Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20Б.11-мм

Фаст Никита Михайлович

Разработка универсального скрипта моделирования систем цифровой связи в среде python

Отчёт по учебной практике

в форме «Производственное задание»

Научный руководитель:
доцент кафедры СП, к.ф.-м.н. Д.В. Луцив

Консультант:
начальник отдела программирования АО "Концерн Гранит" А.С. Кривоногов

Санкт-Петербург
2023

Оглавление

1. Введение

В ходе разработки системы цифровой связи возникает необходимость проверить, удовлетворяют ли ее технические характеристики требованиям заказчика. Одной из важнейших метрик[5] качества в системах цифровой связи является график зависимости вероятности появления ошибочного бита P_b от E_b/N_0, где E_b это энергия бита, а N_0 - спектральная плотность мощности шума. Данная метрика называется помехоустойчивостью системы и одним из способов ее выяснения является осуществление моделирования разрабатываемой системы на компьютере.

Общая структура системы цифровой связи представлена на рисунке 1. Заметим, что система представляет собой набор модулей, соединенных друг с другом. Таким образом, при наличии программных реализаций модулей и наличии механизма, позволяющего соединять модули между собой, удастся сформировать модель разрабатываемой системы. Остается запустить моделирующий алгоритм для полученной модели, что в результате позволит установить помехоустойчивость моделируемой системы цифровой связи.

Данное замечание приводит к задаче по созданию моделирующей системы с библиотекой поддерживаемых модулей, графическим интерфейсом, позволяющим создавать требуемую модель и настраивать параметры каждого элемента модели, а также самим механизмом моделирования помехоустойчивости подготовленной модели. Обозначим требования, предъявляемые к такой системе.

Библиотека поддерживаемых модулей должна быть расширяемой, чтобы пользователь имел возможность использовать в модели модули, реализованные им лично. В перспективе библиотека должна поддерживать программные реализации модулей, написанные на разных языках, например: Python, C, VHDL.

Графический интерфейс системы должен позволять конструировать модель, задавать параметры элементов модели и самого процесса моделирования, указывать на ошибки в построении модели, предоставлять

информацию о ходе моделирования.

Механизм, осуществляющий моделирование помехоустойчивости, должен быть способен задействовать для своей работы по крайней мере все ядра одного процессора, а в идеале – все ядра сразу нескольких машин.

В итоге, концерном "Гранит" было предложено реализовать на языке Python моделирующую систему, удовлетворяющую изложенным выше требованиям и именно этому посвящена данная работа.

