GPL).

### 3.4.2 Опыт использования инструментов

Выбор C++ в качестве языка программирования оказался верным решением. Стандартная библиотека libstdc++ содержит реализации всех необходимых структур данных, кросс-платформенные обёртки над средствами ОС и примитивы синхронизации.  
 Были использованы потоки для асинхронных вычислений модели линии связи, а также атомарные переменные для остановки расчёта по запросу пользователя из потока интерфейса.

Все это позволяет писать код, который можно скомпилировать практически для любой ОС и архитектуры процессора.

Среда разработки Qt Creator позволила без особых усилий сверстать пользовательский графический интерфейс, обеспечила удобным редактором текста с подсветкой синтаксиса, помогла провести отладку программы.

В фреймворке Qt есть все необходимые инструменты для считывания пользовательского ввода – текстовые, числовые, выпадающие списки выбора, гибкая система обработки событий – SLOTS, а также средства вывода графиков. Также, важно отметить, что это динамическая библиотека, однако все требуемые в Т.З. платформы поддерживаются.

Библиотеки Eigen и boost::odeint предоставляют все необходимые примитивы для численных решений задач линейной алгебры и систем дифференциальных уравнений – реализации матриц, векторов и операций над ними, а также удобный интерфейс вызова методов интегрирования дифференциальных уравнений.

## 3.5 Вывод по третьей главе

В данном разделе был описан типовой сценарий взаимодействия пользователя с ПС, создана объектно-ориентированная модель ПС, осуществлен обоснованный выбор инструментов для реализации, а также описан опыт исполнителя при работе с инструментами.

Заключение

В ходе прохождения преддипломной практики были решены следующие задачи.

Проведен обзор предметной области, построена математическая модель длинной линии связи, выбраны необходимые инструменты для создания программного средства.

Обоснован выбор программного инструментария. В качестве языка программирования был выбран C++, для создания графического интерфейса была выбрана библиотека Qt, для решения систем линейных алгебраических уравнений – библиотека Eigen, для решения системы дифференциальных уравнений – библоитека boost::odeint.

Проведено конструирование программного стенда моделирования передачи информации на физическом уровне, отвечающее всем требованиям технического задания.

Разработана документация на программное средство, обоснована безопасность и экологичность реализуемого продукта.

Перечень использованных информационных ресурсов

1. Официальная документация по фреймворку Qt [Электронный ресурс], URL: https://doc.qt.io/qt-6/reference-overview.html (дата обращения: 15.05.2021).
2. Официальная документация по библиотеке Eigen [Электронный ресурс], URL: https://eigen.tuxfamily.org/dox/index.html (дата обращения: 15.05.2021).
3. Официальная документация по библиотеке boost::odeint [Электронный ресурс] URL: https://www.boost.org/doc/libs/1\_66\_0/libs/numeric/odeint/doc/html/index.html (дата обращения: 15.05.2021).
4. Peter Van Weert, Marc Gregoire «C++17 Standard Library Quick Reference: A Pocket Guide to Data Structures, Algorithms, and Functions 2nd ed. Edition», «Apress», 2019.
5. В.П. Попов «Основы теории цепей», ФГУП «Издательство «Высшая школа», 2003.
6. David H. Staelin «Electromagnetics and Applications», «MIT OpenCourseWare», Spring 2009.
7. Картавченко А.Е., Тархов Д.А «Сравнение методов построения приближенных аналитических решений дифференциальных уравнений на примере элементарных функций», Санкт-Петербургский политехнический университет Петра Великого, г. Санкт-Петербург, Россия, 2017.
8. E. Hairer, S.P. Nørsett, G. Wanner «Solving Ordinary Differential Equations I», Springer, Second Revised Edition, 1987.
9. John B. Anderson Б.В «Digital Transmission engineering», Lund University, 2005.
10. Статья по методам кодирования [Электронный ресурс] URL: https://lanberry.ru/lan/standarti\_otkritih\_kabeljnih\_sistem (дата обращения: 15.05.2021).

