[**Введение** 2](#_Toc200917960)

[**Постановка задачи** 7](#_Toc200917961)

[**Анализ основных блоков алгоритма** 8](#_Toc200917962)

[**S2P** 8](#_Toc200917963)

[**Mapper** 8](#_Toc200917964)

[**IFFT** 13](#_Toc200917965)

[**P2S** 24](#_Toc200917966)

[**Оптимизация алгоритма** 25](#_Toc200917967)

[**Проблема синхронизации** 27](#_Toc200917968)

[**Приоритетность** 32](#_Toc200917969)

[**Проверка алгоритма на микроконтроллерах** 35](#_Toc200917970)

[**Заключение** 38](#_Toc200917971)

[**Приложение 1** 39](#_Toc200917972)

[**Принцип работы приемопередатчика на основе OFDM** 39](#_Toc200917973)

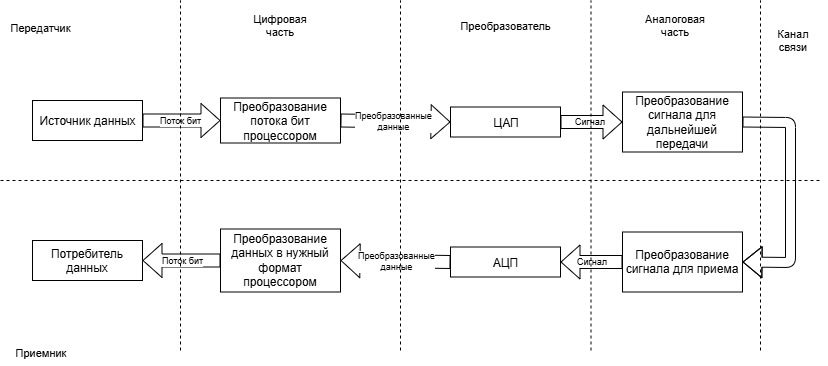
[**Приложение 2.** 43](#_Toc200917974)

[**Модуляция** 43](#_Toc200917975)

# **Введение**

Цифровой модуль приемо-передачи на основе OFDM-модуляции.

Современные информационные потоки передаются в виде двоичных данных. Если взять такую систему передачи информации как смартфон, то его можно условно разделить на аналоговую и цифровую части. В данной работе рассматривается только цифровая часть блока приема и передачи информации.

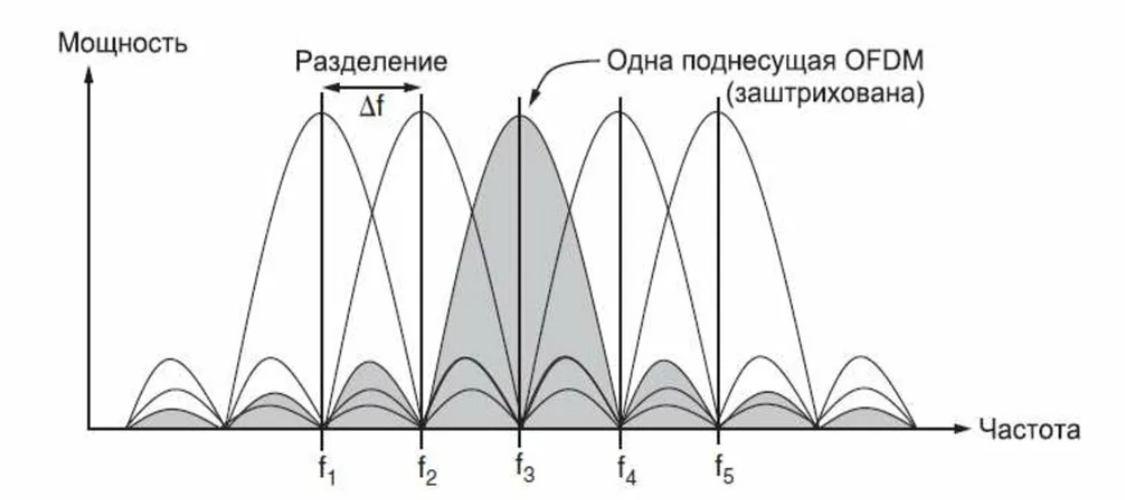


*Рис. 1. Разделение устройства передачи информации на аналоговую и цифровую часть.*

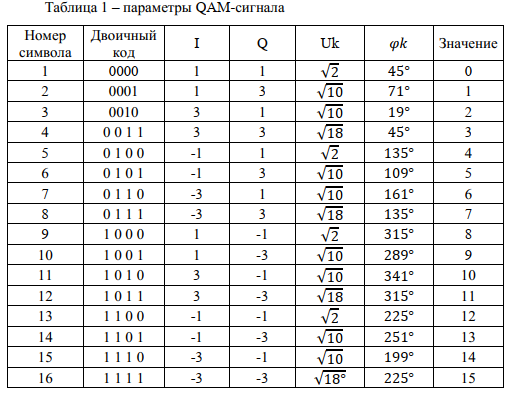
В аналоговой части осуществляется прием сигнала на сверхвысоких частотах (СВЧ, радиоволны в диапазоне частот от 300 МГц до 300 ГГц), преобразование частоты, усиление и фильтрация сигнала. На сигнал действуют шумы, происходит многолучевое распространение в результате переотражения от протяженных предметов – домов, и эффекты смещения частоты – эффект Доплера. В результате частоты, несущие полезную информацию, накладываются друг на друга; происходит потеря информации. Для решения этих проблем используется технология OFDM.

OFDM (мультиплексирование с ортогональным частотным разделением каналов) – технология цифровой связи, в которой сигнал СВЧ образуется из большого количества близко расположенных низкочастотных ортогональных поднесущих – частот, которые занимают ограниченную полосу. Спектры разных поднесущих перекрываются, но не мешают друг другу вследствии правильного выбора временного интервала, на котором все частоты кратны и ортогональны друг другу. OFDM сигнал представлен на рис.2.

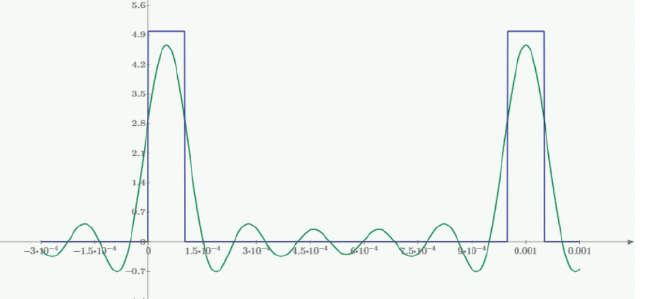
Информация о битах записывается и передается в виде амплитуды и фазы прямоугольных импульсов. Например, при манипуляции 16 QAM все 16 вариантов передачи 4-х бит сведены в таблицу 1.

****

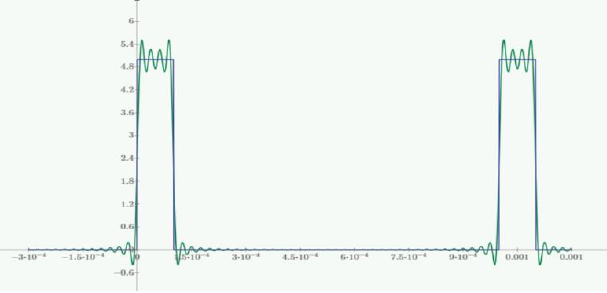
*Рис. 2. Структура OFDM сигнала*

**

Прямоугольные импульсы широкополосны (занимают широкий спектр), поэтому для передачи по узкому каналу их заменяют суммой синусоидальных волн (частот) в полосе текущей поднесущей. Сколько нужно таких волн? Ответить на данный вопрос можно, разложив прямоугольный импульс в ряд Фурье. Используя 8 частот, получится приближение, показанное на рис. 3, оно напоминает начальный сигнал, но не в полной мере его отражает. Хорошее приближение к истинному виду сигнала достигается при использовании 64 частот или более, как представлено на рис. 4.

**

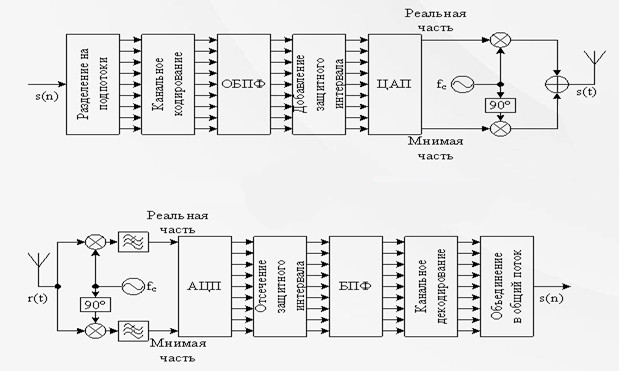
*Рис. 3. Разложение прямоугольного сигнала в ряд Фурье с использованием 8 частот*

**

*Рис. 4. Разложение прямоугольного сигнала в ряд Фурье с использованием 64 частот*

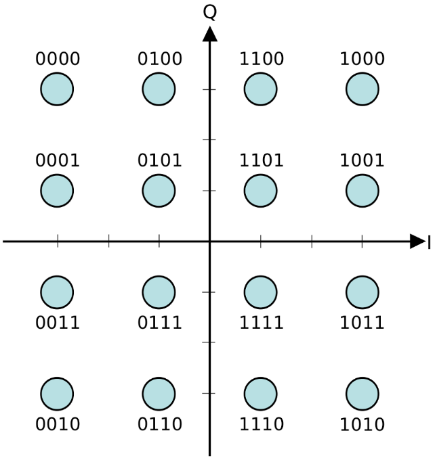
При увеличении числа частот при разложении, итоговый сигнал будет еще точнее повторять исходный, однако при увеличении количества частот увеличивается и время расчета ряда в дальнейших вычислениях. Более подробно эта проблема раскрывается далее в главе «IFFT».