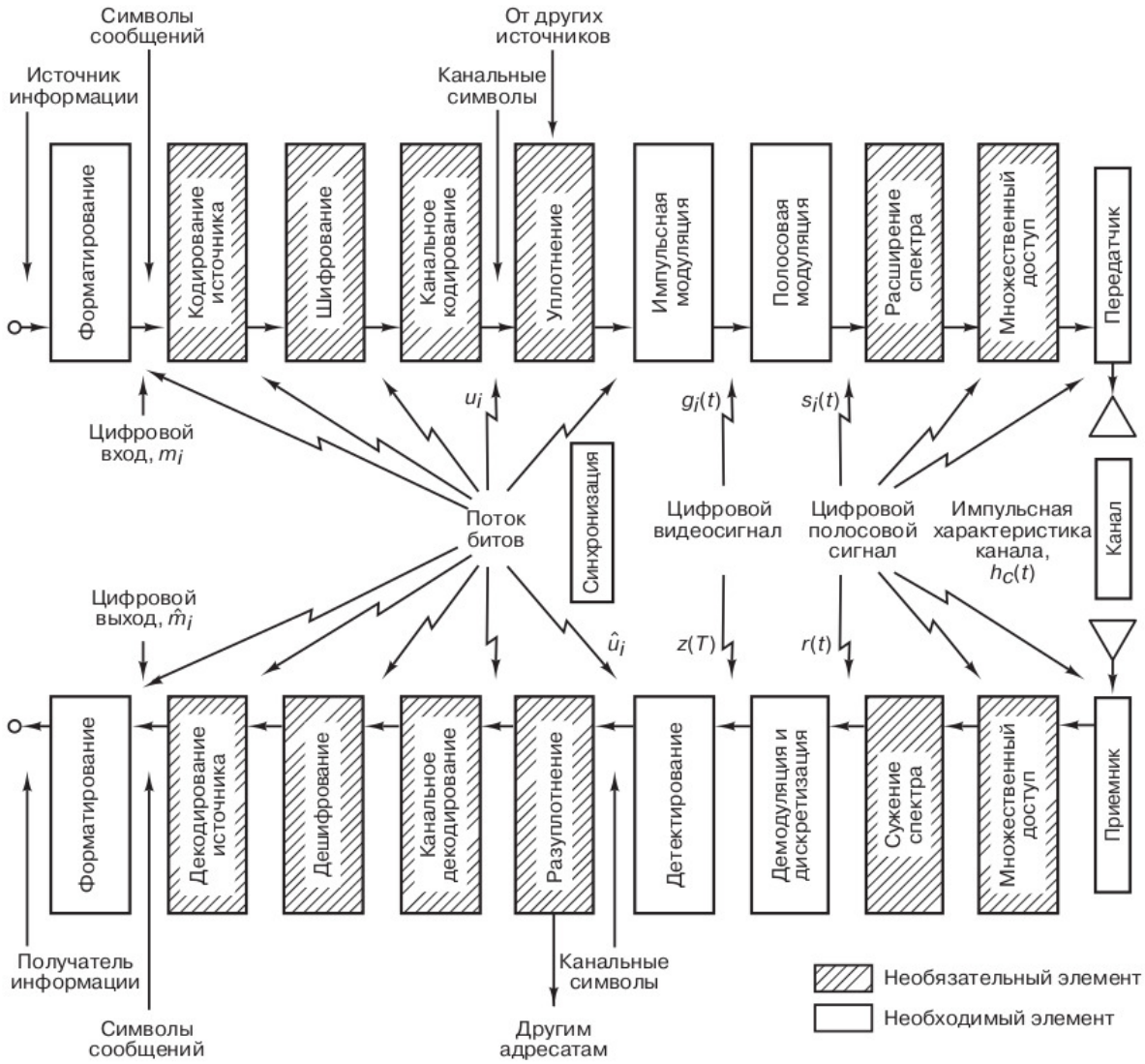

Рис. 1: Структура системы цифровой связи

2. Постановка задачи

Целью работы является реализация[3] моделирующей системы с графическим интерфейсом, которая позволит оперативно создавать требуемую модель из отдельных модулей, а на основе полученной модели осуществит моделирование, результатом которого будет построение кривой помехоустойчивости моделируемой системы. Для её выполнения были поставлены следующие задачи:

1. Разработать архитектуру, позволяющую:
   (a) добавлять в библиотеку пользовательские реализации модулей;
   (b) работать с модулями, реализованными на Python;
   (c) осуществлять валидацию параметров, получаемых от пользователя через GUI;
   (d) используя модель, сформированную через GUI, осуществлять генерацию файла с кодом инициализации модели и кодом выполняющим моделирование;
   (e) генерировать код моделирования, позволяющий задействовать по крайней мере все ядра одного процессора, а в идеале – все ядра сразу нескольких машин;
2. Разработать графический интерфейс, который:
   (a) позволит конструировать модель на основе отдельных модулей и того, как они соединены между собой;
   (b) позволит настраивать параметры моделирования и параметры отдельных элементов модели;
   (c) укажет на ошибки в построении модели при их наличии;
   (d) будет информировать пользователя о ходе моделирования;
3. Провести тестирование разработанной моделирующей системы, используя в качестве опытных моделей классические системы циф-

ровой связи, помехоустойчивость которых известна заранее, поскольку может быть вычислена по готовым формулам

3. Обзор

3.1. Обзор альтернативного подхода

Альтернативным методом выяснения помехоустойчивости системы цифровой связи является строгий теоретический вывод, конечным продуктом которого будет формула, определяющая зависимость между вероятностью появления ошибочного бита P_b и отношением E_b/N_0. Очевидно, что для любой нетривиальной системы подобная процедура трудна и требует привлечения сотрудника, хорошо разбирающегося в теоретических основах цифровой связи. Также при любом изменении системы, выведенная ранее формула станет более неприменимой.

