Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Телекоммуникационные технологии

Отчёт по лабораторной работе № 12

Работу выполнил:
Смирнов Л. Д.
Группа:
3530901/80202
Преподаватель:
Богач Н. В.

Санкт-Петербург
2021

В данной лабораторной работе будут обсуждаться вопросы, связанные с передачей и приёмом сигналов с реальными аппаратными и канальными эффектами. Сначала будет пошагово проведена PSK модуляция, а затем демодуляция сигнала.

1. Передача сигнала

Первым этапов является передача QPSK сигнала. Нам нужно сгенерировать поток битов и смоделировать его в созвездие. Для этого используем блок Constellation Modulator, который использует объект Constellation.

Объект Constellation позволяет нам определить, как символы кодируются. Constellation Modulator ожидает упакованные байты, поэтому у нас есть генератор случайных источников, предоставляющий байты со значениями 0 - 255.

Мы хотим, чтобы количество выборок на символ было как можно меньше (минимальное значение - 2). Здесь мы будем использовать 4, что больше, чем нам нужно, но полезно для визуализации сигнала в разных областях.

Теперь мы устанавливаем значение избыточной пропускной способности. Модулятор созвездия использует RRC (root raised cosine) фильтр, который дает нам один параметр для настройки коэффициента спада фильтра. Следующий рисунок получается из примера mpsk_rrc_rolloff.grc, показывающего различные значения избыточной полосы пропускания. Типичные значения находятся в диапазоне от 0,2 до 0,35:

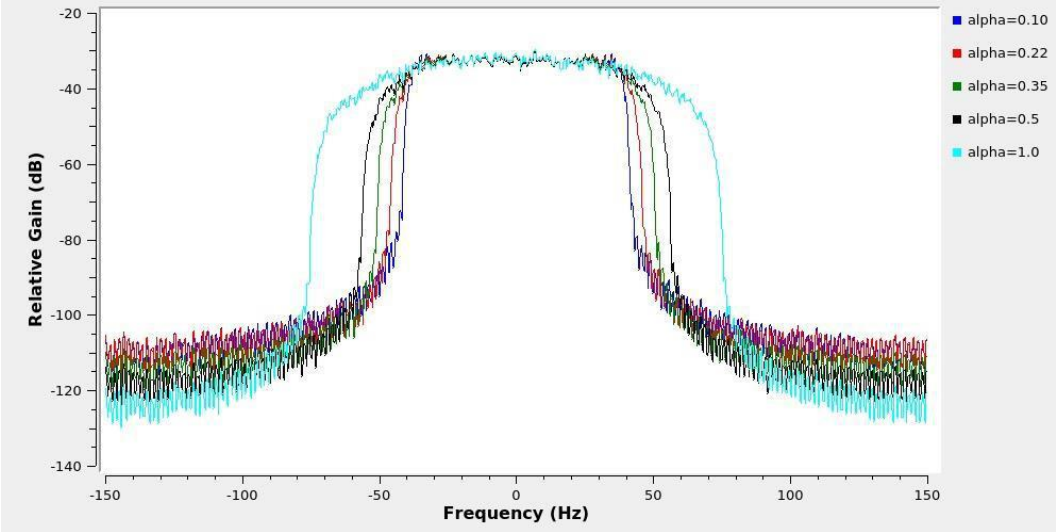

Рис. 1.1

mpsk_stage1.grc отображает как передаваемый сигнал, так и часть цепочки приемника во времени и частоте, а также график созвездия:

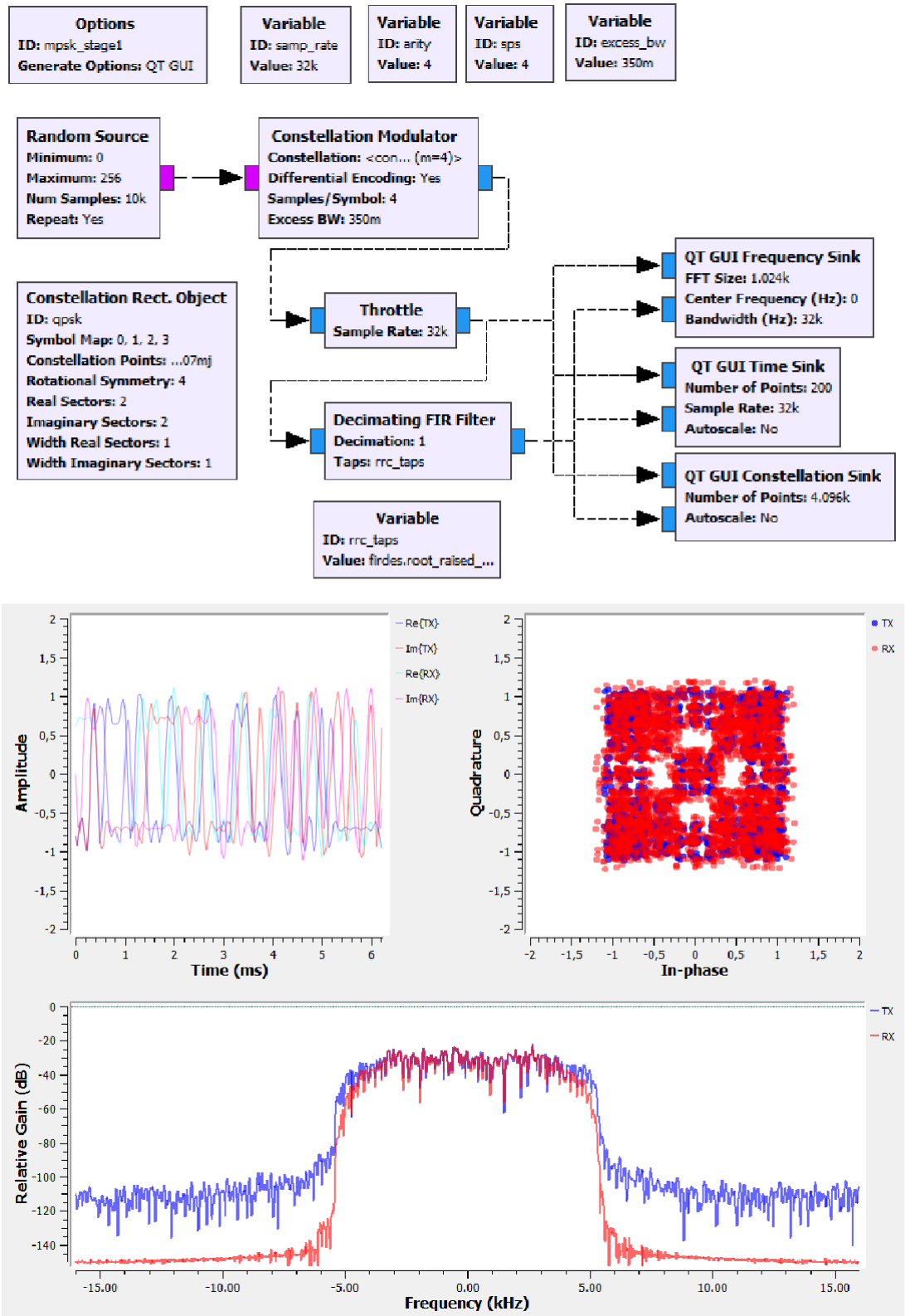

Рис. 1.2.

На графике созвездия видны эффекты повышения частоты дискретизации и фильтрации. Фильтр RRC добавляет преднамеренные собственные помехи, известные как межсимвольные помехи (ISI).

Частотный график показывает сигнал с хорошей формой, который скатывается в шум. Если бы мы не установили формирующий фильтр на сигнал, мы бы передавали прямоугольные волны, которые производят много энергии в соседних каналах. Благодаря уменьшению внеполосных излучений наш сигнал теперь остается в пределах полосы пропускания нашего канала.

С этим сигналом есть небольшая проблема, которая лучше всего видна на графике во временной области. Отключим всё на графике, кроме реальной части передаваемого сигнала (Re {TX}). Затем, щелкнув среднюю кнопку мыши, увеличим количество точек с 200 до 5000. Мы видим, что точки расположены относительно беспорядочно, хотя мы должны были передавать 1 и 0:

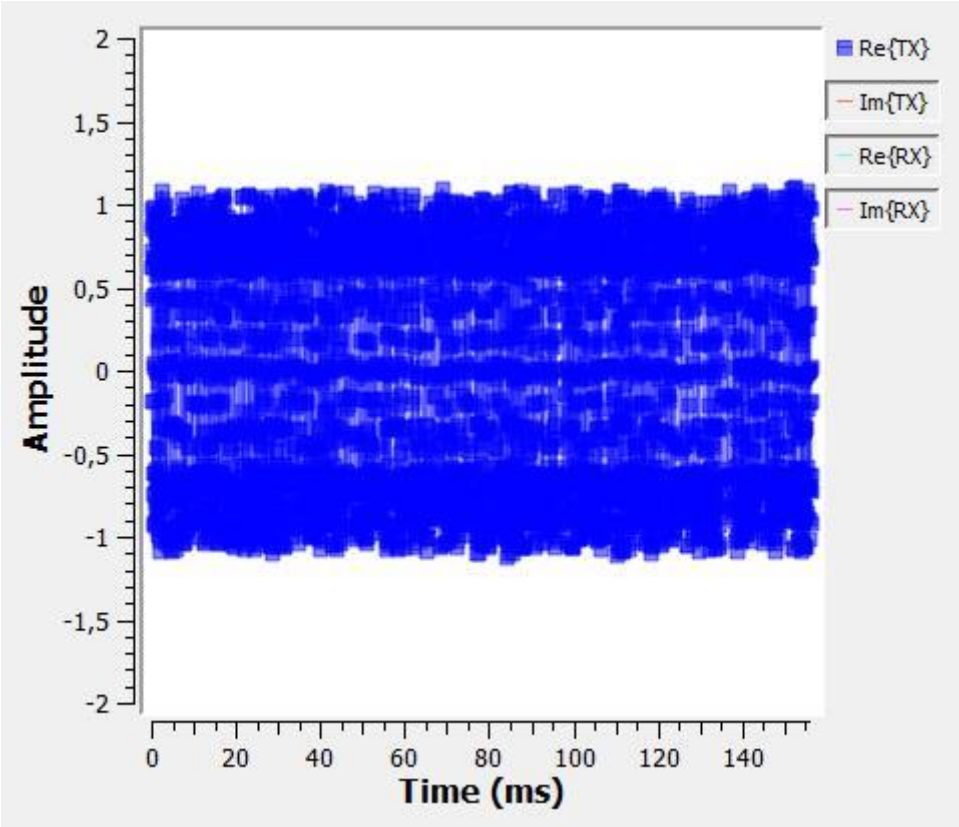

Рис. 1.3.