Из основного свойства OFDM следует, что входной поток бит разделяется на тысячи параллельных подканалов, в которых битовые интервалы замедлены в тысячи раз, передаются в виде низкочастотных колебаний и слабо зависят изменения несущего сигнала во времени, по фазе и частоте.. В каждом подканале скорость обработки в N раз меньше, где N – число подканалов. Распараллеливание потока технически выполняет демультиплексор и он описывается в виде алгоритма на языке С.



*Рис. 5. Схема работы приемника и передатчика по алгоритму OFDM*

В следующем блоке канального кодирования биты группируются в зависимости от отношения сигнал/шум и отображаются точками на комплексной плоскости.



*Рис. 5. Пример, соотношение 4 бит на комплексной плоскости.*

После этого комплексные числа, описывающие амплитуду и фазу каждой точки созвездия с помощью некоторого числа частот в полосе поднесущей при помощи обратного преобразования Фурье (IFFT) преобразуются в амплитудный спектр во временной области, ужимаются (мультиплексируются), подаются на ЦАП и модулируют СВЧ несущую.

В приемнике осуществляются обратные действия с учетом необходимости синхронизации по времени, частоте, амплитуде и фазе. В работе не рассмотрено введение и отсечение защитного интервала.

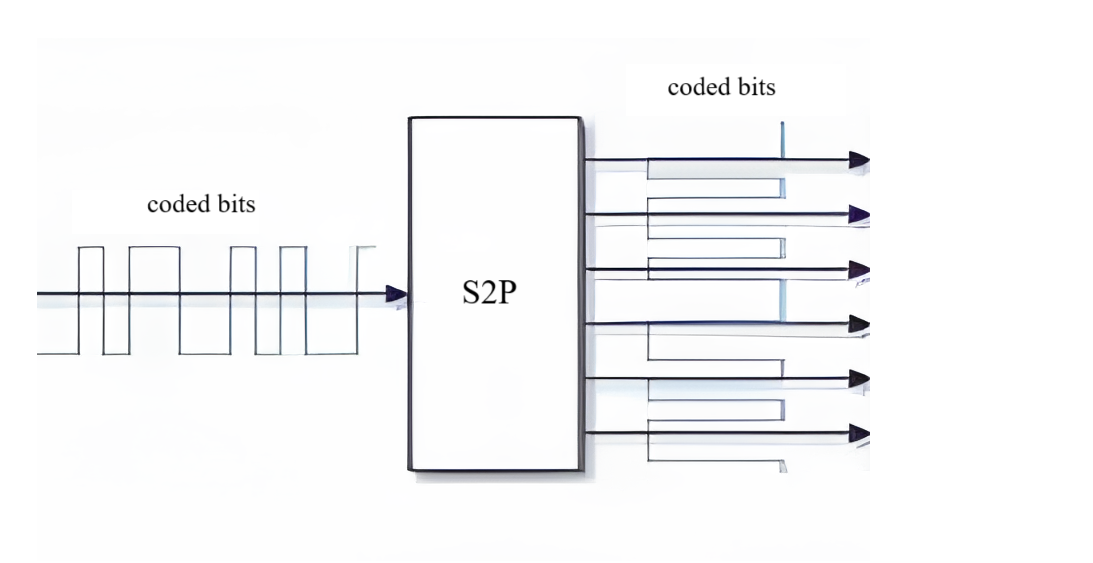
**Задача** ВКРБ заключается в проведении анализа существующих алгоритмов OFDM; реализации алгоритма в одном подканале с использованием процессора STM; реализации алгоритма расчета оптимального соотношения производительности процессоров при организации многопроцессорной версии работы алгоритма. Необходимо заменить аппаратную реализацию программной в части формирования цифрового сигнала передатчика.

# **Постановка задачи**

Проанализировать существующие алгоритмы OFDM как в аппаратной, так и в программной формах; заменить функциональные блоки OFDM-приемопередатчика их программными эквивалентами, заменить аппаратное представление всего приемопередатчика программным в рамках цифровой части обработки. Провести моделирование цифрового программного передатчика с ограниченным количеством параллельных потоков на конкретном микроконтроллере, учитывая требования к производительности, энергопотреблению и занимаемой памяти.

# **Анализ основных блоков алгоритма**

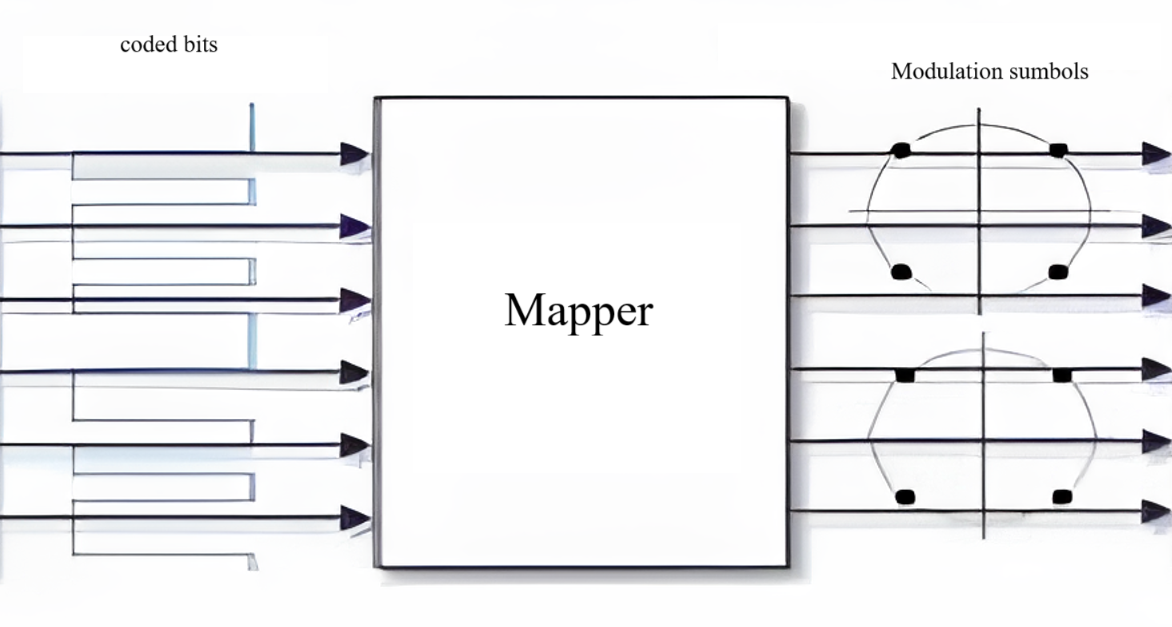
## **S2P**

****

*Рис. 6. Схема блока S2P*

Данный блок разделяет входящий поток бит на группы бит фиксированной длины. Из описания алгоритма OFDM требуется отметить, что длина выходного из блока битового слова зависит от выбранного в блоке Mapper алгоритма. Также, как было ранее описано, важно помнить, что последующая скорость обработки битовых слов будет ниже, чем обработка битового потока, так как иначе блок S2P не будет успевать обрабатывать слова, а последующие блоки будут простаивать, ожидая окончания выполнения работы блока S2P. Это уточнение будет важно при реализации данного алгоритма на микроконтроллере.

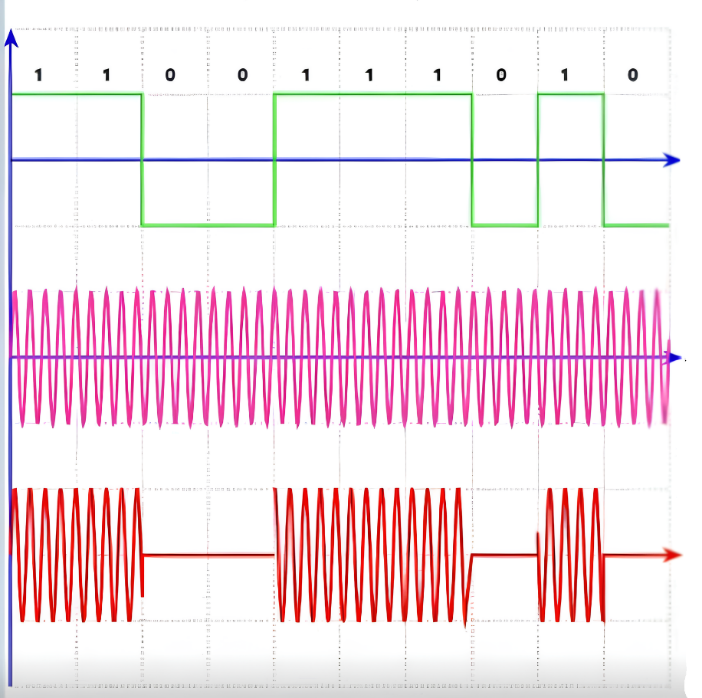
## **Mapper**

****

*Рис. 7. Схема блока Mapper*

Блок Mapper принимает на вход битовые слова и по определенному алгоритму преобразует их в соответствующее комплексное число. Данная операция нужна для реализации передачи слов в виде сигнала.

