|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ   РАДИОЭЛЕКТРОНИКА И ЛАЗЕРНАЯ ТЕХНИКА

КАФЕДРА   РАДИОЭЛЕКТРОННЫЕ СИСТЕМЫ И УСТРОЙСТВА

**ОТЧЕТ**

**по курсовому проекту**

на тему:

**«Разработка приёмо-передатчика OFDM LTE»**

Выполнил:  
 студент группы РЛ1-93 Голобояр А. А.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Проверил: | | (Ф.И.О.) |  | (подпись, дата) |
| преподаватель кафедры РК6 | | Оглоблин Д. И. |  |  |
|  |  | (Ф.И.О.) |  | (подпись, дата) |

Москва, 2024г.

**РЕФЕРАТ**

Отчёт 54 c., 39 рис., 9 использованных источников.

Объект исследования – приёмо-передатчик OFDM LTE на основе быстрого преобразования Фурье.

В данной работе изучается алгоритм цифровой обработки данных для твердотельного модема LTE современного мобильного смартфона, использующего для передачи потока данных концепцию ортогонального частотного мультиплексирования (OFDM).

Артем, привет. Общая оценка «отл», ты реализовал все, что я намечал на группу студентов на пару лет. Особо отмечу, что тебе не нужен Simulink для обучения (восстановления) функции программирования аппаратуры на С. Для описания параллельно работающих устройств понадобятся расширения языка С в виде SystemC или Verilog.

Все принимается, после стр. 21 есть пожелания по применению результатов к конкретной задаче OFDM, а именно к выбору числа точек частотного разбиения на основе гл. 4.

Я нигде не встречал обоснование выбора количества частот для конкретного сигнала LTE с конкретной модуляцией, можно это опубликовать.

На стр. 44 есть пожелания по поводу обоснования перехода к одному последовательному каналу и его выполнению на основе МК

**Содержание**

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4](#_Toc185885319)

[ВВЕДЕНИЕ 5](#_Toc185885320)

[1 Маркетинговые исследования о технологии LTE 7](#_Toc185885321)

[1.1 Основные концепции маркетинговых исследований 7](#_Toc185885322)

[1.2 Типы модемов LTE 8](#_Toc185885323)

[1.3 Тенденции и инновации 9](#_Toc185885324)

[2 Патентные исследования технологии LTE 10](#_Toc185885325)

[2.1 Ключевые области патентных исследований 10](#_Toc185885326)

[2.2 Ключевые игроки на рынке 10](#_Toc185885327)

[2.3 Примеры патентов 11](#_Toc185885328)

[2.4 Патентное законодательство и рынок 11](#_Toc185885329)

[3 Расчет надежности 12](#_Toc185885330)

[4 Понимание преобразования Фурье 13](#_Toc185885331)

[5 Реализация преобразований Фурье 22](#_Toc185885332)

[5.1 Реализация дискретного преобразования Фурье 23](#_Toc185885333)

[5.2 Реализация БПФ на языке С 25](#_Toc185885334)

[5.3 Анализ результатов ДПФ и БПФ 29](#_Toc185885335)

[6 Реализация блока S2P 32](#_Toc185885336)

[7 Реализация OFDM в Matlab 33](#_Toc185885337)

[8 Реализация OFDM на языке C++ 39](#_Toc185885338)

[9 Реализация OFDM на микроконтроллере STM32 44](#_Toc185885339)

[ЗАКЛЮЧЕНИЕ 53](#_Toc185885340)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 54](#_Toc185885341)

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

|  |  |  |
| --- | --- | --- |
| IFFT | – | inverse fast Fourier transform |
| LTE | – | long-term evolution |
| OFDM | – | orthogonal frequency division multiplexing |
| QAM | – | quadrature amplitude modulation |
| АЦП | – | аналого-цифровой преобразователь |
| БПФ | – | быстрое преобразование Фурье |
| ОБПФ | – | обратное быстрое преобразование Фурье |
| ПЛИС | – | программируемая логическая интегральная схема |
| ЦАП | – | цифро-аналоговый преобразователь |

**ВВЕДЕНИЕ**

В настоящее время технология LTE широко используется для беспроводной передачи данных в мобильных сетях. Одним из ключевых аспектов LTE является метод модуляции-демодуляции с использованием ортогонального частотного разделения (OFDM), который обеспечивает высокую скорость передачи данных и эффективное использование частотного спектра.

Данный курсовой проект посвящен изучению твердотельного модема LTE современного мобильного смартфона, использующего для передачи потока данных концепцию ортогонального частотного мультиплексирования (OFDM). Условно модем разбивается на цифровую и аналоговую часть. В цифровую часть передатчика входит блок избыточного кодирования и скрэмблирования (шифрования), цифровой модулятор и ЦАП. В аналоговой части формируется вещественный сигнал, осуществляется преобразование частоты, усиление и фильтрацию сигнала. В приемнике осуществляются обратные действия. Все части выполняются в виде системы на кристалле, не предполагающей настройку пользователем.

Бизнес-функциональные требования:

1. Модель Matlab Simulink;
2. Большая скорость передачи данных;
3. Широкая полоса;
4. Маленькая задержка;
5. Большое количество поднесущих;
6. Выбор модуляции;
7. Выбор программируемого устройства.

Техническое задание:

1. Готовый код на языке C;
2. Пиковая спектральная эффективность:

* Нисходящий канал 16 ;
* Восходящий канал 4 ;

1. Ширина канала до 20 МГц;
2. Задержка данных не более 5 мс;
3. Количество поднесущих – 12;
4. Модуляция 16-QAM.

# **1 Маркетинговые исследования о технологии LTE**

## **1.1 Основные концепции маркетинговых исследований**

1) Рост пользователей:

- Увеличение числа пользователей смартфонов и других мобильных устройств создает постоянный спрос на высокоскоростные мобильные сети.

-По прогнозам, к 2025 году более 70% всех мобильных подключений будут осуществляться через LTE и его последующие версии.

2) Региональные особенности:

- В развивающихся странах наблюдается быстрый рост LTE-сетей, поскольку операторы стремятся улучшить качество связи и удовлетворить растущий спрос на мобильный интернет.

- В развитых странах LTE становится основным стандартом, а операторы уже начинают внедрять 5G.

3) Конкуренция:

- Основные операторы связи активно запускают LTE-сети, конкурируя друг с другом по скорости и качеству услуг.

- Существуют также альтернативные технологии, такие как WiMAX, которые могут оспаривать позиции LTE в определенных сегментах рынка.

4) Преимущества LTE

- Высокая скорость передачи данных: LTE может обеспечивать скорости до 300 Мбит/с.

- Снижение задержки: LTE значительно сокращает время отклика, что особенно важно для приложений, требующих высокой скорости.

- Поддержка множества устройств: LTE работает на широком спектре мобильных устройств, что делает его универсальным.

5) Тенденции и будущее

- Развитие 5G: Переход к 5G приносит новые возможности, но стандарт LTE станется важным в переходный период.

6) Интернет вещей (IoT):

LTE также будет поддерживать рост IoT, обеспечивая связь для миллионов устройств.

7) Инвестиции в инфраструктуру:

Операторы будут продолжать инвестировать в расширение и модернизацию своих сетей, чтобы поддерживать растущий трафик данных.

8)Обзор продукции модемов LTE

Модемы LTE представляют собой ключевые устройства для обеспечения доступа к высокоскоростному мобильному интернету. Они позволяют пользователям подключаться к LTE-сетям на различных устройствах, таких как компьютеры, ноутбуки, смартфоны и маршрутизаторы. Ниже представлен обзор различных типов модемов LTE и их особенностей.

## **1.2 Типы модемов LTE**

1) USB-модемы

- Компактные устройства, которые подключаются напрямую к USB-порту компьютера.

- Идеально подходят для мобильного доступа, так как их легко использовать в путешествиях.

- Примеры: модель Huawei E3372, ZTE MF833V.

2) Мобильные маршрутизаторы (MiFi)

- Позволяют создать точку доступа Wi-Fi, используя LTE-сигнал.

- Поддерживают несколько подключений одновременно, что делает их идеальными для группового доступа к интернету.

- Примеры: Huawei E5785, Netgear Nighthawk M1.

3) Стационарные LTE-модемы

- Устройства, которые подключаются к фиксированным источникам питания обеспечивают стабильное соединение.

- Подходят для домашнего или офисного использования, могут поддерживать более мощные антенны для улучшения сигнала.

- Примеры: TP-Link TL-MR6400, ZTE MF286R.

Характеристики модемов LTE

1) Скорость передачи данных: Современные модемы LTE поддерживают скорости до 300 Мбит/с (LTE Cat 6) и выше, что позволяет комфортно использовать интернет для потокового видео и онлайн-игр.

2) Поддержка сетей: Многие модемы совместимы с несколькими стандартами связи, такими как 3G и 4G, что гарантирует более стабильное соединение в различных условиях.

3) Портативность: USB-модемы и мобильные маршрутизаторы часто имеют встроенные аккумуляторы, что позволяет использовать их в любом месте без привязки к сети электропитания.

Ключевые бренды на рынке

1) Huawei: Известен широким ассортиментом модемов, включая USB-модемы и мобильные маршрутизаторы высокого качества.

2) ZTE: Предлагает доступные и надежные варианты модемов с хорошими характеристиками.

3) Netgear: Специализируется на мощных маршрутизаторах, которые

беспечивают высокоскоростное соединение.

4) TP-Link: Известен своими стационарными LTE-модемами и маршрутизаторами, что делает их популярными для домашнего использования.

## **1.3 Тенденции и инновации**

- Увеличение поддержки 5G в модемах, позволяющее пользователям готовиться к следующему поколению мобильной связи.

- Развитие технологий, обеспечивающих более высокую скорость и лучшую устойчивость сигнала.

- Появление устройств с поддержкой IoT, что открывает новые возможности для использования LTE в умных домах и городах.

# **2 Патентные исследования технологии LTE**

## **2.1 Ключевые области патентных исследований**

1) Модуляция и кодирование

- Разработки в области методов модуляции, таких как OFDMA (Orthogonal Frequency Division Multiple Access) и SC-FDMA (Single Carrier Frequency Division Multiple Access).

- Патенты, касающиеся схемы кодирования и декодирования, включая технологии защиты данных, такие как Turbo-коды и LDPC (Low-Density Parity-Check) коды.

2)Технологии радиосвязи

- Патенты, связанные с управлением радиочастотами, распределением ресурсов в сети и обработкой сигналов.

- Системы MIMO (Multiple Input Multiple Output) и технологии для повышения эффективности использования спектра.

3)Управление сетью

- Инновации в области управления сетевыми ресурсами, включая динамическое распределение загрузки и автоматическую настройку сетей.

- Патенты, касающиеся архитектур сетей и их интеграции с предыдущими поколениями технологий 3G и 2G.

4)Оптимизация и безопасность

- Технологии, связанные с оптимизацией пользовательского опыта и качеством обслуживания (QoS).

- Патенты в области систем безопасности, шифрования и аутентификации пользователей в LTE-сетях.