На данном изображении мы видим ISI, о котором упоминали ранее. Мы избавляемся от ISI, используя другой фильтр на приемнике. По сути, мы целенаправленно использовали фильтр на передатчике, создающий ISI, но когда мы сворачиваем два фильтра RRC вместе, мы получаем RC (raised cosine) фильтр, который является формой фильтра Найквиста. Зная это свойство фильтра RRC, мы можем использовать другой фильтр RRC на приемнике. Выходной сигнал RRC-фильтра на принимающей стороне представляет собой сигнал с минимальным ISI.

Теперь стоит разобраться, как же выглядит сигнал после фильтрации на приёмнике. На графике времени снова включим сигнал «Re {RX}». В отличие от беспорядочного передаваемого сигнала, он имеет различимые линии:

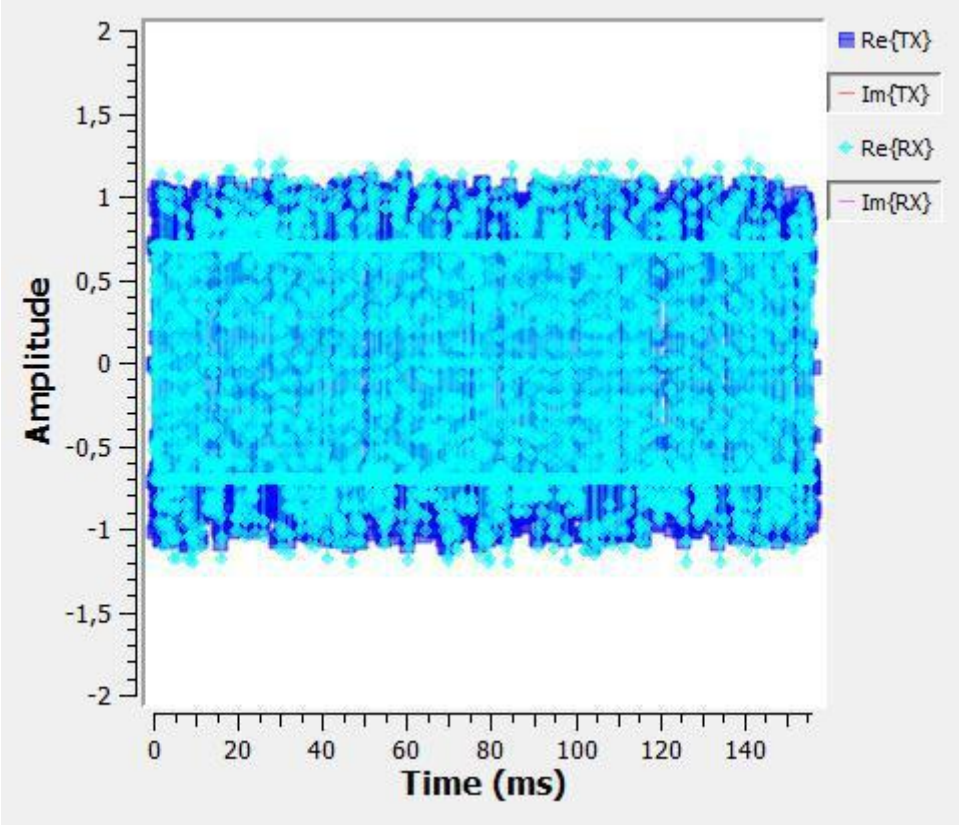

Рис.1.4.

2. Добавление канала

В первом пункте мы рассмотрели только механизм передачи QPSK сигнала. Теперь рассмотрим эффекты канала и то, как сигнал искажается между тем, когда он был передан, и тем, когда мы получаем его на приемнике. Первым шагом будет добавление модели канала, что делается на примере mpsk_stage2.grc. Для начала используем самый базовый блок Channel Model в GNU Radio.

Этот блок позволяет нам смоделировать несколько основных проблем, с которыми нам приходится иметь дело. Первая проблема с приемниками - шум. Тепловой шум в приемнике вызывает шум, который мы называем аддитивным белым гауссовым шумом (AWGN). Мы устанавливаем мощность шума, регулируя значение шумового напряжения модели канала. Здесь мы указываем напряжение вместо мощности, потому что нам нужно знать полосу пропускания сигнала, чтобы правильно рассчитать мощность.

Другая существенная проблема между двумя радиостанциями - это разные тактовые сигналы, управляющие частотой радиостанций. Тактовые

сигналы несовершенны, и, следовательно, отличаются между радиостанциями. Одно радио передает номинально на fc (скажем, 450 МГц), но действительно оно передает на fc + f_delta_1. Между тем, другая радиостанция имеет другой тактовый сигнал и, следовательно, другое смещение, f_delta_2. В конце концов, полученный сигнал будет на f_delta_1 + f_delta_2 от того, где мы думаем, что он должен быть.

С проблемой тактовых сигналов связана проблема идеальной точки выборки. Мы увеличили частоту дискретизации нашего сигнала в передатчике, но при его получении нам необходимо произвести выборку сигнала в исходной точке выборки, чтобы максимизировать мощность сигнала

и минимизировать межсимвольные помехи. После добавления второго фильтра RRC, мы можем видеть, что среди 4 выборок на символ одна из них находится в идеальной точке выборки. Но, опять же, две радиостанции работают на разных скоростях, поэтому идеальная точка выборки неизвестна.

Запустим mpsk_stage2.grc и добавим немного шума, некоторое смещение частоты и некоторое тактовое смещение.

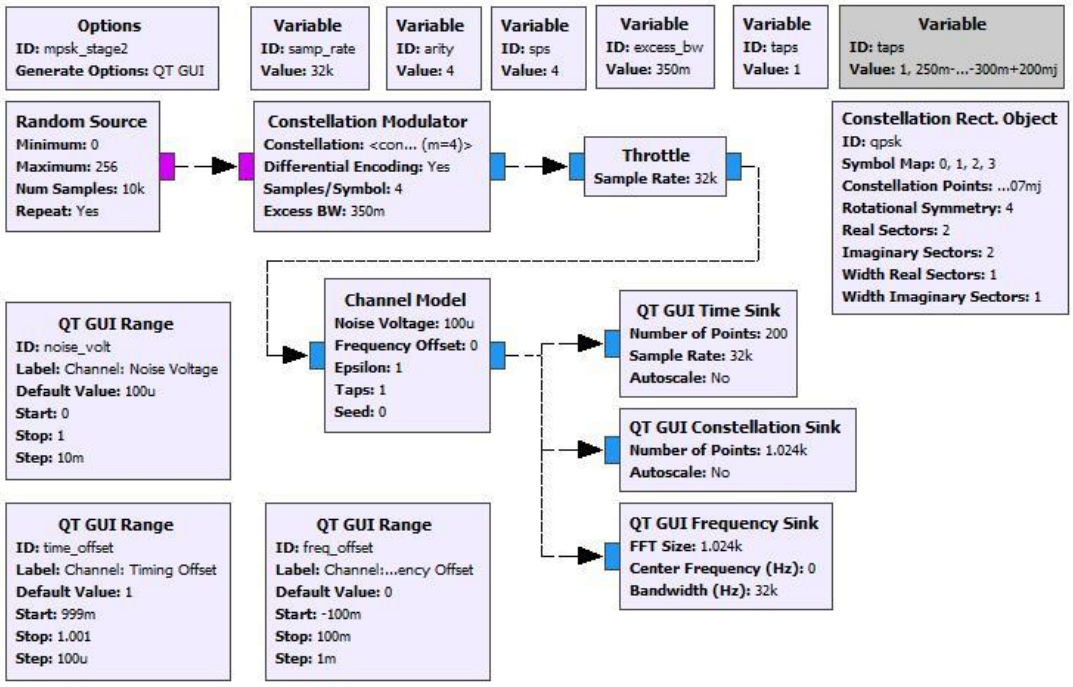

Рис. 2.1.

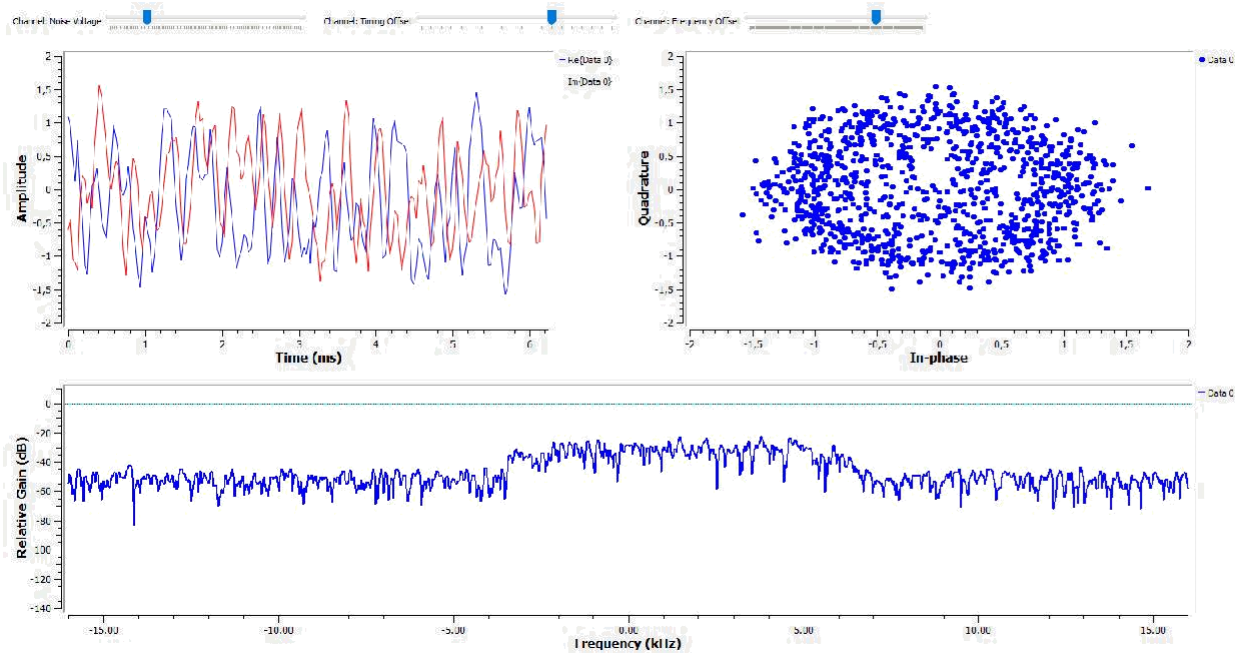

Рис.2.2.