К преимуществам данного подхода можно отнести возможность анализа системы при очень низких значениях вероятности битовой ошибки, в то время как моделирование в данной ситуации потребует больших временных затрат, поскольку для расчета одной точки на кривой помехоустойчивости необходимо обработать такой объем данных, который позволит получить не менее 100 битовых ошибок, так как в противном случае результаты моделирования могут значительно отличаться от раза к разу и не могут быть признаны достоверными. Таким образом, если моделирование осуществляется при $P_b = 10^{-10}$, то необходимо осуществить обработку как минимум $10^{10} * 100 = 10^{12}$ бит, что определенно потребует существенных временных затрат.

3.2. Обзор существующих решений

- Simulink[4] это графическая среда на основе MATLAB предназначенная для моделирования, симуляции и анализа динамических систем. Является проприетарным программным обеспечением и в данный момент приобретение лицензии на Simulink в РФ невозможно из-за политики компании MathWorks. Данный инструмент позволяет проводить моделирование не только систем цифровой связи, но и очень широкого спектра других систем. Такая универсальность делает моделируемые сущности более тяжеловесными

и не лучшим образом сказывается на скорости моделирования. Simulink осуществляет моделирование по тактам, в ходе которых осуществляется проход по всей системе, что также повышает трудоемкость моделирования и снижает скорость. Отметим, что в рамках задачи, поставленной в данной работе, используемый в Simulink подход к моделированию избыточен.

- GNU Radio это свободное ПО, распространяемое по лицензии GPL v3. Оно предоставляет модули обработки сигналов, которые позволяют реализовать SDR и другие системы обработки сигналов. Пользователь может самостоятельно создавать модули обработки сигнала. Содержит встроенный инструмент BER Tool предназначенный для определения помехоустойчивости системы посредством моделирования. Имеет инструмент с графическим интерфейсом под названием GNU Radio Companion, позволяющий создать требуемую систему на экране и по ней сгенерировать код на C++ или Python. Поддерживает параллельные, но не поддерживает распределенные вычисления. Не может генерировать код на VHDL. Подведем итоги, многое из того, что требуется от разрабатываемой системы, в GNU Radio уже имеется. Однако имеются и нереализованные требования. Поскольку GNU Radio это чужой проект с большим объемом исходников, то трудно спрогнозировать время, необходимое для анализа проекта и корректирования его под нужды организации. Кроме того, использование GNU Radio в разрабатываемой системе осложнит ее коммерциализацию, поскольку GNU Radio использует лицензию GPL v3 и из-за этого код системы должен будет быть открытым для ее покупателей.

4. Реализация

4.1. Используемые технологии

При создании ПО предназначенного для анализа систем цифровой связи предпочтительнее использовать интерпретируемые языки программирования, поскольку они дают большую гибкость в манипулировании данными и упрощают отладку кода. Обычно это Python или MATLAB. Поскольку Python распространяется под открытой лицензией, то был выбран он.

Для реализации графического интерфейса используется PySide 2[2], поскольку данная технология предоставляет всю необходимую функциональность и используется в проектах концерна "Гранит".

4.2. Архитектура приложения

Требуется, используя графический интерфейс, создать модель из доступных в библиотеке модулей. Затем, проверив корректность построения модели, осуществить генерацию кода, получив в результате файл со всем необходимым для подготовки модели и осуществления самого моделирования. Задача кодогенерации это избавление пользователя от монотонной работы по подготовке всех объектов, участвующих в моделировании. К такой работе можно отнести: создание модулей и задание их начальных параметров, прописывание соединений между портами модулей.