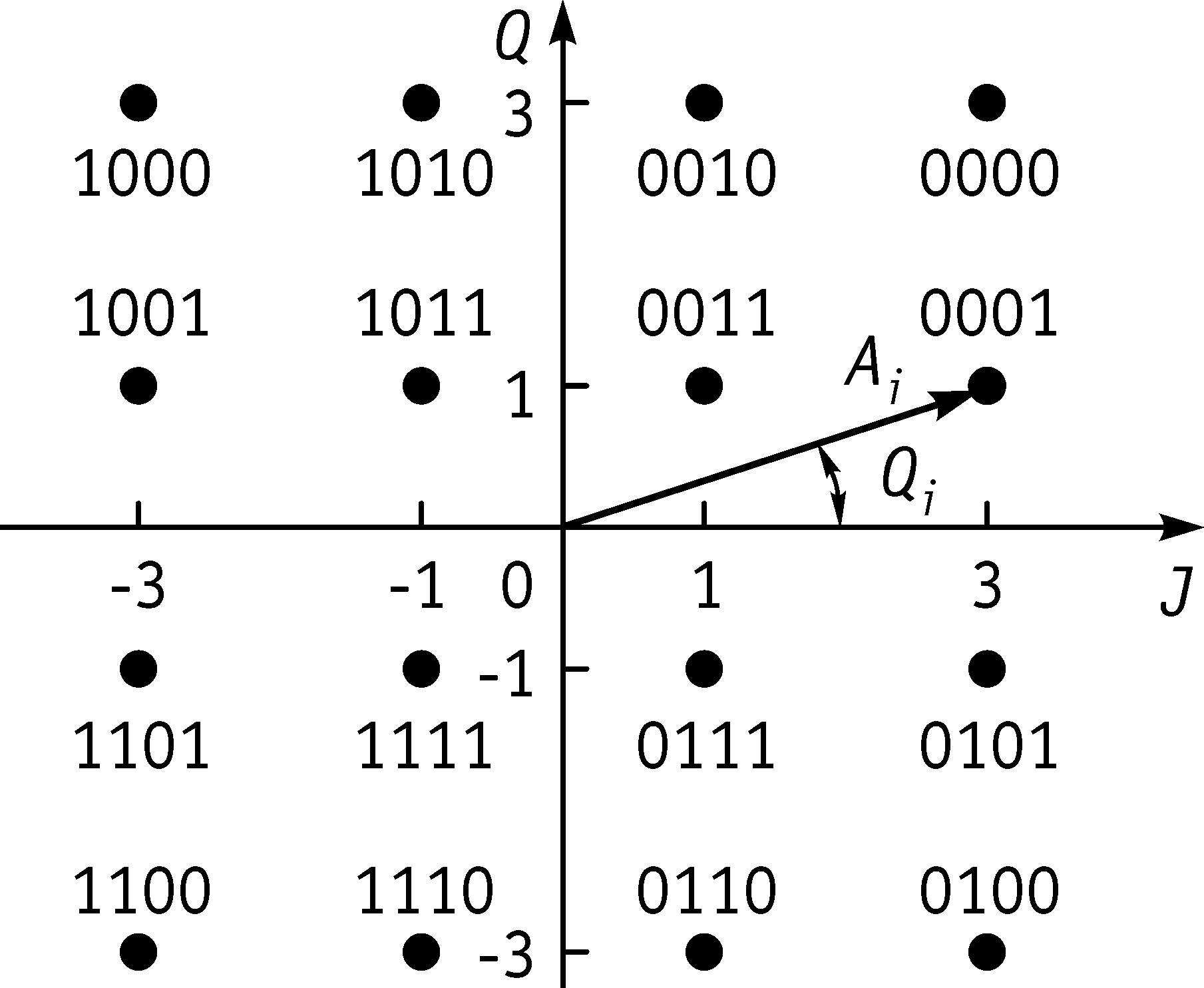
Может возникнуть вопрос, почему бы не передавать сигнал без преобразования сразу в битовом виде, как, например, на рисунке 8.

****

*Рис. 8. Представление потока бит в виде «On-Off»*

Этот способ передачи данных неэффективен, так как если передавать данные в виде «1 – есть сигнал, 0 – нет сигнала», то это будет в разы проигрывать в скорости передачи информации методам кодирования (модуляции), которые реализуются в данном блоке.

Для рассмотрения способов модуляции, рассмотрим комплексную плоскость. На ней имеется ось реальных значений и ось мнимых значений.



*Рис. 9. Представление алгоритма 16-QAM на комплексной плоскости.*

Ось реальных значений определяет амплитуду волны. Ось мнимых значений - фазу волны. На основе этого имеется 3 вида алгоритмов, в основе которых: изменения фазы при фиксированной амплитуде; изменение амплитуды при фиксированной фазе; изменение фазы и амплитуды. Описание данных видов алгоритмов представлено в Приложении 2.

Заметим, что самое эффективное кодирование является кодирование изменением и амплитуды, и фазы, так как использование обоих параметров позволит кодировать больше данных при меньших изменениях одного из параметров. В данных алгоритмах битовые слова сопоставляются определенным точкам на комплексной плоскости. Изображение сопоставления битовых слов определенным точкам на комплексной плоскости называют картой. Пример карты для алгоритма 16-QAM, который и будет реализован, представлен на рисунке 9.

Стоит отметить, что данная карта составлена так, чтобы соседние точки в ней отличались лишь на 1 бит. Данный способ кодирования позволяет уменьшить вероятность ошибки при передаче и облегчает восстановление данных при возможных ошибках при приеме и дешифровки сигнала.

Программная реализация данного блока:

inline void Modulation(unsigned char \* cadrs, fftw\_complex \*symbolCoding)

{

    // Вариант 1: swithc-case.

    for (i = 0; i < COUNT\_CADR; ++i)

    {

        switch (cadrs[i]) {

            case 0b0000: symbolCoding[i][0] = -3.0; symbolCoding[i][1] =  3.0; break; // 0

            case 0b0001: symbolCoding[i][0] = -1.0; symbolCoding[i][1] =  3.0; break; // 1

            case 0b0010: symbolCoding[i][0] = -3.0; symbolCoding[i][1] =  1.0; break; // 2

            case 0b0011: symbolCoding[i][0] = -1.0; symbolCoding[i][1] =  1.0; break; // 3

            case 0b0100: symbolCoding[i][0] = -3.0; symbolCoding[i][1] = -3.0; break; // 4

            case 0b0101: symbolCoding[i][0] = -1.0; symbolCoding[i][1] = -3.0; break; // 5

            case 0b0110: symbolCoding[i][0] = -3.0; symbolCoding[i][1] = -1.0; break; // 6

            case 0b0111: symbolCoding[i][0] = -1.0; symbolCoding[i][1] = -1.0; break; // 7

            case 0b1000: symbolCoding[i][0] =  3.0; symbolCoding[i][1] =  3.0; break; // 8

            case 0b1001: symbolCoding[i][0] =  1.0; symbolCoding[i][1] =  3.0; break; // 9

            case 0b1010: symbolCoding[i][0] =  3.0; symbolCoding[i][1] =  1.0; break; // 10

            case 0b1011: symbolCoding[i][0] =  1.0; symbolCoding[i][1] =  1.0; break; // 11

            case 0b1100: symbolCoding[i][0] =  3.0; symbolCoding[i][1] = -3.0; break; // 12

            case 0b1101: symbolCoding[i][0] =  1.0; symbolCoding[i][1] = -3.0; break; // 13

            case 0b1110: symbolCoding[i][0] =  3.0; symbolCoding[i][1] = -1.0; break; // 14

            case 0b1111: symbolCoding[i][0] =  1.0; symbolCoding[i][1] = -1.0; break; // 15

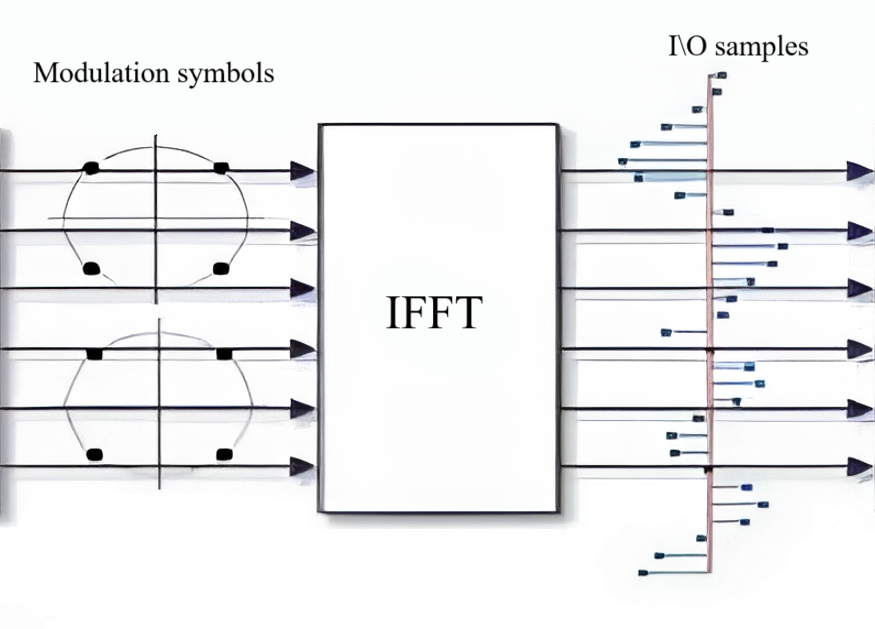
            default:   symbolCoding[i][0] = 0.0; symbolCoding[i][1] = 0.0; // Обработка неожиданной ситуации (опционально)

        }

    }

}

## **IFFT**

****

*Рис. 10. Схема блока IFFT.*

Данный блок преобразовывает комплексное число или комплексные числа, полученные на прошлом этапе, в временной сигнал. Это позволяет сделать обратное преобразование Фурье. Рассмотрим преобразование Фурье. Для дискретного сигнала  из  отсчетов, где , дискретное преобразование Фурье (ДПФ), обозначаемое , определяется как:

для , где

- -ый отсчет входной последовательности (сигнала во временной области);

- -ый отсчет выходной последовательности (спектр сигнала в частотной области);

- длина последовательности;

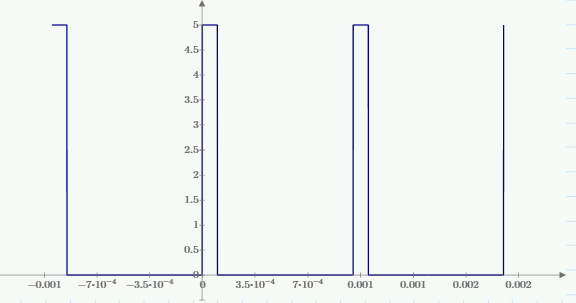
Обратное дискретное преобразование Фурье (ОДПФ) определяется как:

для , где .