Мы получили график созвездия, выглядящий гораздо хуже, чем тот, который был на первом этапе.

3. Временное восстановление

Теперь нужно восстановить исходный сигнал из полученного искажённого сигнала. Есть много алгоритмов, которые мы могли бы использовать для нивелирования каждого из негативных эффектов. Некоторые могут осуществлять совместное восстановление от нескольких эффектов одновременно. Мы будем использовать алгоритм восстановления многофазных часов.

Начнем с временного восстановления. Сначала нужно найти наилучшее время для выборки входящих сигналов, что позволит максимально увеличить отношение сигнала к шуму (SNR) каждой выборки, а также уменьшить влияние межсимвольных помех (ISI).

Мы можем проиллюстрировать проблему ISI, используя пример symbol_sampling.grc, где мы создаем четыре отдельных символа единицы и затем фильтруем их. На первом этапе фильтрации выполняется повышение частоты дискретизации до 'sps’ выборок на символ и используется RRC фильтр. Затем применяется ещё один RRC фильтр. Сворачивая два RRC фильтра вместе, мы получаем RC фильтр, который является формой фильтра Найквиста.

Графики, приведённые ниже, показывают различия между RRC- и RC-фильтрованными символами. Без фильтрации Найквиста в идеальной точке

выборки каждого символа другие символы имеют некоторую энергию. Если мы сложим эти символы вместе, что мы и сделаем, имея непрерывный поток выборок, энергия других символов исказит символ в данной точке. Но в RC-фильтрованном выходе энергия других символов равна 0 в идеальной точке выборки для данного символа во времени. Это означает, что, если мы производим выборку в идеальной точке выборки, мы получаем энергию только от текущего символа без помех от других символов в потоке.

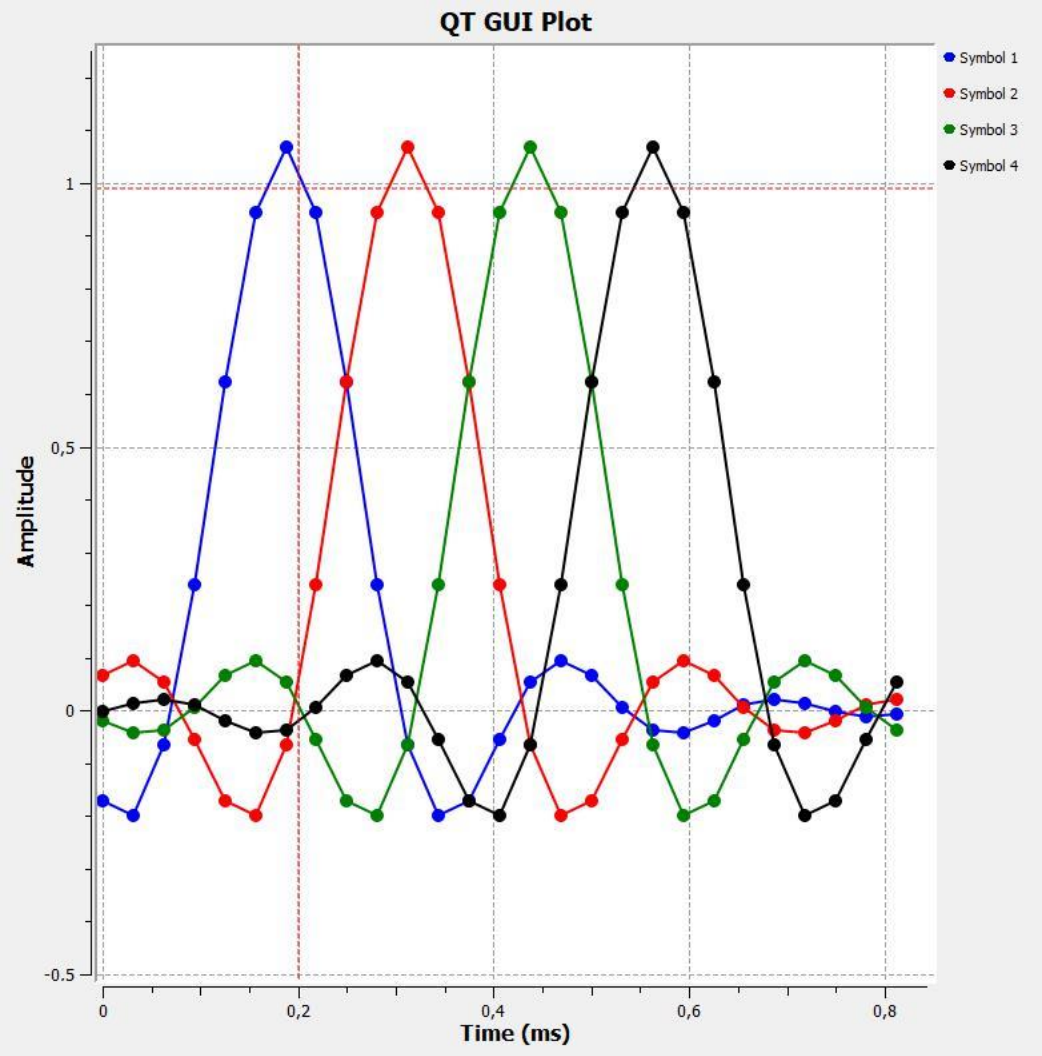

Рис. 3.1

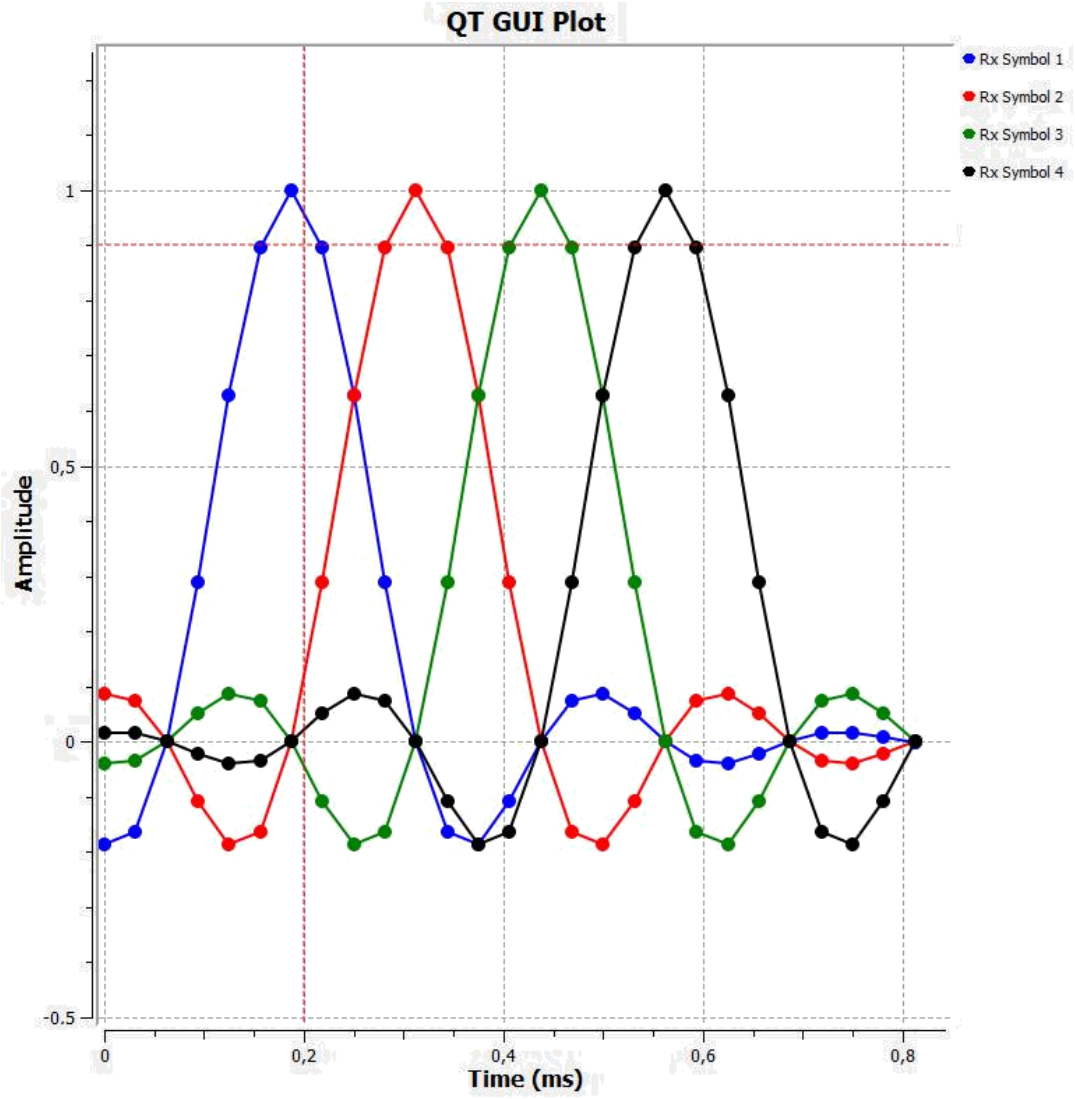

Рис. 3.2.

Теперь давайте посмотрим, что происходит из-за различий в тактовых сигналах между передатчиком и приёмником и как это влияет на точки выборки. Смоделируем это, используя пример symbol_sampling_diff.grc. Тактовые сигналы несовершенны, и поэтому они а) начинаются в разные моменты времени и б) дрейфуют относительно друг друга. Мы моделируем это, добавляя ресемплер, который немного настраивает время выборки символа между передаваемым сигналом и приемником. Разница тактовых сигналов, показанная здесь, в 1.125, является очень большой, это сделано для визуализации. На самом деле, временные различия находятся в пределах миллионных долей. Заметим, что при таких выборках, собираемых в разные моменты времени, идеальный период выборки неизвестен, и любая сделанная выборка также будет включать ISI.