Приложение А Техническое задание

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент кафедры «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_ Н.В. Кудинов  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г. | УТВЕРЖДЕНО  Зав. кафедрой «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_ В.В. Долгов  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2021 г. |

## А.1. Общие сведения

### А.1.1 Имя программы

«PLDTM»

### А.1.2 Полное наименование ПС

Программный стенд моделирования передачи информации на физическом уровне «PLDTM»

### А.1.3 Перечень документов, на основании которых создается система

Основанием для разработки является задание к преддипломной практике, согласованное с руководителем ВКР доцентом кафедры «ПОВТиАС» Кудиновым Никитой Валерьевичем с одной стороны, студентом гр. ВПР41 Бачуриным Данилой Дмитриевичем, именуемым в дальнейшем исполнителем с другой стороны, утвержденному заведующим кафедрой «Информационные технологии» Долговым Василием Валерьевичем.

### А.1.4 Порядок оформления и предъявления заказчику результатов работ по созданию системы

Система передается в виде функционирующего комплекса на базе средств вычислительной техники Заказчика и Исполнителя. Приемка системы осуществляется комиссией в составе уполномоченных представителей Заказчика и Исполнителя.

### А.1.5 Перечень нормативно–технических документов, методических материалов, использованных при разработке ТЗ

При разработке автоматизированной системы и создании проектно–эксплуатационной документации Исполнитель должен руководствоваться требованиями следующих нормативных документов:

ГОСТ 19.201–78. Техническое задание. Требования к содержанию и оформлению.

ГОСТ 34.601–90. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания;

ГОСТ 34.201–89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплексность и обозначение документов при создании автоматизированных систем;

РД 50–34.698–90. Методические указания. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов.

## А.2 Назначение программного средства

Функциональным назначением программного средства является изучение свойств линий передачи информации. Эксплуатироваться программное средство будет в учебной организации.

## А.3.Требования к программному средству

### А.3.1 Требования к программе в целом

#### А.3.1.1 Требования к дизайну графического интерфейса ПС

Дизайна интерфейса ПС находится в ответственности Исполнителя.

#### А.3.1.2 Требования к шрифтовому оформлению ПС

Основным шрифтом должен стать Times New Roman*.*

Размер (кегль) шрифтов должен обеспечивать удобство восприятия текста при минимально допустимом размере экрана.

#### А.3.1.3 Требования к средствам выполнения ПС

Программа должна корректно выполняться в следующих ОС:

* Linux (версия 2.6 и выше);
* Windows 10;
* macOS (версия 10.0 и выше).

#### А.3.1.4 Требования к архитектуре ПС

ПС должна представляться единым монолитным исполняемым модулем, статически скомпилированным.

#### А.3.1.5 Язык интерфейса программы

Русский.

### А.3.2 Требования к функциям (возможностям), выполняемых программой

#### А.3.2.1 Основные требования

Программа должна представлять собой исполняемый файл с исходными кодами, размещенными на физическом накопителе.

## A.4 Требования к программе или программному изделию

### А.4.1 Требования к функциональным характеристикам

Программное средство должно осуществлять следующие функции:

* предоставлять пользователю форму для ввода параметров линии передачи, кодера, моделирования;
* вывод результата моделирования в виде графиков функций входного напряжения и выходного, зависящих от времени.

### А.4.2 Требования к надежности

Надежное функционирование программы должно быть обеспечено выполнением совокупности нижеописанных мероприятий:

* выполнение требований ГОСТ 51188-98;
* защита информации;
* организацией бесперебойного питания технических средств;
* использованием лицензионного программного обеспечения.

### А.4.3 Условия эксплуатации

Для функционирования программного продукта необходимо соблюдение всех требований и правил эксплуатации компьютерной техники.

Высокая квалификация пользователя программного средства не требуется. Дополнительных требований и ограничений не вводится.

Требования к персоналу, работающему с данным программным продуктом – общие знания вычислительной техники.