## **2.2 Ключевые игроки на рынке**

- Qualcomm: Один из лидеров в патентовании технологий LTE, включает разработки в области чипсетов и радиосистем.

- Huawei: Активно патентует инновации в радиосвязи и управления сетями.

- Ericsson: Разработки, связанные с управлением сетевым трафиком и архитектурой сетей.

- Nokia: Патенты, касающиеся интеграции LTE и других технологий, таких как 5G.

## **2.3 Примеры патентов**

1) Патент US 8213510 B2 - "Method and apparatus for wireless communication in a mobile communication system utilizing multiple transmit/receive antennas".

- Описание: Этот патент описывает методы передачи данных с использованием нескольких антенн в системе мобильной связи LTE, что улучшает качество связи и скорость передачи.

2) Патент US 8355681 B2 - "Scheduling in a wireless communication system".

- Описание: Этот патент касается методов планирования ресурсов в беспроводной сети LTE, что позволяет оптимально распределять доступные каналы между пользователями.

3) Патент US 7751725 B2 - "Method for sending control information in a wireless communication system".

- Описание: Этот патент описывает способ передачи управляющей информации в сетях LTE, что улучшает эффективность и надежность связи.

## **2.4 Патентное законодательство и рынок**

Исследования патентов свидетельствуют о высоком уровне конкуренции в области LTE. Компании активно защищают свои разработки с помощью патентов, что создает определенные вызовы для новых игроков на рынке, стремящихся вникнуть в уже существующие технологии.

# **3 Расчет надежности**

|  |  |  |
| --- | --- | --- |
|  | Интенсивность отказа элементов () | количество |
| Резистор | 0,008 | 20 |
| Конденсатор | 0,008 | 22 |
| Танталовый конденсатор | 0,007 | 2 |
| Выходная клемма | 0,06 | 40 |
| Светодиод | 0,001 | 10 |
| Порт USB | 0,01 | 2 |
| Кварц | 0,012 | 1 |
| Пайка печатного монтажа | 0,01 | 173 |
| 2 кнопки | 0,008 | 2 |
| микроконтроллер | 0,01 | 1 |

Общая интенсивность отказа схемы

Вероятность безотказной работы

Пусть время работы 10000 часов

(2)

# **4 Понимание преобразования Фурье**

Основа перехода из временной области в частотную область - это преобразование Фурье. Формула преобразования Фурье представлена ниже.

(2)

Цель преобразования Фурье – это получить коэффициенты ряда Фурье. Общий вид коэффициентов ряда Фурье представлен ниже.

(3)

Разложим в ряд Фурье последовательность прямоугольных импульсов амплитудой 5, частотой следования импульсов 1 КГц и скважностью 10.

Функция с параметрами, задающая последовательность прямоугольных импульсов в среде Matcad prime представлена на рисунке 3, получившийся сигнал представлен на рисунке 4.

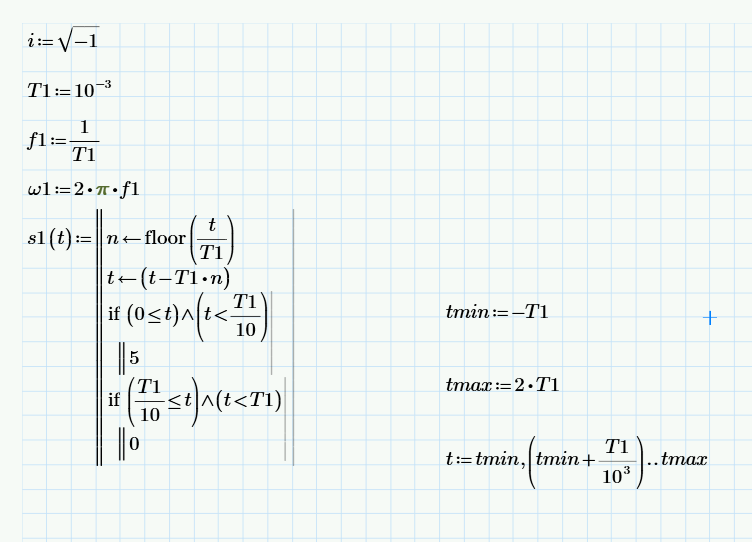


Рис.1 – Функция задающая входной сигнал

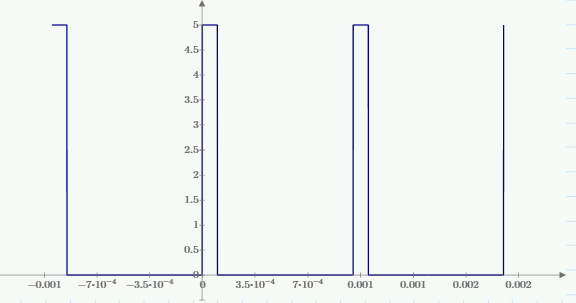


Рис.2 – График последовательности прямоугольных импульсов

Создадим функцию выполняющую преобразование Фурье. Функция принимает число гармоник используемых для разложения в ряд Фурье. Преобразование Фурье используем в комплексной форме по формуле (2). Такая форма очень удобно тем, что в результате получаются комплексные амплитуды, которые содержат информацию как по амплитуде самой гармоники, так и о начальной фазе. Функция с параметрами, выполняющая преобразование Фурье в среде Matcad prime, представлена на рисунке 5.

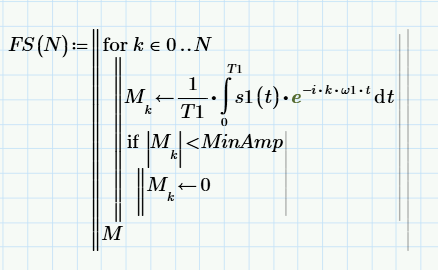


Рис.3 – Функция выполняющая преобразование Фурье

Ограничимся рассмотрением только нулевой и положительных гармоник, отрицательные номера, то есть отрицательные частоты рассматривать не будем по причине, что мы раскладываем в ряд Фурье вещественный сигнал. А как известно комплексные амплитуды для положительных и отрицательных гармоник будут комплексно сопряженными числами, поэтому нет необходимости вычислять комплексные амплитуды для гармоник имеющие отрицательные номера. Для устранения ошибок накопления сразу отсеем комплексные амплитуды, значения которых меньше 0,001. По итогу работы функция вернет вектор значений комплексной амплитуды каждой составляющей.

Создадим функцию возвращающую аргумент (угол) комплексной амплитуды, а также функцию корректирующую значение комплексной амплитуды. Реализация вышеперечисленных функций в среде Matcad prime, представлена на рисунке 6.

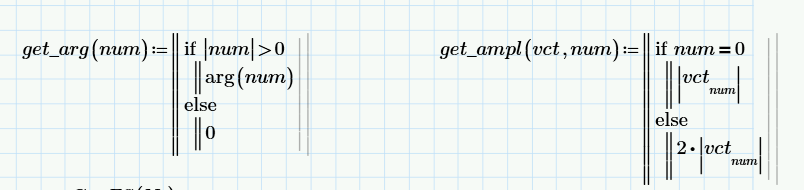


Рис.4 – Функции, возвращающие угол комплексной амплитуды и корректирующие значение комплексной амплитуды

Функция get\_arg принимает значение какой-либо комплексной амплитуды и вычисляет её модуль, если модуль больше нуля, то она возвращает аргумент (угол) этой комплексной амплитуды, иначе возвращает ноль - это сделано для корректной работы функции arg.

Функция get\_ampl принимает значение какой-либо комплексной амплитуды и её номер. Если в функцию подаётся постоянная составляющая, то она возвращает модуль этой постоянной составляющей. А если извлекаются первая и высшие гармоники, то функция вычисляет её модуль и умножает на 2, это необходимое условие, так как рассматриваются только положительные гармоники.

Теперь получим конечный ряд Фурье вычислив сумму постоянной составляющей и остальных гармоник, функция формирующая ряд коэффициентов Фурье представлена на рисунке 7.

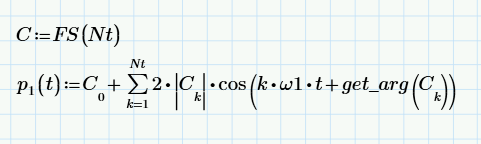


Рис.5 – Функция, формирующая ряд коэффициентов Фурье

В данной реализации мы рассматривали 40 гармоник. Входная последовательность импульсов и сигнал разложенный на ряд коэффициентов Фурье представлен на рисунке 8.

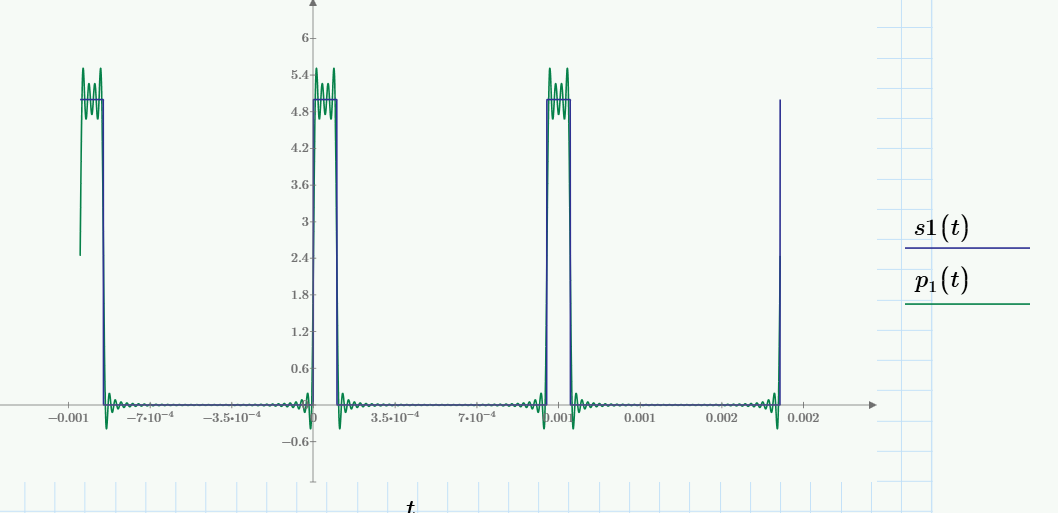


Рис.6 – График входного сигнала и сигнал разложенный на ряд коэффициентов Фурье

Синим цветом изображена исходная последовательность прямоугольных импульсов, а зеленым цветом сигнал, полученный в результате разложения в ряд Фурье. На рисунках 9 и 10 представлены амплитуды и углы комплексных амплитуд соответственно.