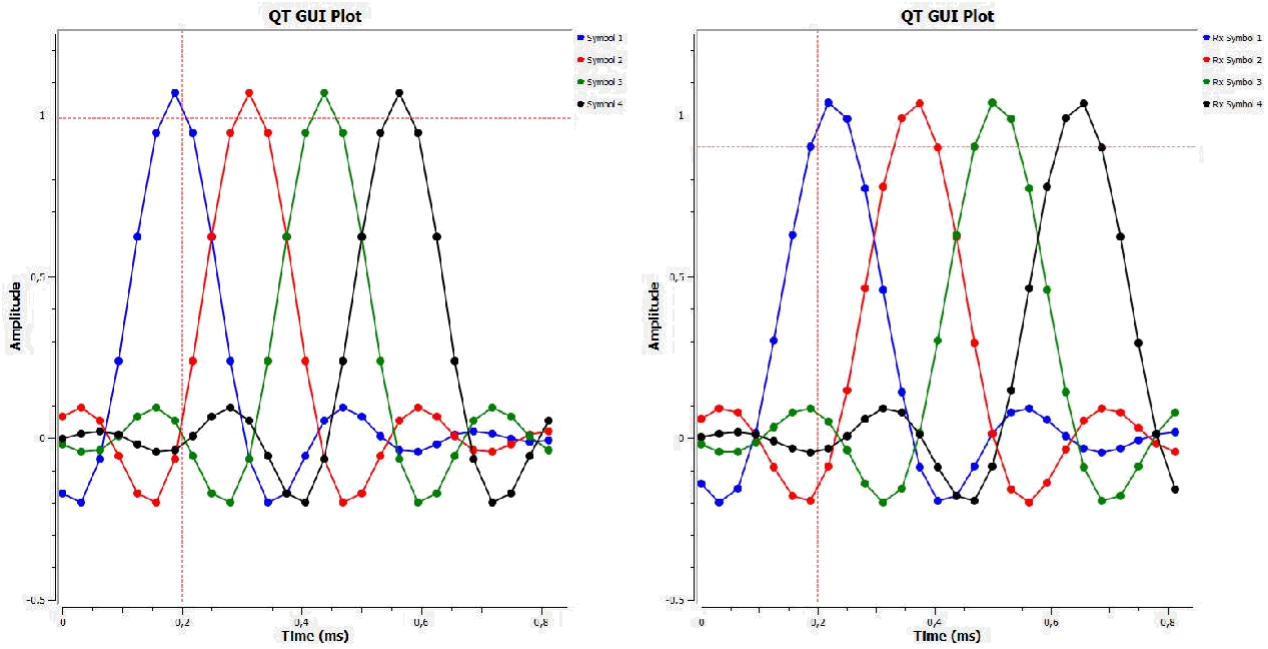

Рис. 3.3.

Наша задача состоит в том, чтобы синхронизировать тактовые сигналы передатчика и приемника, используя только информацию на приемнике из входящих выборок. Эта задача известна как временное восстановление.

4. Детали блока синхронизации многофазных часов

Существуют различные алгоритмы, которые мы можем использовать для восстановления тактового сигнала на приемнике, и почти все они включают в себя некоторый цикл управления с обратной связью. Те, которые не включают, обычно используют некоторое известное слово. Мы будем использовать метод ‘polyphase filterbank clock recovery”. Этот метод позволяет сделать сразу 3 вещи. Во-первых, он выполняет временное восстановление. Во-вторых, он согласовывает фильтр приемника и, тем самым, устраняет ISI. В-третьих, он понижает дискретизацию сигнала и производит выборки с частотой 1 sps.

Блок синхронизации многофазных часов работает, вычисляя первую производную входящего сигнала, которая связана с тактовым смещением сигнала. Рассмотрим работу блока на примерах. В примере symbol_differential_filter.grc мы можем видеть, как идеально всё выглядит, когда нет смещения тактовой частоты. Точка выборки, которая нам нужна, очевидно, находится на 0,25 мс. Дифференциальный фильтр ([-1, 0, 1]) генерирует дифференциал символа, и, как показано на следующем рисунке, результат этого фильтра в правильной точке выборки равен 0. Мы можем

проинвертировать этот оператор и вместо этого сказать, что если на выходе дифференциального фильтра 0, то мы нашли оптимальную точку выборки.

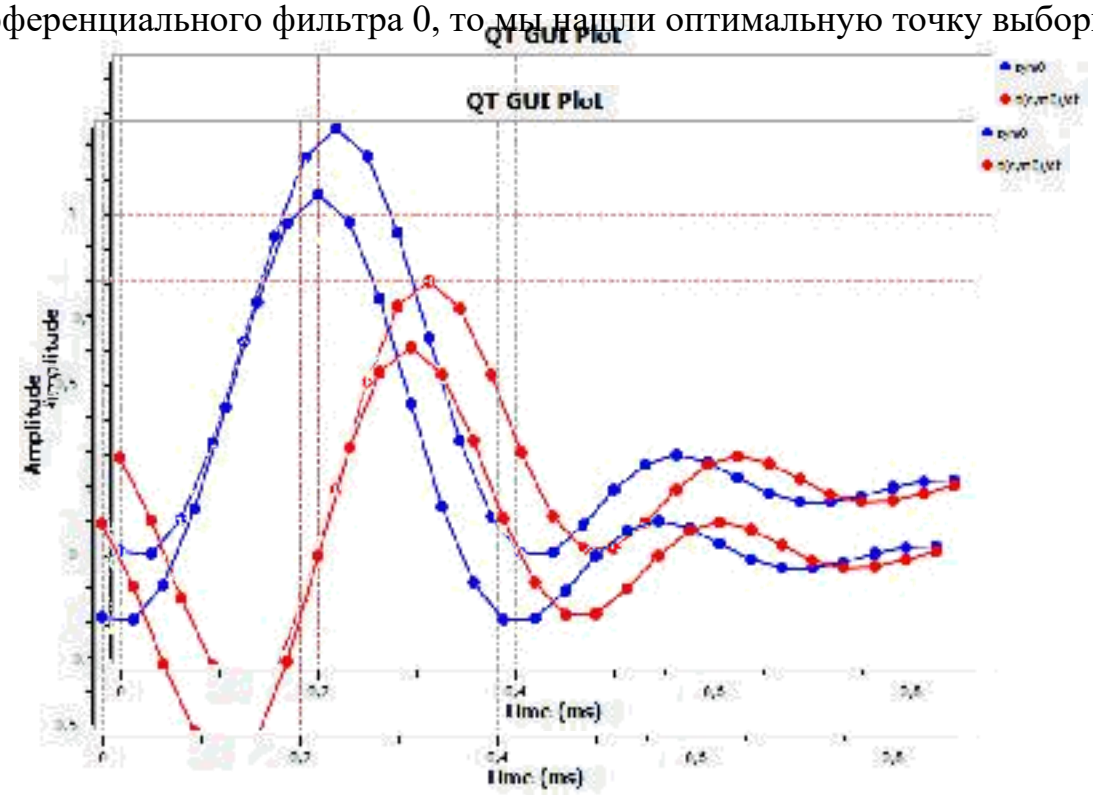

Рис. 4.1.

Но что происходит, когда у нас есть тактовое смещение? В пиковой точке дифференциальный фильтр уже не будет давать 0:

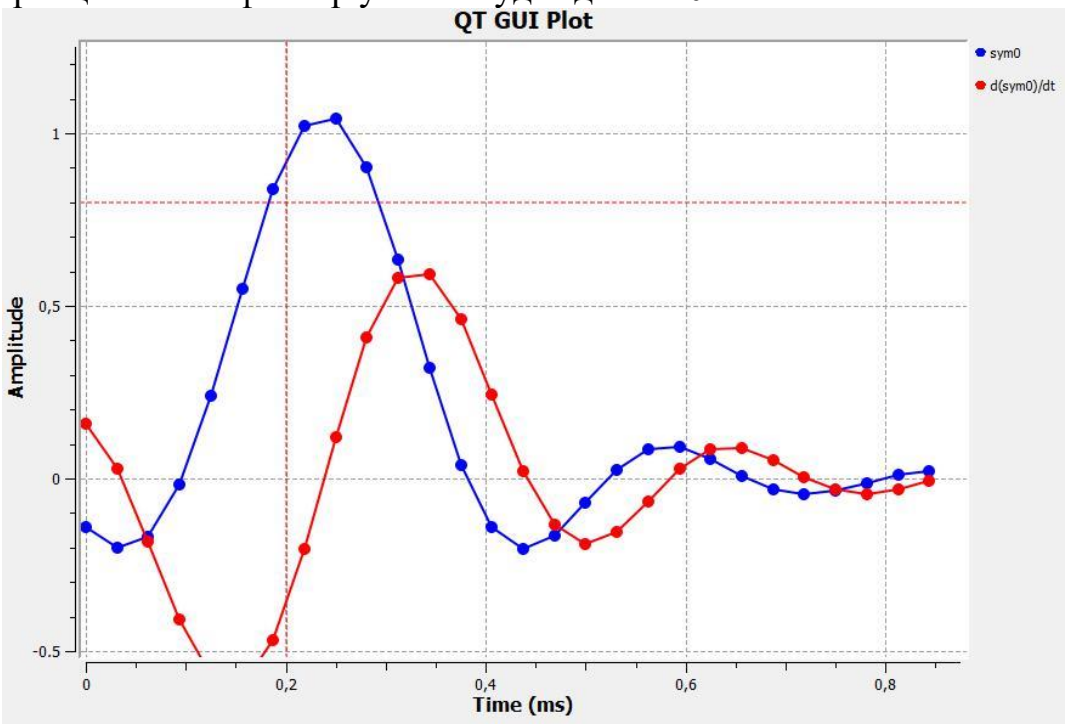

Рис. 4.2.

Вместо того, чтобы использовать один фильтр, мы можем создать серию фильтров, каждый из которых будет иметь свою фазу. И если у нас будет

достаточно фильтров с разными фазами, одна из них окажется правильной и даст нам желаемое значение. Рассмотрим симуляцию, которая строит 5 фильтров, что означает 5 различных фаз (пример symbol_differential_filter_phase.grc).

