GNU radio

Чернышев Ярослав

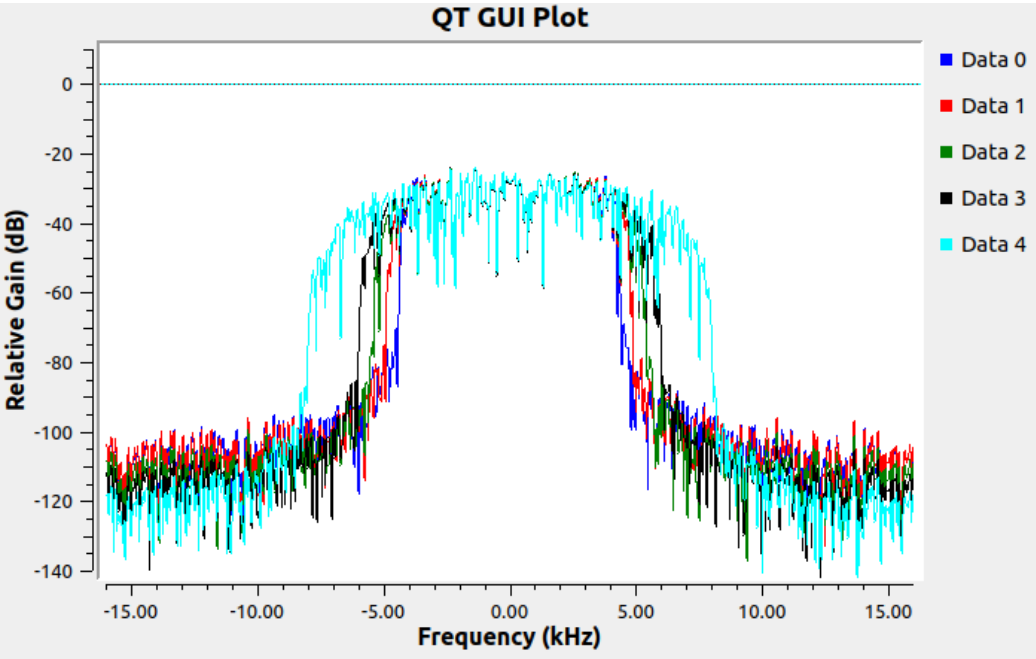
1 июня 2021 г.