### А.4.4 Требования к составу и параметрам технических средств

### В состав технических средств должен входить IВМ-совместимый персональный компьютер (ПЭВМ), выполняющий роль сервера, включающий в себя:

### процессор Pentium - 2.0 GHz, не менее;

### оперативную память объемом, 2 Гигабайт, не менее;

* ПЗУ объемом не менее 256 Гигабайт, со свободным дисковым пространством не менее 100Мегабайт.

### А.4.5 Требования к информационной и программной совместимости

#### А.4.5.1 Язык программирования

Для разработки программного средства должен быть использован язык программирования С++.

#### А.4.5.2 Операционная система

Программное средство должно работать под управлением любой ОС из списка:

* Linux (версия 2.6 и выше);
* Windows 10;
* macOS (версия 10.0 и выше).

### А.4.6 Требования к упаковке и маркировке

Требования к упаковке и маркировке программного средства не предъявляется.

### А.4.7 Требования к транспортировке и хранению

Условия транспортирования, места хранения, условия складирования и сроки хранения в различных условиях должны соответствовать требованиям, предъявляемым к носителям информации, на которых будет содержаться данное программное изделие.

Допустимы все способы транспортирования и хранения, не нарушающие целостность используемого носителя данных. Программное средство может храниться на любом носителе информации, имеющее возможность подключения к персональному компьютеру.

### А.4.8 Специальные требования

Специальных требований не предъявляется.

## А.5 Требования к программной документации

Программная документация должна состоять из следующих листов:

* титульный лист;
* пояснительная записка к производственной практике;
* техническое задание по ГОСТ 19.201-78 ЕСПД;
* исходный код программного средства по ГОСТ 19.401-79 ЕСПД.

## А.6 Стадии и этапы разработки

* постановка задачи (с 20.04.21 по 22.04.21);
* изучение предметной области (с 23.04.21 по 26.04.21);
* разработка алгоритмов решения задачи (с 27.04.21 по 02.05.21);
* разработка программы (с 04.05.21 по 11.05.21);
* тестирование программы (с 11.05.21 по 14.05.21).

## А.7 Порядок контроля и приемки

Порядок и контроль приёмки определяются заведующим кафедрой «ПОВТиАС» и основаны на демонстрации знаний технологии и умении создавать программные средства для различных предметных областей.

Главным требованием к приемке является наличие правильно работающего программного средства с тестовым примером и отчета, представленного в печатном виде.

Разработчик технического задания /Бачурин Д.Д./

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Приложение Б Листинг программы

Листинг Б.1 – Заголовочный файл кодеров

#ifndef ENCODER\_H

#define ENCODER\_H

#include <vector>

#include <string>

using namespace std;

class Encoder {

public:

virtual vector<double> encode(const string& s) = 0;

};

class RZEncoder : Encoder {

private:

double uPos;

double uNeg;

double zero;

size\_t halfCnt;

public:

vector<double> encode(const string& s) override;

RZEncoder(double zero, double step, size\_t halfCnt) : uPos(zero+step), uNeg(zero-step), zero(zero), halfCnt(halfCnt) {}

};

#endif // ENCODER\_H

Листинг Б.2 – Исходный код кодеров

#include "encoder.h"

vector<double> RZEncoder::encode(const string &s) {

auto res = vector<double>();

for (auto c : s) {

double val = c == '0' ? uPos : uNeg;

for (int i = 0; i < halfCnt; i++) {

res.push\_back(val);

}

for (int i = 0; i < halfCnt; i++) {

res.push\_back(zero);

}

}

return res;

};

Листинг Б.3 – Заголовочный файл расчетного модуля моделирования

#ifndef MACHINE\_H

#define MACHINE\_H

#include <array>

#include <iostream>

#include <eigen3/Eigen/Dense>

#include "types.h"

#include <stop\_token>

#include <optional>

#include <atomic>

using namespace std;

using namespace Eigen;

class Machine {

typedef vector<double> state\_type;

RLC opt;

int N;

double RN;

double NL;

double NC;