Для поддержки пользовательских модулей введено специальное описание модуля, называемое дескриптором. Дескриптор (Рис. 2) это .py файл, который содержит набор обязательных полей, набор опциональных полей и произвольный питоновский код. Суть обязательных и опциональных полей в предоставлении моделирующей системе необходимой информации о данном модуле, которая позволит интегрировать данный модуль в подсистему визуального построения модели, подсистему кодогенерации и подсистему моделирования. В качестве произвольного кода выступает сама реализация модуля, например: описание

набора функций или класса; импорт файлов, содержащих реализацию модуля. Такая структура дескриптора позволяет уменьшить количество кода, которое пользователь должен написать для интеграции своего модуля в данную систему.

```
name = "Binary Generator"
language = "Python"
module_type = 'function'
entry_point = gen_binary_data

output_ports = [
    {
        "label": "Выход",
        "type": List[int]
    },
]
module_parameters = [
    {
        'name': 'bits_num',
        'type': int,
        'has_default_value': True,
        'default_value': 64_000,
        'validator': lambda x: isinstance(x, int) and (0 < x < 1_000_000_000)
    }
]
```

Рис. 2: Пример дескриптора для модуля, генерирующего бинарные данные

Опишем детально устройство системы. Перед проведением моделирования, необходимо построить модель, которая представляет собой список модулей, список установленных значений параметров каждого модуля и список исходящих соединений для каждого модуля.

Каждый модуль имеет определенное количество входных и выходных портов с помощью которых модули получают и передают данные. Выходной порт одного модуля может быть соединен с одним или несколькими входными портами других модулей, однако входной порт может быть соединен лишь с одним выходным портом. Соединение портов описывается специальной одноименной сущностью.

Конструирование модели и конфигурация модулей осуществляется на таком элементе системы как сцена. Графический интерфейс, позволяющий установить значения параметров модуля, генерируется системой, основываясь на дескрипторе данного модуля. Отметим, что в дескрипторе зарезервировано опциональное поле, позволяющее пользователю предоставить свой графический интерфейс, однако на данный момент такая возможность отключена. Графический интерфейс модуля визуально выделяет параметры, значения которых еще не предоставлены. Также при вводе значения параметра осуществляется его проверка при помощи валидатора, указанного для данного параметра в дескрипторе. Также на сцене осуществляется подключение модулей при помощи соединений их портов. При создании нового соединения производится проверка типов данных между соединяемыми портами, что позволяет указать пользователю на некорректные соединения и, как следствие, на неправильно настроенные модули. Ошибки настройки модуля отображаются в его графическом интерфейсе. Корректным считается модуль у которого каждый параметр имеет валидное значение и все порты которого подключены корректными соединениями. Вся модель в совокупности считается корректной, если каждый входящий в нее модуль является корректным. Таким образом система предоставляет инструмент визуального построения модели (Рис. 3) и уведомляет пользователя об ошибках в сконструированной модели.

Для корректной модели можно осуществить кодогенерацию. На этом этапе информация о соединениях портов модуля сериализуется с помощью json, а информация о параметрах сериализуется в бинарном виде при помощи pickle, поскольку json не может сериализовать некоторые питоновские объекты, например, комплексные числа. Затем в отдельный файл пишется код, состоящий из нескольких частей. В первой части импортируются необходимые файлы, такие как дескрипторы модулей и вспомогательные классы обёртки. Во второй части производится создание объектов модулей из которых состоит модель и настройка всех этих объектов в соответствии с параметрами, заданными пользователем. В третьей части модули оборачиваются специальным классом