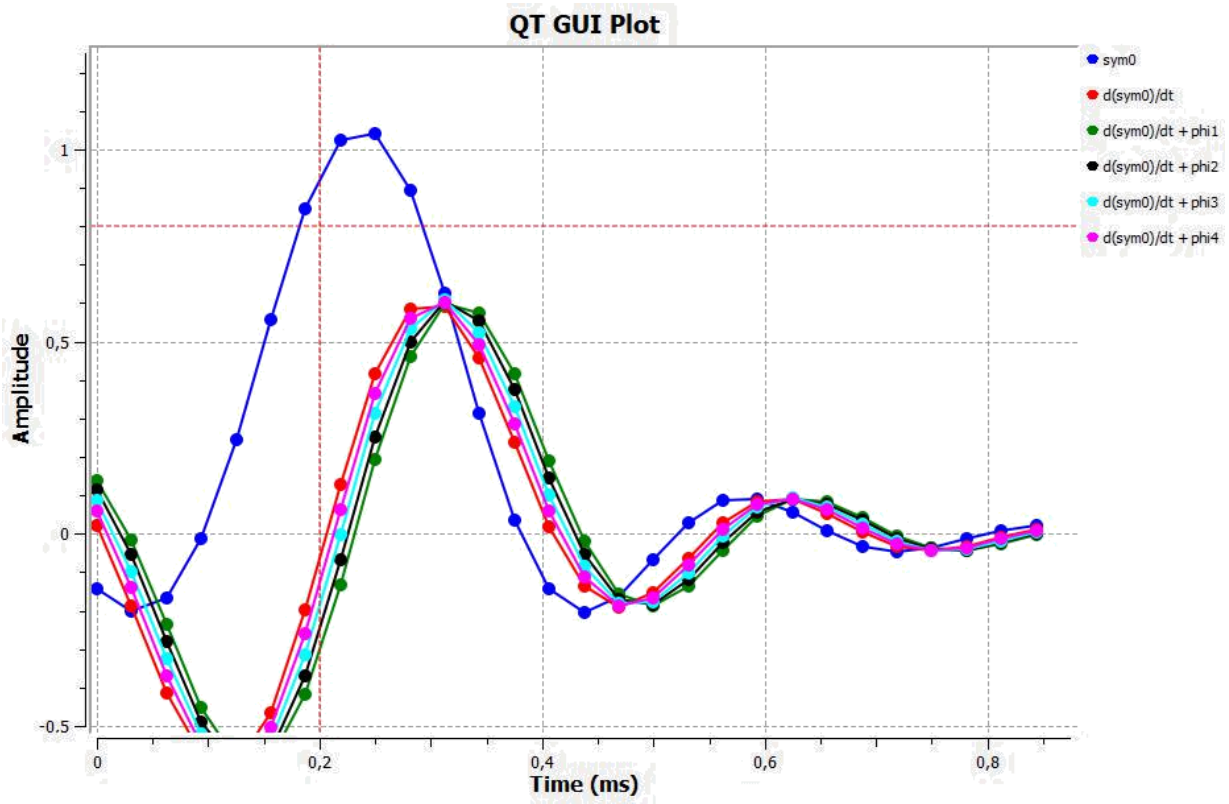

Рис. 4.3.

Видно, что сигнал, помеченный как d (sym0) / dt + phi3 в правильной точке выборки равен 0. Это говорит о том, что наша идеальная точка выборки возникает при этом сдвиге фазы. Поэтому, если мы возьмем фильтр RRC нашего приемника и настроим его фазу на phi3 (которая равна 3
* 2pi / 5), то мы сможем исправить несоответствие синхронизации и выбрать идеальную точку выборки.

Но это лишь моделируемое приближение, в действительности выборки каждого фильтра не будут происходить в один и тот же момент времени. Чтобы действительно увидеть такое поведение, мы должны увеличить частоту дискретизации в количество раз, равное количеству фильтров. Мы можем рассматривать эти разные фильтры как части одного большого фильтра с частотой дискретизации в M раз больше, где M = 5 в нашем простом примере. Мы могли бы увеличить частоту нашего входящего сигнала и выбрать момент времени, когда мы получаем 0 на выходе фильтра, но проблема в том, что мы говорим о большой сложности вычислений, поскольку она пропорциональна

нашей частоте дискретизации. Вместо этого, будем работаем с фильтрами с разными фазами на входящей частоте дискретизации, но, имея такой набор фильтров, мы сможем получить эффект от работы с фильтром с увеличенной частотой дискретизации без дополнительных вычислительных затрат.

Таким образом, в примере выше мы сместили частоту дискретизации на некоторый известный коэффициент 1,2 и обнаружили, что можем использовать один из пяти фильтров в качестве идеальной точки выборки. К сожалению, у нас действительно есть только 5 различных фаз, которые мы можем точно исправить. Любое смещение выборки между этими фазами все равно приведет к неправильной выборке с добавлением ISI. Поэтому в алгоритме восстановления тактовой частоты мы используем большее количество фильтров. Не углубляясь в математику, мы можем использовать 32 фильтра, чтобы максимальный шум от ISI был меньше шума квантования 16-битного значения. Если нам нужна точность более 16 бит, мы можем использовать больше фильтров.

В итоге у нас есть большой банк фильтров, один из которых находится очень близко к идеальному смещению фазы дискретизации. Как мы автоматически выберем нужный фильтр? Используя цикл управления 2-го порядка (2nd order control loop), как мы почти всегда делаем в ситуациях восстановления. Сигналом ошибки восстановления будет являться выход дифференциального фильтра. Цикл управления запускается на одном из фильтров и вычисляет выходной сигнал как сигнал ошибки. Затем он перемещается вверх или вниз по банку фильтров в зависимости от полученного сигнала ошибки и пытается найти, где сигнал ошибки наиболее близок к нулю, тем самым находя оптимальный фильтр.

5. Использование блока синхронизации многофазных часов в приемнике

Применим рассмотренный блок в нашей симуляции. Пример mpsk_stage3.grc берет выходные данные модели канала и пропускает их через блок синхронизации. Сам блок использует 32 фильтра по причинам, которые мы обсуждали ранее и принимает на вход число ожидаемых выборок на символ, но это число - всего лишь наше предположение, внутри блок будет адаптировать его, основываясь на скоростях входящего сигнала. Заметьте, однако, что мы настроили эту симуляцию так, что оценка немного отличается от 4-х sps, которые мы передаем. Это делается для имитации начального тактового смещения между передатчиком и приемником.

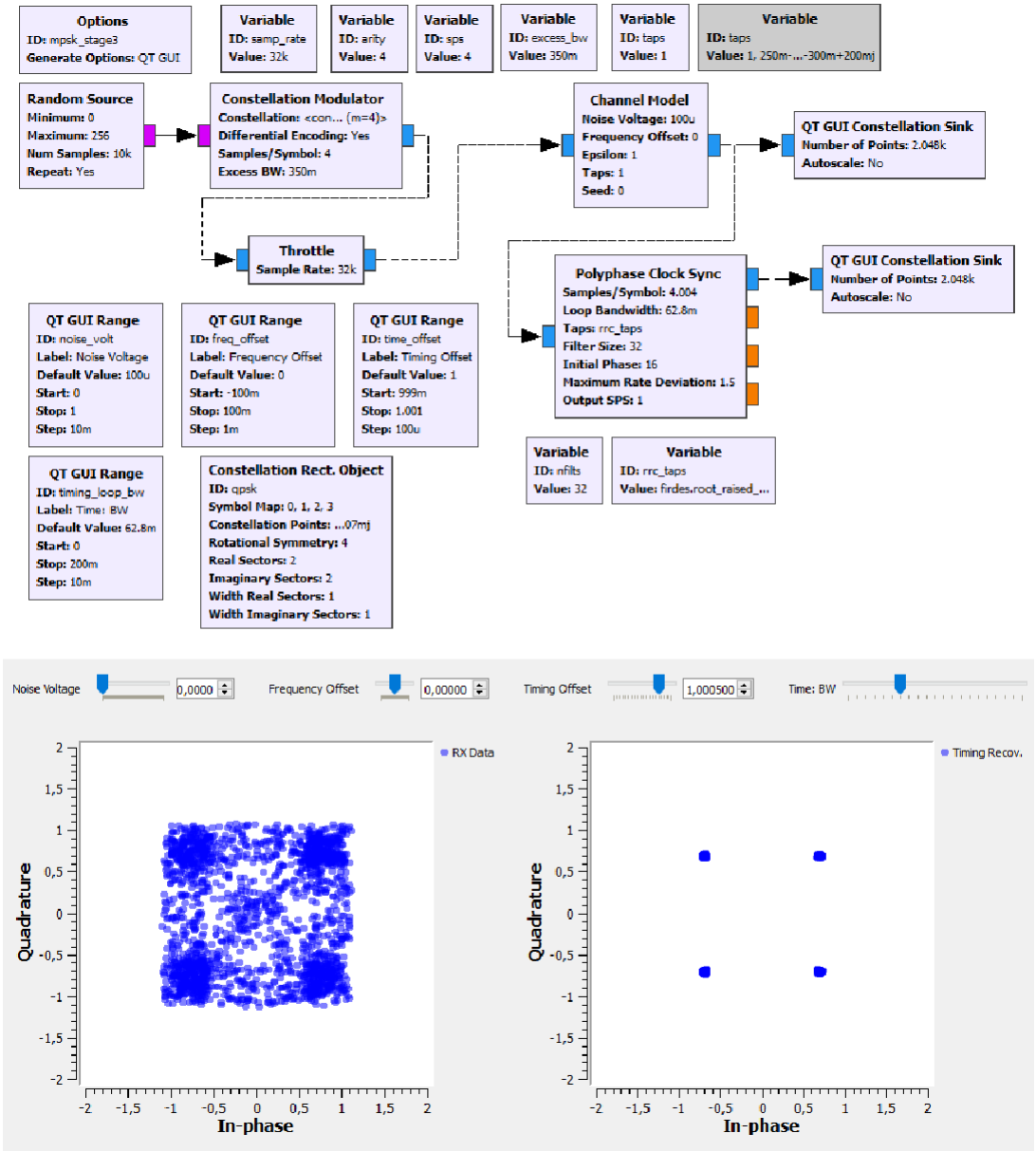

Рис. 5.1.