Оглавление

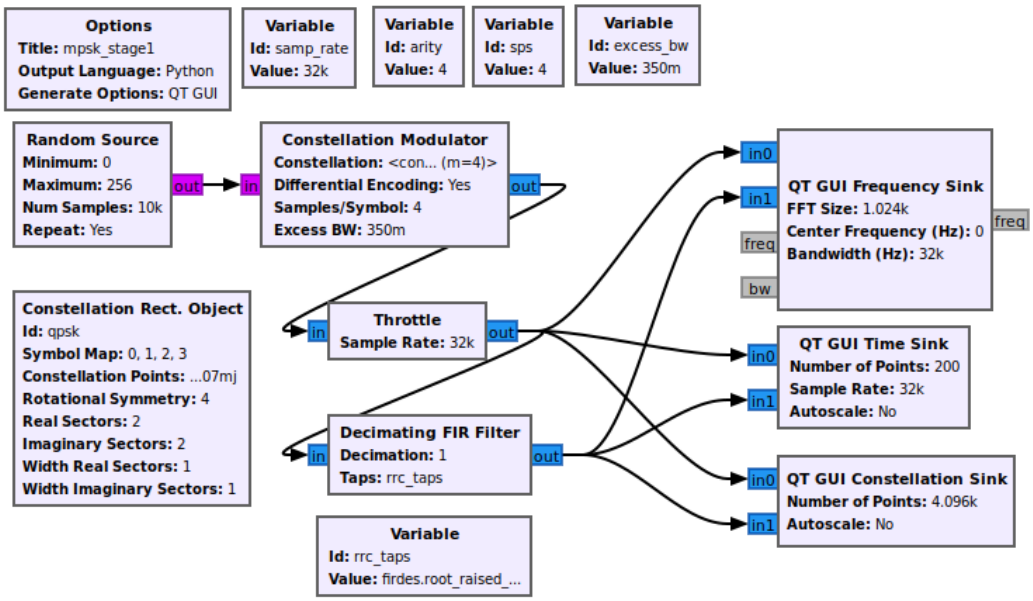
Глава 1

Передача сигнала

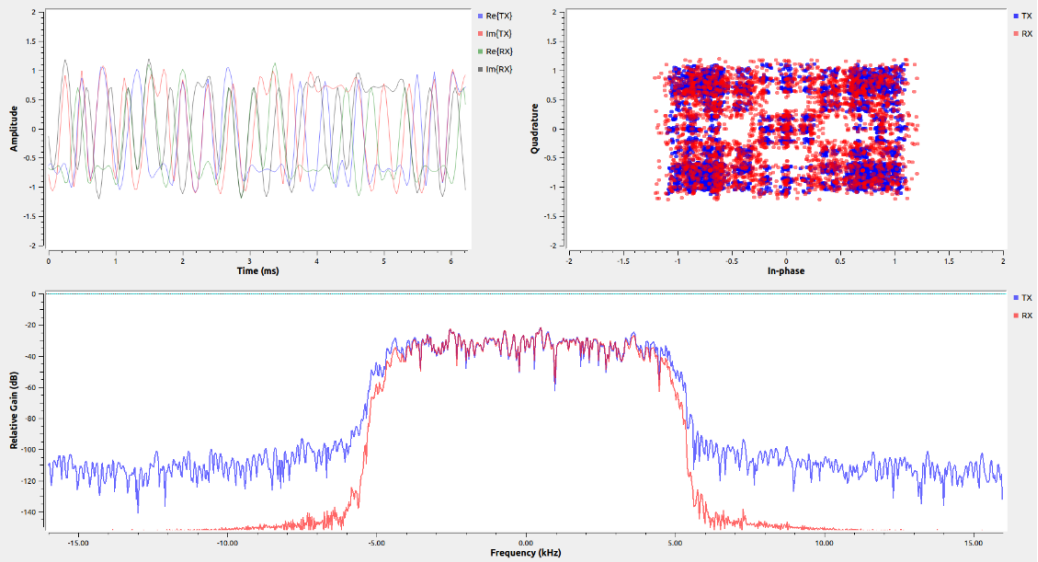
Первым этапом является передача QPSK сигнала. О нас требуется сгенерировать поток битов и отобразить его на, как я понял, двумерную плоскость, на которой комплексным числам будут соответствовать наши символы. Для этого используем блок Constellation Modulator, который использует объект Constellation. Этот объект позволяет определить, как именно символы кодируются. Также, Constellation Modulator работает с байтами, поэтому для работы с ним может понадобиться генератор байтовых значений. Количество сэмплов оставлено равным 4. Теперь можно перейти к экспериментам с выбором разной избыточной пропускной способности. Этот параметр влияет на крутизну краев фильтра.



Как показывают эксперименты, чем больше excess bandwidth, тем более пологие края. Следующий Flow Graph осуществляет передачу QPSK-созвездия. Он показывает переданный сигнал, полученный на приемнике сигнал во времени и частотах и созвездие.



В результате, ниже, мы получаем следующее созвездие:



На рисунке можно видеть эффект увеличения частоты дискретизации и процесс фильтрации. В нашем случае RRC-фильтр специально добавляет межсимвольные помехи с самим собой. Для минимизации погрешности на приемной стороне используется еще один RRC-фильтр. При помощи свертки мы получаем импульсы приподнятого косинуса с минимизированным ISI.

Глава 2

Добавление искажений

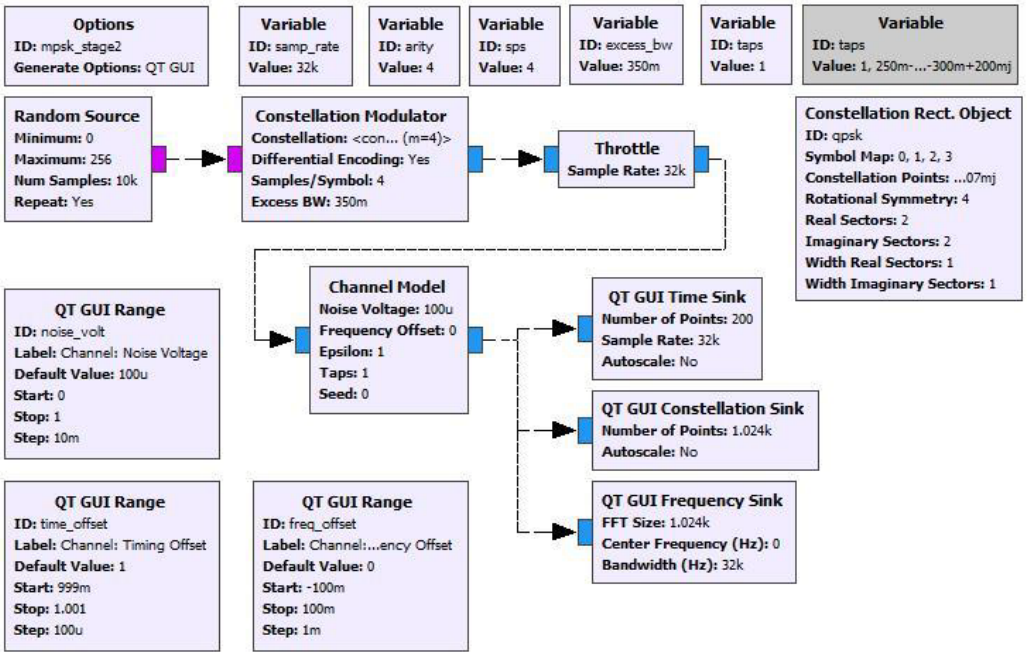
В первом пункте мы рассмотрели только механизм передачи QPSK сигнала. Теперь рассмотрим эффекты канала и то, как сигнал искажается между тем, когда он был передан, и тем, когда мы получаем его на приемнике. Первым шагом будет добавление модели канала, что делается на примере mpsk_stage2.grc. Для начала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