Разлагая комплексную экспоненту по формуле Эйлера () получим:

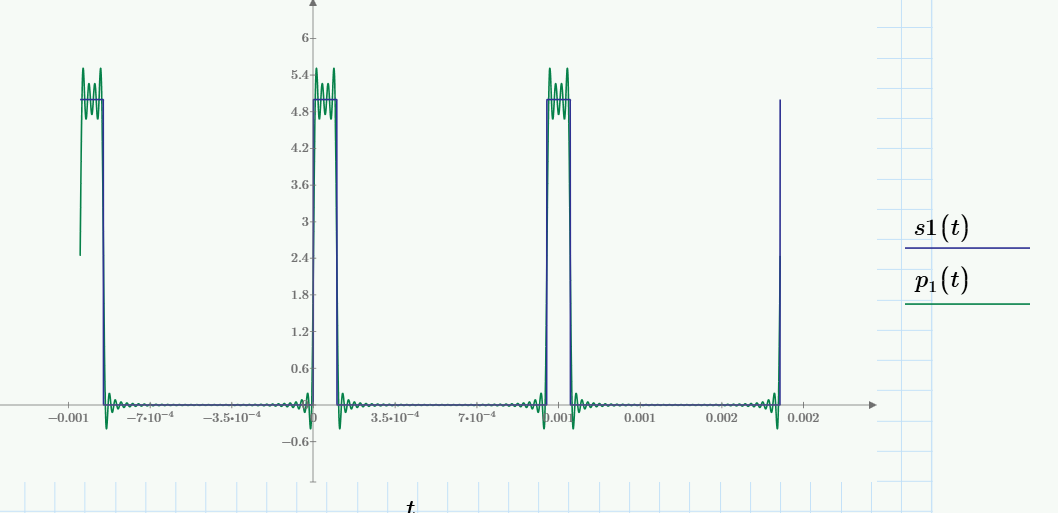
Данную формулу можно разложить на реальную и мнимую составляющие. Они будут равняться соответственно:

Рассмотрим на практическом примере. Имеем следующий сигнал, представленный на рисунке 11, который необходимо передать.



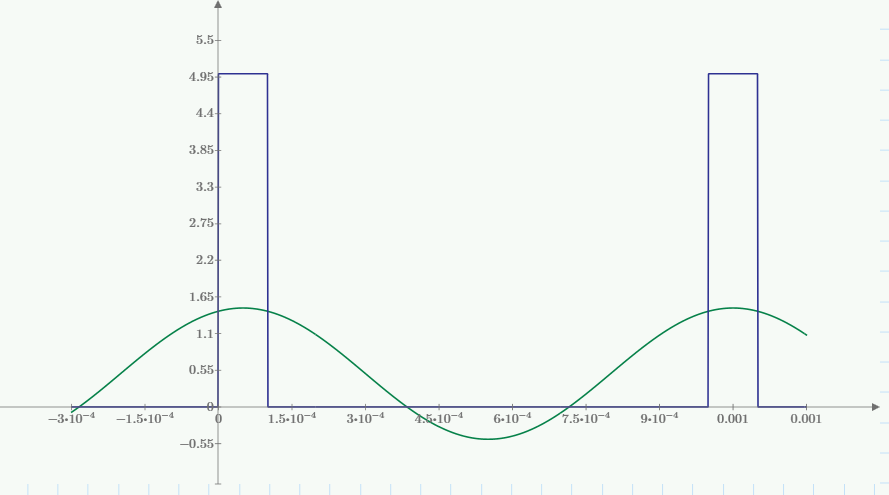
*Рис. 11. Прямоугольный импульс, являющийся входным сигналом.*

При преобразовании данного сигнала с помощью обратного преобразования Фурье (ОПФ) мы получим сигнал, представленный на рисунке 12, по форме напоминающий исходный, но который возможно передать в цифро-аналоговый преобразователь для последующей передачи.



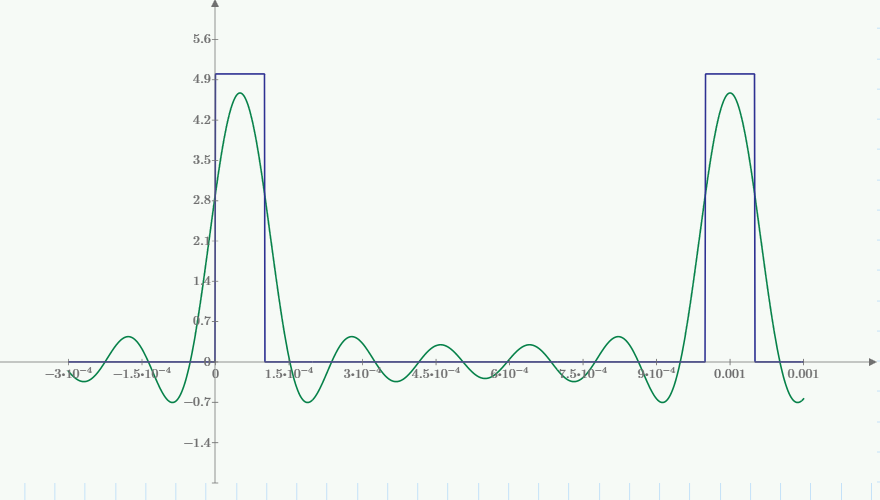
*Рис. 12. Сигнал после ОПФ при использовании 40 гармоник.*

Важным параметром в вычислениях ОПФ является число гармоник, на которые будет разложен входной сигнал. Как можно увидеть по рисунку 12, изначально прямоугольный сигнал разложился на сигнал, являющийся суммой синусоид. Эти синусоиды и есть гармоники сигнала. От их количества зависит то, насколько точно выходной сигнал повторит входной. Преобразование на рисунке 12 использует 40 гармоник. В качестве примера, рассмотрим преобразование того же входного сигнала, но используя одну гармонику. Получим вид, представленные на рисунке 13.



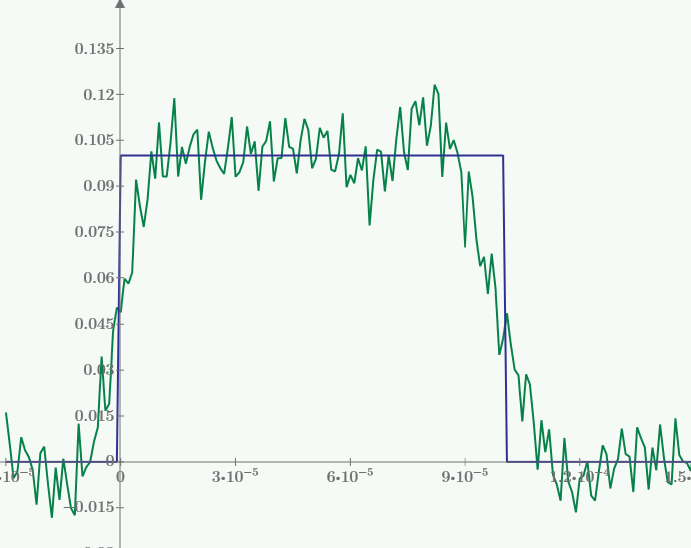
*Рис. 13. Сигнал после ОПФ при использовании 1 гармоники.*

По данному выходному сигналу очень сложно определить изначальный вид. На рисунке 14 представлен результат ОПФ при использовании 4 гармоник.



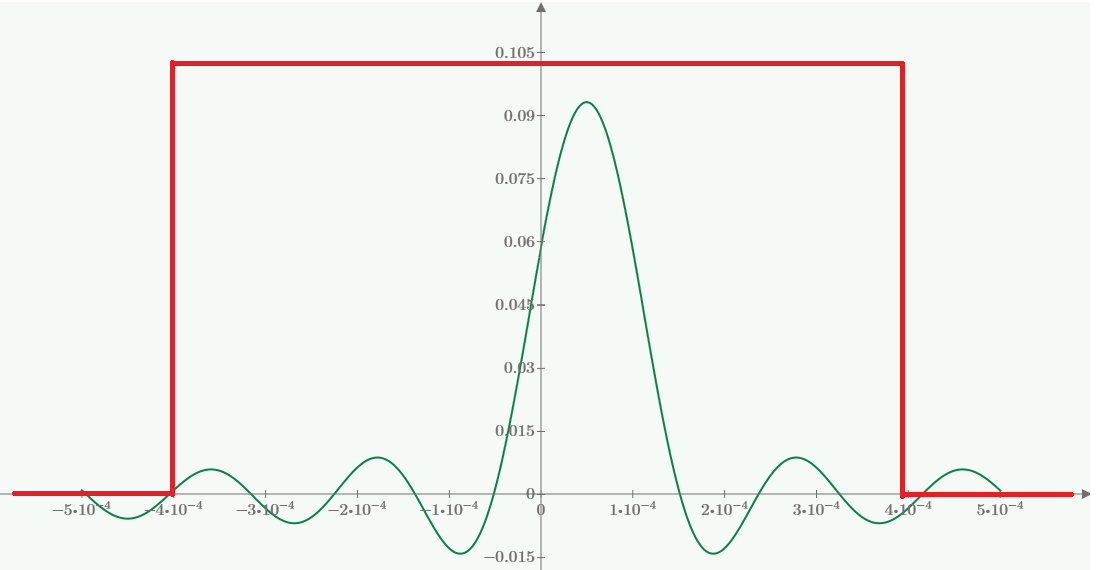
*Рис. 14. Сигнал после ОПФ при использовании 4 гармоник.*

Данный сигнал больше похож на первоначальный, но все еще не так точно его повторяет. Первоначальное разложение на 40 гармоник хорошо раскладывает входной сигнал, однако, при увеличении количества точек вплоть до бесконечности, на которые разбивается сигнал, не удастся идеально восстановить исходных сигнал. График исходного и восстановленного сигнала при разложении на 4000 точек представлен на рисунке 15.



*Рис. 15. Сигнал после ОПФ при использовании 4000 гармоник.*

При дальнейшем увеличении точек будет увеличиваться количество пульсаций, а их амплитуда останется той же. Это связанно с тем, что то, что получается после преобразования, является сверткой частотной характеристики и прямоугольного окна. То есть, когда ограничивается число гармоник, это эквивалентно умножению бесконечной частотной характеристики на прямоугольную функцию, которая равна 1 в пределах выбранного диапазона частот и 0 за его пределами. Это вырезание части спектра и есть применение прямоугольного окна. Наглядный пример на рисунке 16. Также можно использовать другие окна (Ханна, Хэмминга), это поможет побороть пульсации. Свертка – комбинирование двух сигналов.