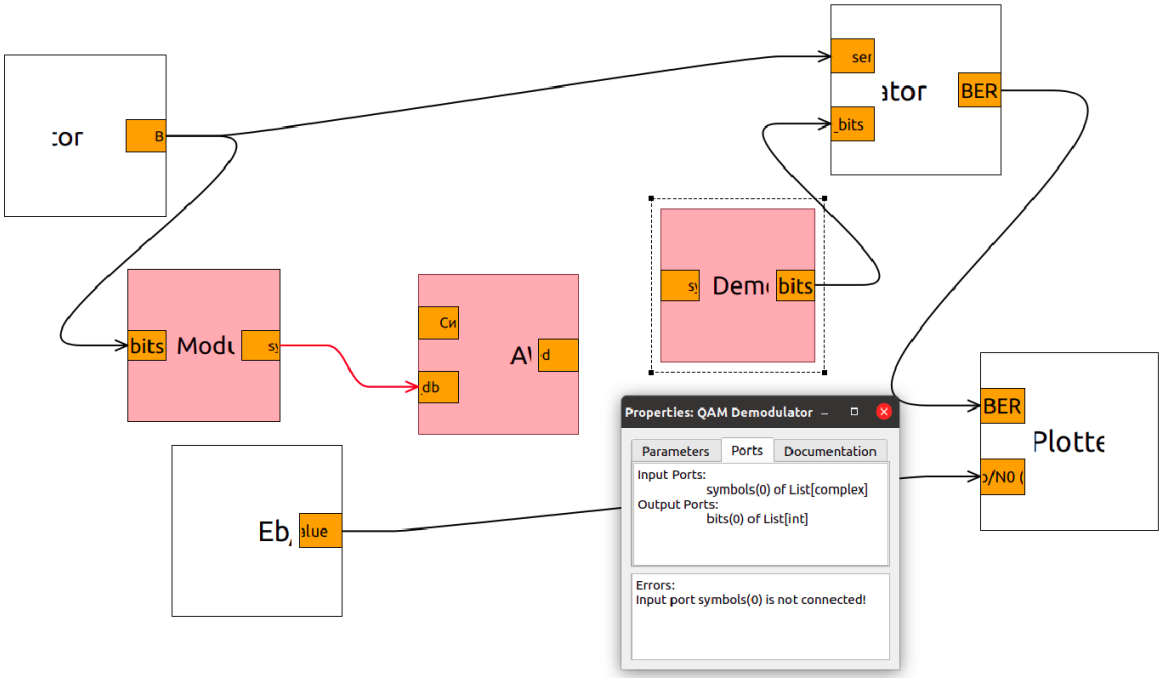

Рис. 3: Сцена на которой строится модель. Розовым подсвечены некорректно настроенные модули. Некорректные соединения портов подсвечены красным

ModuleWrapper, который предоставляет общий интерфейс, используемый модулирующей системой для взаимодействия с модулем. Затем конструируется объект класса FlowGraph, который внутри себя аккумулирует модули и информацию о том, как эти модули соединены друг с другом. Также данный класс содержит метод run, который осуществляет само моделирование.

После генерации кода, можно запустить моделирование. При этом указывается число процессов, которое будет использовано для вычислений. Далее средствами Qt создается нужное количество процессов, каждый из которых получает сгенерированный код и исполняет его. Процедура моделирования для всех процессов аналогична. Каждый процесс исполнит определенное число итераций моделирования, которое зависит от диапазона значений отношения сигнал шум, на котором мы исследуем помехоустойчивость. Во время итерации производится запуск всех модулей в определенном порядке. Данный порядок устанавливается при конструировании объекта класса FlowGraph, создаваемого на

этапе кодогенерации.