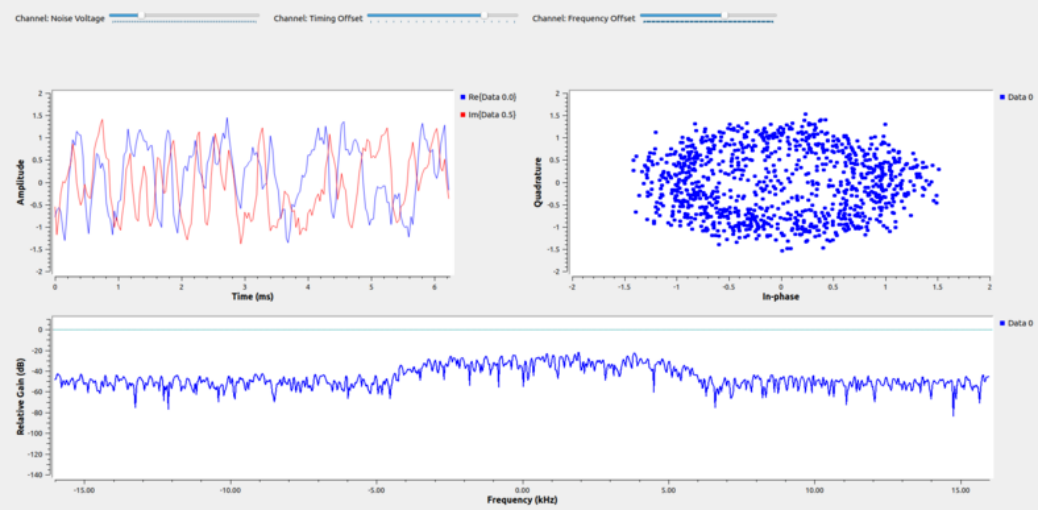
Этот блок позволяет нам смоделировать несколько основных проблем, с которыми нам приходится иметь дело. Первая проблема с приемниками - шум. Тепловые помехи в приемнике вызывает шумы, которые мы называем аддитивным белым гауссовым шумом (AWGN). Мы устанавливаем мощность шума, регулируя значение шумового 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 в разных устройствах не связаны между собой, и, следовательно, отличаются между радиостанциями. Это также может привести к помехам.

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.





Получившийся график созвездия хуже, чем тот, что мы получили на первом этапе.

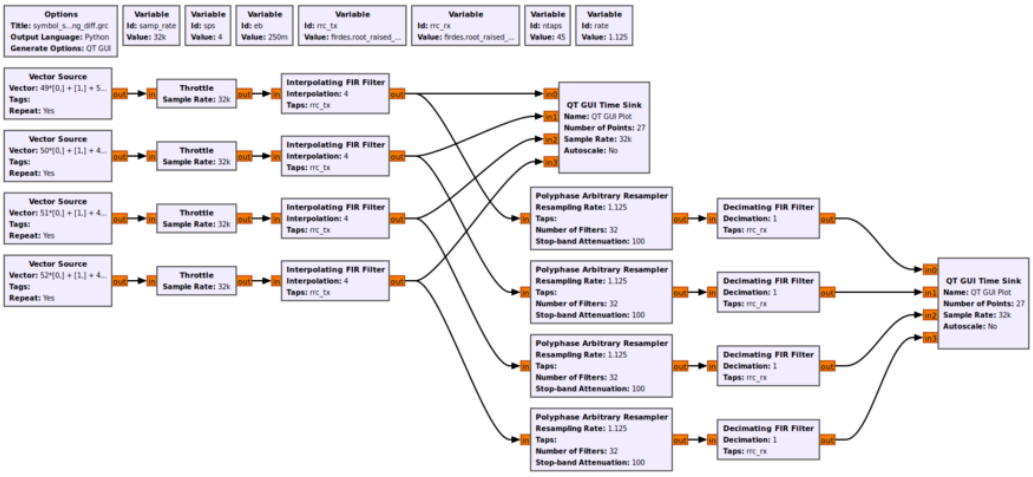
Глава 3

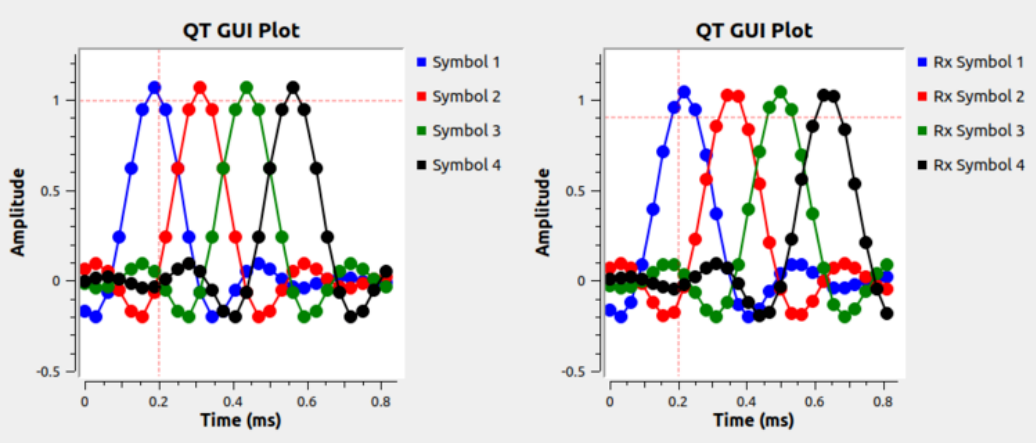
Временная коррекция

В рамках данной работы, для осуществления временной коррекции воспользуемся алгоритмом полифазного восстановления тактового сигнала.