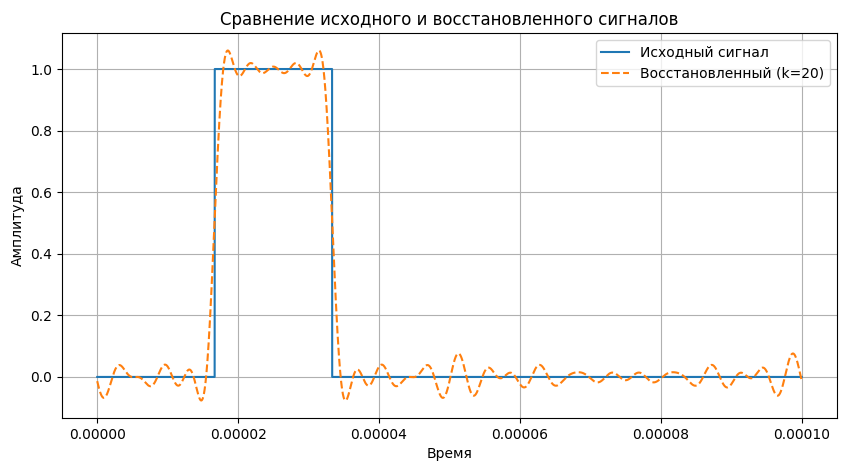
*Рис. 16. Сигнал после ОПФ с использованием прямоугольного окна.*

Учитывая то, что с увеличением числа гармоник сигнал станет менее определяемый, требуется использовать ограниченное число гармоник, но такое, чтобы сигнал однозначно определялся.

Возьмем такт в 20 мкс. За это время данные должны пройти через все модули и выйти из системы, преобразовавшись в единый временной сигнал. Соответственно выходной сигнал будет передавать данные с периодом в 20 мкс. Окно должно быть в 2 раза больше полезного сигнала, поэтому требуется найти число гармоник для сигнала, длительностью 10 мкс, причем такое, чтобы все 16 сигналов алгоритма 16-QAM были однозначно определимы.

Для дискретных сигналов используется теорема Котельникова: непрерывный сигнал с ограниченным спектром можно точно восстановить по его дискретным отсчётам, если они были взяты с частотой дискретизации, превышающей максимальную частоту сигнала минимум в 2 раза. То есть минимальное число гармоник для шифрования одного сигнала – 8. ­­

Попробовав разложить сигнал в ряд Фурье и задав порог дисперсии в 98%, получается, что 20 гармоник при обычном разложении хватает для того, чтобы добиться данной точности. Визуализация представлена на рисунке 17.



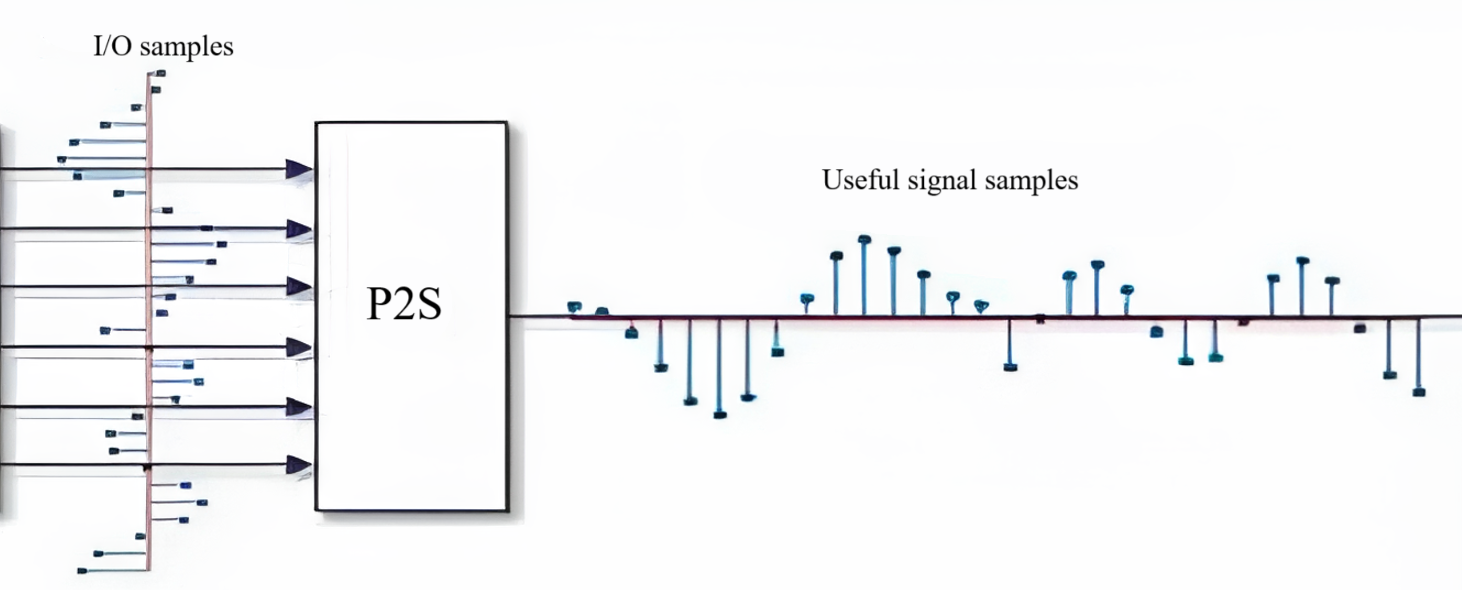
*Рис. 17. Сравнение исходного и восстановленного сигнала с 20 гармониками.*

Из-за конкретизированных преобразований в блоке Mapper на вход данного блока могут поступать 16 конкретных комплексных чисел. Имея фиксированные данные, можно заранее рассчитать все возможные варианты сигнала на выходе блока IFFT. Данное упрощение не повлияет на точность вычислений, но вместо расчета ОПФ можно использовать заранее посчитанные данные, из-за чего множество операций умножения, сложения и возведения в степень заменится операцией присваивания в более меньшем объеме, что в разы увеличит скорость работы блока, хоть и потребует хранить некоторый объем данных.

Для подтверждения корректности работы программы в основную реализацию был добавлен вывод данных на каждом этапе с обратным процессом.

|  |
| --- |
| out data:  0, 1  2, 3  4, 5  6, 7  8, 9  10, 11  12, 13  14, 15  Now data:  1  35  69  103  After S2P:  cadr 0, digit 1  cadr 1, digit 0  cadr 2, digit 3  cadr 3, digit 2  cadr 4, digit 5  cadr 5, digit 4  cadr 6, digit 7  cadr 7, digit 6  After modulation:  cadr 0, Re: -1.000000, Im: 3.000000  cadr 1, Re: -3.000000, Im: 3.000000  cadr 2, Re: -1.000000, Im: 1.000000  cadr 3, Re: -3.000000, Im: 1.000000  cadr 4, Re: -1.000000, Im: -3.000000  cadr 5, Re: -3.000000, Im: -3.000000  cadr 6, Re: -1.000000, Im: -1.000000  cadr 7, Re: -3.000000, Im: -1.000000  After OFFT:  cadr 0, Re: -5.656854, Im: 0.000000  cadr 1, Re: -2.707107, Im: 3.121320  cadr 2, Re: 0.000000, Im: 0.000000  cadr 3, Re: -1.292893, Im: 1.121320  cadr 4, Re: 2.828427, Im: 0.000000  cadr 5, Re: 1.292893, Im: 1.121320  cadr 6, Re: 0.000000, Im: 0.000000  cadr 7, Re: 2.707107, Im: 3.121320  RECOWER...  After FFT:  cadr 0, Re: -1.000000, Im: 3.000000  cadr 1, Re: -3.000000, Im: 3.000000  cadr 2, Re: -1.000000, Im: 1.000000  cadr 3, Re: -3.000000, Im: 1.000000  cadr 4, Re: -1.000000, Im: -3.000000  cadr 5, Re: -3.000000, Im: -3.000000  cadr 6, Re: -1.000000, Im: -1.000000  cadr 7, Re: -3.000000, Im: -1.000000  After demodulation:  cadr 0, digit 1  cadr 1, digit 0  cadr 2, digit 3  cadr 3, digit 2  cadr 4, digit 5  cadr 5, digit 4  cadr 6, digit 7  cadr 7, digit 6  Now data:  137  171  205  239  After S2P:  cadr 0, digit 9  cadr 1, digit 8  cadr 2, digit 11  cadr 3, digit 10  cadr 4, digit 13  cadr 5, digit 12  cadr 6, digit 15  cadr 7, digit 14  After modulation:  cadr 0, Re: 1.000000, Im: 3.000000  cadr 1, Re: 3.000000, Im: 3.000000  cadr 2, Re: 1.000000, Im: 1.000000  cadr 3, Re: 3.000000, Im: 1.000000  cadr 4, Re: 1.000000, Im: -3.000000  cadr 5, Re: 3.000000, Im: -3.000000  cadr 6, Re: 1.000000, Im: -1.000000  cadr 7, Re: 3.000000, Im: -1.000000  After OFFT:  cadr 0, Re: 5.656854, Im: 0.000000  cadr 1, Re: -2.707107, Im: 3.121320  cadr 2, Re: 0.000000, Im: 0.000000  cadr 3, Re: -1.292893, Im: 1.121320  cadr 4, Re: -2.828427, Im: 0.000000  cadr 5, Re: 1.292893, Im: 1.121320  cadr 6, Re: 0.000000, Im: 0.000000  cadr 7, Re: 2.707107, Im: 3.121320  RECOWER...  After FFT:  cadr 0, Re: 1.000000, Im: 3.000000  cadr 1, Re: 3.000000, Im: 3.000000  cadr 2, Re: 1.000000, Im: 1.000000  cadr 3, Re: 3.000000, Im: 1.000000  cadr 4, Re: 1.000000, Im: -3.000000  cadr 5, Re: 3.000000, Im: -3.000000  cadr 6, Re: 1.000000, Im: -1.000000  cadr 7, Re: 3.000000, Im: -1.000000  After demodulation:  cadr 0, digit 9  cadr 1, digit 8  cadr 2, digit 11  cadr 3, digit 10  cadr 4, digit 13  cadr 5, digit 12  cadr 6, digit 15  cadr 7, digit 14 |

## **P2S**

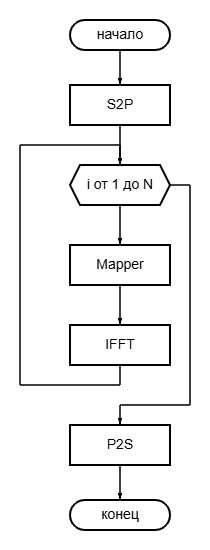
****

*Рис. 18. Схема блока P2S.*

Данный блок получает на вход сигналы, вычисленные в предыдущем блоке, и объединяет их для последующей передачи. По основному свойству OFDM, сигналы располагаются довольно плотно, но при этом так, чтобы основные части сигнала не перекрывали друг друга. Сигналы будут передаваться подряд в функцию отправки данных. Задача данной функции – правильно расположить данные, чтобы функция передачи данных без лишних операций работала с единым набором данных.

