

Лабораторная работа 12

Луняк Николай

16 мая 2021 г.

Оглавление

1	Общее	5
1.1	Цели	5
1.2	Окружение	5
2	Передача QPSK-сигнала	10
3	Искажения в канале	13
4	Временная коррекция	15
4.1	Немного о блоке синхронизации многофазного тактового сигнала	17
4.2	Использование блока синхронизации полифазного тактирующего сигнала в нашем приемнике	19
5	Множество путей	21
6	Эквалайзеры	23
7	LMS-DD Эквалайзер	25
8	Подгонка фазы и частоты	27
9	Декодер	29
10	Выводы	31

Список иллюстраций

1.1	Проблема 1	6
1.2	Проблема 2	6
1.3	Что-то	7
1.4	Опять	7
1.5	Опять	8
1.6	Очередные бульканья GNU Radio	9
2.1	Flow Graph mpsk_rrc_rolloff	11
2.2	График mpsk_rrc_rolloff	11
2.3	Flow Graph mpsk_stage1	12
2.4	График mpsk_stage1	12
3.1	Flow Graph mpsk_stage2	13
3.2	График mpsk_stage2	14
4.1	Flow Graph symbol_sampling	15
4.2	График symbol_sampling	16
4.3	Flow Graph symbol_sampling_diff	16
4.4	График symbol_sampling_diff	17
4.5	Flow Graph symbol_differential_filter	17
4.6	График symbol_differential_filter	18
4.7	Flow Graph symbol_differential_filter_phases	18
4.8	График symbol_differential_filter_phases	19
4.9	Flow Graph mpsk_stage3	19
4.10	График mpsk_stage3	20
4.11	График mpsk_stage3 x2	20
5.1	Flow Graph multipath_sim	21
5.2	График multipath_sim	22
6.1	Flow Graph mpsk_stage4	23
6.2	График mpsk_stage4	24

7.1	Flow Graph mpsk_stage4_lms_dd	26
7.2	График mpsk_stage4_lms_dd	26
8.1	Flow Graph mpsk_stage5	28
8.2	График mpsk_stage5	28
9.1	Flow Graph mpsk_stage6	29
9.2	График mpsk_stage6	30

Листинги

1.1	Опять какая-то фигня	8
-----	--------------------------------	---

Глава 1

Общее

1.1 Цели

Суть данной работы заключается в том, чтобы:

- получить представление о проблемах искажения сигналов и «канальных эффектах»
- определять стадии, необходимые для восстановления сигналов:
 - временное восстановление
 - многопутевые каналы
 - коррекция фазы и частоты
 - декодирование символов и порядок бит

1.2 Окружение

Так как официально Windows не поддерживается, буду работать на Arch. Вроде бы достаточно только установить **gnuradio** и все... а нет, говорит:

```

luna_koly in ~
$ gnuradio-companion
Traceback (most recent call last):
  File "/usr/bin/gnuradio-companion", line 59, in check_gtk
    gi.require_version('Gtk', '3.0')
AttributeError: module 'gi' has no attribute 'require_version'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/bin/gnuradio-companion", line 100, in <module>
    check_gtk()
  File "/usr/bin/gnuradio-companion", line 67, in check_gtk
    die(err, "Failed to initialize GTK. If you are running over ssh, "
  File "/usr/bin/gnuradio-companion", line 41, in die
    gi.require_version('Gtk', '3.0')
AttributeError: module 'gi' has no attribute 'require_version'

```

Рис. 1.1: Проблема 1

В интернетах пишут, нужно установить `python-gobject`. Пробуем.

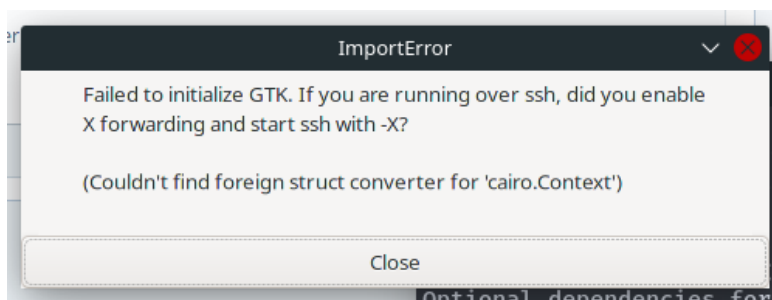


Рис. 1.2: Проблема 2

Классное ПО.

О, я не одинок: <https://bbs.archlinux.org/viewtopic.php?id=252150>. Что ж, перезапустим... неа, то же самое.

В консоли написано: «Gtk-Message: 14:22:04.664: Failed to load module "appmenu-gtk-module"». Установим. Все равно та же картинка, только теперь еще и в консоли нет никаких сообщений. Классное ПО.

Попробуем установить еще и `python-cairo`.

Ура, что-то открылось:

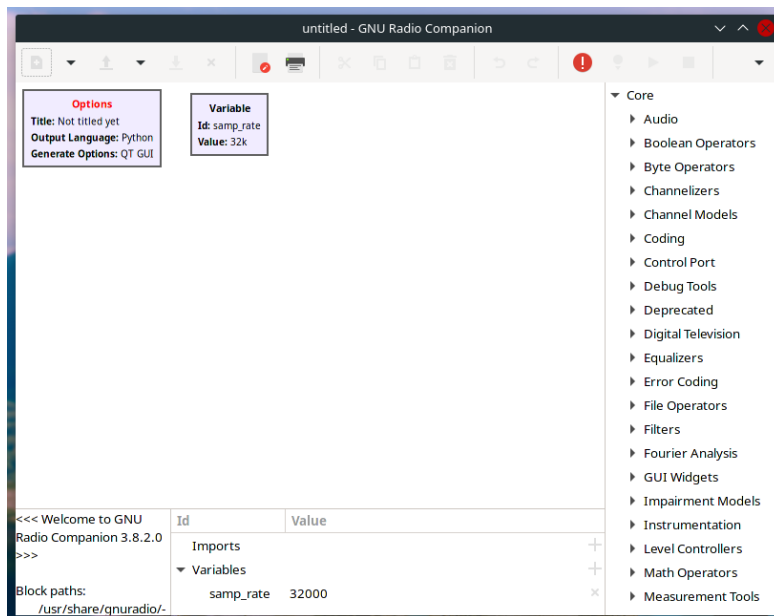


Рис. 1.3: Что-то

Попробовал сделать простейший граф, а получил это:

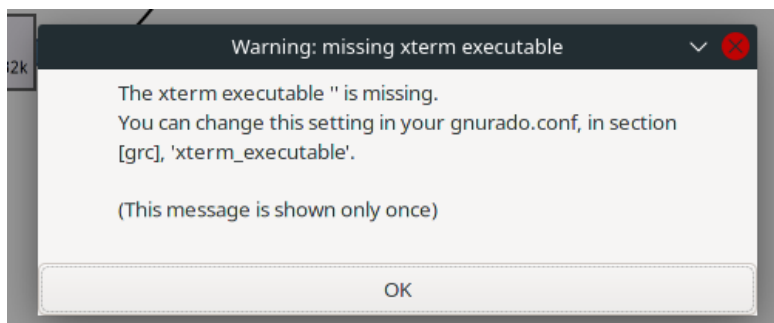


Рис. 1.4: Опять

Какого дьявола я обязан устанавливать `xterm`, я совершенно не понимаю, это абсурд.

Мало этого,


```
Generating: /home/luna_koly/Desktop/test_1.py

Executing: /usr/bin/python3 -u /home/luna_koly/Desktop/test_1.py

Traceback (most recent call last):
  File "/home/luna_koly/Desktop/test_1.py", line 23, in <module>
    from PyQt5 import Qt
ModuleNotFoundError: No module named 'PyQt5'

>>> Done (return code 1)
```

Рис. 1.5: Опять

Выбора нет, придется все это ставить. Если что, я не случайно пользуюсь Arch'ем, у меня места осталось 1ГБ, и если оно закончится, я буду *очень* расстроен.

```
1 Executing: /usr/bin/python3 -u /home/luna_koly/Desktop/test_1
  .py
2
3 Traceback (most recent call last):
4   File "/usr/lib/python3.9/site-packages/gnuradio/qtgui/
    __init__.py", line 32, in <module>
5     from .qtgui_swig import *
6   File "/usr/lib/python3.9/site-packages/gnuradio/qtgui/
    qtgui_swig.py", line 13, in <module>
7     from . import _qtgui_swig
8 ImportError: libqwt.so.6: cannot open shared object file: No
    such file or directory
9
10 During handling of the above exception, another exception
    occurred:
11
12 Traceback (most recent call last):
13   File "/home/luna_koly/Desktop/test_1.py", line 24, in <
    module>
14     from gnuradio import qtgui
15   File "/usr/lib/python3.9/site-packages/gnuradio/qtgui/
    __init__.py", line 36, in <module>
16     from .qtgui_swig import *
17   File "/usr/lib/python3.9/site-packages/gnuradio/qtgui/
    qtgui_swig.py", line 13, in <module>
18     from . import _qtgui_swig
19 ImportError: libqwt.so.6: cannot open shared object file: No
    such file or directory
20
```

21 >>> Done (return code 1)

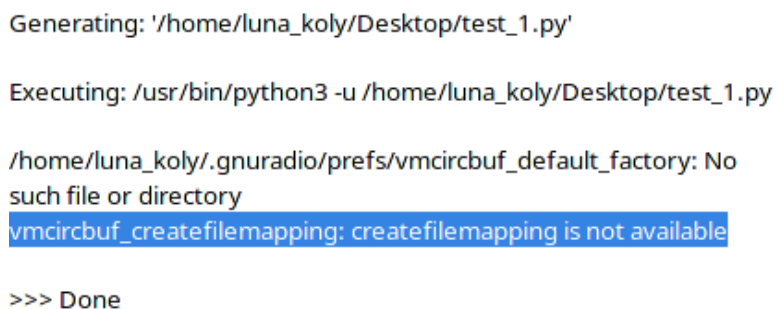
Листинг 1.1: Опять какая-то фигня

Это просто сюр какой-то, это ну невозможно просто.

Хорошо, сделаем так, как говорят тут: <https://github.com/gnuradio/gnuradio/issues/3117>, и установим еще и `qwt`.

Вот теперь я получил какой-то график, наконец.

Но нельзя просто так взять и запустить GNU Radio...



```
Generating: '/home/luna_koly/Desktop/test_1.py'

Executing: /usr/bin/python3 -u /home/luna_koly/Desktop/test_1.py

/home/luna_koly/.gnuradio/prefs/vmcircbuf_default_factory: No
such file or directory
vmcircbuf_createfilemapping: createfilemapping is not available

>>> Done
```

Рис. 1.6: Очередные бульканья GNU Radio

Ну и дьявол с ним!

Глава 2

Передача QPSK-сигнала

QPSK означает Quadrature Phase-Shift Keying[Вик], что есть фазовая модуляция, подразумевающая использование 4 точек constellation diagram (двумерной плоскости, на которой комплексным числам будут соответствовать наши символы).

Для простой передачи сигнала нам потребуется модулятор (Constellation Modulator Block) и Constellaton Rect. Object, в который отвечает за то, как кодируются символы. Также нам потребуется генератор байтовых значений.

Количество сэмплов на символ я оставлю тем же, что и в tutorialе (там этот выбор обосновывается наглядностью).

Поэкспериментируем с выбором разной избыточной пропускной способности. Как я понимаю, этот параметр влияет на крутизну краев фильтра.

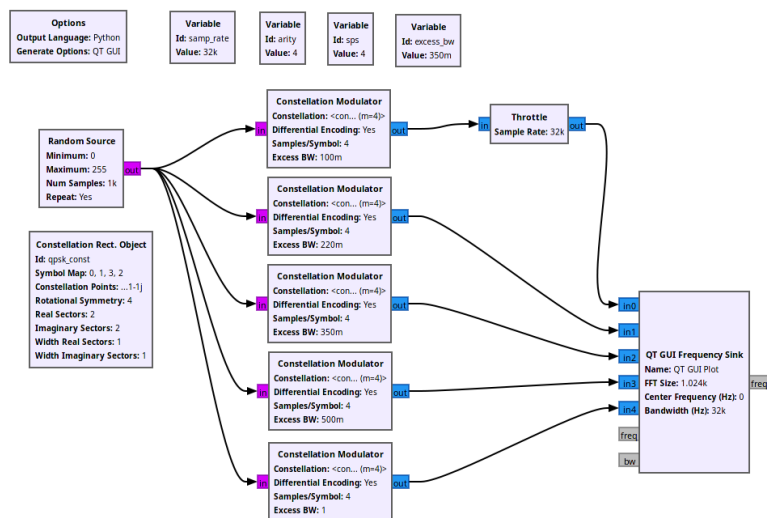


Рис. 2.1: Flow Graph mpsk_rrc_rolloff

И вот такой результат мы наблюдаем.