Payload pl;

auto initState(double U\_in, double I\_out);

auto f(double Uin, double y, double z);

auto g(double Iout, double y, double z);

auto iter(const state\_type &x, state\_type &dxdt, double t);

public:

void appendPayload(const Payload& p);

Payload processNextPayload();

optional<Payload> processNextPayloadStoppable(reference\_wrapper<atomic\_bool> shouldStop);

Machine(int amount, const RLC& lOptions);

~Machine();

};

#endif // MACHINE\_H

Листинг Б.4 – Исходный код расчетного модуля моделирования

#include <array>

#include <iostream>

#include <eigen3/Eigen/Dense>

#include "types.h"

#include "machine.h"

#include <boost/array.hpp>

#include <boost/numeric/odeint.hpp>

using namespace std;

using namespace Eigen;

using namespace boost::numeric::odeint;

auto Machine::initState(double U\_in, double I\_out) {

const int l = 2\*N;

MatrixXd A = MatrixXd::Zero(l, l);

VectorXd b = VectorXd::Zero(l);

b(0) = U\_in;

b(l - 1) = I\_out;

A(0,0) = RN;

A(0,1) = 1;

A(1,0) = 1;

if (N > 1) {

A(1,2) = -1;

A(l-2, l-3) = 1;

A(l-2, l-2) = -RN;

A(l-2, l-1) = -1;

A(l-1, l-2) = 1;

}

for (int i = 1; i < N-1; i++) {

const int idx = i\*2;

A(idx, idx-1) = 1;

A(idx, idx) = -RN;

A(idx, idx+1) = -1;

A(idx+1, idx) = 1;

A(idx+1, idx+2) = -1;

}

VectorXd x = A.householderQr().solve(b);

vector<double> I = vector<double>(N);

vector<double> U = vector<double>(N);

for (int i = 0; i < N; i++) {

const int idx = i\*2;

I[i] = x(idx);

U[i] = x(idx+1);

}

return make\_pair(I,U);

}

void Machine::appendPayload(const Payload& p) {

pl.tau = p.tau;

pl.I = p.I;

pl.U = p.U;

}

optional<Payload> Machine::processNextPayloadStoppable(reference\_wrapper<atomic\_bool> shouldStop) {

Payload res;

res.tau = pl.tau;

auto [I0, U0] = initState(pl.U[0], pl.I[0]);

state\_type state = vector<double>(2\*N);

boost::numeric::odeint::runge\_kutta4\_classic<state\_type> rk;

for (int i = 0; i < N; i++) {

const int idx = i\*2;

state[idx] = I0[i];

state[idx+1] = U0[i];

}

auto& U\_in = pl.U;

auto& I\_out = pl.I;

double h = pl.tau;

size\_t l = min(U\_in.size(), I\_out.size());

vector<double> I\_last = vector<double>(l);

vector<double> U\_last = vector<double>(l);

size\_t i = 0;

const int lIdx = N\*2 - 1;

do {

atomic\_bool& stop = shouldStop.get();

if (stop.load()) {

return nullopt;

}

const double t = i\*h;

I\_last[i] = state[lIdx-1];

U\_last[i] = state[lIdx];

rk.do\_step(

[&](const state\_type &x, state\_type &dxdt, double t) {

double time = (t/h);

size\_t i = (size\_t)time;

dxdt[0] = (U\_in[i] - RN\*x[0] - x[1])/NL;

dxdt[1] = (x[0] - x[2])/NC;

for (int i = 1; i < N-1; i++) {

const int idx = i\*2;

dxdt[idx] = (x[idx - 1] - RN\*x[idx] - x[idx+1])/NL;

dxdt[idx+1] = (x[idx] - x[idx+2])/NC;

}

const int lIdx = N\*2 - 1;

dxdt[lIdx - 1] = (x[lIdx - 2] - RN\*x[lIdx - 1] - x[lIdx])/NL;

dxdt[lIdx] = (x[lIdx - 1] - I\_out[i])/NC;

},

state,

t,

h

);

i++;