# **Оптимизация алгоритма**

Как ранее отмечалось, алгоритм должен выполняться быстро. Рассмотрим блок-схему алгоритма, представленную на рисунке 19.

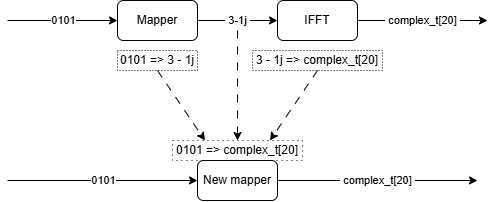


*Рис. 19. Блок-схема алгоритма OFDM.*

Из описания блоков можно сказать, что самым нагруженным местом в алгоритме является IFFT, где происходит обратное преобразование Фурье. Существует множество способов оптимизации, начиная от заранее просчитанных коэффициентов экспонент и заканчивая алгоритмами БПФ или специализированными библиотеками, к примеру, FFTW3 для ускорения вычислений.

Условия реализации данного алгоритма устанавливают, что в сигнале ОПФ будет передаваться только один сигнал из 16 возможных, что все 16 сигналов заранее определены и неизменны и что они передаются с использованием одного и того же числа гармоник (что означает, что размер выходных массивов для каждого сигнала одинаковый). То есть каждый раз при вычислениях в функции IFFT может прийти один из 16 возможных сигналов и в зависимости от этого сигнала будет получен соответствующий этому сигналу массив комплексных чисел, число вариантов которых также 16. Значит, вместо попыток оптимизации самого алгоритма ОПФ можно заранее посчитать 16 возможных вариантов выходных массивов, сохранить их в памяти и присваивать соответствующему сигналу свой массив.

Более того, теперь можно объединить блоки Mapper и IFFT и вместо преобразования набора бит в соответствующее комплексное число, передачи его в блок IFFT и присваивания этому числу соответствующий массив можно сразу присваивать набору бит соответствующий массив. Условная схема оптимизации блоков изображена на рисунке 20.



*Рис. 20. Схема оптимизации IFFT и Mapper*

В итоге за счет хранения 16 массивов по 20 комплексных чисел получилось сократить число операций с: передачи данных в функцию Mapper, нахождение соответствующего сигнала, передачи этого сигнала в функцию IFFT, проведение ОПФ над сигналом с 20-тью гармониками (20\*20 вычислений по формуле ОПФ), передача данных из IFFT; до: передачи данных в функцию New mapper, присваивание сигналу соответствующего ему массива ОПФ, передача данных из New mapper.

# **Проблема синхронизации**

Сложность реализации данного алгоритма на STM32 заключается в проблеме синхронизации. Необходимо обеспечить одновременное выполнение нескольких задач: прием данных из входного битового потока с заданной скоростью передачи данных, передачу обработанных данных и выполнение основной вычислительной функции с определенной частотой. Основная трудность заключается в имитации многопоточности на одноядерном микроконтроллере, где отсутствует встроенный планировщик задач, характерный для многопроцессорных систем. Хотя эмуляция многопоточности на компьютере с мощным процессором и развитыми технологиями распараллеливания процессов не представляет особых проблем, для STM32 требуются альтернативные подходы.

Существует несколько решений для организации параллельной работы на STM32.

RTOS (Операционная система реального времени): RTOS представляет собой специализированное программное обеспечение, предназначенное для управления ресурсами вычислительной системы в условиях жестких временных ограничений. Она обеспечивает детальное управление ресурсами процессора, предоставляет абстракции для упрощения реализации многопоточности, включая сложный, но эффективный встроенный планировщик задач, и обеспечивает развитые средства синхронизации потоков (мьютексы, семафоры, очереди сообщений). В основе RTOS лежит компактное ядро, которое отвечает за выполнение функций планирования (выбор задачи для выполнения в данный момент времени), управления памятью (выделение и освобождение памяти для задач) и обработки прерываний. Архитектура RTOS позволяет разработчикам создавать сложные системы, состоящие из множества параллельно выполняющихся задач, каждая из которых имеет свой приоритет и свое время, к которому задача должна быть выполнена. При выборе RTOS следует тщательно учитывать факторы, такие как размер ядра (влияет на потребление памяти), потребление ресурсов (влияет на производительность системы), доступность библиотек (облегчает разработку) и качество инструментальных средств разработки (упрощает отладку и тестирование). Однако, использование RTOS приводит к дополнительному потреблению ресурсов на переключение между задачами, которое, хоть и оптимизировано с использованием аппаратных возможностей, все равно требует процессорного времени. Примерами популярных RTOS для STM32 являются FreeRTOS (бесплатная и широко используемая), Micrium µC/OS (коммерческая, с расширенной функциональностью) и Zephyr (с открытым исходным кодом, ориентирована на безопасность и надежность). RTOS представляет собой удобный инструмент для разработки сложных многопоточных приложений. Но в данной задаче, где требуется максимальная производительность, а логика программы предсказуема и не предполагает значительных изменений, использование RTOS может оказаться избыточным, так как это может значительно снизить производительность системы, что критично для данной задачи.

В контексте данной задачи предлагается сделать собственную эмуляция многопоточности на основе системы прерываний. В рамках данного подхода используются прерывания для контроля. Прерывание – это аппаратный механизм, позволяющий микроконтроллеру временно приостановить выполнение текущей основной программы и переключиться на выполнение специальной подпрограммы, называемой обработчиком прерывания. Этот механизм позволяет микроконтроллеру реагировать на внешние события, например, поступление данных по последовательному порту, нажатие кнопки, или внутренние события, например, срабатывание таймера, завершение операции ввода-вывода, без необходимости постоянно опрашивать соответствующие флаги или регистры. Когда происходит прерывание, микроконтроллер сохраняет текущее состояние программы (значение регистров, адрес возврата) в стеке, а затем переходит к выполнению обработчика прерывания. После завершения работы обработчика прерывания микроконтроллер восстанавливает сохраненное состояние и продолжает выполнение основной программы с того места, где она была приостановлена. Прерывания бывают аппаратные (вызываемые внешними устройствами, например, датчиками или периферийными устройствами), программные (вызываемые командами в программе, используются для переключения между задачами или вызова системных функций) и системные (вызываемые ядром микроконтроллера, используются для обработки ошибок или исключительных ситуаций).

Всего в системе используется три прерывания.

* Инициализация: активирует цикл работы программы. Запускает работу всех функций. Подача сигнала на выбранный канал как инициализация начала передачи;
* Отправка данных: отправляет на передающий модуль данные, полученные на предыдущем цикле работы вычислительной функции. Частота определяется необходимой скоростью передачи данных;
* Прием битового потока: отвечает за прием входящего бита и его запись в память для последующей обработки. Частота этого прерывания зависит от скорости входящего битового потока. Когда данные сформированы, передает их в переменную в вычислительную функцию и выставляет флаг принятых данных в значение 1.

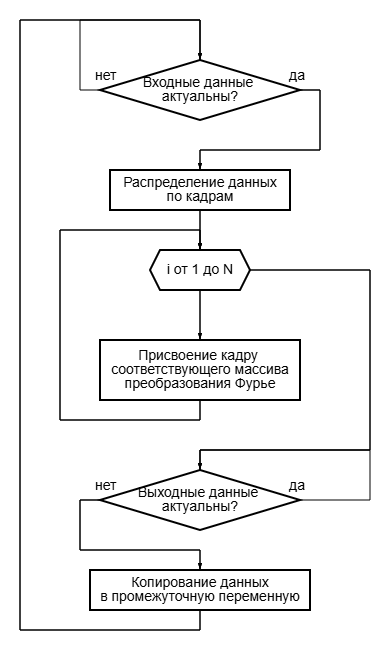


Рисунок 21. Блок-схема основной вычислительной функции

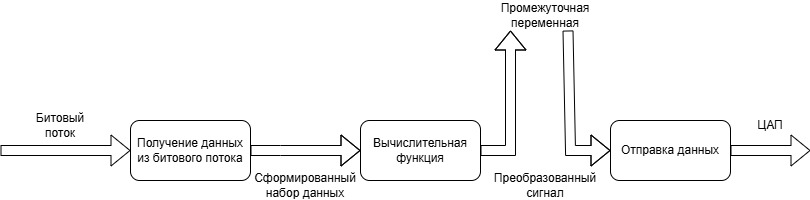


Рисунок 22. Схема взаимосвязи блоков.

Алгоритм работы системы представлен на рисунке 22. Данные в систему входят в виде битового потока с определенной частотой. Функция получения данных из битового потока записывает данные в определенную переменную (или массив, в зависимости от числа каналов). Далее после того, как необходимое число бит считано и записано, данные передаются в вычислительную функцию и выставляется флаг актуальности данных для данной переменной. Такой подход возможен, так как в вычислительной функции входящие данные на первом же этапе преобразуются в необходимый набор бит в отдельных переменных.