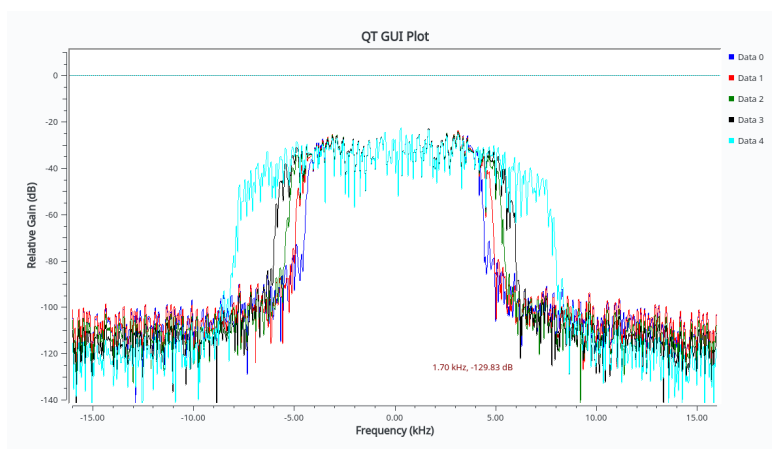


Рис. 2.2: График mpsk_rrc_rolloff

Чем больше excess bandwidth, тем более пологие края.

Следующий Flow Graph осуществляет передачу QPSK-«созвездия» (честно, не знаю, насколько корректны мои попытки перевести термины на русский, поэтому дальше я буду использовать оба языка вперемешку, чтобы минимизировать путаницу). Он показывает переданный сигнал, полученный на приемнике сигнал во времени и частотах и созвездие.

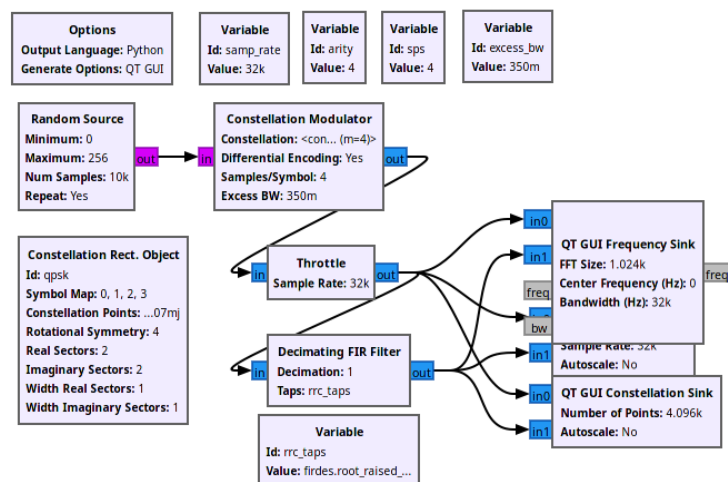


Рис. 2.3: Flow Graph mpsk_stage1

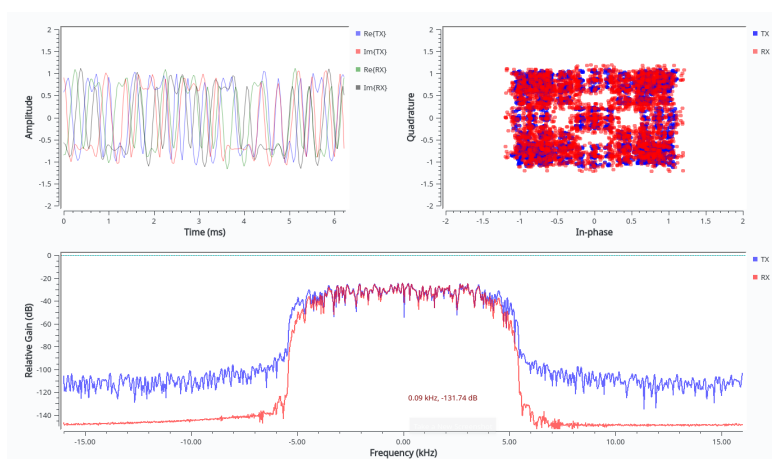


Рис. 2.4: График mpsk_stage1

На созвездии можно видеть эффект увеличения частоты дискретизации (4 сэмпла на символ) и процесс фильтрации. В нашем случае RRC-фильтр специально добавляет помехи с самим собой («межсимвольные помехи») ISI. Это плохо для принятого сигнала, потому что тогда символы размываются.

Чтобы избавиться от ISI мы на приемной стороне используем еще один RRC-фильтр. При помощи свертки мы получаем импульсы приподнятого косинуса с минимизированным ISI.

Глава 3

Искажения в канале

До этого мы рассматривали саму передачу сигнала, а теперь мы добавим искажения в канале переноса. Модель канала можно добавить при помощи **Channel Model**.

Канал позволит проиллюстрировать несколько проблем. Одна из них - шум. Мощность шума регулируется при помощи напряжения.

Другая проблема - разные домены тактовых сигналов в приемнике и передатчике. Мало того, что может быть сдвиг, так еще и частоты не могут быть идеально точными, и потому содержат небольшую разницу.

Третья проблема заключена в идеальной точке сэмпирования, которая не известна принимающей стороне.

Flow Graph ниже позволяет манипулировать этими эффектами и наблюдать их влияние на сигнал:

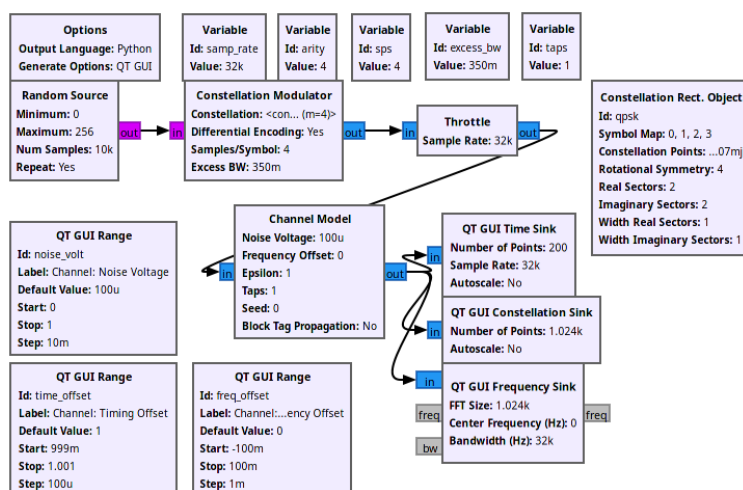


Рис. 3.1: Flow Graph mpsk_stage2

И вот результат.