Опишем алгоритм, определяющий порядок запуска модулей. Рассмотрим модель как ориентированный граф, где модули являются вершинами, а соединения - ребрами. В данном случае не важно, какие порты связывает соединение, имеет значение на каких модулях данные порты находятся. С каждым модулем ассоциируем такую величину как уровень. Модули с меньшим уровнем должны быть запущены строго раньше модулей с большим уровнем. Модули, имеющие одинаковый уровень, могут быть запущены в произвольном порядке относительно друг друга. Изначально все модули имеют уровень равный бесконечности. Затем выделяем модули источники, это модули не имеющие входных портов. Все модули источники имеют уровень 0. Все модули источники добавляются в очередь на обработку. Алгоритм будет работать до опустошения очереди. В ходе работы алгоритм извлекает из очереди первый модуль m. Рассмотрим все модули $succ$, с которыми m связан исходящим соединением. Для каждого модуля $succ$ требуется вычислить его уровень. Для этого рассмотрим предшественников модуля $succ$, т.е. такие модули $pred$, из которых исходит соединение в модуль $succ$. Уровень модуля $succ$ определяется как максимум среди уровней его предшественников плюс один. После этого добавляем модуль $succ$ в конец очереди. Результатом работы алгоритма является рассчитанный уровень для каждого модуля. Далее группируем модули по уровню, а группы сортируем по росту уровня.

В итоге, во время итерации моделирования обходим модули в установленном порядке. Выбранный модуль запускается, а результаты его работы отправляются на входные порты соответствующих модулей. Итерация считается завершенной, когда все модули будут обработаны. Для завершения моделирования необходимо дождаться завершения всех процессов. После чего можно агрегировать вычисленные значения BER-а и построить график помехоустойчивости моделируемой системы.