Для восстановления времени, нам нужно отыскать наилучшее время для сэмплинга входящего сигнала, чтобы максимизировать SNR и минимизировать ISI.

Flow Graph на рисунке ниже иллюстрирует проблему ISI. В нём создается 4 символа-единицы, при этом, приемник и передатчик находятся в разных доменах:





Можно видеть, как из-за различий в тактовых сигналах между передатчиком и приёмником была допущена ощутимая погрешность. Заметим, что при таких выборках, собираемых в разные моменты времени, идеальный период выборки неизвестен, и любая сделанная выборка также будет включать ISI.

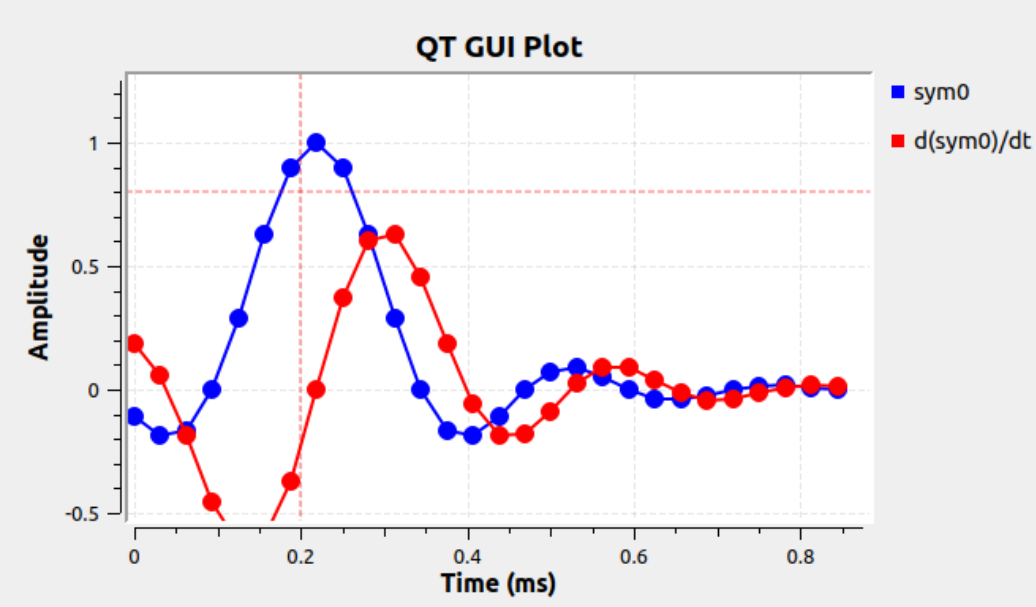
Наша задача состоит в том, чтобы синхронизировать тактовые сигналы передатчика и приемника, используя только информацию на приемнике из входящих выборок. Эта задача известна как временное восстановление.