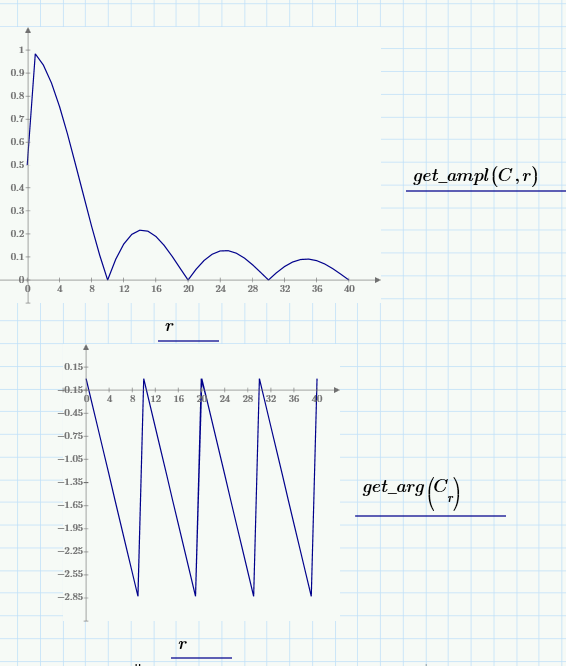


Рис.7 – График амплитуды сигнала после преобразования Фурье

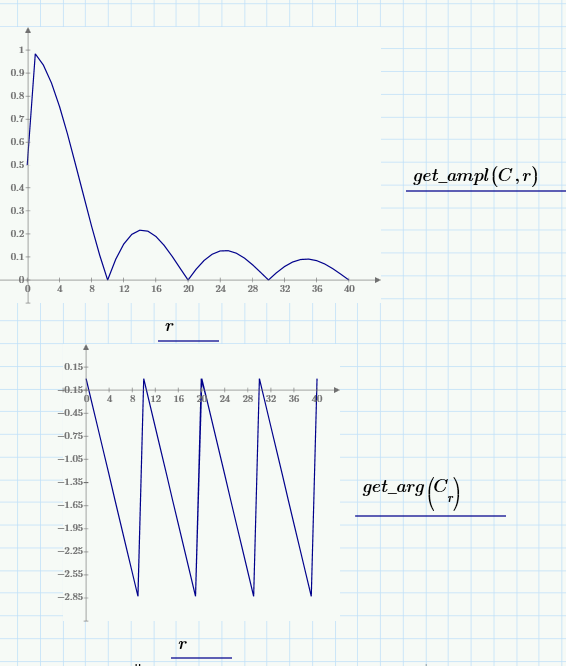


Рис.8 – График углов комплексной амплитуды сигнала

Рассмотрим, как восстанавливаемся исходный сигнал от количества гармоник, на которые разбиваем сигнал.

Если разбивать сигнал на постоянную составляющую и первую гармонику, то он восстановится очень плохо и мало чем похож на первоначальный, если только фазой. График это иллюстрирующий представлен на рисунке 10.

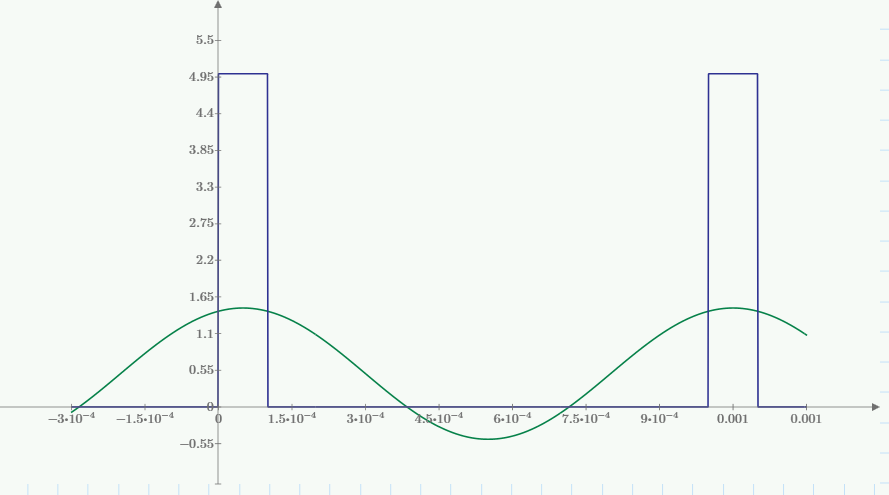


Рис.9 – График исходного и восстановленного сигнала при разложении на 1 гармонику

Если разбивать сигнал на постоянную составляющую, первую гармонику и 4 высших гармоник, то сигнал восстановится уже лучше и будет более похож на первоначальный. График это иллюстрирующий представлен на рисунке 11.

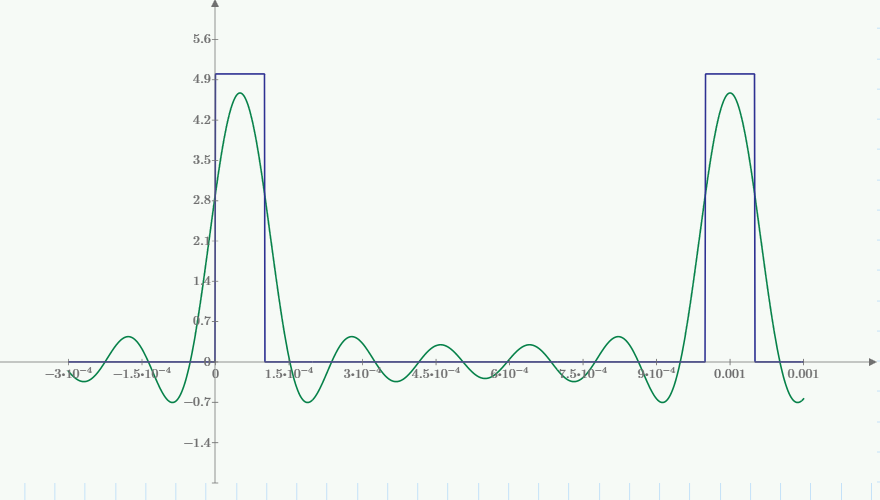


Рис.10 – График исходного и восстановленного сигнала при разложении на 4 гармоники

При разложении сигнала на 40 гармоник, сигнал восстановится почти идеально. График это иллюстрирующий представлен на рисунке 12.

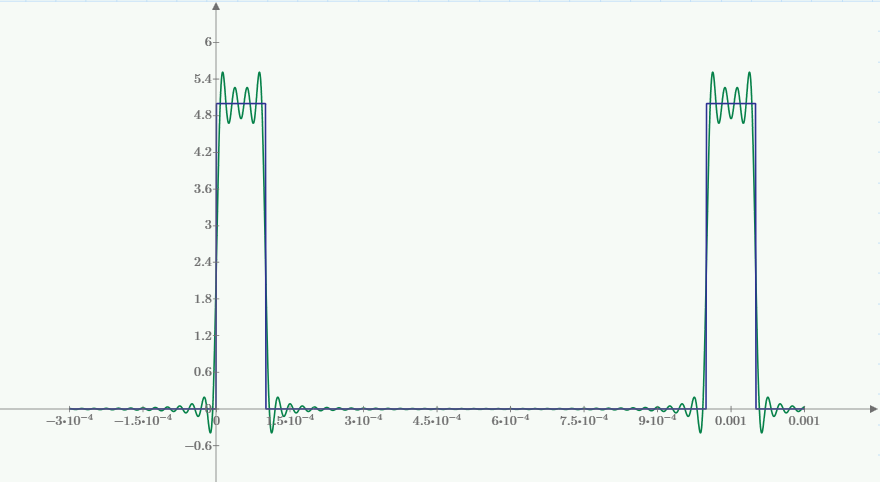
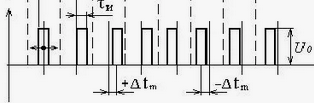


Рис.11 – График исходного и восстановленного сигнала при разложении на 40 гармоник

Так же, помимо важности **числа гармоник**, **следует отметить важность параметра частоты дискретизации**. Нижний порог частоты дискретизации следует выбирать в соответствии с теоремой Котельникова. И чем частота дискретизации выше, то есть чем больше количество точек разбиения, тем лучше сигнал восстановится. Однако, при увеличении количества точек вплоть до бесконечности, на которые мы разбиваем сигнал, не удастся идеально восстановить исходных сигнал. График исходного и восстановленного сигнала при разложении на 4000 точек представлен на рисунке 12.

Попробуем перейти к нашему сигналу: 4 бита имеют 16 комбинаций на комплексной плоскости в виде амплитуды и фазы. Поскольку имеем цифровую обработку, имеется где-то частота дискретизации в виде тактового интервала – на тактовом интервале имеем прямоугольный импульс непонятной пока длительности с амплитудой Аn и фазой 𝞿n. Если брать первый ресурсный элемент (12 частот от 16 до 192 кГц, то для первой частоты 16 кГц полоса от 12 до 24 кГц –> пусть частота дискретизации 50 кГц – такт 20 мкс и импульс с амплитудой Аn и фазой от + π/4 до + 2π-π/4) см рис. 36 данной РПЗ. Всего комбинаций -16 для QAM16.

 Этот импульс раскладывается по sin и cos частот в полосе от 12 до 24 кГц, причем число частот разложения неизвестно, от 8 до 1024 – все кратно 2n . Нужна информация о числе гармоник.

Твои исследования говорят, что 40 достаточно, а 4000- много. Исходя из практических интересов надо или один реальный импульс с амплитудой и фазой (пусть длительность равно 1/10 такта) или все 16 вариантов амплитуды и фазы импульсов поставить друг за другом и подобрать число частот 8-16-32-64-128, которые бы обеспечивали приемлемое описание и восстановление. Отсюда можно, например, упростить приемник, если сразу по временной выборке определять «изображение» четверки бит и сразу их дешифровать без БПФ и демодуляции.

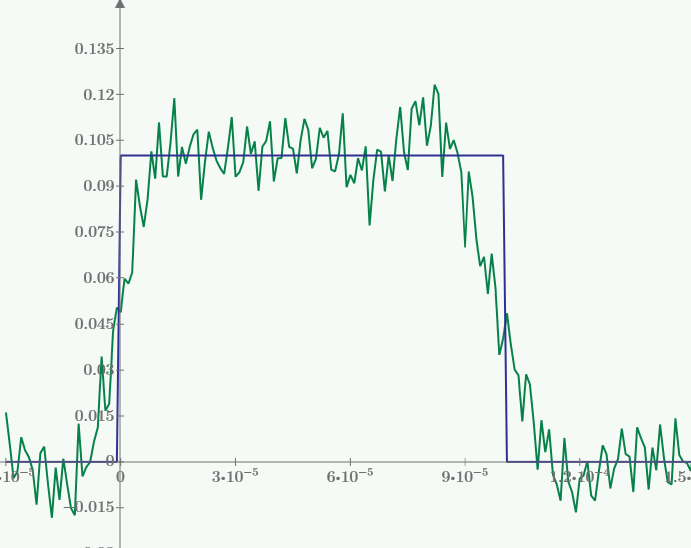


Рис.12 – График исходного и восстановленного сигнала при разложении на 4000 точек

Как много отсчетов не бери, от пульсаций никак не избавиться. При дальнейшем увеличении точек будет увеличиваться количество пульсаций, а их амплитуда останется той же. Это связанно с тем, что то что мы получаем это свертка частотной характеристики и прямоугольного окна.

На рисунке представлена идеальная частотная характеристика и оконная появившееся в результате того, что мы вынуждены брать ограниченное число коэффициентов.