Далее работает вычислительная функция. По включенному флагу функция начинает выполнение алгоритма. Блок-схема итогового алгоритма представлена на рисунке 21. В нем сначала обнуляется флаг актуальности входных данных, полученный битовый поток раскладывается на N переменных. Далее каждой переменной во время модуляции сразу присваиваются необходимые гармоники, которые рассчитываются перед началом работы цикла. После этого эти гармоники объединяются в единый массив. При выключенном флаге актуальности выходных данных готовые данные копируются во временную переменную.

Функция отправки данных переносит данные с определенной частотой в ЦАП для дальнейшей передачи. После передачи, если это была последняя переменная, данные копируются из временной в основную переменную этой функции и выставляется флаг неактуальности выходных данных. Её тактирование жестко задается, так как данная функция определяет синхронизацию сигнала с приемником и его стабильность. Частота работы данной функции также зависит от числа гармоник, числа каналов, выбранной модуляции и входной частоты. Если взять N каналов в системе, P гармоник, модуляцию 16-QAM и скорость входного потока – K Гц, то частота работы данной функции должна быть . По данной формуле рассчитывается необходимая для данного набора скорость, так как при уменьшении скорости не все данные будут успевать доходить, а при увеличении данные не будут успевать обрабатываться входной функцией.

## **Приоритетность**

Для установки приоритетов прерываний углубимся в работу функций. На STM32 нельзя задавать сдвиг тактирования, то есть нельзя сделать так, чтобы прерывания одной и той же частоты вызывались в разное время. Визуализация представлена на рисунке23. Эту проблему можно решить расстановкой правильных приоритеты для каждого прерывания. Приоритеты прерывания определяют, какое прерывание может быть прервано другим прерыванием.

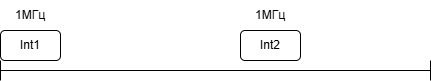


Рисунок 23. Визуализация невозможной ситуации тактирования.

Рассмотрим функциональные блоки. Ранее было сказано, что вычислительная функция работает не как прерывание, а в качестве основной функции работы программы. Функция начала вычислений также не будет рассматриваться, так как это механизм инициализации вычислений, который не входит в основной цикл работы программы.

Рассмотрим отправку данных в ЦАП. За счет отправки данных из данного блока обеспечивается основная синхронизация между приемником и передатчиком. Из этого можно сказать, что данное прерывание должно иметь наивысший приоритет и минимум действий.

Рассмотрим получение данных из битового потока. Для этого проанализируем схему входа битового потока, представленную на рисунке 24. По ней видно, что данные можно фиксировать не в конкретной точке, а на протяжении всего отрезка, пока данные не изменились. Можно сделать вывод, что данное прерывание должно иметь приоритет ниже, чем прерывание отправки данных в ЦАП.

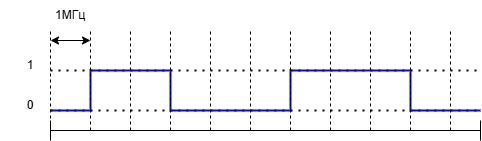


Рисунок24. Визуализация битового потока со скоростью передачи данных 1МГц.

Применив данные правила приоритетности и тактирования, получим схему распределения задач процессора, представленную на рисунке 25. Здесь представлено то, как процессор будет обрабатывать задачи за четверть цикла основного алгоритма, то есть за то время, в течении которого входящий бит из потока не будет меняться. В начальной точке представленной схемы вызывается 2 прерывания, OUT и IN. Из-за приоритетности, управление будет передано прерыванию OUT, после чего будет работать прерывание IN. Далее в оставшееся время работы цикла будет работать основная вычислительная функция main, которая будет прерываться OUT.

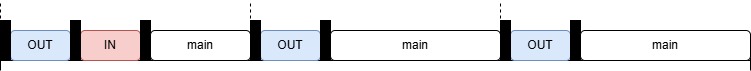


Рисунок 25. Схема распределения задач на процессоре.

Рассмотрим иную ситуацию, когда частота процессора не слишком высокая и схема распределения задач будет выглядеть так, как представлено на рисунке 26. Так как функция main занимается основной вычислительной частью алгоритма, может показаться, что при данной схеме рабочий цикл точно не успеет пройти за необходимую частоту. Но так как основной алгоритм был преобразован и упрощен, что показано на рисунке 21, а на схеме рассматривается четвертая часть вычислительного цикла, нельзя быть уверенным в том, что вычислительной функции не хватит времени. Для полного подтверждения требуется провести тесты производительности на разных микроконтроллерах с разной частотой тактирования.

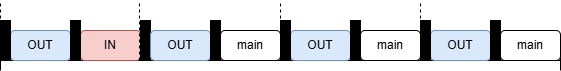


Рисунок 26. Схема распределения задач на процессоре с меньшей частотой.

# **Проверка алгоритма на микроконтроллерах**

Для тестирования алгоритма был выбран микроконтроллер на базе ядра Cortex-M3: STM32F103RB.

Данное ядро является одним из самых популярных в использовании как в учебных или личных целей, так и в различных повседневных устройствах. Оно построено на 32-битной архитектуре, имеет набор инструкций Thumb-2 (совмещает в себе 16-битные инструкции для компактности кода и 32-битные инструкции для повышения производительности, обеспечивая баланс между размером кода и вычислительной мощностью), энергоэффективно, оптимизировано для работы с ограниченными ресурсами.

Микроконтроллер STM32F103RB имеет следующие характеристики:

* Максимальная частота работы ядра до 72 МГц;
* Объем Flash-памяти и SRAM - 128 КБ, 20 КБ.

Проект конфигурировался с использованием STM32CubeMX. На обоих микроконтроллерах был подключен USART-порт для получения данных, 2 модуля таймера для тактирования прерываний. При тестировании менялись коэффициенты тактирования так, чтобы удовлетворять следующему условию: , где – частота прерывания, получающего данные, – частота прерывания, отправляющее данные. Частота работы ядра была установлена 64 МГц (частота 72 МГц недоступна к выставлению в самой программе), тактирование осуществлялось от внутренней RC цепи.

После конфигурации и полной отладки, в режиме сборки Release и с выставлением дополнительных флагов компиляции (-O2) программа была запущена и протестирована. Тестирование проходило следующим образом: заранее составлена таблица возможных таблиц, удовлетворяющих условию выше; в прерывания была добавлена проверка, которая активируется при запросе новых данных; если данные нельзя запросить – поток выводит об этом информацию в последовательный порт, иначе просто получает данные и продолжает работать.

|  |  |  |
| --- | --- | --- |
| In(KHz) | OUT(KHz) | Прохождение теста |
| 1 | 5 | + |
| 2 | 10 | + |
| 4 | 20 | + |
| 5 | 25 | + |
| 8 | 40 | + |
| 10 | 50 | + |
| 16 | 80 | + |
| 20 | 100 | + |
| 25 | 125 | + |
| 32 | 160 | + |
| 40 | 200 | + |
| 50 | 250 | + |
| 64 | 320 | + |
| 80 | 400 | - |
| 100 | 500 | - |

Из результатов можно увидеть, что на данных настройках максимальные возможности данного микроконтроллера – 64 КГц на прерывание получения одного бита, то есть скорость получения и обработки данных на данном микроконтроллере - 64000 бит\с. Получаем, что если один микроконтроллер STM32F103RB будет заниматься приемом, передачей и обработкой данных, то он сможет принимать и отправлять данные со скоростью 64 000 бит\с.

Также было проверено время работы только вычислительной функции. Тесты показали, что функция без прерываний и ожиданий данных справляется с данной задачей за 2 микросекунды.

# **Заключение**

В рамках данной работы был проведен анализ и реализована оптимизация алгоритма OFDM. Экспериментальным путем было выяснено, что микроконтроллер STM32F103RB на базе ядра Cortex-M3 может получать, обрабатывать и передавать данные со скоростью 64000 бит\с. Следующим шагом данной работы является использование полученных данных и алгоритмов для реализации каждой задачи на отдельном микроконтроллере. Данная реализация позволит в разы увеличить скорость работы всей системы.

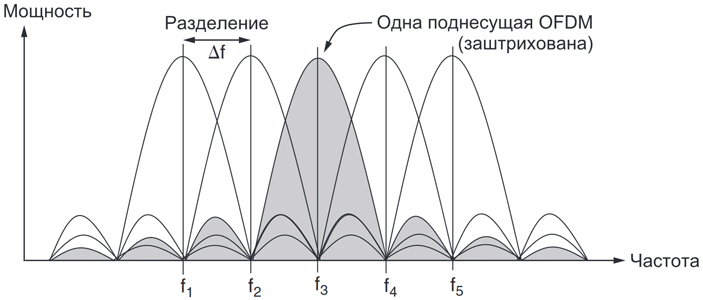
# **Приложение 1**

## **Принцип работы приемопередатчика на основе OFDM**

Прежде чем перейти к детальному рассмотрению алгоритмов формирования и обработки сигналов OFDM на микропроцессорной платформе, необходимо разобрать теоретические основы данной технологии модуляции.

Ортогональное частотное мультиплексирование (OFDM) - это метод цифровой модуляции, который делит широкополосный канал связи на множество узкополосных, ортогональных поднесущих. Вместо передачи одного высокоскоростного потока данных по всему каналу, OFDM передает несколько низкоскоростных потоков данных параллельно по этим поднесущим.

В основе OFDM лежит принцип разделения полосы пропускания на множество узкополосных поднесущих, каждая из которых модулируется независимо. Эти поднесущие выбираются таким образом, чтобы они были ортогональны друг другу. Ортогональность означает, что в точке максимальной амплитуды одной поднесущей амплитуда всех остальных поднесущих равна нулю. Это позволяет поднесущим перекрываться по частоте, не создавая взаимных помех.



Основной принцип работы OFDM заключается в следующем:

1. Высокоскоростной поток данных делится на несколько параллельных низкоскоростных потоков;
2. Каждый низкоскоростной поток данных модулирует свою поднесущую. Поднесущие выбираются таким образом, чтобы они были ортогональными друг другу;
3. Все модулированные поднесущие суммируются вместе с использованием обратного быстрого преобразования Фурье (IFFT). Это создает OFDM-сигнал во временной области;
4. К каждому OFDM-символу добавляется циклический префикс, который представляет собой копию части OFDM-символа, добавленную в начало. CP помогает бороться с межсимвольной интерференцией и межподнесущей интерференцией, вызванными многолучевым распространением.