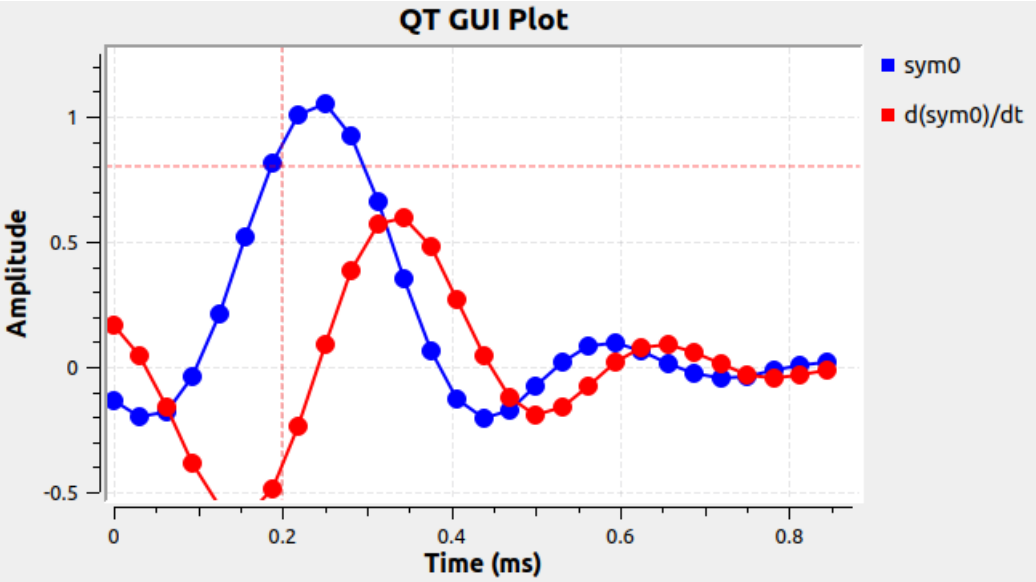
Глава 4

Временная синхронизация

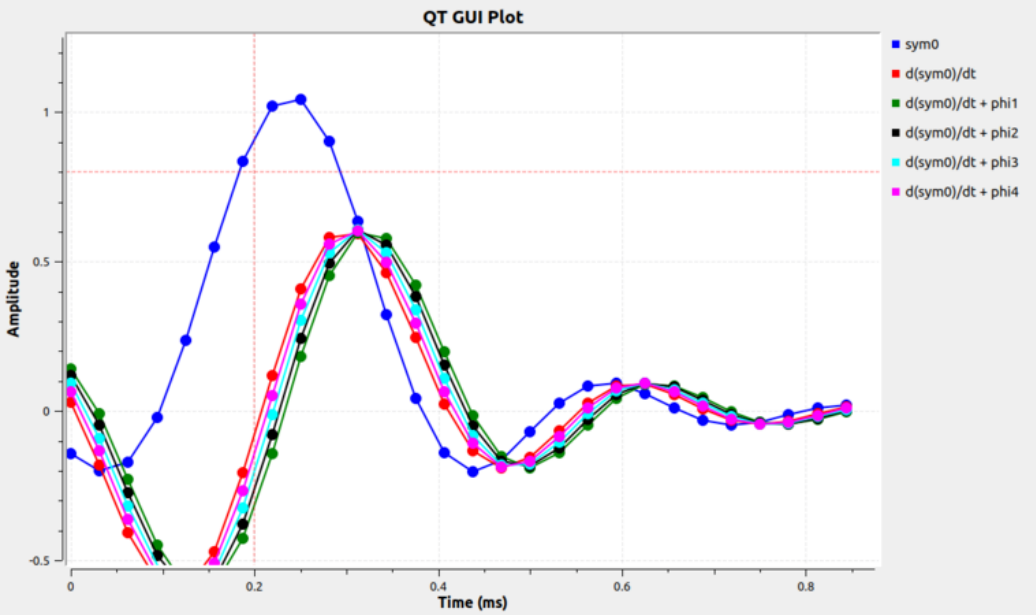
Существуют различные алгоритмы, которые мы можем использовать для восстановления тактового сигнала на приемнике, и почти все они включают в себя некоторый цикл управления с обратной связью. Те, которые не включают, обычно используют некоторое известное слово. Мы будем использовать метод polyphase filterbank clock recovery. Этот метод позволяет сделать сразу 3 вещи: выполнить временное восстановление; согласовать фильтр приемника и, тем самым, устранить ISI; понизить дискретизацию сигнала и произвести 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.





На самом деле, можно использовать серию фильтров с разными сдвигами по фазам - если их будет достаточно, то одна из этих фаз окажется правильной и даст желаемое значение. Рассмотрим симуляцию, которая строит 5 фильтров, что означает 5 различных фаз (пример symbol_differential_filter_phase.grc).



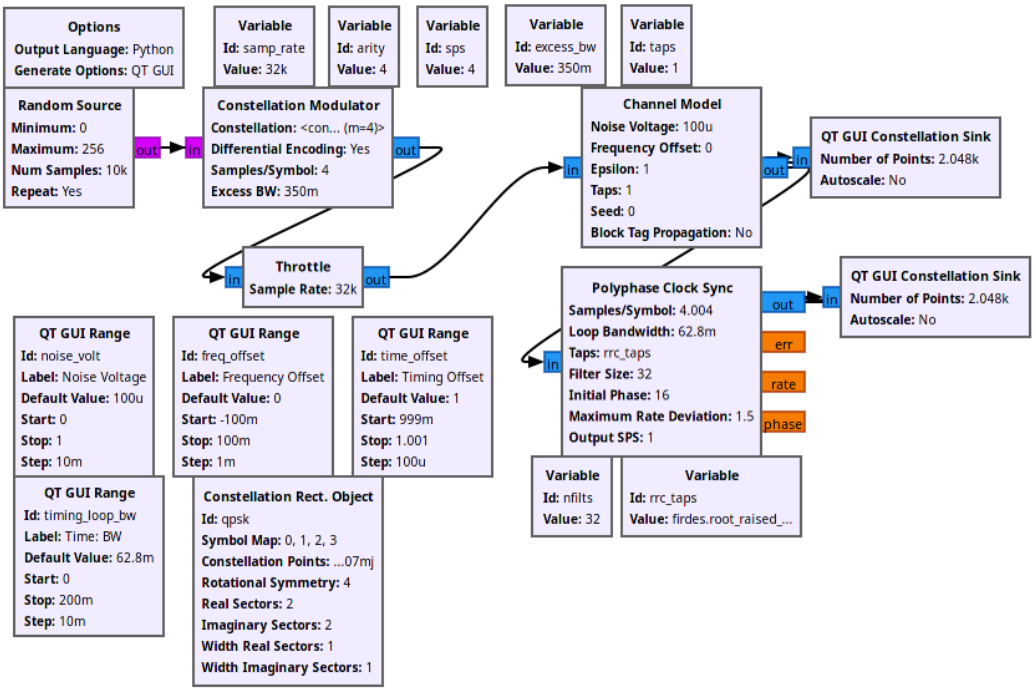
Видно, что сигнал, помеченный как d (sym0) / dt + phi3 в правильной точке выборки равен 0. Из этого можно сделать вывод, что наша идеальная точка выборки возникает при этом сдвиге фазы. Поэтому, если мы возьмем фильтр RRC нашего приемника и настроим его фазу на phi3 (которая равна 3 * 2pi / 5), то мы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

Однако, мы имеем дело лишь с моделью, 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.