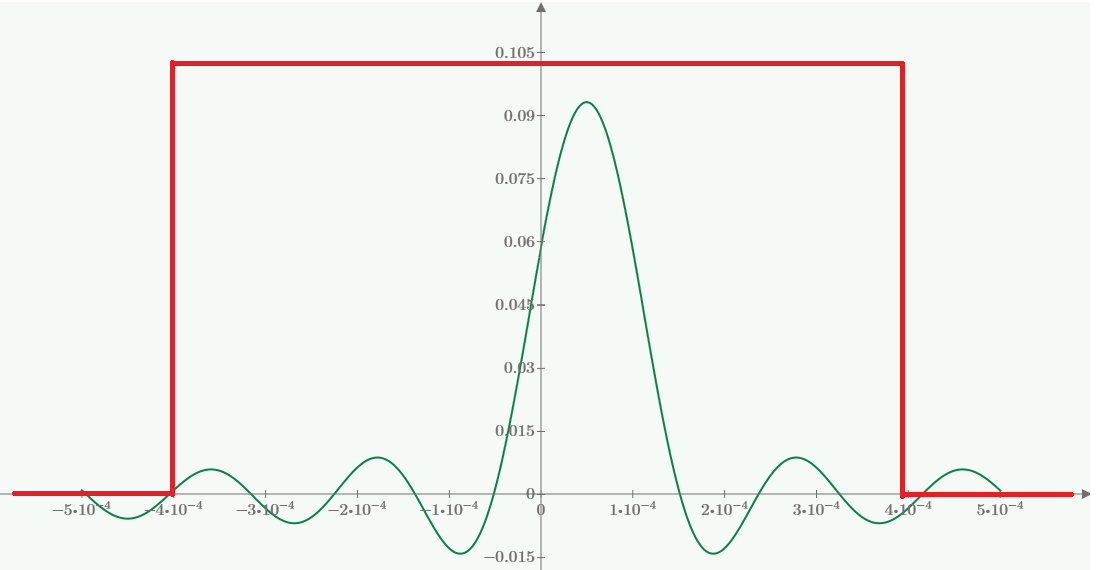


Рис.13 – График идеальной частотной характеристики и оконной функции

Произведение бесконечного числа коэффициентов идеальной частотной характеристики на оконную функцию приводит к свертке их частотных характеристик в частотной области, а свертка приводит к пульсациям.

Вывод: берем большее число коэффициентов, следовательно берем большее окно, однако всё равно будет прямоугольник, а значит пульсации никуда не денутся.

Для борьбы с пульсациями следует использовать не прямоугольные окна, а, к примеру, окна Ханна, Хэмминга, тогда пульсаций можно избежать.

# **5. Реализация преобразований Фурье**

В качестве входного сигнала возьмём сумму 3х синусоид с разными амплитудами, частотами и начальными фазами. Затем будем рассчитывать спектр этого сигнала при помощи разных алгоритмов преобразования Фурье. Формула входного сигнала представлена ниже.

(1)

Необходимо дискретизировать сигнал, для этого будем брать 8 отсчетов на период, то есть , график сигнала во временной области представлен ниже.

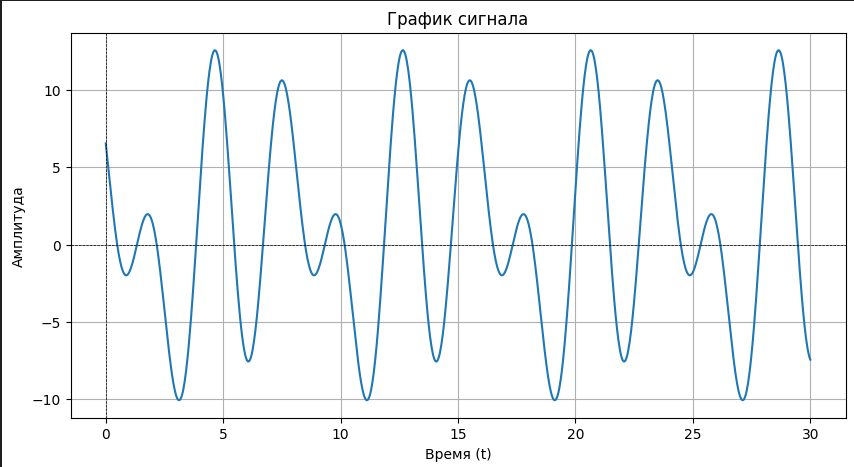


Рис.14 – График входного сигнала во временной области

График представленный на рисунке 1, не корректно отображает входной сигнал, мы имеем дело не с непрерывными сигналами, а с дискретными. Имеются только численные значения сигнала в определенные моменты времени (отсчеты), а какое значение сигнал принимает в моменты времени между отсчетами не известно. Корректный график дискретного представления входного сигнала представлен ниже.

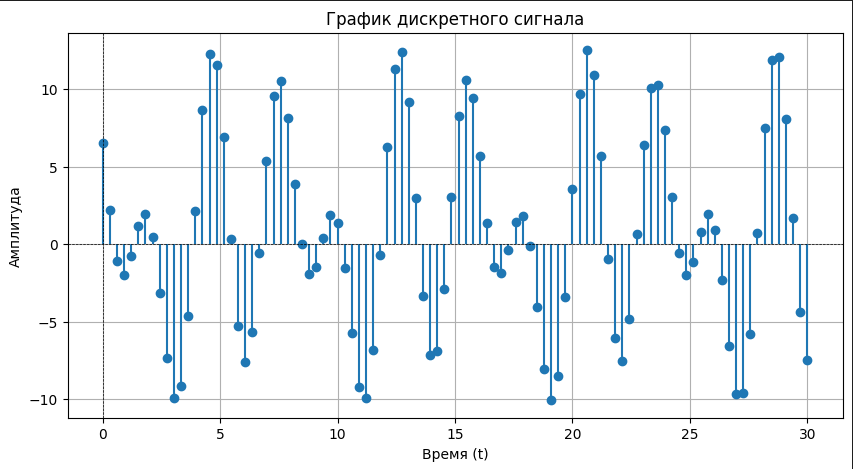


Рис.15 – Дискретное представление входного сигнала во временной области

## **5.1 Реализация дискретного преобразования Фурье**

Применим ДПФ к сигналу из прошлого пункта. Алгоритм основан на том, что мы берем корреляционный интеграл между входным сигналом и функциями sin(x) и cos(x).

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define M\_PI 3.14159265358979323846

#define N 8 // Размерность сигнала

#define SAMPLE\_RATE 100.0 // Частота дискретизации

void dft(const double\* x\_real, const double\* x\_imag, double\* X\_real, double\* X\_imag) {

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

X\_real[k] = 0.0;

X\_imag[k] = 0.0;

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

X\_real[k] += x\_real[n] \* cos(2 \* M\_PI \* k \* n / N) - x\_imag[n] \* sin(2 \* M\_PI \* k \* n / N);

X\_imag[k] += x\_real[n] \* sin(2 \* M\_PI \* k \* n / N) + x\_imag[n] \* cos(2 \* M\_PI \* k \* n / N);

}

}

}

int main() {

double x\_real[N];

double x\_imag[N] = { 0.0 }; // Мнимые значения равны нулю

float p = 2 \* M\_PI / 8;

// Генерация входного сигнала как сумма трёх синусоид

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

x\_real[i] = 1 + 3 \* cos(p \* i + 3.14159265 / 2) + 5 \* cos(2 \* p \* i - M\_PI / 3) + 7 \* cos(3 \* p \* i + 0);

}

double X\_real[N], X\_imag[N];

dft(x\_real, x\_imag, X\_real, X\_imag);

// Вывод результата

printf("Результаты ДПФ:\n");

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

printf("X[%d] = %.4f + %.4fi\n", k, sqrt(X\_real[k]\* X\_real[k]/64 + X\_imag[k]\* X\_imag[k]/64));

}

return 0;

}

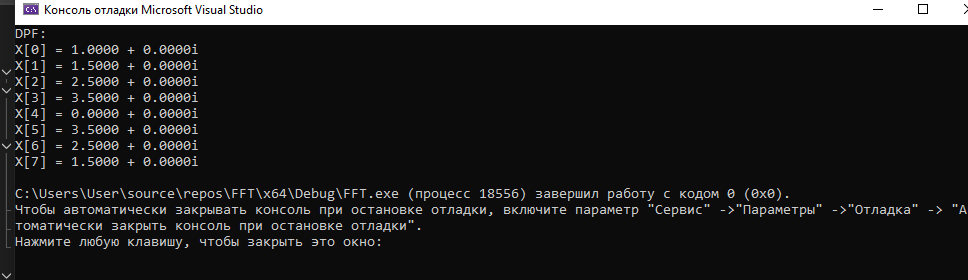


Рис.16 - Результаты ДПФ

Получили численные значения действительной и мнимой составляющих спектра сигнала. Построим спектр сигнала, сложив квадраты действительной и мнимой части.

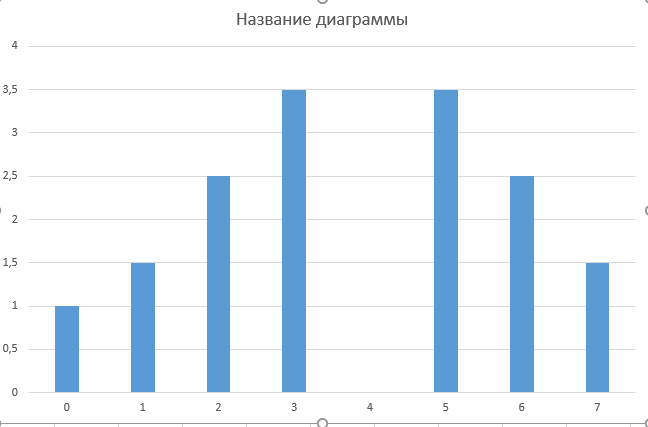


Рис.17 – Спектр сигнала после ДПФ

Здесь на графике по оси X расположены гармоники.

## **5.2 Реализация БПФ на языке С**

#include <stdio.h>

#include <math.h>

#include <stdbool.h>

#define NUMBER\_IS\_2\_POW\_K(x) ((!((x)&((x)-1)))&&((x)>1)) // x is pow(2, k), k=1,2, ...

#define FT\_DIRECT -1 // Direct transform.

#define FT\_INVERSE 1 // Inverse transform.