} while(i < l);

res.I = I\_last;

res.U = U\_last;

return res;

}

Payload Machine::processNextPayload() {

Payload res;

res.tau = pl.tau;

auto [I0, U0] = initState(pl.U[0], pl.I[0]);

state\_type state = vector<double>(2\*N);

boost::numeric::odeint::runge\_kutta4\_classic<state\_type> rk;

for (int i = 0; i < N; i++) {

const int idx = i\*2;

state[idx] = I0[i];

state[idx+1] = U0[i];

}

auto& U\_in = pl.U;

auto& I\_out = pl.I;

double h = pl.tau;

size\_t l = min(U\_in.size(), I\_out.size());

vector<double> I\_last = vector<double>(l);

vector<double> U\_last = vector<double>(l);

size\_t i = 0;

const int lIdx = N\*2 - 1;

do {

const double t = i\*h;

I\_last[i] = state[lIdx-1];

U\_last[i] = state[lIdx];

rk.do\_step(

[&](const state\_type &x, state\_type &dxdt, double t) {

double time = (t/h);

size\_t i = (size\_t)time;

dxdt[0] = (U\_in[i] - RN\*x[0] - x[1])/NL;

dxdt[1] = (x[0] - x[2])/NC;

for (int i = 1; i < N-1; i++) {

const int idx = i\*2;

dxdt[idx] = (x[idx - 1] - RN\*x[idx] - x[idx+1])/NL;

dxdt[idx+1] = (x[idx] - x[idx+2])/NC;

}

const int lIdx = N\*2 - 1;

dxdt[lIdx - 1] = (x[lIdx - 2] - RN\*x[lIdx - 1] - x[lIdx])/NL;

dxdt[lIdx] = (x[lIdx - 1] - I\_out[i])/NC;

},

state,

t,

h

);

i++;

} while(i < l);

res.I = I\_last;

res.U = U\_last;

return res;

}

Machine::~Machine() {

}

Machine::Machine(int amount, const RLC& lOptions)

: opt(lOptions),

N(amount)

{

RN = lOptions.R/N;

NL = lOptions.L/N;

NC = lOptions.C/N;

}

Листинг Б.5 – Заголовочный файл главного окна

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "qcustomplot.h"

#include <atomic>

#include <QVector>

using namespace std;

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_startButton\_clicked();

void on\_stopButton\_clicked();

private:

QCustomPlot\* inputPlot;

QCustomPlot\* outputPlot;

QDoubleSpinBox\* resistanceSpinBox;

QDoubleSpinBox\* inductanceSpinBox;

QDoubleSpinBox\* capacitySpinBox;

QDoubleSpinBox\* accuracySpinBox;

QDoubleSpinBox\* minStepSpinBox;

QDoubleSpinBox\* maxStepSpinBox;

QDoubleSpinBox\* lenSpinBox;

QDoubleSpinBox\* freqSpinBox;

QDoubleSpinBox\* uStepSpinBox;

QDoubleSpinBox\* uNullSpinBox;

QDoubleSpinBox\* timeSpinBox;

QLineEdit\* bitsLineEdit;

QPushButton\* startButton;

QPushButton\* stopButton;

QComboBox\* coderComboBox;

Ui::MainWindow \*ui;

QLabel\* loader;

atomic\_bool stopped;

void initControls();

void setDemonstrationValues();

void setOscilatorStylePlot(QCustomPlot\* plt);

void clearPlot(QCustomPlot\* plt);

void setLoading(bool);

void showResult(QCustomPlot\* plt, QVector<double> x, QVector<double> y);

};

#endif // MAINWINDOW\_H

Листинг Б.6 – Исходный код главного окна

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QPushButton>

#include <QtConcurrent>

#include "qcustomplot.h"

#include "encoder.h"

#include "machine.h"

#include "types.h"