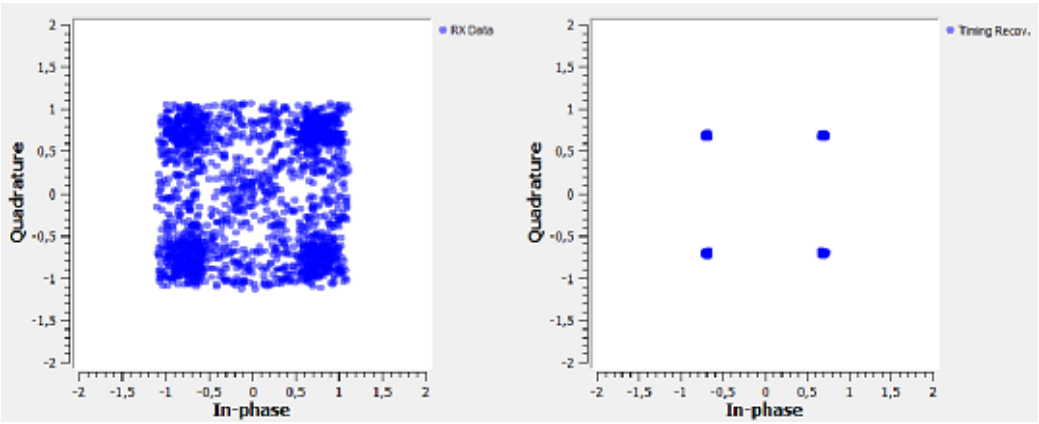
Глава 5

Использование блока синхронизации полифазного тактирующего сигнала в нашем приемнике

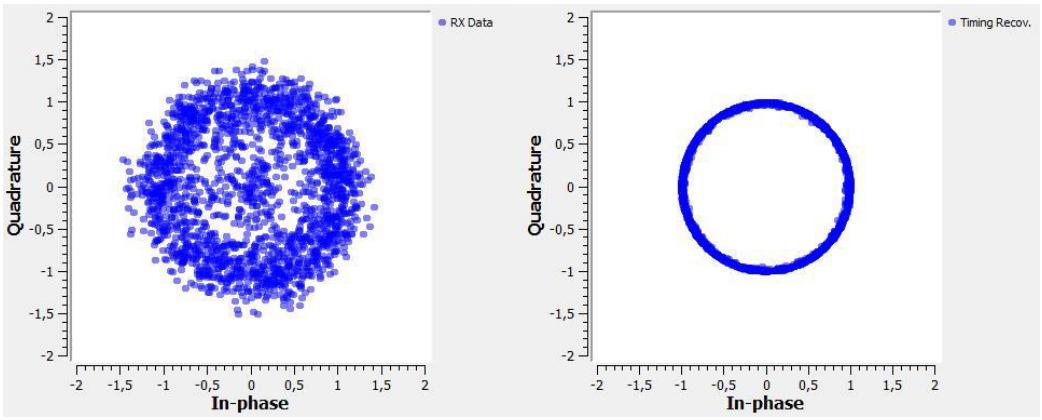
Применим рассмотренный блок в симуляции, настроив его на использование 32 фильтра по причинам, которые мы обсуждали ранее. Вход - число ожидаемых выборок на символ - всего лишь наше предположение, внутри блок будет адаптировать его, основываясь на скоростях входящего сигнала. Заметьте, однако, что мы 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.



Результат (полученный сигнал до и после):



При добавлении смещения частоты созвездие становится окружностью. Созвездие всё еще будет находится на единичной окружности, что означает, что тактовая синхронизация сохраняется, но блок не позволяет нам корректировать смещение частоты. Далее придётся придумать решение этой проблемы.


RX Data
Quadrature
In-phase
Timing Recov.
Quadrature
In-phase

Глава 6

Множество путей

На практике часто происходит так, что на приемник сигнал попадает с нескольких направлений сразу, что может приводить к искажениям. Проблему можно решить, используя механизм, похожий на стереоэквалайзер. С его помощью возможено изменить усиление определенных частот, чтобы подавить или усилить сигналы.

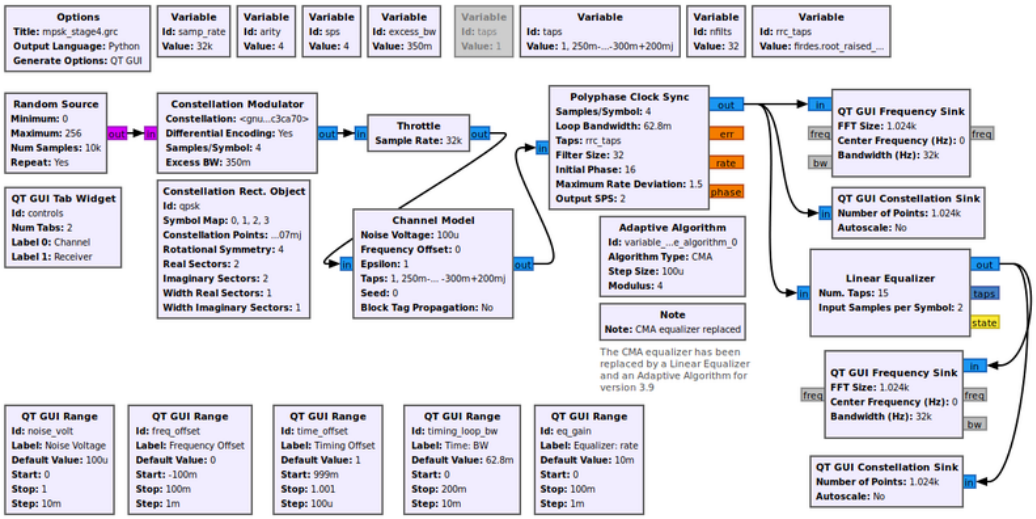
В качестве примера рассмотрим Multipath_sim.grc. Он создаёт модель канала, обеспечивая канал пятью частотными полосами для эквалайзера, четыре из которых можно контролировать. При значении 1 полоса будет пропускать частоты без помех. При значении 0 полоса будет давать глубокий ноль в спектре, который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