#define PI 3.14159265358979323846

bool FFT(float\* Rdat, float\* Idat, int N, int LogN, int Ft\_Flag)

{

// parameters error check:

if ((Rdat == NULL) || (Idat == NULL)) return false;

if ((N > 16384) || (N < 1)) return false;

if (!NUMBER\_IS\_2\_POW\_K(N)) return false;

if ((LogN < 2) || (LogN > 14)) return false;

if ((Ft\_Flag != FT\_DIRECT) && (Ft\_Flag != FT\_INVERSE)) return false;

register int i, j, n, k, io, ie, in, nn;

float ru, iu, rtp, itp, rtq, itq, rw, iw, sr;

static const float Rcoef[14] =

{ -1.0000000000000000F, 0.0000000000000000F, 0.7071067811865475F,

0.9238795325112867F, 0.9807852804032304F, 0.9951847266721969F,

0.9987954562051724F, 0.9996988186962042F, 0.9999247018391445F,

0.9999811752826011F, 0.9999952938095761F, 0.9999988234517018F,

0.9999997058628822F, 0.9999999264657178F

};

static const float Icoef[14] =

{ 0.0000000000000000F, -1.0000000000000000F, -0.7071067811865474F,

-0.3826834323650897F, -0.1950903220161282F, -0.0980171403295606F,

-0.0490676743274180F, -0.0245412285229122F, -0.0122715382857199F,

-0.0061358846491544F, -0.0030679567629659F, -0.0015339801862847F,

-0.0007669903187427F, -0.0003834951875714F

};

nn = N >> 1;

ie = N;

for (n = 1; n <= LogN; n++)

{

rw = Rcoef[LogN - n];

iw = Icoef[LogN - n];

if (Ft\_Flag == FT\_INVERSE) iw = -iw;

in = ie >> 1;

ru = 1.0F;

iu = 0.0F;

for (j = 0; j < in; j++)

{

for (i = j; i < N; i += ie)

{

io = i + in;

rtp = Rdat[i] + Rdat[io];

itp = Idat[i] + Idat[io];

rtq = Rdat[i] - Rdat[io];

itq = Idat[i] - Idat[io];

Rdat[io] = rtq \* ru - itq \* iu;

Idat[io] = itq \* ru + rtq \* iu;

Rdat[i] = rtp;

Idat[i] = itp;

}

sr = ru;

ru = ru \* rw - iu \* iw;

iu = iu \* rw + sr \* iw;

}

ie >>= 1;

}

for (j = i = 1; i < N; i++)

{

if (i < j)

{

io = i - 1;

in = j - 1;

rtp = Rdat[in];

itp = Idat[in];

Rdat[in] = Rdat[io];

Idat[in] = Idat[io];

Rdat[io] = rtp;

Idat[io] = itp;

}

k = nn;

while (k < j)

{

j = j - k;

k >>= 1;

}

j = j + k;

}

if (Ft\_Flag == FT\_DIRECT) return true;

rw = 1.0F / N;

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

{

Rdat[i] \*= rw;

Idat[i] \*= rw;

}

return true;

}

// Пример вычисления БПФ от одного периода косинусного

// действительного сигнала

int main()

{

int const len = 100;

static float Re[len];

static float Im[len];

float p = 2 \* PI / len; // будет 8 отсчетов на период

int i;

// формируем сигнал

for (i = 0; i < len; i++)

{

Re[i] = 1+3\*cos(p\*i+ 3.14159265/3)+5\*cos(2\*p\*i- PI /5)+7\*cos(3\*p\*i+ PI /2); // заполняем действительную часть сигнала

Im[i] = 0.0; // заполняем мнимую часть сигнала

}

FFT(Re, Im, len, 3, FT\_DIRECT); // вычисляем прямое БПФ

// выводим действительную и мнимую части спектра и спектр мощности

FILE\* f = fopen("spectrum.txt", "w");

fprintf(f, "# Re Im Abs\n", Re[i], Im[i]);

for (i = 0; i < len; i++)

{

fprintf(f, "%d-> %10.3f %10.3f %10.3f\n", i, Re[i]/8, Im[i]/8, sqrt(Re[i] \* Re[i]/64 + Im[i] \* Im[i]/64));

}

fclose(f);

printf("...FFT Complete");

return 0;

}

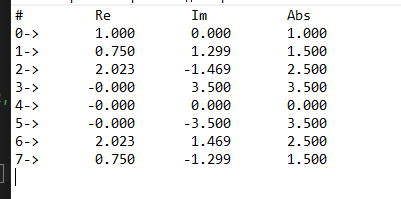


Рис.18 - Результаты БПФ

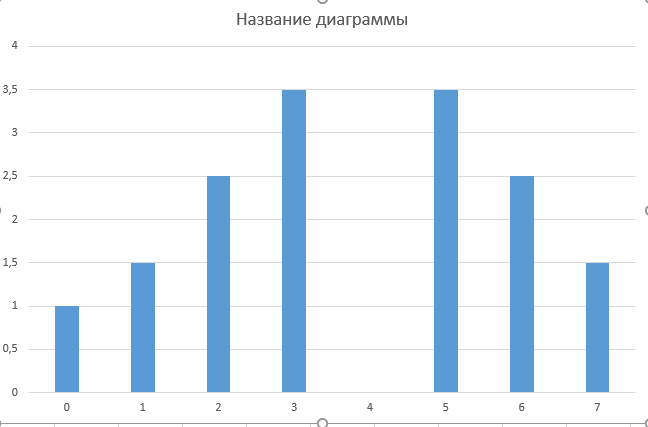


Рис.19 – Спектр сигнала после БПФ

## **5.3 Анализ результатов ДПФ и БПФ**

Спектры, представленные на рисунках 17, 19 полностью совпадают.

Вспомним, что входной сигнал был задан следующим образом.

Где p=, 8 отсчетов на период

На графике по оси X отложены номера гармоник, рассчитаем их частоты

(Fд/количество отсчетов на период) \*номер гармоники

K=0 f=0

K=1 f=(1000/8)\*1=125 Гц

К=2 f=(1000/8)\*2=250 Гц

По графику видно, что амплитуда постоянной составляющей равна 1 и да это верно в исходном сигнале эта величина равна 1 .

Первая, вторая и третья гармоники по частотам вычислились правильно, однако амплитуды у них получились в 2 раза меньше, чем должны были быть.

Пятая, шестая и седьмая гармоники образовались вследствие существования отрицательных частот. Они отображены верно. Пятая гармоника это первая гармоника. Шестая гармоника это вторая гармоника и т.д. Они новой информации не несут.

А что такое четвертая гармоника эта гармоника соответствующая частоте 500 Гц, добавим её в исходный сигнал.

И получим спектр

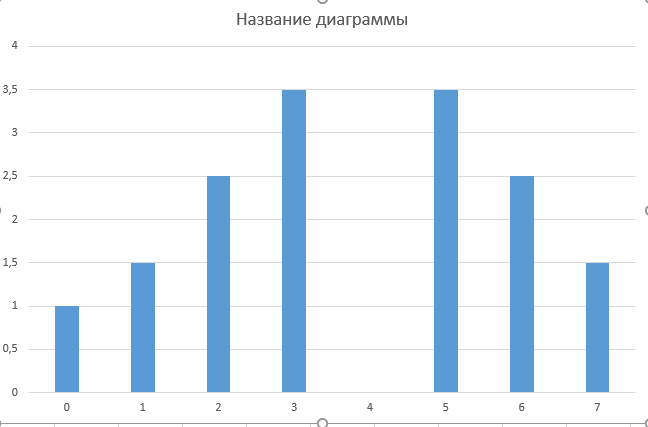


Рис.20 – Спектр сигнала из четырех гармоник

На графике почему-то нет четвертой гармоники, а мы её задавали в исходном сигнале. Теперь изменим начальные фазы сигналов

И получим спектр

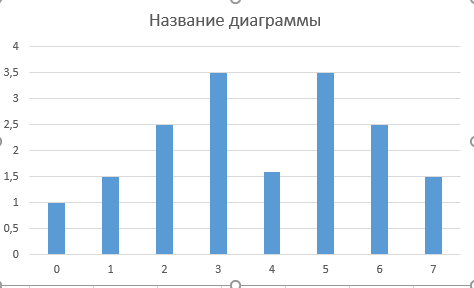


Рис.21 – Спектр сигнала из четырех гармоник с изменением начальной фазы

Как видим, она появилась, а всё дело в том, что четвертая гармоника находится на частоте Найквиста.

# **6. Реализация блока S2P**

Блок S2P принимает входной последовательный поток бит и разбивает его на несколько параллельных потоков. Каждый из этих потоков будет модулирован на отдельной поднесущей и содержать по 4 бита для дальнейшего преобразования этой группы бит в символ.

Ниже представлен код, написанный на языке С++ реализующий группировку входных данных на основе динамического выделения памяти и цикла перебора по 4 элемента.

int numRows = arrayLength / 4; // Количество строк для mas

int\*\* mas = new int\* [numRows]; // Динамическое выделение памяти для mas

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

mas[i] = new int[4]; // Выделение памяти для каждого ряда

for (int j = 0; j < 4; j++) {

mas[i][j] = dataBits[i \* 4 + j];

}

}

std::cout << "Массив mas:\n";

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

printBits(mas[i], 4);

}

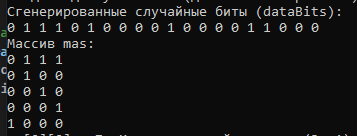


Рис.22 - Данные полученные в результате прохождения входной последовательности через блок S2P

# **7. Реализация OFDM в Matlab**

Код работает при помощи встроенных функций Matlab достаточно корректно за исключением того, что очень странно преобразует биты в символы, однако преобразует символы в биты обратно он таким же странным образом и в результате всё сходится. Входной сигнал совпадает с выходным.

% Параметры OFDM

N = 40; % Число поднесущих

M = 16; % Размерность 16-QAM

numSymbols = 10; % Количество OFDM символов

% Генерация случайных битов

dataBits = randi([0 1], numSymbols \* log2(M) \* N, 1);

disp('Сгенерированные случайные биты (dataBits):');

disp(dataBits);

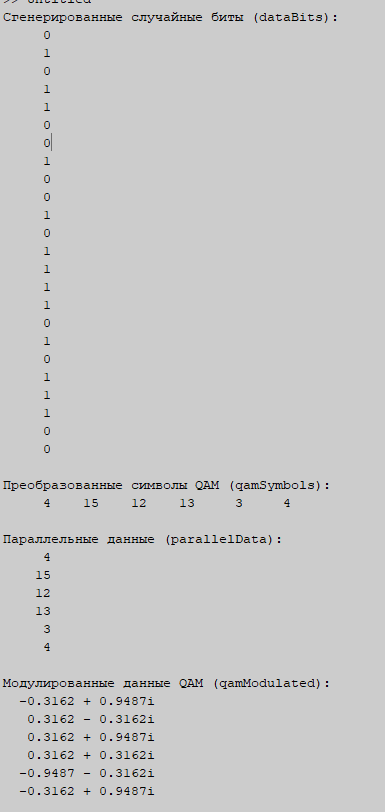


Рис.23 – Сгенерированные случайные биты

% Сериализация в параллель

parallelData = reshape(qamSymbols, N, []).'; % Преобразование в N поднесущих

disp('Параллельные данные (parallelData):');

disp(parallelData);