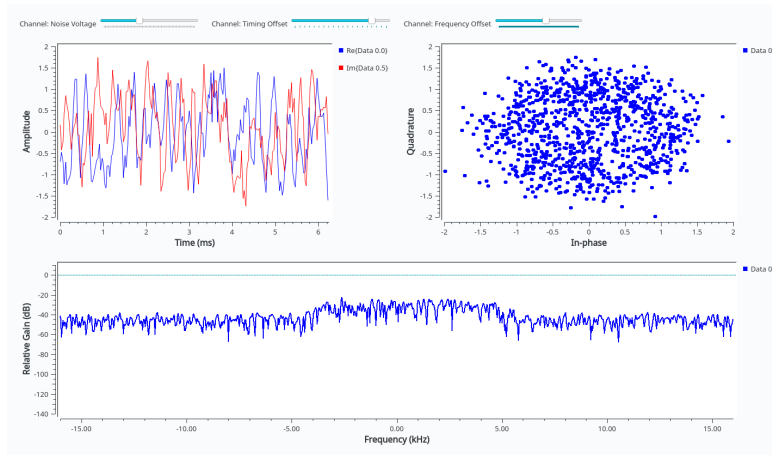


Рис. 3.2: График mpsk_stage2

Видимо, насколько перемешанное получилось облако. Будем исправлять.

Глава 4

Временная коррекция

Вообще, есть много разных алгоритмов, некоторые даже могут произвести восстановление нескольких эффектов сразу, но мы воспользуемся алгоритмом полифазного восстановления тактового сигнала.

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

Следующий Flow Graph иллюстрирует проблему ISI, мы тут создаем 4 символа-единицы подряд, а затем фильтруем их:

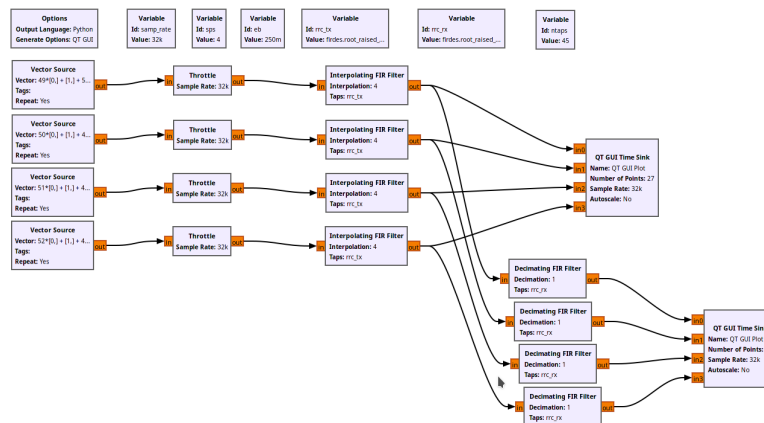


Рис. 4.1: Flow Graph symbol_sampling

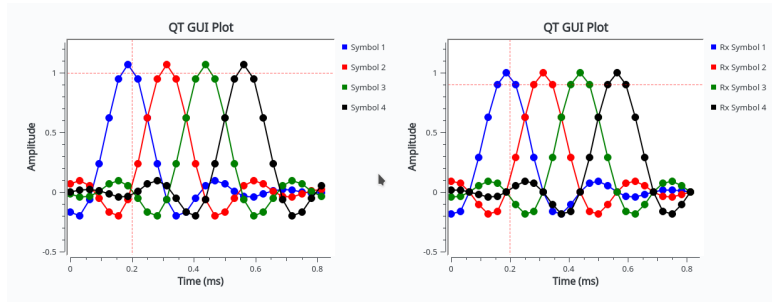


Рис. 4.2: График symbol_sampling

Теперь посмотрим на эффект разных доменов тактовых сигналов на передатчике и приемнике. Разница во времени специально завышена для наглядности.

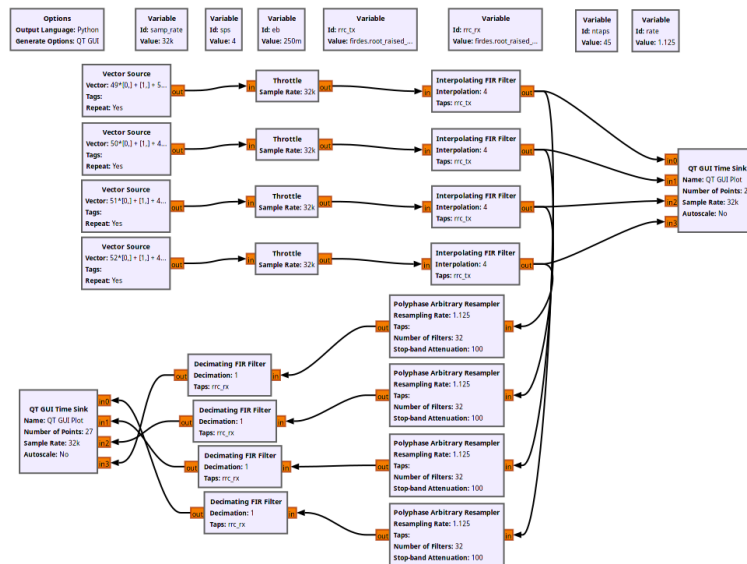


Рис. 4.3: Flow Graph symbol_sampling_diff

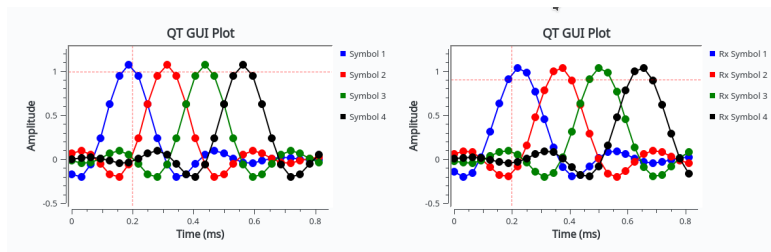


Рис. 4.4: График `symbol_sampling_diff`

Теперь нам надо синхронизировать все это дело.