#include <thread>

#include <atomic>

#include <limits>

void MainWindow::initControls() {

inputPlot = findChild<QCustomPlot\*>("inputPlot");

outputPlot = findChild<QCustomPlot\*>("outputPlot");

resistanceSpinBox = findChild<QDoubleSpinBox\*>("rSpinBox");

inductanceSpinBox = findChild<QDoubleSpinBox\*>("lSpinBox");

capacitySpinBox = findChild<QDoubleSpinBox\*>("cSpinBox");

lenSpinBox = findChild<QDoubleSpinBox\*>("lenSpinBox");

accuracySpinBox = findChild<QDoubleSpinBox\*>("accuracySpinBox");

minStepSpinBox = findChild<QDoubleSpinBox\*>("minStepSpinBox");

maxStepSpinBox = findChild<QDoubleSpinBox\*>("maxStepSpinBox");

freqSpinBox = findChild<QDoubleSpinBox\*>("freqSpinBox");

bitsLineEdit = findChild<QLineEdit\*>("bitsLineEdit");

startButton = findChild<QPushButton\*>("startButton");

stopButton = findChild<QPushButton\*>("stopButton");

coderComboBox = findChild<QComboBox\*>("codeComboBox");

uStepSpinBox = findChild<QDoubleSpinBox\*>("stepUSpinBox");

uNullSpinBox = findChild<QDoubleSpinBox\*>("nullUSpinBox");

timeSpinBox = findChild<QDoubleSpinBox\*>("timeSpinBox");

loader = findChild<QLabel\*>("loader");

resistanceSpinBox->setSingleStep(0.1);

inductanceSpinBox->setSingleStep(0.1);

capacitySpinBox->setSingleStep(0.1);

lenSpinBox->setSingleStep(0.1);

accuracySpinBox->setSingleStep(0.01);

minStepSpinBox->setSingleStep(0.0001);

minStepSpinBox->setDecimals(4);

maxStepSpinBox->setDecimals(4);

maxStepSpinBox->setSingleStep(0.0001);

freqSpinBox->setSingleStep(0.1);

uStepSpinBox->setSingleStep(0.1);

uNullSpinBox->setSingleStep(0.1);

timeSpinBox->setSingleStep(1.0);

freqSpinBox->setSingleStep(10.0);

freqSpinBox->setMaximum(std::numeric\_limits<double>::max());

timeSpinBox->setMaximum(std::numeric\_limits<double>::max());

resistanceSpinBox->setMaximum(std::numeric\_limits<double>::max());

inductanceSpinBox->setMaximum(std::numeric\_limits<double>::max());

capacitySpinBox->setMaximum(std::numeric\_limits<double>::max());

lenSpinBox->setMaximum(1000.0);

freqSpinBox->setMaximum(std::numeric\_limits<double>::max());

minStepSpinBox->setMaximum(1);

maxStepSpinBox->setMaximum(1);

accuracySpinBox->setMaximum(1);

}

void MainWindow::setDemonstrationValues() {

resistanceSpinBox->setValue(10.0);

inductanceSpinBox->setValue(250.0);

capacitySpinBox->setValue(1.0);

lenSpinBox->setValue(500.0);

freqSpinBox->setValue(100.0);

uStepSpinBox->setValue(0.5);

uNullSpinBox->setValue(12.0);

timeSpinBox->setValue(1000);

accuracySpinBox->setValue(10.0);

minStepSpinBox->setValue(0.04);

maxStepSpinBox->setValue(0.04);

coderComboBox->setCurrentIndex(1);

}

void MainWindow::setOscilatorStylePlot(QCustomPlot \*plt) {

//plt->setBackground( QColor(0,0,0) );

QColor gridColor(255, 255, 255);

QPen gridPen;

gridPen.setWidthF(0.5);

gridPen.setColor(gridColor);

gridPen.setStyle(Qt::PenStyle::SolidLine);

plt->xAxis->grid()->setPen(gridPen);

plt->yAxis->grid()->setPen(gridPen);

QCPGraph\* graph = plt->addGraph();

QColor graphColor(0, 0, 255);