На графике слева - сигнал до синхронизации, справа - после. Он всё-ещё немного зашумлён из-за ISI, но этот шум быстро поглощается другим шумом при установке Noise Voltage для каналов отличным от нуля:

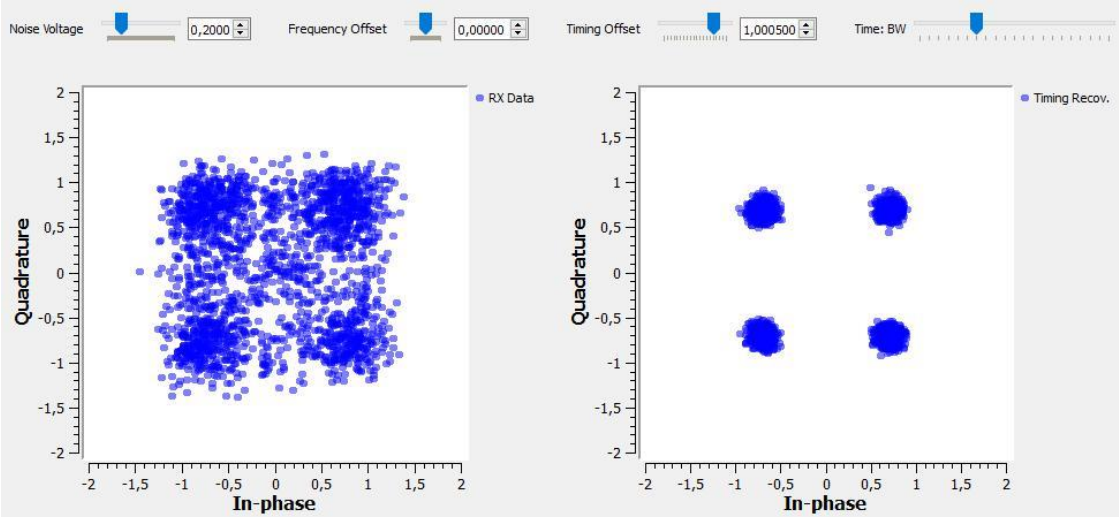

Рис. 5.2

Если добавить смещение частоты, то созвездие станет окружностью. Созвездие всё еще будет находится на единичной окружности, что означает, что тактовая синхронизация сохраняется, но блок не позволяет нам корректировать смещение частоты. Позже нам придётся разобраться с этим.

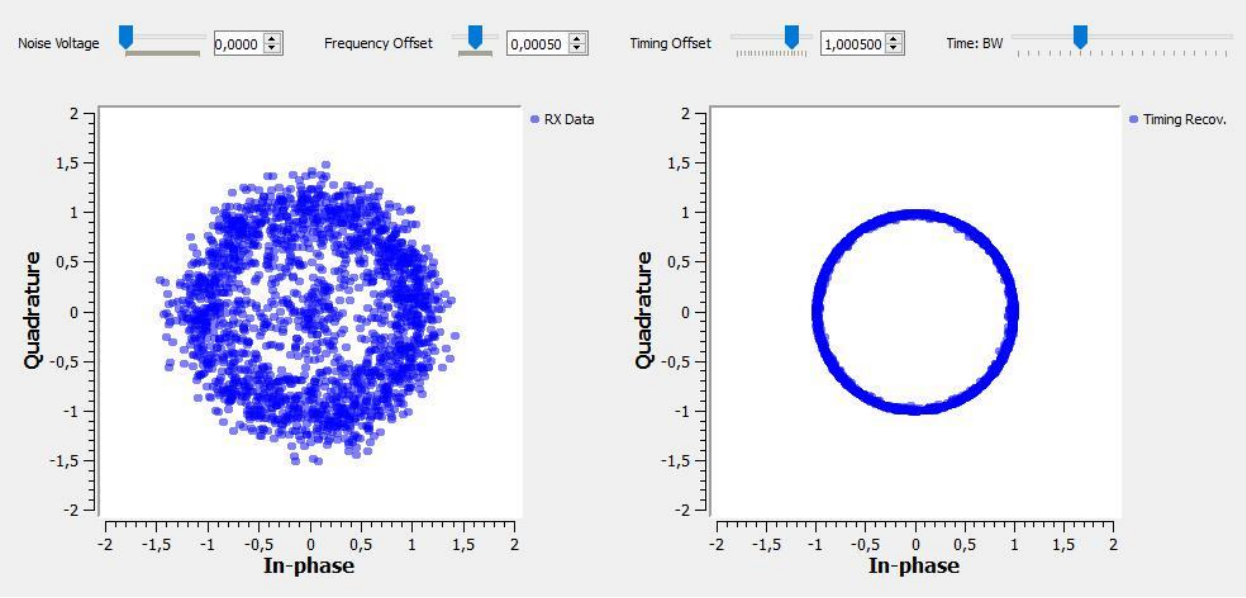

Рис. 5.3.

Кроме того, с помощью изменения переменной taps мы можем добавить многолучевое распространение и увидеть, что блок восстановления тактового сигнала устойчив к многолучевому распространению, но не исправляет его, поэтому нам снова нужно что-то ещё.

6. Многолучевое распространение

Сначала узнаем, что такое многолучевое распространение, откуда оно берется и как оно влияет на наши коммуникационные возможности.

Многолучевое распространение обусловлено тем, что в большинстве сред связи у нас нет единого пути прохождения сигнала от передатчика к приемнику. Как показано на рисунке ниже, каждый раз, когда появляется объект, отражающий сигнал, между двумя узлами может быть установлен новый путь. Поверхности, такие как здания, знаки, деревья, люди, кошки и т.д. могут создавать отражения сигналов. Сигнал, проходя по каждому из путей, будет доходить до приемника за разное время, в зависимости от длины пути.

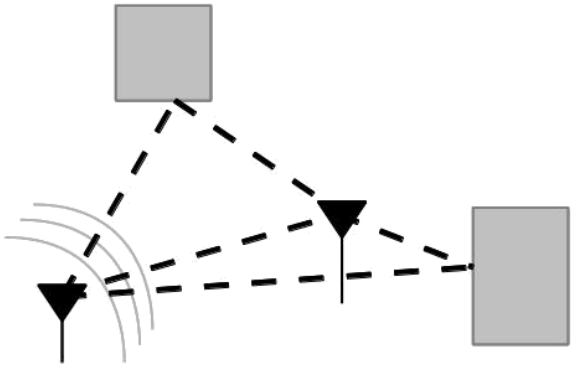

Рис. 6.1.

Воздействие комбинации таких сигналов на приемник вызывает искажение сигнала.

Мы можем исправить это, используя механизм, очень похожий на стереоэквалайзер. На самом деле, мы называем их эквалайзерами. С помощью стереоэквалайзера мы можем изменить усиление определенных частот, чтобы подавить или усилить сигналы. Рассмотрим очень простой пример multipath_sim.grc, чтобы посмотреть, как это выглядит в частотной области.

Multipath_sim.grc создаёт модель канала, обеспечивая канал пятью частотными полосами для эквалайзера, четыре из которых можно контролировать. При значении 1 полоса будет пропускать частоты без помех. При значении 0 полоса будет давать глубокий ноль в спектре, который будет влиять на все частоты вокруг.

Хотя в этом примере мы явно контролируем частотную область, мы действительно можем создать эквалайзер, способный регулировать частотную характеристику принимаемого сигнала. Наша цель показана на рисунке ниже, где канал многолучевого распространения создает некоторое искажение в сигнале. Задача эквалайзера - инвертировать это канал. По сути, мы хотим отменить искажение, вызванное каналом, чтобы выход эквалайзера был плоским. Наша задача - использовать правильный алгоритм эквалайзера и настроить параметры. Одним из важных параметров является количество частотных полос в эквалайзере. Как мы видим из нашего моделирования, пять полос дают довольно грубый контроль над частотной характеристикой. Чем

больше полос, тем глубже контроль, но тем больше времени требуется для их вычисления.

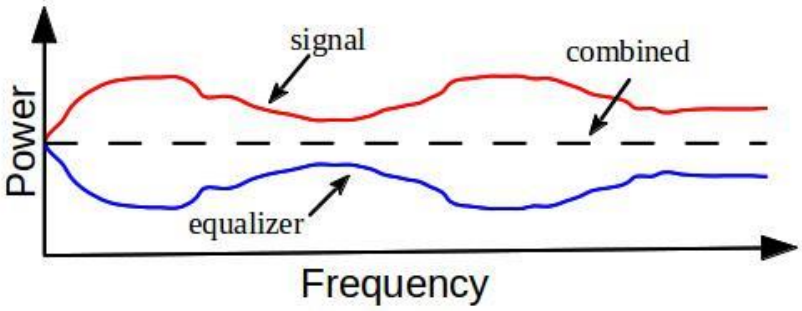

Рис. 6.2.

7. Эквалайзеры

GNU Radio поставляется с двумя легко используемыми эквалайзерами, CMA и DD LMS. Мы будем использовать DD LMS (Decision-Directed Least Mean Squared).

CMA и DD LMS довольно похожи в плане параметров, за исключением одной важной особенности. DD LMS, в отличие от CMA – не слепой эквалайзер, он требует информации о принимаемом сигнале. Ему необходимо знать точки созвездия.

Этот эквалайзер отлично подходит для сигналов, которые не соответствуют требованию постоянного модуля алгоритма CMA. С другой стороны, если SNR достаточно плох, принимаемые решения будут неправильными, что может ухудшить производительность приемника. Кроме того, DD LMS более сложен в вычислительном отношении. Однако, когда у нас есть сигнал хорошего качества, этот эквалайзер может генерировать более высококачественные сигналы, нежели CMA, поскольку у него есть информация о сигнале.