5. Тестирование

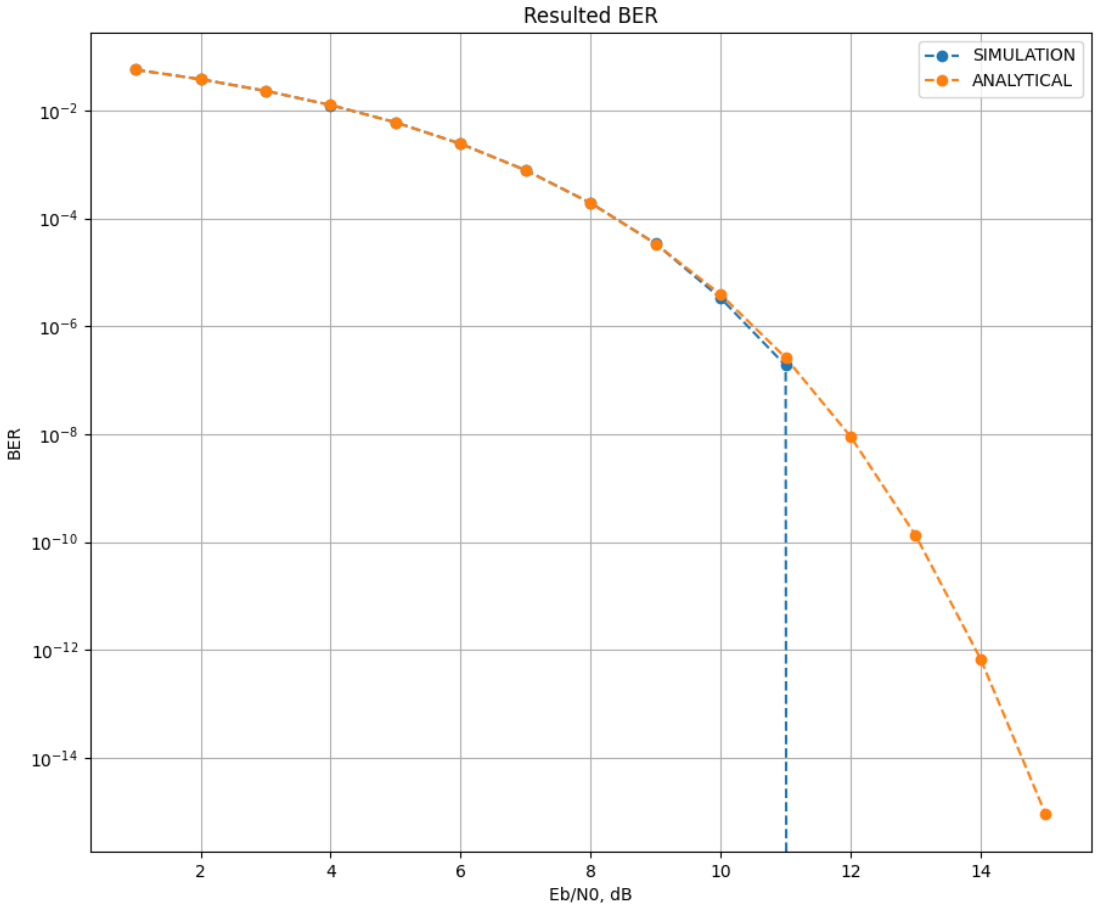

Рис. 4: Теоретическая вероятность появления ошибочного бита для модуляции КАМ-4 vs результаты моделирования

Для тестирования моделирующей системы запустим ее на классических системах цифровой связи и сравнив полученные результаты с результатами, выведенными с помощью специализированных формул для расчета вероятности битовой ошибки. Одним из примеров такой системы является модулятор и демодулятор для КАМ-4 над АБГШ каналом. Вероятность битовой ошибки для М-КАМ модуляции над АБГШ каналом вычисляется по формуле[1] $p_b = \frac{2}{n} * (1 - \frac{1}{\sqrt{M}}) * erfc(\sqrt{\frac{3*n}{M-1} * \frac{E_b}{N_0}})$, где M это порядок модуляции, $n = \log_2 M$ число бит на символ, $erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-x^2} dx$ это комплементарная функция ошибок. На рис. 4 синим цветом представлен BER, полученный в результате моделирования, а

оранжевым цветом - вероятность появления ошибочного бита рассчитанная по формуле. Как видим графики практически совпадают. Резкий обрыв синего графика говорит о том, что при таком уровне E_b/N_0 во время моделирования не возникло ни одной битовой ошибки, что ожидаемо, поскольку количество бит, проходящих через систему ограничено, а вероятность битовой ошибки стремится к нулю с ростом E_b/N_0. Соответствие практических результатов, полученных с помощью моделирования, и теоретических результатов, полученных по формулам, позволяет судить о корректной работе моделирующей системы.

6. Заключение

По задачам сделано следующее:

1. Разработана архитектура, позволяющая:
   (a) добавлять в библиотеку пользовательские реализации модулей;
   (b) работать с модулями, реализованными на Python;
   (b) осуществлять валидацию параметров, получаемых от пользователя через GUI;
   (c) используя модель, сформированную через GUI, осуществлять генерацию файла с кодом инициализации модели и кодом выполняющим моделирование;
   (f) генерировать код моделирования, позволяющий задействовать по крайней мере все ядра одного процессора, а в идеале – все ядра сразу нескольких машин;
2. Разработан графический интерфейс, который позволяет:
   (a) конструировать модель на основе отдельных модулей и того, как они соединены между собой;
   (b) настраивать параметры отдельных элементов модели;
   (b) укажет на ошибки в построении модели при их наличии;
   (d) будет информировать пользователя о ходе моделирования;
3. Проведено тестирование разработанной моделирующей системы, с использованием в качестве опытных моделей классических систем цифровой связи, помехоустойчивость которых известна заранее, поскольку может быть вычислена по готовым формулам

Список литературы

[1] Moon Todd K. Error Correction Coding: Mathematical Methods and Algorithms. — 1st Edition edition. — Wiley-Interscience, 2005. — June 6. — P. 14–21. — ISBN: 978-0471648000.

[2] PySide 2. — URL: https://wiki.qt.io/Qt_for_Python (online; accessed: 2023-03-23).

[3] SimLab. — URL: https://github.com/Nikita-Fast/SimLab (online; accessed: 2023-06-02).

[4] Simulink. — URL: https://www.mathworks.com/products/simulink.html (online; accessed: 2023-03-23).

[5] Sklar B. Digital Communications: Fundamentals and Applications. — Second Edition edition. — Communications Engineering Services, Tarzana, California and University of California, Los Angeles : Prentice Hall, 2001. — P. 146. — ISBN: 978-0134724058.