На приемной стороне выполняются обратные операции: удаление CP, быстрое преобразование Фурье (FFT), демодуляция поднесущих и объединение низкоскоростных потоков данных в высокоскоростной поток.

Преимущества и недостатки OFDM:

Преимущества:

* OFDM эффективно борется с многолучевым распространением, так как узкополосные поднесущие менее подвержены частотно-избирательному замиранию (явление, при котором разные частотные компоненты сигнала затухают по-разному из-за многолучевого распространения);
* Так как поднесущие ортогональны друг другу, данные плотно упаковываются в частотный диапазон;
* Простота реализации с использованием БПФ/ОБПФ;
* OFDM позволяет адаптировать параметры передачи (модуляцию, кодирование) к условиям канала для каждой поднесущей.

Недостатки:

* Небольшие отклонения частоты могут привести к потере ортогональности между поднесущими и ухудшению производительности системы.
* OFDM-сигнал может иметь высокий пик-фактор, что требует использования линейных усилителей мощности.
* Фазовый шум может привести к межподнесущей интерференции (ICI).

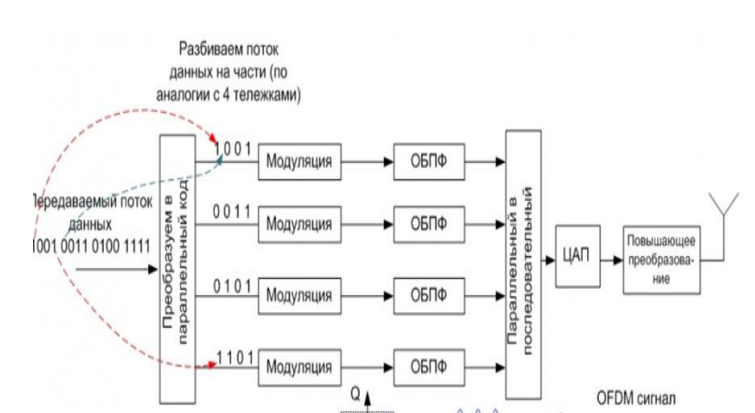
Пример:

Представим, что существует полоса частот шириной 1 МГц, которую необходимо использовать для передачи данных. Вместо того чтобы использовать один широкий канал, эта полоса разделяется, допустим, на 128 узких каналов (поднесущих) шириной примерно 7.8 кГц каждый. Каждая из этих поднесущих модулируется независимо с использованием определенного алгоритма модуляции.

Таким образом, вместо передачи одного символа с высокой скоростью передачи данных, передается 128 символов с более низкой скоростью передачи данных на каждой поднесущей. Это снижает влияние многолучевого распространения, так как узкополосные поднесущие менее подвержены частотно-избирательному замиранию (явление, при котором разные частотные компоненты сигнала затухают по-разному из-за многолучевого распространения).

В итоге, суммарная скорость передачи данных остается прежней, но система становится более устойчивой к помехам и многолучевому распространению.

Пример строения блоков:



# **Приложение 2.**

### **Модуляция**

Модуляция поднесущих - это процесс изменения одного или нескольких параметров несущего сигнала (амплитуды, частоты, фазы) в соответствии с информацией, которую необходимо передать. В OFDM модуляция выполняется на каждой поднесущей независимо, что позволяет передавать несколько параллельных потоков данных. Выбор метода модуляции влияет на скорость передачи данных, устойчивость к шуму и сложность системы.

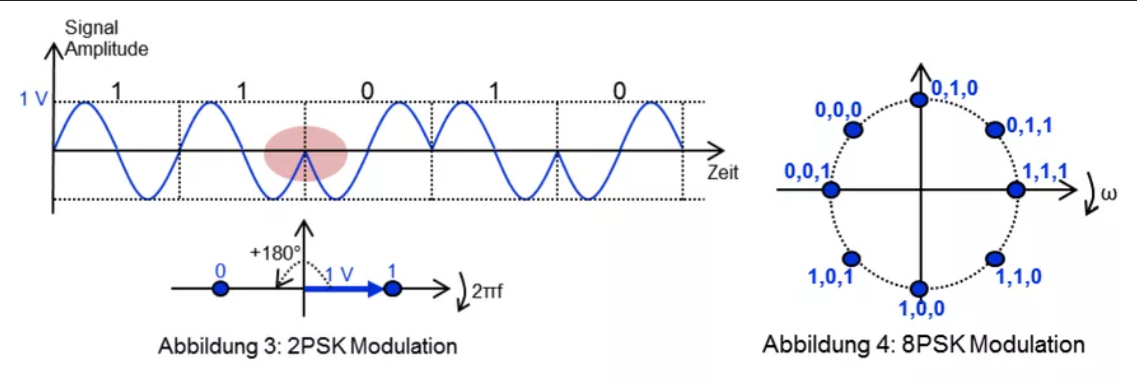
Рассмотрим наиболее распространенные методы модуляции, используемые в OFDM:

**Фазовая модуляция (Phase-Shift Keying - PSK)**

Данные кодируются путем изменения фазы несущей частоты. Амплитуда остается постоянной.

Типы PSK:

* BPSK (Binary PSK): Использует две фазы (0 и 180 градусов) для представления битов 0 и 1.
* QPSK (Quadrature PSK): Использует четыре фазы (0, 90, 180 и 270 градусов) для представления двух битов информации на символ.
* 8-PSK: Использует восемь фаз для представления трех битов на символ.
* 16-PSK: Использует шестнадцать фаз для представления четырех битов на символ.



Пример в виде алгоритма QPSK:

* Входящий поток битов делится на пары.
* Каждой паре битов присваивается определенная фаза:

00 -> 45°

01 -> 135°

10 -> 225°

11 -> 315°

* Сигнал генерируется как: , где - амплитуда, - частота поднесущей, - фаза, соответствующая дибиту.

Преимущества: Простота реализации, постоянная огибающая (что упрощает работу усилителей мощности).

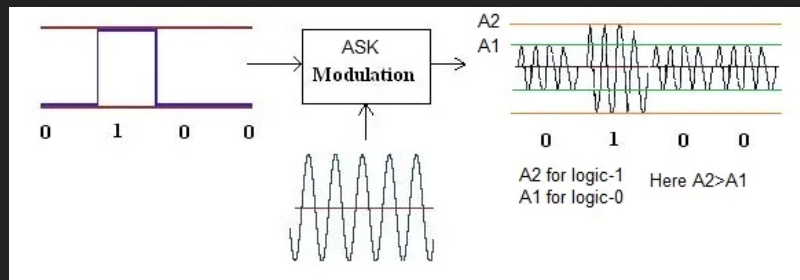
Недостатки: Ограниченная спектральная эффективность (малое количество битов на символ), чувствительность к фазовому шуму.

Вычислительные ресурсы: Низкие. Требуются простые операции для генерации синусоидальных сигналов с определенной фазой.

**Амплитудная модуляция (Amplitude-Shift Keying - ASK)**

Данные кодируются путем изменения амплитуды несущей частоты. Фаза остается постоянной.

* OOK (On-Off Keying): Простейший тип ASK, где наличие сигнала представляет бит 1, а отсутствие сигнала - бит 0.



Пример алгоритма:

Бит 1: (сигнал есть)

Бит 0: s(t) = 0 (сигнала нет)

Преимущества: Очень простая реализация.

Недостатки: Низкая спектральная эффективность, очень чувствительна к шуму и изменениям уровня сигнала. Практически не используется в OFDM.

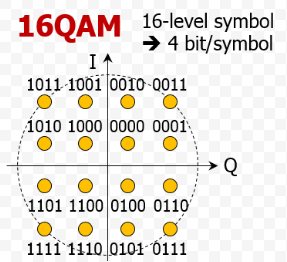
Вычислительные ресурсы: Очень низкие.

**Квадратурная амплитудная модуляция (Quadrature Amplitude Modulation - QAM)**

Данные кодируются путем изменения как амплитуды, так и фазы несущей частоты.

Типы QAM:

* 16-QAM: Использует 16 комбинаций амплитуды и фазы для представления четырех битов на символ.
* 64-QAM: Использует 64 комбинации амплитуды и фазы для представления шести битов на символ.
* 256-QAM: Использует 256 комбинаций амплитуды и фазы для представления восьми битов на символ.



Пример алгоритма 16-QAM:

1. Входящий поток битов делится на группы по 4 бита.
2. Каждой группе битов присваивается определенная амплитуда и фаза в соответствии с картой созвездия.
3. Сигнал генерируется как сумма двух квадратурных компонент: где *I* и - синфазная и квадратурная компоненты, определяемые амплитудой и фазой.

Преимущества: Высокая спектральная эффективность (большое количество битов на символ).

Недостатки: Более сложная реализация, чувствительность к шуму и искажениям, требует линейных усилителей мощности.

Вычислительные ресурсы: Средние. Требуются более сложные операции для генерации сигналов с переменной амплитудой и фазой.

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Биты | Устойчивость к шуму | Вычислительные ресурсы |
| BPSK | 1 | Высокая | Низкие |
| QPSK | 2 | Средняя | Низкие |
| 16-QAM | 4 | Средняя | Средние |
| 64-QAM | 6 | Низкая | Средние |
| 256-QAM | 8 | Очень низкая | Высокие |

Таблица 1. Сравнение методов модуляции

Выбор метода модуляции для каждой поднесущей в OFDM зависит от:

* Состояния канала: Если канал характеризуется высоким SNR, можно использовать более сложные схемы модуляции (например, 64-QAM) для увеличения скорости передачи данных. Если канал зашумлен, следует использовать более надежные схемы (например, QPSK или BPSK).
* Требований к скорости передачи данных: Если требуется высокая скорость передачи данных, используются схемы модуляции с большим количеством битов на символ.
* Требований к надежности: Если требуется высокая надежность передачи данных, используются схемы модуляции с меньшим количеством битов на символ и канальным кодированием.