Возьмём пример mpsk_stage4.grc и заменим CMA на LMS-DD:

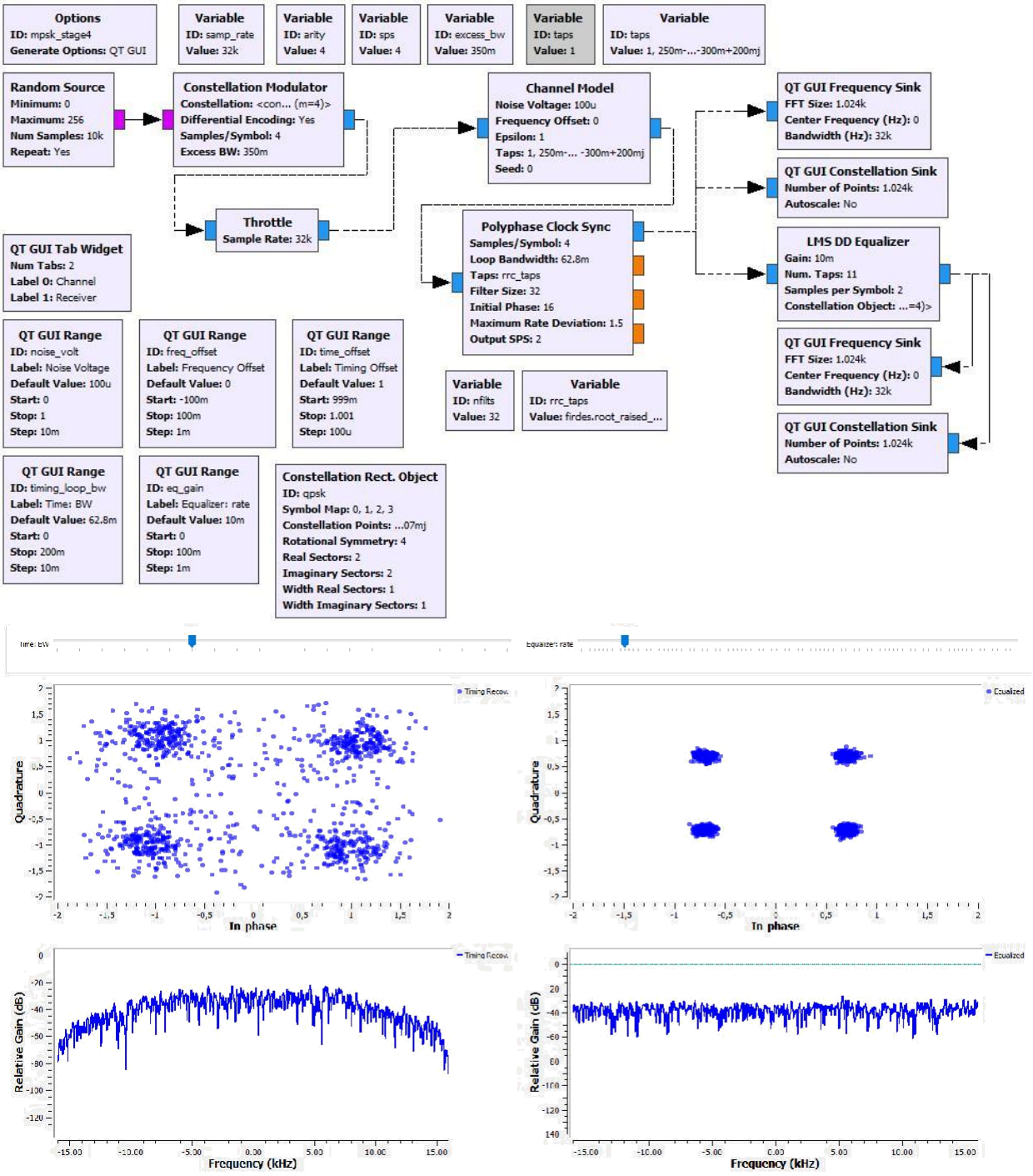

Рис.7.1.

Noise Voltage = 0.2, Frequency Offset=0, Timing Offset = 1.0005

Мы можем увидеть, как сходится DD LMS. Поскольку у нас есть и блок синхронизации, и блок эквалайзера, они сходятся независимо, но первый этап влияет на второй. В конце мы можем увидеть влияние сигнала многолучевого распространения до и после эквалайзера. Перед эквалайзером у нас весьма уродливый сигнал даже без шума, но эквалайзер способен нивелировать помехи, чтобы у нас снова был чистый сигнал.

8. Фазовая частотная коррекция

У нас все ещё остаётся проблема сдвига фазы и частоты. Эквалайзеры, как правило, не адаптируются быстро, поэтому смещение частоты может легко превысить их возможности.

Две вещи об этом этапе. Во-первых, мы будем использовать цикл второго порядка, чтобы мы могли отслеживать как фазу, так и частоту, которая является производной фазы по времени. Во-вторых, тип восстановления, с которым мы здесь будем иметь дело, предполагает, что мы делаем точную коррекцию частоты. Поэтому мы должны быть уверены, что мы уже находимся достаточно близко к идеальной частоте.

Мы будем использовать цикл Костаса (пример mpsk_stage5.grc). Блок Costas Loop может синхронизировать BPSK, QPSK и 8PSK.

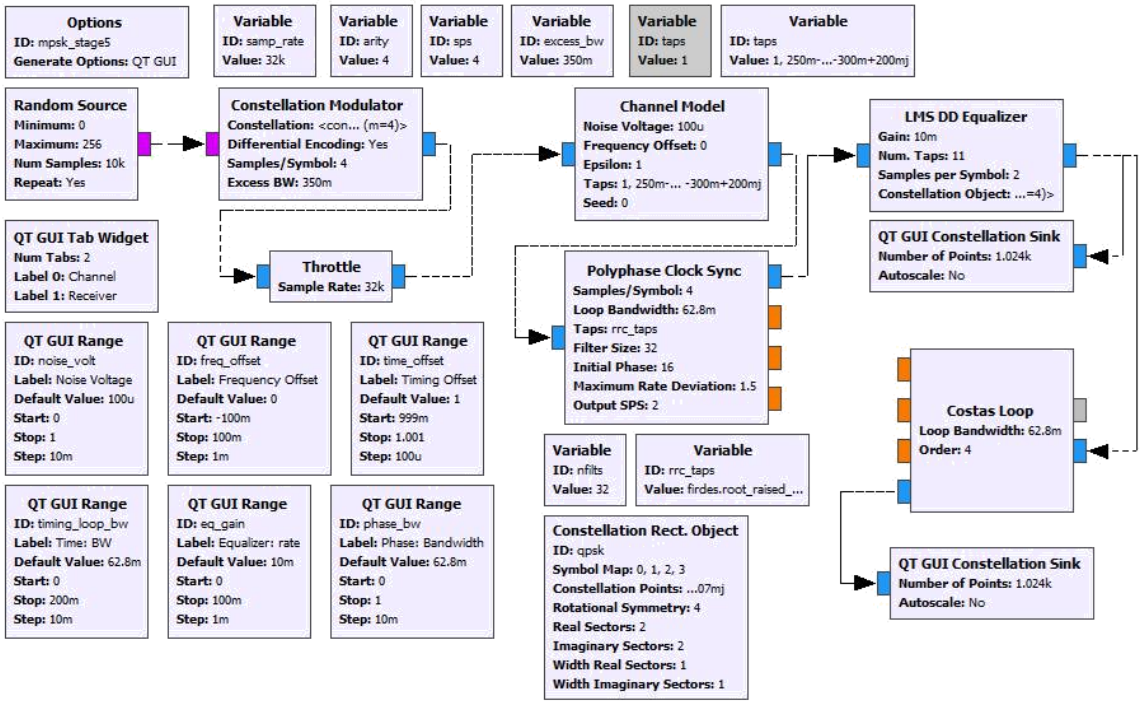

Рис.8.1.

Этот блок использует цикл второго порядка и поэтому определяется параметром пропускной способности цикла. Другая вещь, которая ему необходима - это порядок модуляции PSK (2 для BPSK, 4 для QPSK и 8 для 8PSK). На следующем рисунке мы установили шум, тактовое смещение, простой многолучевой канал и смещение частоты. После эквалайзера мы видим, что все символы находятся на единичной окружности, но вращаются из-за смещения частоты, которое пока не исправляется. На выходе блока Costas Loop мы можем увидеть исходное созвездие, хоть и с дополнительным шумом, с которым мы ничего не можем поделать.

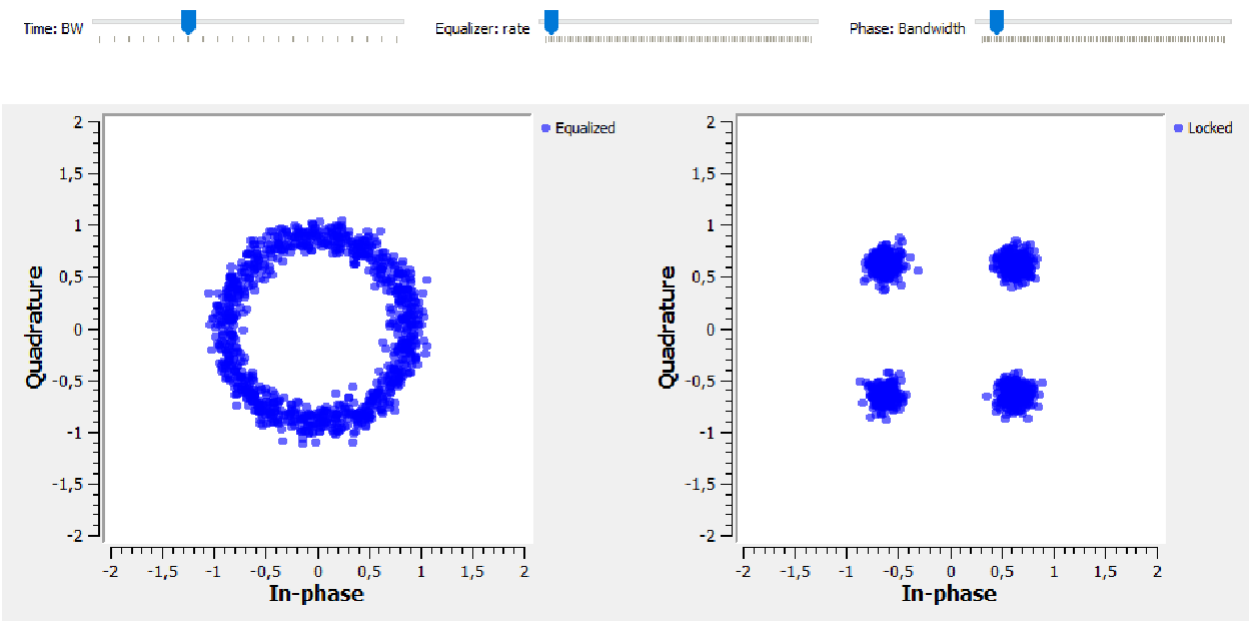

Рис. 8.2.