4.1 Немного о блоке синхронизации многофазного тактового сигнала

Тут тоже много разных алгоритмов, но почти все они основаны на обратной связи (остальные получают в свое распоряжение дополнительную информацию). Мы сопользуея техникой восстановления `polyphase filterbank` (кстати, она `deprecated` в GNU Radio 3.9). Она послужит для нескольких целей. Во-первых, она решит проблему с тактовыми доменами. Во-вторых, уменьшит ISI. В-третьих, понизит частоту дискретизации сигнала и будет производить по 1 сэмплу на символ.

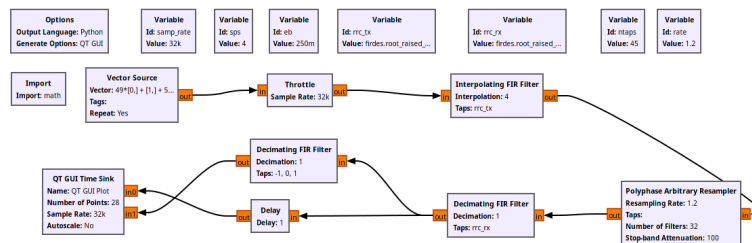


Рис. 4.5: Flow Graph `symbol_differential_filter`

«Идеальный» случай, где все хорошо и сдвига тактирующих сигналов нет должен был бы нам засечь значение в пике и нарисовать точку на графике производной при пересечении нуля, но из-за временного сдвига мы теперь ее не видим:

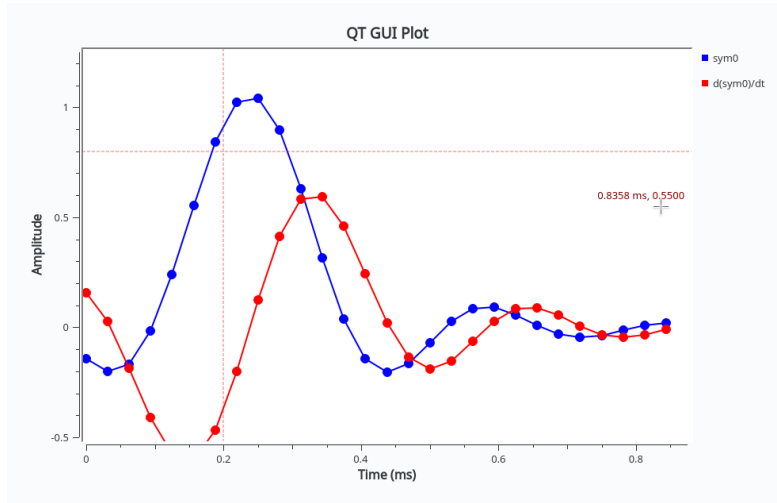


Рис. 4.6: График `symbol_differential_filter`

Можно попробовать использовать серию фильтров с разными сдвигами по фазам. Если их *достаточно* много, можно надеяться, что какой-нибудь засечет правильную

Тут пришлось вносить правки, потому что исходный Flow Graph был не доделан.

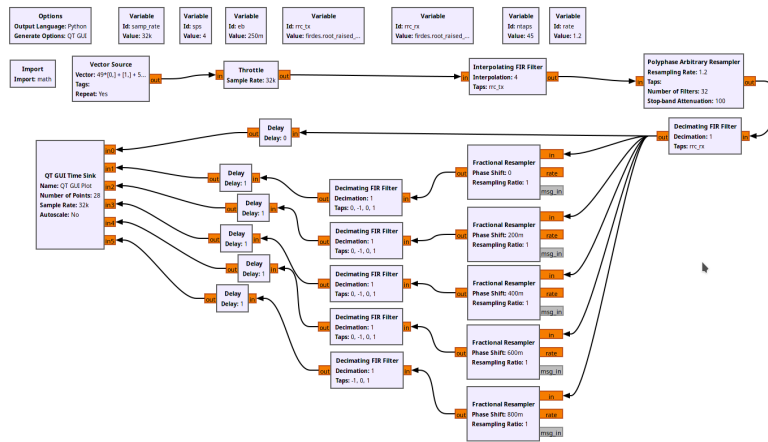


Рис. 4.7: Flow Graph `symbol_differential_filter_phases`

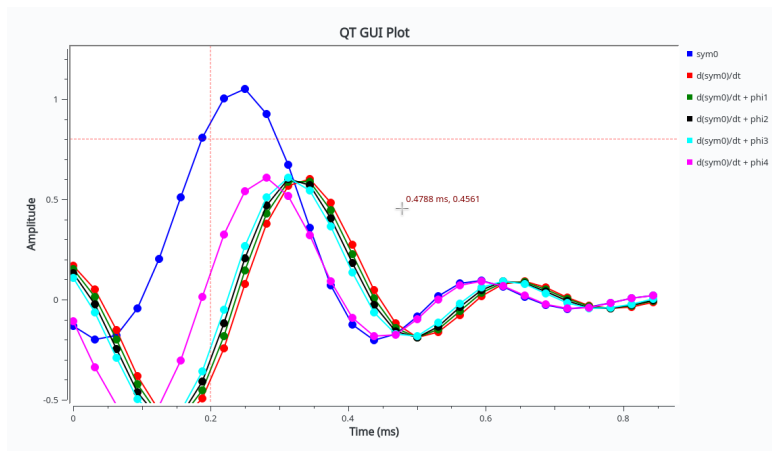


Рис. 4.8: График `symbol_differential_filter_phases`

4.2 Использование блока синхронизации полифазного тактирующего сигнала в нашем приемнике

Настроим блок на использование 32 фильтров:

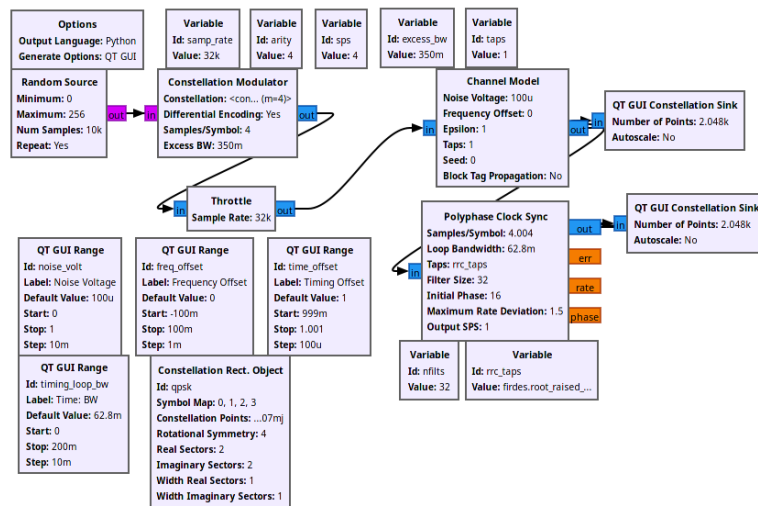


Рис. 4.9: Flow Graph `mpsk_stage3`

И вот такой результат мы наблюдаем.

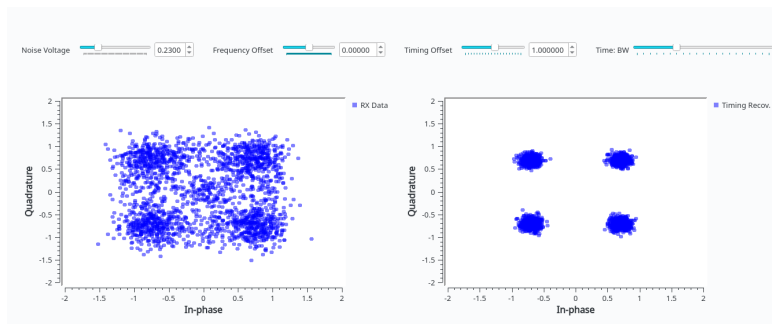


Рис. 4.10: График `mpsk_stage3`

Вот они слева направо: полученный сигнал «до» восстановления времени и «после».

Если подвигать ползунок `Timing Offset`, то разница особо не заметна, а вот `Frequency Offset` сразу размывает точки по всей окружности. Причем окружность еще и утолщается, что говорит и об ошибках в интерпретации абсолютных значений.

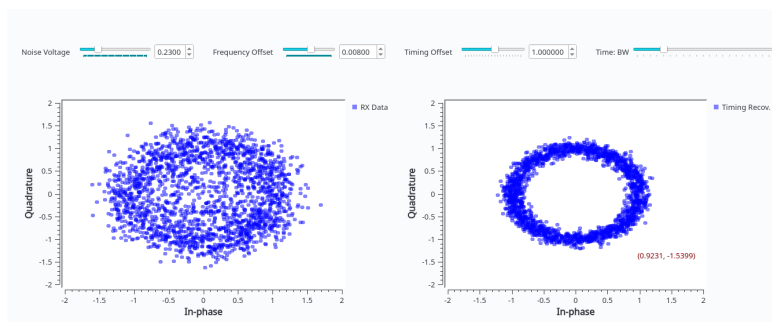


Рис. 4.11: График `mpsk_stage3 x2`

Глава 5

Множество путей

Тут смысл в том, что обычно наш сигнал поступает на приемник с множества направлений сразу, что может приводить к искажениям.

Чтобы справиться с искажениями, можно воспользоваться приемом, схожим со стерео-эквалайзерами. Как это выглядит в пространстве частот, можно видеть при помощи схемы ниже:

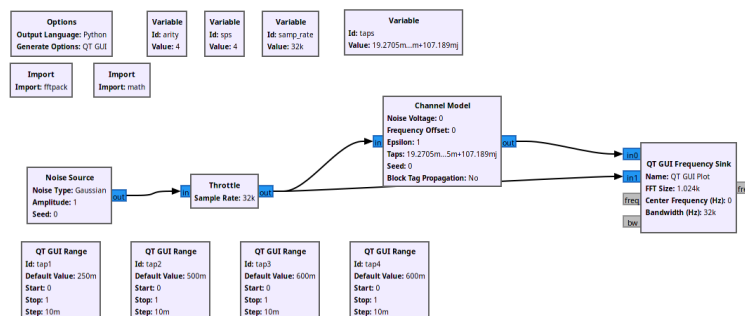


Рис. 5.1: Flow Graph multipath_sim

В этой симуляции у нас создается канал с 4 управляемыми параметрами, если установить которые в 1, то соответствующие частоты смогут пройти без помех, а при 0 они будут производить «глубокий ноль» в спектре, что будет влиять так же и на соседние частоты.

Цель в том, чтобы сделать эквалайзер, после работы которого искаженный принятый сигнал бы отражался снова как горизонтальная прямая.

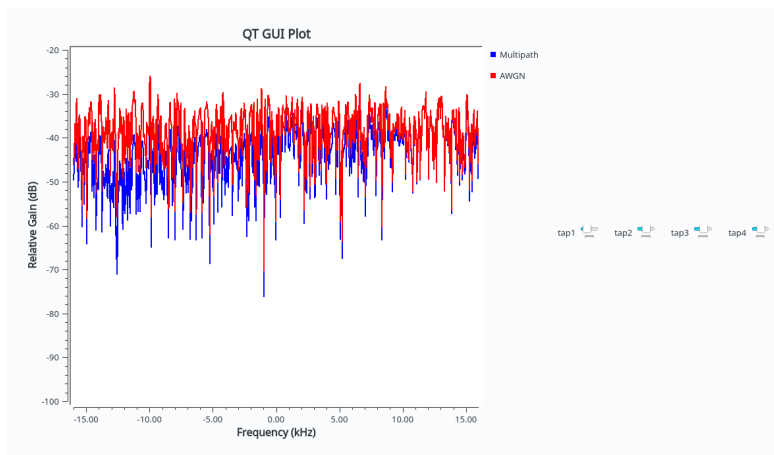


Рис. 5.2: График `multipath_sim`

выглядит некрасиво, но эквалайзеру удастся успешно инвертировать и избавиться от канала.