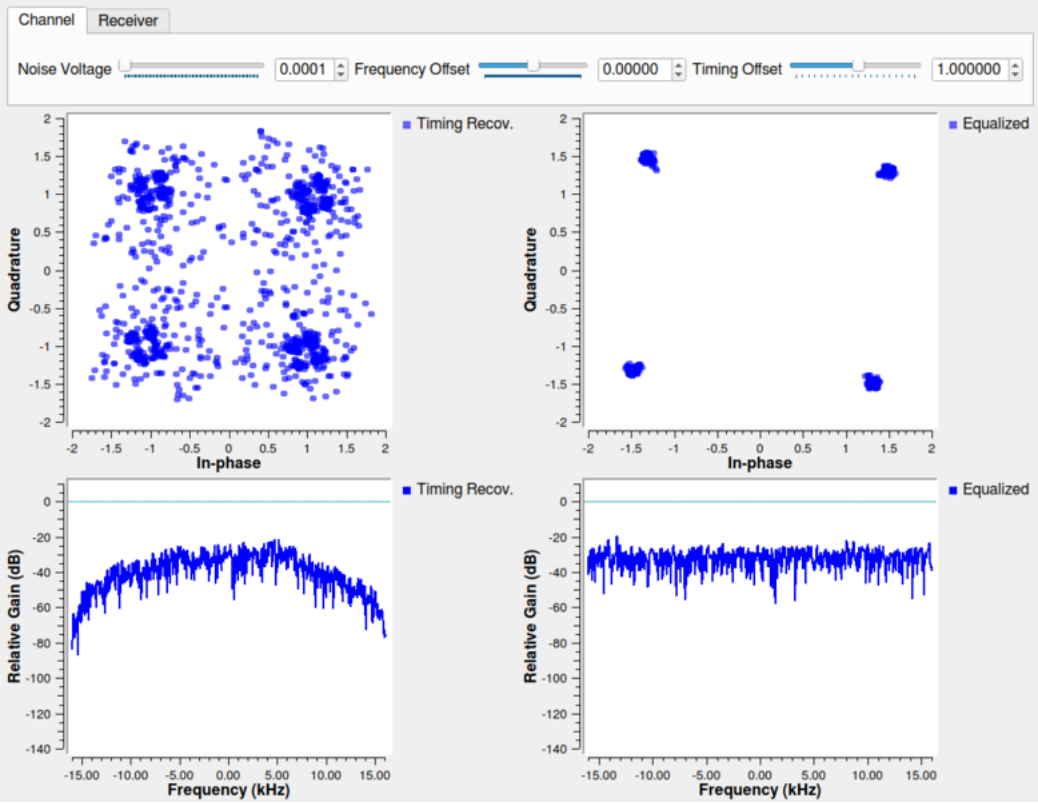
Хотя в этом примере мы явно контролируем частотную область, мы действительно можем создать эквалайзер, способный регулировать частотную характеристику принимаемого сигнала. На примере, канал многолучевого распространения создает некоторое искажение в сигнале. Задача эквалайзера - инвертировать это канал. По сути, мы хотим отменить искажение, вызванное каналом, чтобы выход эквалайзера был плоским. Наша задача - использовать правильный алгоритм эквалайзера и настроить параметры. Важным из них является количество частотных полос в эквалайзере. Чем больше полос, тем глубже контроль, но тем больше времени требуется для их вычисления.

Глава 7

Эквалайзеры

GNU Radio поставляется с двумя легко используемыми эквалайзерами, CMA и DD LMS. В данной работе будет использоваться второй. Возьмём пример mpsk_stage4.grc и заменим CMA на LMS-DD:





Эксперементируя с настройками, можно 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.

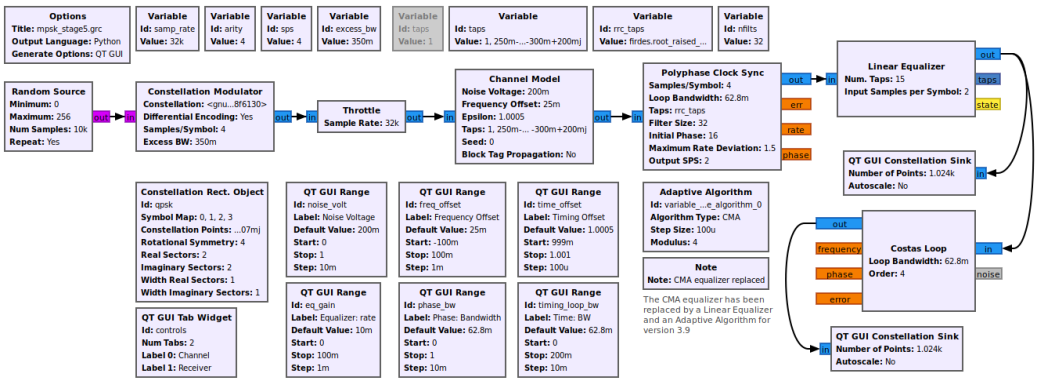
Глава 8

Фазовая частотная коррекция

После применения эквалайзера у нас все еще остается проблема со смещением в фазе и частоте. Эквалайзеры часто медленно адаптируются, так что из-за сдвига частот они могут быстро перестать успевать.