QPen graphPen;

graphPen.setWidthF(2);

graphPen.setColor(graphColor);

graph->setPen(graphPen);

plt->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom | QCP::iSelectPlottables);

//plt->axisRect()->setAutoMargins(QCP::msNone);

//plt->axisRect()->setMargins(QMargins(-5,0,0,0));

}

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

initControls();

setDemonstrationValues();

stopped.store(true);

setOscilatorStylePlot(inputPlot);

setOscilatorStylePlot(outputPlot);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::setLoading(bool state) {

if (state) {

if (loader->movie())

return;

//QMovie\* movie = new QMovie(":/Resource/loading.gif");

//loader->setMovie(movie);

loader->setText("loading");

//movie->start();

} else {

// loader->movie();

loader->setText(nullptr);

//loader->setMovie(nullptr);

}

}

void MainWindow::on\_startButton\_clicked()

{

bool running = !stopped.load();

if (running)

return;

stopped.store(false);

setLoading(true);

clearPlot(inputPlot);

clearPlot(outputPlot);

auto trd = std::thread([&]() {

double R = resistanceSpinBox->value();

double L = inductanceSpinBox->value();

double C = capacitySpinBox->value();

double accuracy = accuracySpinBox->value();

double length = lenSpinBox->value();

double minStep = minStepSpinBox->value();

double maxStep = maxStepSpinBox->value();

double frequency = freqSpinBox->value();

double uStep = uStepSpinBox->value();

double uNull = uNullSpinBox->value();

double T = timeSpinBox->value();

double h = (minStep+maxStep)/2;

//nrz, rz, manchester2

size\_t codeIndex = coderComboBox->currentIndex();

double halfStep = 1/(2\*h\*frequency);

RZEncoder rz(uNull, uStep, 100);

vector<double> vals = rz.encode(bitsLineEdit->text().toStdString());

int tCount = (T/h);

Payload p = {

.tau = h

};

RLC rlc = {

.R = R,

.L = L,

.C = C,

};

p.I = vector<double>(tCount);

p.U = vector<double>(tCount);

std::fill(p.I.begin(), p.I.end(), 1.0);

for (int i = 0; (i < vals.size()) && (i < tCount); i++) {

p.U[i] = vals[i];

}

for (int i = vals.size(); i < tCount; i++) {

p.U[i] = 12.0;

}

Machine m(length, rlc);

m.appendPayload(p);

optional<Payload> lastOpt = m.processNextPayloadStoppable(ref(stopped));

if (!lastOpt.has\_value()) {

return;

}

Payload last = lastOpt.value();

double maxU = 0;

QVector<double> x(tCount), U(tCount), I(tCount), U\_in(tCount);

for (int i=0; i<tCount; ++i) {

x[i] = i\*h; // x goes from -1 to 1

I[i] = last.I[i]; // let's plot a quadratic function

U[i] = last.U[i]; // let's plot a quadratic function

U\_in[i] = p.U[i];

if (U[i] > maxU)

maxU = U[i];

}

outputPlot->graph(0)->setData(x, U);

outputPlot->xAxis->setRange(0, T);

outputPlot->yAxis->setRange(0, (maxU\*1.1));

outputPlot->replot();

inputPlot->graph(0)->setData(x, U\_in);

inputPlot->xAxis->setRange(0, T);

inputPlot->yAxis->setRange(0, (uNull+uStep)\*1.1);

inputPlot->replot();

stopped.store(true);

setLoading(false);

});

trd.detach();

}

void MainWindow::clearPlot(QCustomPlot \*plt) {

plt->graph(0)->data()->clear();

plt->replot();

}

void MainWindow::on\_stopButton\_clicked()

{

stopped.store(true);

setLoading(false);

clearPlot(inputPlot);

clearPlot(outputPlot);

}

Листинг Б.8 – Исходный код приложения

#include "mainwindow.h"

#include <QApplication>

#include <stdlib.h>

int main(int argc, char\* argv[])

{

QApplication app(argc, argv);

MainWindow w;

w.show();

return app.exec();

}