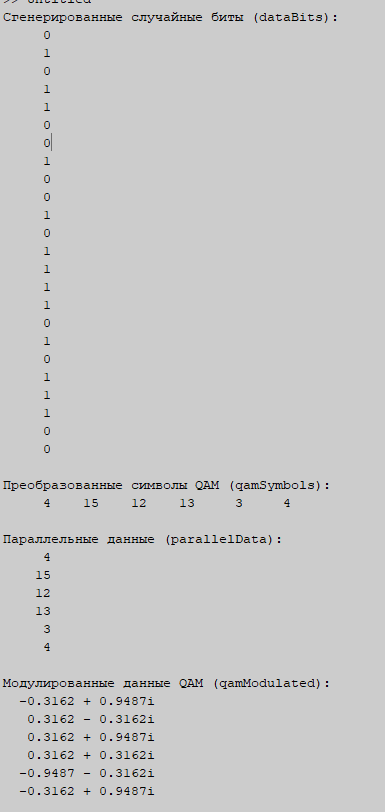


Рис.24 – Биты после S2P

% Преобразование бит в символы

dataSymbols = reshape(dataBits, [], log2(M));

qamSymbols = bi2de(dataSymbols)'; % Преобразование бинарных символов в десятичные

disp('Преобразованные символы QAM (qamSymbols):');

disp(qamSymbols);

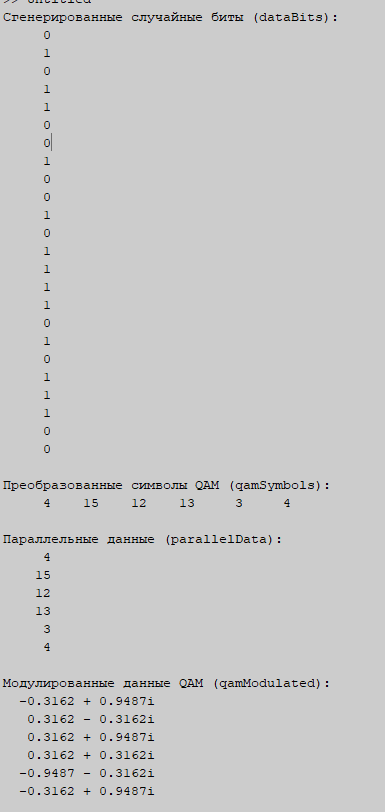


Рис.25 – Символы полученные из бит

% Модуляция 16-QAM

qamModulated = qammod(parallelData(:), M, 'UnitAveragePower', true); % Модуляция

disp('Модулированные данные QAM (qamModulated):');

disp(qamModulated);

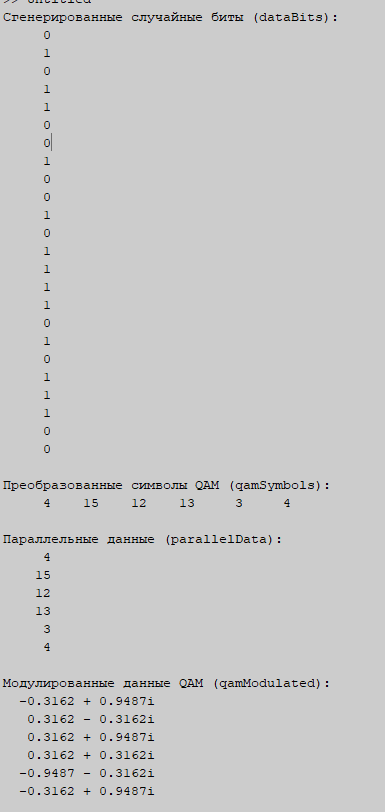


Рис.26 – Данные после 16QAM

% Применение IFFT к каждому OFDM символу

ofdmSymbols = ifft(reshape(qamModulated, [], N), N, 2); % IFFT по строкам

disp('OFDM символы (ofdmSymbols):');

disp(ofdmSymbols);

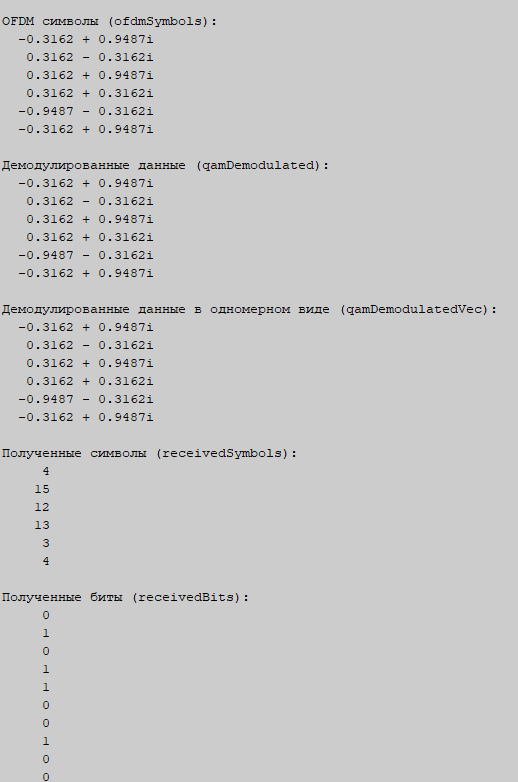


Рис.27 – Данные после 16QAM

% Визуализация

figure;

subplot(2,1,1);

plot(real(ofdmSymbols(:)), imag(ofdmSymbols(:)), 'o');

title('Символы OFDM');

xlabel('Действительная часть');

ylabel('Мнимая часть');

grid on;

subplot(2,1,2);

plot(abs(ofdmSymbols(:)));

title('Амплитуда символов OFDM');

xlabel('Индекс');

ylabel('Амплитуда');

grid on;

% Применение FFT к каждому OFDM символу для демодуляции

ifftData = reshape(ofdmSymbols, [], N); % Возвращаем из строкового вида

qamDemodulated = fft(ifftData, N, 2); % FFT по строкам

disp('Демодулированные данные (qamDemodulated):');

disp(qamDemodulated);

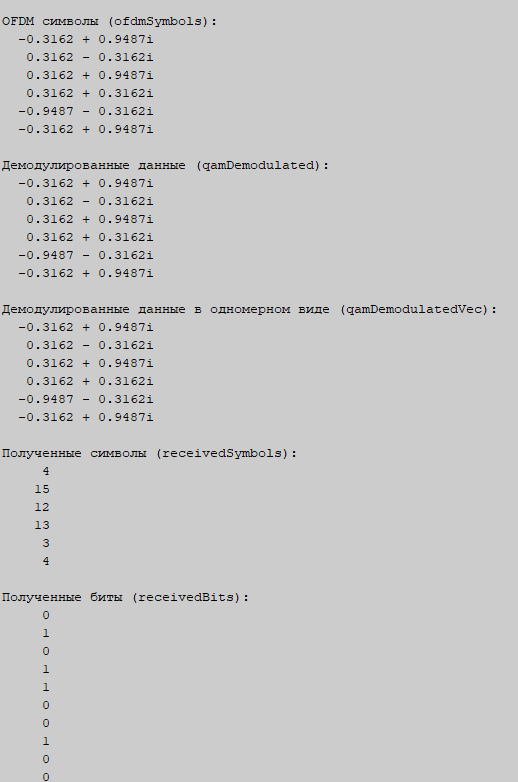


Рис.28 – Данные после демодуляции 16QAM

% Преобразование обратно в одно измерение

qamDemodulatedVec = qamDemodulated(:);

disp('Демодулированные данные в одномерном виде (qamDemodulatedVec):');

disp(qamDemodulatedVec);

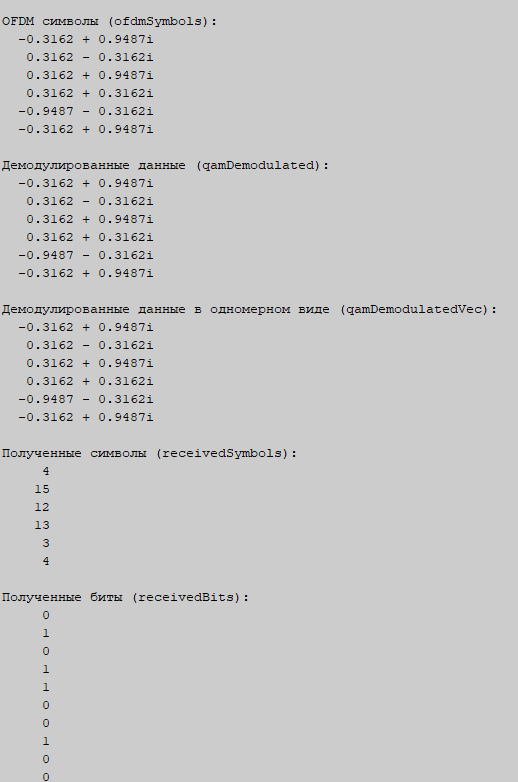


Рис.29 – Демодулированные данные в одномерном виде

% Демодуляция 16-QAM

receivedSymbols = qamdemod(qamDemodulatedVec, M, 'UnitAveragePower', true); % Демодуляция

disp('Полученные символы (receivedSymbols):');

disp(receivedSymbols);

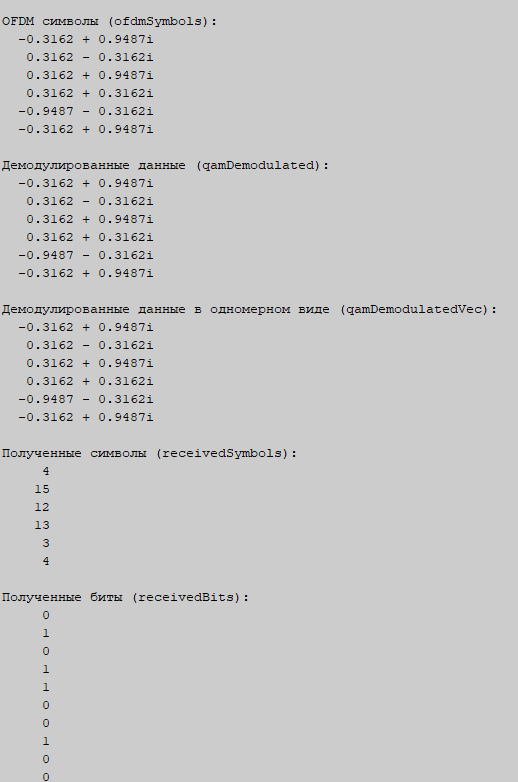


Рис.30 – Символы полученные из демодулированных данных

% Преобразование символов обратно в биты

receivedBits = de2bi(receivedSymbols', log2(M)); % Преобразование десятичных символов обратно в двоичные

receivedBits = receivedBits(:); % Приведение к одномерному виду

disp('Полученные биты (receivedBits):');

disp(receivedBits);



Рис.31 – Биты полученные после работы всех функций

% Восстановление исходного сигнала

% Проверим, совпадает ли с оригинальными битами

isEqual = isequal(receivedBits, dataBits);

if isEqual

disp('Восстановленный сигнал совпадает с исходным.');

else

disp('Восстановленный сигнал НЕ совпадает с исходным.');

end



Рис.32 – Результат проверки на совпадение входных и выходных данных

Графики, отображающие действительное и мнимое значение символов OFDM, а так же их амплитуды представлены ниже.