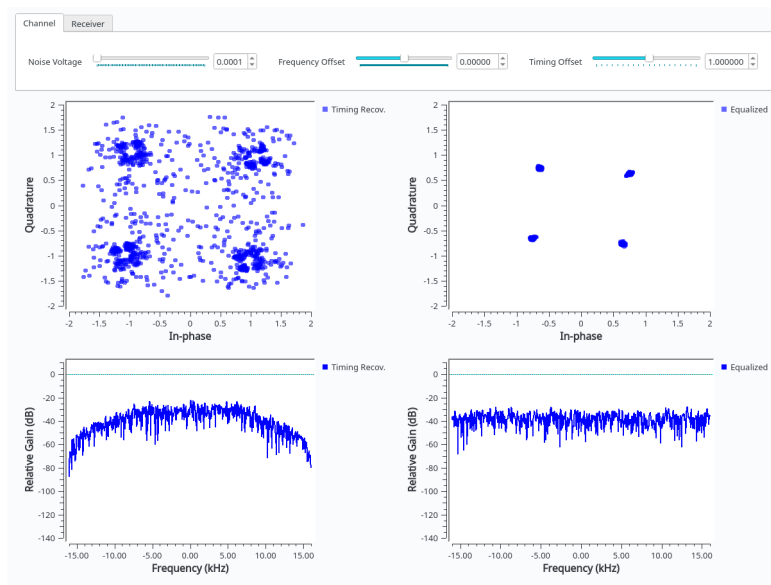


Рис. 6.2: График mpsk_stage4

Глава 7

LMS-DD Эквалайзер

Хорошим упражнением было бы попробовать применить Least Mean Squared Decision-Directed эквалайзер. Многие параметры схожи, но есть одно важное отличие от СМА: теперь нам нужна дополнительная информация о принятом сигнале. Эквалайзеру нужно знать точки созвездия.

Этот эквалайзер хорошо подходит для сигналов, которые не подходят под требование постоянной амплитуды (как в СМА), поэтому его можно использовать и для QAM-модуляции. С другой стороны, если все достаточно плохо с SNR, принимаемые эквалайзером решения могут быть некорректными, что заметно снизит производительность приемника. Да и сам блок тоже вычислительно более сложный. Впрочем, если сигнал хороший, то этот эквалайзер может выдать заметно более хороший результат (у него же есть информация о сигнале). Часто сначала используют слепой эквалайзер, когда еще ничего неясно с сигналом, а потом можно переключиться на LMS-DD, но мы не будем так все усложнять.

Заменим СМА на LMS-DD:

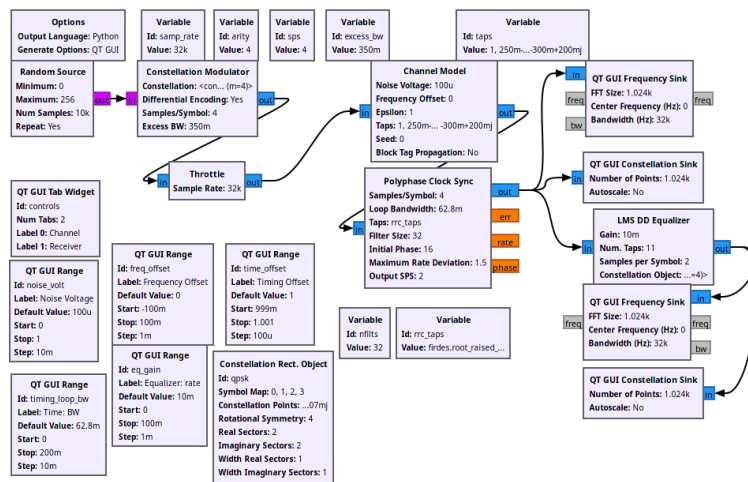


Рис. 7.1: Flow Graph mpsk_stage4_lms_dd

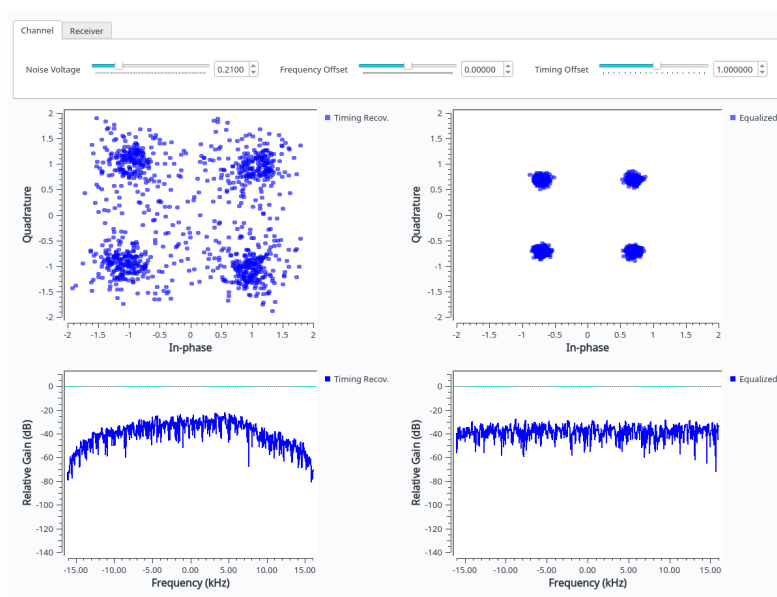


Рис. 7.2: График mpsk_stage4_lms_dd

Глава 8

Подгонка фазы и частоты

После применения эквалайзера у нас все еще остается проблема со смещением в фазе и частоте. Эквалайзеры часто небыстро адаптируются, так что из-за сдвига частот они могут быстро перестать успевать. К тому же, СМА-эквалайзер ничего не знает о точках созвездия, так что могут возникать проблемы и с тем, какую фазу он начнет использовать как точку отсчета.

Есть два нюанса. Во-первых, мы будем использовать цикл второго порядка, чтобы следить за фазой и частотой с течением времени. Во-вторых, восстановление подразумевает, что мы будем делать *достаточно* хорошую коррекцию частот, поэтому нам нужно будет убедиться, что мы достаточно близки к идеальной частоте. Иначе наш цикл не будет сходиться.