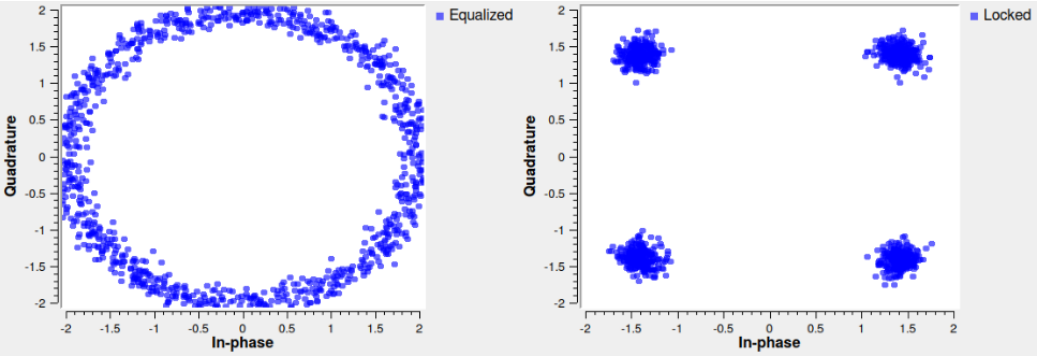
Можно отметить ещё две вещи. Во-первых, мы будем использовать цикл второго порядка, чтобы мы могли отслеживать как фазу, так и частоту, которая является производной фазы по времени. Во-вторых, тип восстановления, с которым мы здесь будем иметь дело, предполагает, что мы делаем точную коррекцию частоты.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

В этом задании воспользуемся циклом Костаса. Соответствующий блок может синхронизировать BPSK, QPSK и 8PSK.



Этот блок, как и все наши другие, использует цикл второго порядка, а потому и имеет соответствующий параметр, связанный с пропускной способностью. Ему также нужно знать степень PSK-модуляции (4 для QPSK).

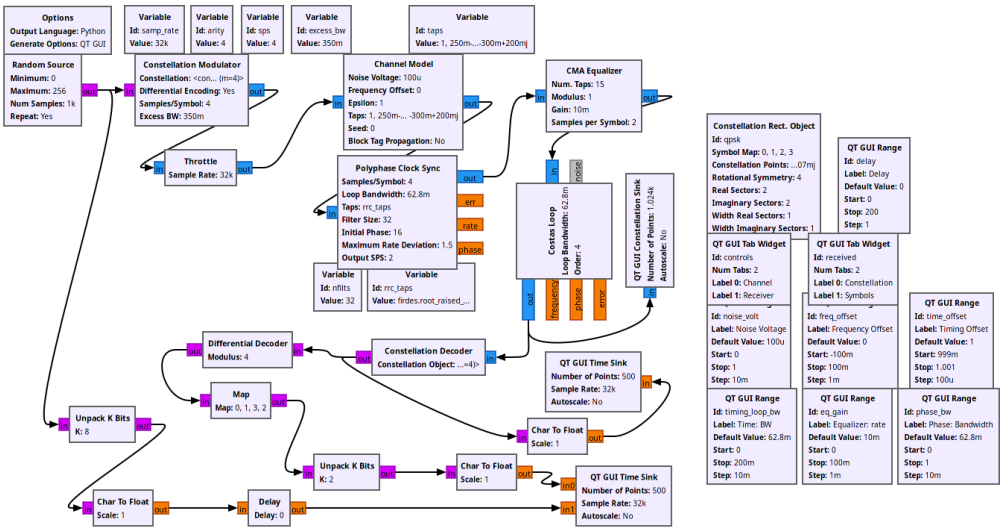
На рисунке ниже можно видеть, как после работы эквалайзера все символы расположены на единичной окружности, но из-за сдвига частот, они как бы размазаны по ней, а не соответствуют нашим 4 точкам. После цикла Костаса мы уже видим ровно 4 исходные точки, но некоторый шум тоже присутствует.



Глава 9

Декодирование

Осталась последняя часть - декодер. После цикла Костаса мы вставляем Constellation Decoder, но этого всё ещё недостаточно. На этом этапе мы получаем символы от 0 до 3, потому что это размер нашего алфавита в схеме QPSK, но у нас нет никакой уверенности, что они правильно соответствуют тем 0...3, что были закодированы изначально. Ранее мы использовали дифференциальное кодирование в Constellation Modulator - теперь мы его выключаем.



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. Задача решена. 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 создает упакованные биты, поэтому при помощи Unpack Bit мы должны их распаковать из 8 бит на байт в 1 бит на байт, затем явно типизировать их как float 0.0 и 1.0. Но напрямую сравнивать значения пока нельзя, потому что в приемнике есть еще много узлов, которые задерживают данные, поэтому нам нужен еще и блок Delay.