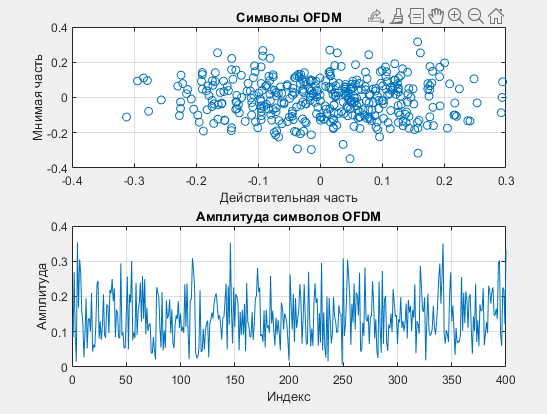


Рис.33 – Графики сигналов OFDM для 40 поднесущих

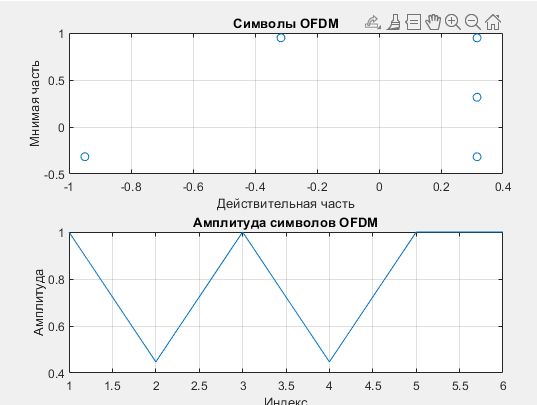


Рис.34 – Графики сигналов OFDM для 1 поднесущей

# **8. Реализация OFDM на языке C++**

Код выполняет блоки S2P, mapper, 16QAM, IFFT

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cmath>

#include <cstring>

#include <complex>

#include <vector>

//функция вывода массива битов

void printBits(const int\* bits, int length) {

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

std::cout << bits[i] << " ";

}

std::cout << std::endl;

}

//функция перевода из двоичной системы в десятичную

int binaryToDecimal(const int\* binaryArray, int length) {

int decimalValue = 0;

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

decimalValue = (decimalValue << 1) | binaryArray[i];

}

return decimalValue;

}

// Функция для модуляции 16QAM

std::complex<double> modulate16QAM(int symbol) {

int I = (symbol % 4) \* 2 - 3; // I-компонента

int Q = (symbol / 4) \* 2 - 3; // Q-компонента

return std::complex<double>(I, Q);

}

// Функция для выполнения обратного БПФ

ifft(const std::vector<std::complex<double>>

std::vector<std::complex<double>>& input) {

int N = input.size();

std::vector<std::complex<double>> output(N);

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

output[k] = 0;

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

output[k] += input[n] \* std::exp(std::complex<double>(0, 2 \* M\_PI \* k \* n / N));

}

output[k] /= N; // Нормализация

}

return output;

}

// Функция для добавления циклического префикса

std::vector<std::complex<double>> addCyclicPrefix(const std::vector<std::complex<double>>& symbols, int cpLength) {

std::vector<std::complex<double>> txSymbols(cpLength + symbols.size());

std::copy(symbols.end() - cpLength, symbols.end(), txSymbols.begin());

std::copy(symbols.begin(), symbols.end(), txSymbols.begin() + cpLength);

return txSymbols;

}

void printComplexArray(const std::vector<std::complex<double>>& array) {

for (const auto& value : array) {

std::cout << value << " ";

}

std::cout << std::endl;

}

int main() {

std::setlocale(LC\_ALL, "Russian");

int arrayLength;

std::cout << "Введите длину массива (должна быть кратна 4): ";

std::cin >> arrayLength;

// Проверка на кратность 4

while (arrayLength % 4 != 0) {

std::cout << "Длина должна быть кратна 4. Повторите ввод: ";

std::cin >> arrayLength;

}

// Генерируем массив случайных битов

int\* dataBits = new int[arrayLength];

std::srand(static\_cast<unsigned int>(std::time(0))); // Инициализация генератора случайных чисел

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

dataBits[i] = std::rand() % 2; // Случайное значение 0 или 1

}

std::cout << "Сгенерированные случайные биты (dataBits):\n";

printBits(dataBits, arrayLength);

int numRows = arrayLength / 4; // Количество строк для mas

int\*\* mas = new int\* [numRows]; // Динамическое выделение памяти для mas

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

mas[i] = new int[4]; // Выделение памяти для каждого ряда

for (int j = 0; j < 4; j++) {

mas[i]

[j] = dataBits[i \* 4 + j];

}

}

std::cout << "Массив mas:\n";

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

printBits(mas[i], 4);

}

int\*\* ww = new int\* [numRows]; // Динамическое выделение памяти для ww

std::complex<double>\* modulatedSymbols = new std::complex<double>[numRows];

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

ww[i] = new int[1]; // Выделение памяти для каждого элемента

ww[i][0] = binaryToDecimal(mas[i], 4);

modulatedSymbols[i] = modulate16QAM(ww[i][0]); // Модуляция 16QAM

}

// Выполнение преобразования Фурье (IFFT)

std::vector<std::complex<double>> ifftResult = ifft(std::vector<std::complex<double>>(modulatedSymbols, modulatedSymbols + numRows));

// Вывод результатов после IFFT

std::cout << "Результат после IFFT:\n";

printComplexArray(ifftResult);

// Добавление циклического префикса

int cpLength = 4; // длина циклического префикса

std::vector<std::complex<double>> txSymbols = addCyclicPrefix(ifftResult, cpLength);

// Вывод символов с циклическим префиксом

std::cout << "Символы с циклическим префиксом:\n";

printComplexArray(txSymbols);

// Вывод результата

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

std::cout << "ww[" << i << "][0] = " << ww[i][0] << ", Модулированный символ: "

<< modulatedSymbols[i] << std::endl;

}

std::cout << "Преобразование Фурье и добавление циклического префикса завершено.\n";

// Освобождение памяти

return 0;

}

Результат выполнения кода

В массиве mas находятся сгруппированный по 4 биты

В массиве WW находятся десятичная запись двоичных чисел из массива WW

А затем осуществляется модуляция 16 QAM

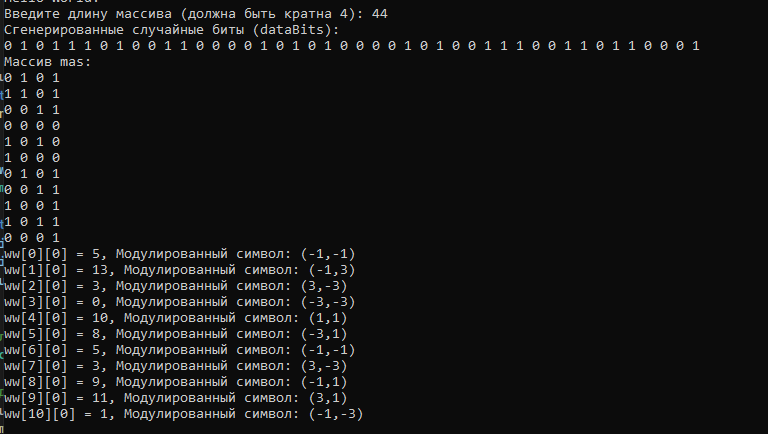
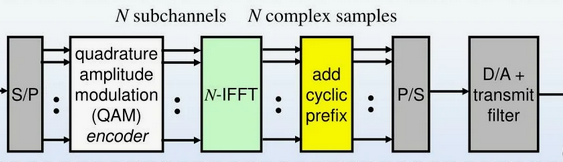


Рис.36 – Данные полученные в результате выполнения кода

Описана процедура модуляции (представления бит в виде временного изображения на основании ОБПФ) и обратной демодуляции сигнала (БПФ) и выделением этой же группы бит.

Рассмотрено преобразование одной группы бит во временнОе изображение на основе ОБПФ в комплексной форме с разделением цифрового сигнала на синфазный и квадратурный в одном подканале. В таком одноканальном варианте возможно применение микроконтроллера для поэтапного последовательного выполнения всех заданных функций цифровой части формирования модулирующего сигнала. Эти функции хорошо известны:



Комплексное БПФ потребует последовательного применения одного - двух микроконтроллеров (или одного DSP IP-блока) для вычисления двух массивов комплексной части ОБПФ и, соответственно, оценки времени вычисления, которое ограничено (время одного подкадра OFDM не превышает 0.5 мсек). Ограничения вводят и частоты дискретизации в подканалах, которые в первом РЕ изменяются от 50 до 600 кГц

Пока, для отладки последовательного кода и оценки производительности, используем весьма производительный и распространенный микроконтроллер STM32 с тактовой частотой 72 МГц (до 432 МГц для высших серий) и развитым ПО.

**9. Реализация OFDM на микроконтроллере STM32**

В качестве железа был выбран микроконтроллер STM32 с отладочной платой Nucleo 64.

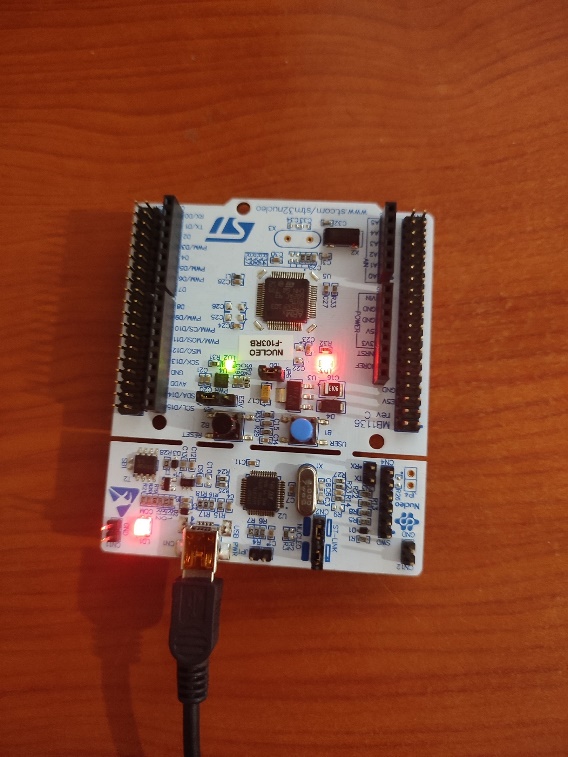


Рис.37 – Отладочная плата Nucleo-64 с микроконтроллером STM32

Среда в которой производилась прошивка микроконтроллера STM32CubeIDE. После загрузки всех пакетов для корректной работы микроконтроллера и теста моргания диодами при помощи функций SET, RESET и DELAY пришли к выводу, что микроконтроллер работает корректно. Далее, необходимо было загрузить стандартную библиотеку <stdio.h> для написания программ на языке C.

Как правило, при применении микроконтроллера проводят оценку необходимой при загрузке памяти с учетом всех применяемых библиотек и предполагаемого кода.