9. Декодирование

Декодируем сигнал. Используя пример диаграммы mpsk_stage6.grc , мы добавляем квадратурный демодулятор после Costas Loop вместо my_qpsk_demodulator. На этом этапе мы получаем символы от 0 до 3, потому что это размер нашего алфавита в схеме QPSK. Но из этих 0-3 символов, как мы можем точно знать, что у нас такое же отображение символов в точках созвездия, которое мы делали при передаче? К счастью, мы избежали этой проблемы, передав дифференциальные символы. На самом деле мы не передавали само созвездие, мы передавали разницу между символами созвездия, установив для параметра «Differential» в блоке «Constellation Modulator» значение «True».

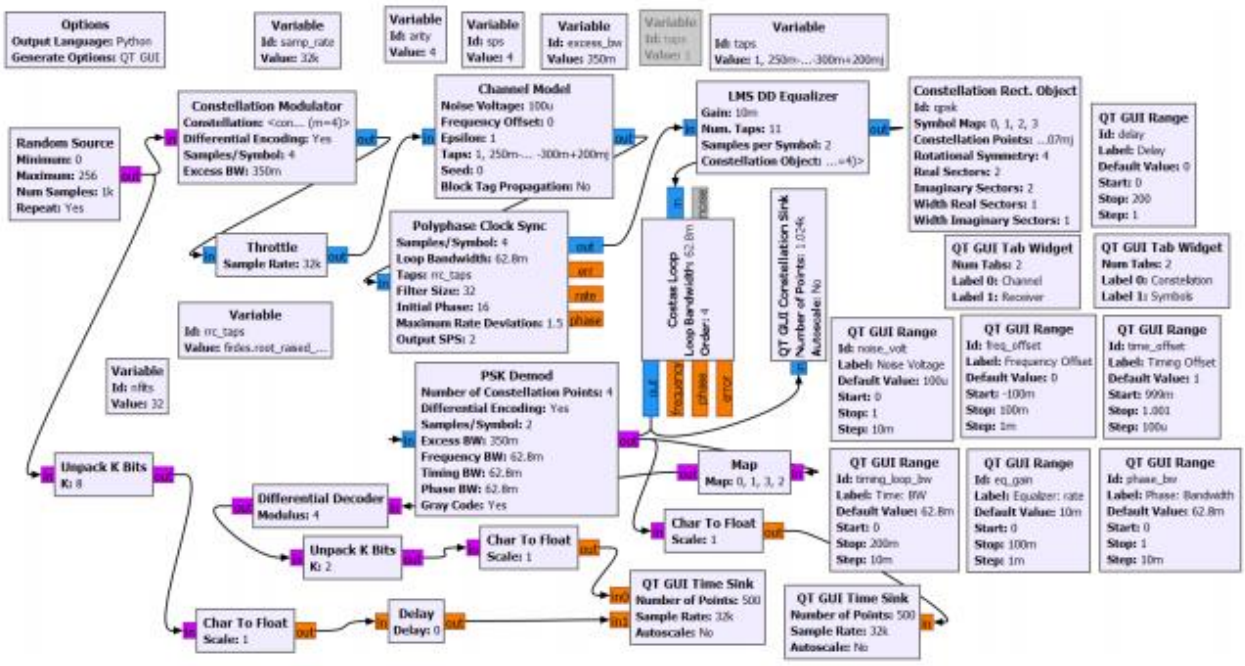

Рис.9.1.

Блок-схема использует блок “PSK Demod” для преобразования дифференциально-кодированных символов обратно в их исходные символы. Но даже отсюда наши символы не совсем верны. Это самая сложная часть демодуляции. На этапах синхронизации физика и математика была на нашей стороне. Теперь, однако, мы должны интерпретировать некоторый символ, основываясь на том, что это сказал кто-то другой. Мы, в основном, просто должны знать это отображение. И, к счастью, мы это делаем, поэтому мы используем блок Map для преобразования символов из дифференциального декодера в исходные. Теперь у нас есть исходные символы от 0 до 3, поэтому давайте распакуем эти 2 бита на символ в биты, используя блок распаковки битов. Теперь у нас есть оригинальный поток данных!

Остался вопрос: как узнать, что это оригинальный битовый поток? Для этого сравним его с входным потоком битов, что мы можем сделать, т.к. это симуляция, и у нас есть доступ к переданным данным. Передатчик генерировал упакованные биты, поэтому мы снова используем битовый блок распаковки. Затем мы преобразуем эти потоки в значения с плавающей запятой 0,0 и 1,0. Сравнение этих двух потоков непосредственно ничего не покажет. Почему? Из-за того, что цепочка приемника имеет много блоков и фильтров, которые задерживают сигнал, принятый сигнал отстает на некоторое количество битов. Чтобы компенсировать это, мы должны задержать переданный сигнал на ту же величину, используя блок “Delay”.

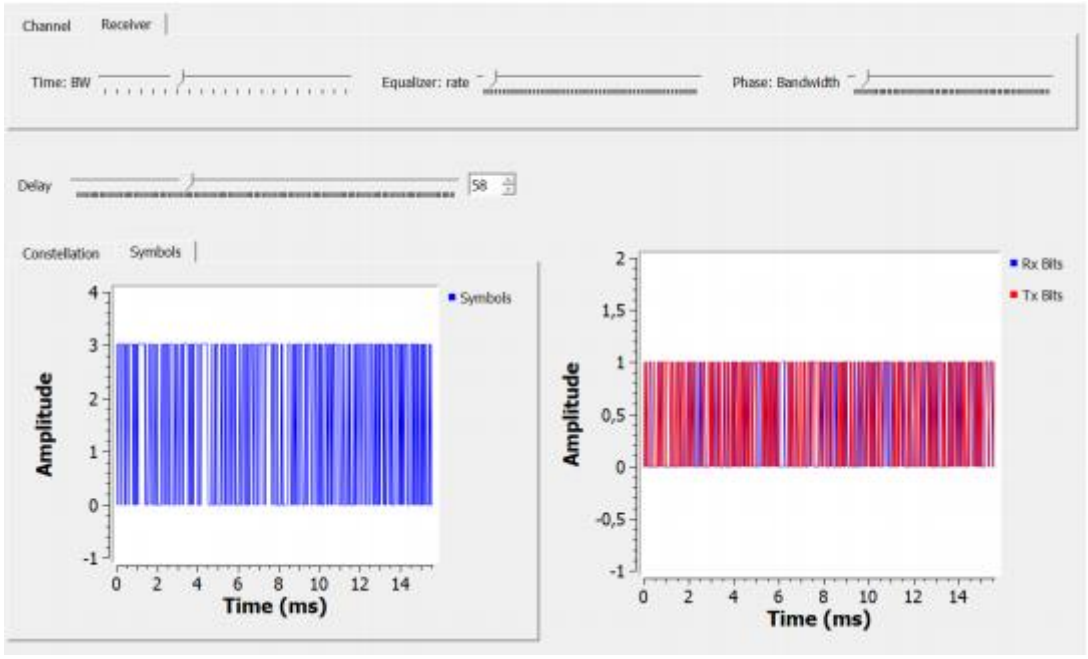

Рис. 9.2.

Результат немного отличается от ожидаемого, поэтому заключим, что в этой версии есть некоторая необнаруженная проблема с синхронизацией.

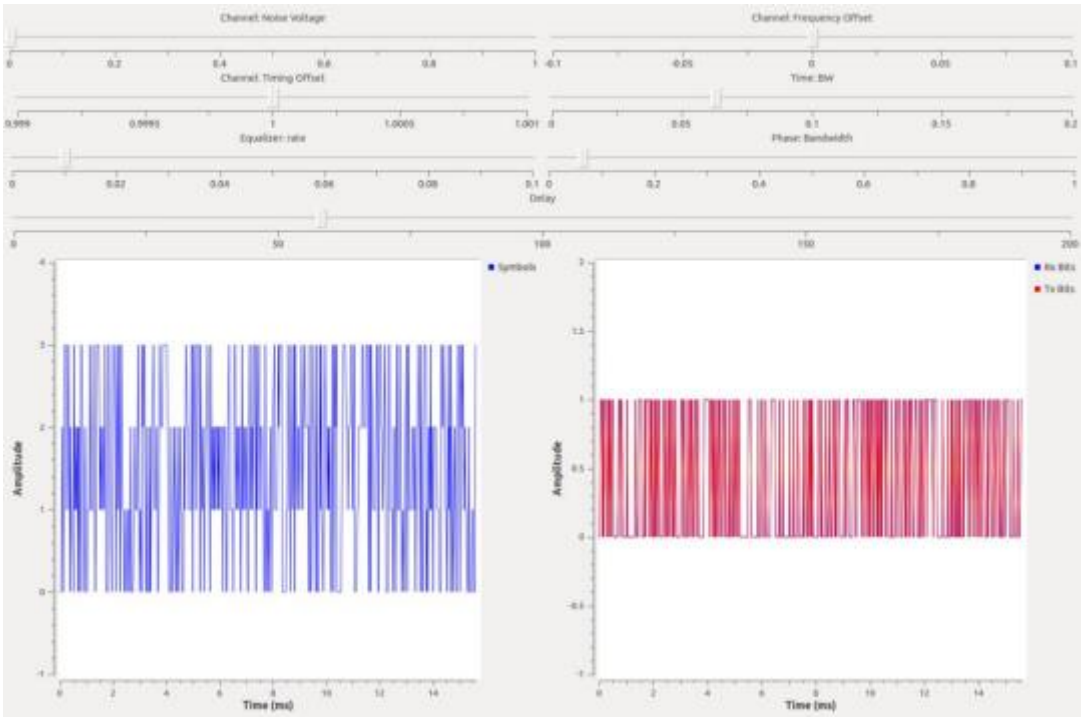

Рис. 9.3.

Данный рисунок показывает то, что мы ожидали увидеть. Здесь можно вычесть один сигнал из другого, чтобы увидеть, когда они синхронизируются, поскольку выходной сигнал будет равен 0. Добавление шума и других влияний на канал можно увидеть в виде помех, когда сигнал не равен 0. После подбора значение задержки, мы видим, что принятый поток совпадает с исходным.