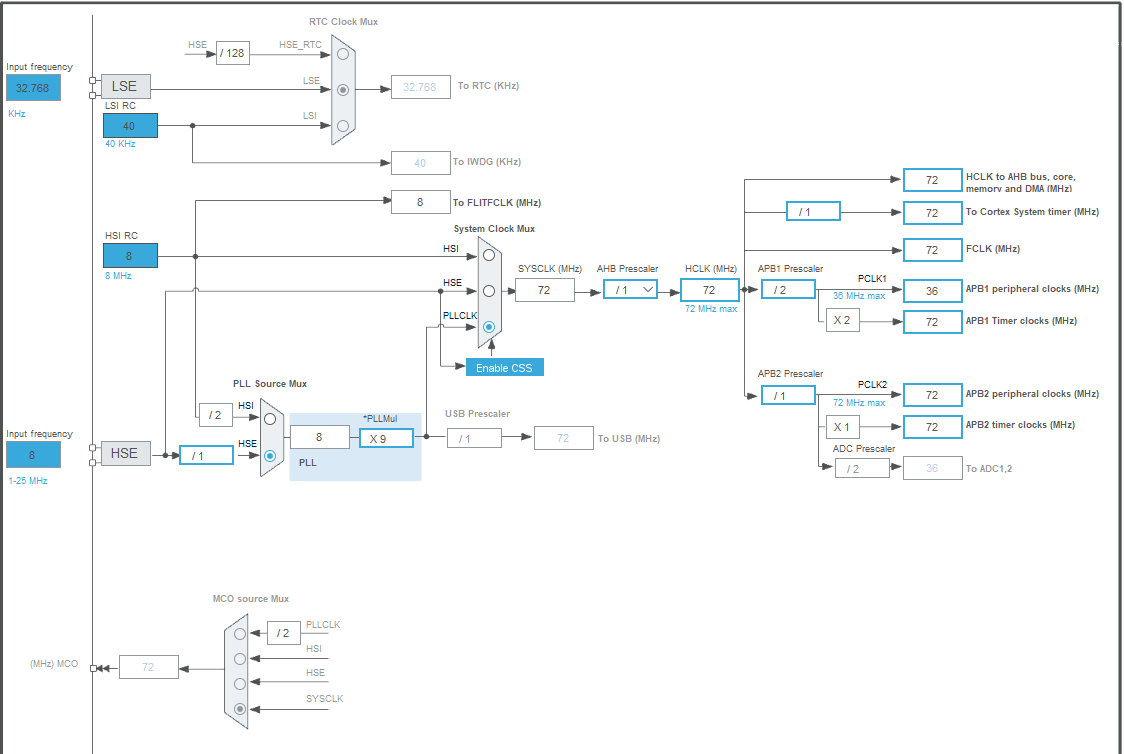


Рис.38 – Конфигурация тактового сигнала

Были внесены изменения в заголовочный файл syscalls.c, а именно

**void** **ITM\_SendChar**(uint8\_t ch)

{

DEMCR |= (1 <<24);

ITM\_TRACE\_EN |= (1<<0);

**while**(!(ITM\_STIMULUS\_PORT0 & 1));

ITM\_STIMULUS\_PORT0 = ch;

}

**for** (DataIdx = 0; DataIdx < len; DataIdx++)

{

//\_\_io\_putchar(\*ptr++);

ITM\_SendChar(\*ptr++);

}

Далее в main файла main.c в цикле while(1) запишем наш алгоритм OFDM.

Полный main файл представлен ниже.

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2024 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

**#include** "main.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

UART\_HandleTypeDef huart2;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

**void** **SystemClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_USART2\_UART\_Init**(**void**);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

**int** **main**(**void**)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

**HAL\_Init();**

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

**SystemClock\_Config();**

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_USART2\_UART\_Init();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

**void** **printBits**(**const** **int**\* bits, **int** length) {

**for** (**int** i = 0; i < length; i++) {

**printf**("%d ", bits[i]);

}

**printf**("\n");

}

**int** **binaryToDecimal**(**const** **int**\* binaryArray, **int** length) {

**int** decimalValue = 0;

**for** (**int** i = 0; i < length; i++) {

decimalValue = (decimalValue << 1) | binaryArray[i];

}

**return** decimalValue;

}

**int** arrayLength=16;

**int** dataBits[arrayLength];

**for**(**int** i = 0; i < arrayLength; i++) {

dataBits[i] = 0;

}

**for** (**int** i = 0; i < arrayLength; i++) {

dataBits[i] = **rand**() % 2; // Случайное значение 0 или 1

}

**printf**("Bits (dataBits):\n");

printBits(dataBits, arrayLength);

**const** **int** numRows = arrayLength / 4;

**int** mas[numRows][4];

**for** (**int** i = 0; i < numRows; i++) {

**for** (**int** j = 0; j < 4; j++) {

mas[i][j] = dataBits[i \* 4 + j]; // Копируем данные

}

}

**printf**("mas:\n");

**for** (**int** i = 0; i < numRows; i++) {

printBits(mas[i], 4); // Выводим каждую строку массива

}

**int** ww[numRows][1];

**int** mod\_16\_I[numRows][1];

**int** mod\_16\_Q[numRows][1];

**for** (**int** i = 0; i < numRows; i++) {

ww[i][0] = binaryToDecimal(mas[i], 4);

mod\_16\_I[i][0]=(ww[i][0] % 4) \* 2 - 3;

mod\_16\_Q[i][0]=(ww[i][0] / 4) \* 2 - 3;

}

**printf**("ww:\n");

**for** (**int** i = 0; i < numRows; i++) {

printBits(ww[i], 1); // Выводим каждую строку массива

}

**printf**("I:\n");

**for** (**int** i = 0; i < numRows; i++) {

printBits(mod\_16\_I[i], 1);

}

**printf**("Q:\n");

**for** (**int** i = 0; i < numRows; i++) {

printBits(mod\_16\_Q[i], 1);

}

**printf**("HELLO WORLD1\n");

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, *GPIO\_PIN\_SET*);

HAL\_Delay(1000);

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, *GPIO\_PIN\_RESET*);

HAL\_Delay(1000);

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

**void** **SystemClock\_Config**(**void**)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_BYPASS;

RCC\_OscInitStruct.HSEPredivValue = RCC\_HSE\_PREDIV\_DIV1;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL9;

**if** (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != *HAL\_OK*)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

**if** (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != *HAL\_OK*)

{

Error\_Handler();

}

}

/\*\*

\* @brief USART2 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_USART2\_UART\_Init**(**void**)

{

/\* USER CODE BEGIN USART2\_Init 0 \*/

/\* USER CODE END USART2\_Init 0 \*/

/\* USER CODE BEGIN USART2\_Init 1 \*/

/\* USER CODE END USART2\_Init 1 \*/

huart2.Instance = USART2;

huart2.Init.BaudRate = 115200;

huart2.Init.WordLength = UART\_WORDLENGTH\_8B;

huart2.Init.StopBits = UART\_STOPBITS\_1;

huart2.Init.Parity = UART\_PARITY\_NONE;

huart2.Init.Mode = UART\_MODE\_TX\_RX;

huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;

**if** (HAL\_UART\_Init(&huart2) != *HAL\_OK*)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART2\_Init 2 \*/

/\* USER CODE END USART2\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_GPIO\_Init**(**void**)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(LD2\_GPIO\_Port, LD2\_Pin, *GPIO\_PIN\_RESET*);

/\*Configure GPIO pin : B1\_Pin \*/

GPIO\_InitStruct.Pin = B1\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_RISING;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(B1\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : LD2\_Pin \*/

GPIO\_InitStruct.Pin = LD2\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(LD2\_GPIO\_Port, &GPIO\_InitStruct);

/\* EXTI interrupt init\*/

HAL\_NVIC\_SetPriority(*EXTI15\_10\_IRQn*, 0, 0);

HAL\_NVIC\_EnableIRQ(*EXTI15\_10\_IRQn*);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

**void** **Error\_Handler**(**void**)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

**while** (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

**#ifdef** USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

**void** assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

**#endif** /\* USE\_FULL\_ASSERT \*/

Результат работы микроконтроллера представлен ниже

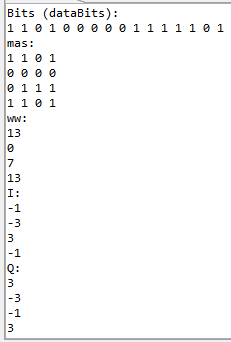


Рис.39 – Данные полученные в результате работы микроконтроллера STM-32

Как видим, результат полностью соответствует тому, что мы получали в коде, реализованном на языке C++ в среде VS Code.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта были детального изучены и реализованы алгоритмы ДПФ и БПФ. Была создана модель LTE OFDM в среде Matlab, а также код полностью реализующий алгоритм формирования OFDM сигнала на языке C++. Проведена работа по настройки микроконтроллера STM-32, а также осуществилась прошивка микроконтроллера на языке С для реализации OFDM. Проект позволил глубже понять принципы работы OFDM, включая модуляцию, демодуляцию, а также обработку сигналов, основанных на ортогональных частотах. Использование STM32 обеспечило необходимые вычислительные ресурсы и гибкость для реализации данных алгоритмов.

В будущем стоит рассмотреть возможность исследования всех 16 точек созвездия и определения количества гармоник для их однозначной реализации, а также оптимизации алгоритмов для повышения их мощности и адаптации системы к различным условиям передачи. Также можно исследовать применение современных подходов, таких как адаптивные схемы модуляции и кодирования, что позволит дополнительно улучшить характеристики системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ричард Лайонс “Цифровая обратока сигналов”, 2006 г. - 656 с.: ил.
2. Лохвицкий М.С., Мардер Н.С. “Сотовая связь: от поколения к поколению”, 2014. – C.
3. H. Elshaer, F. Boccardi, M. Dohler, R. Irmer “Downlink and Uplink Decoupling: A disruptive architectural design for 5G networks”, Global Communications Conference (GLOBECOM), pp. 1798 – 1803, 2014.
4. Ahmed Almajdoob: “Design and FPGA Implementation of an OFDM System Based on 3GPP LTE Standard over Multipath Fading Channel”, pp. 1-119, 2012.
5. K. Jallouli1, M. Mazouzi, J. Diguet, A. Monemi and S. Hasnaoui1 “MIMO-OFDM LTE System based on a parallel IFFT/FFT on NoC-based FPGA”, pp. 1-10, 2021.
6. Huang J, Ruan F, Su M, Yang X, Yao S, Zhang J (2016) Analysis of orthogonal frequency division multiplexing (OFDM) technology in wireless communication process. In: 2016 10th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID), IEEE, pp 122–125.
7. Hui Liu, Guoqing Li OFDM-Based Broadband Wireless Networks: Design and Optimization. - 2005. - 264 с.
8. K. Fazel, S. Kaiser Multi-Carrier and Spread Spectrum Systems: From OFDM and MC-CDMA to LTE and WiMAX. - 2-е изд. - Ericsson GmbH, Germany: A John Wiley and Sons, Ltd, Publication, 2008. - 380 с.
9. Шахнович И. Современные технологии беспроводной связи. - 2-е изд. - Техносфера, 2006. - 288 с.