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

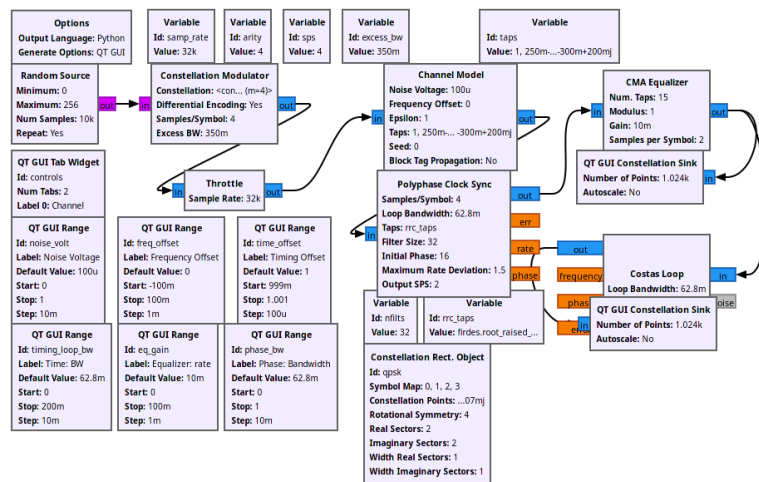


Рис. 8.1: Flow Graph mpsk_stage5

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). 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.

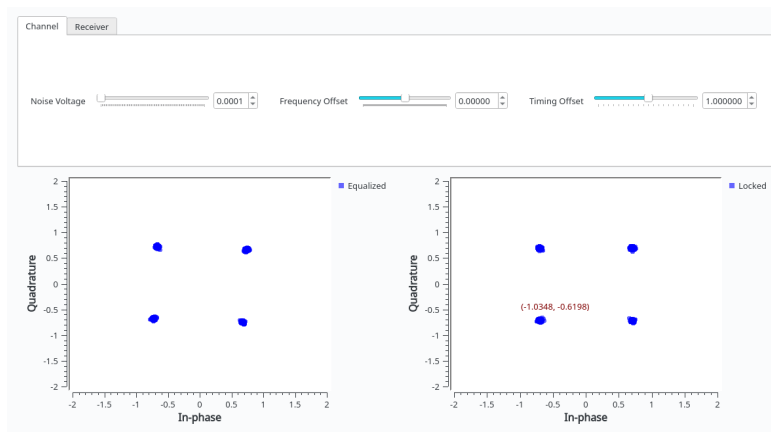


Рис. 8.2: График mpsk_stage5

Глава 9

Декодер

Теперь, когда сложная часть пройдена, можно делать декодер. После цикла Костаса мы вставляем еще **Constellation Decoder**, но это еще не все. На этом этапе мы видим символы 0..3, потому что на это рассчитан наш алфавит, но у нас нет никакой уверенности, что эти 0..3 правильно соответствуют тем 0..3, что мы передавали изначально. Нам удавалось обходить этот вопрос, потому что мы использовали дифференциальное кодирование в **Constellation Modulator**'е. Выключим теперь это.

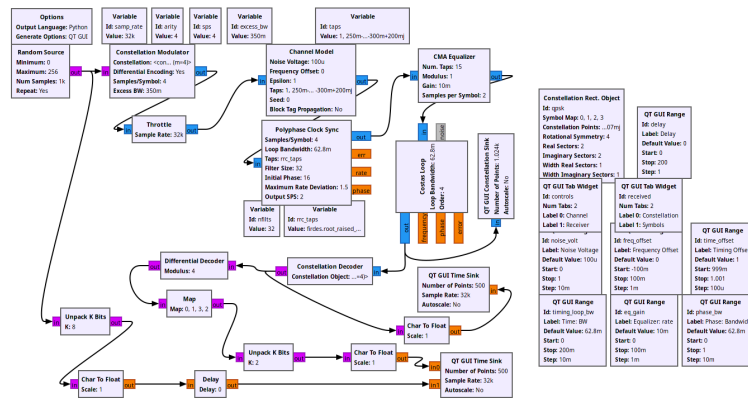


Рис. 9.1: Flow Graph mpsk_stage6

Flow Graph использует блок **Differential Decoder**, чтобы перевести дифференциально-закодированные символы обратно в исходные при помощи сдвигов фаз, а не абсолютной фазы. Но даже так наши символы еще не совсем верные. Во время синхронизации математика и физика была на нашей стороне, а теперь нам надо интерпретировать символы, основываясь на том, что кто-то оскзал, какими они были. То есть, нам

просто надо знать, как они соотносятся. Для этого мы воспользуемся блоками Map и Unpack Bit.

Чтобы убедиться, что теперь мы действительно получаем *тот самый поток данных*, мы просто сравним, что было в начале, с тем, что получили (ведь это симуляция, мы имеем доступ к любой информации). Передатчик создает упакованные биты, поэтому при помощи Unpack Bit мы должны их распаковать из 8 бит на байт в 1 бит на байт, затем превратить их в float 0.0 и 1.0, потому что Time Sink умеет только float и complex принимать. Но напрямую сравнивать значения пока нельзя, потому что в приемнике есть еще много узлов, которые задерживают данные, поэтому нам нужен еще и блок Delay.

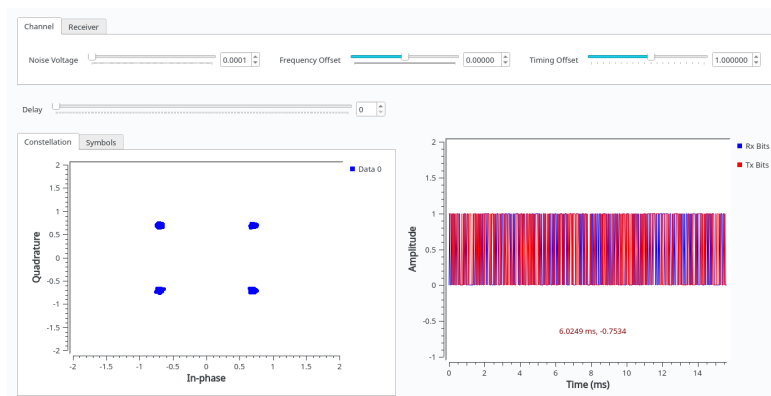


Рис. 9.2: График mpsk_stage6

Глава 10

Выводы

В процессе выполнения работы мы разобрались с тем, как произвести установку и настройку GNU Radio, а также рассмотрели множество явлений, связанных с передачей QPSK-сигнала. Симулируя настоящие условия передачи данных, мы столкнулись с рядом возможных трудностей, а также рассмотрели и пути их решения, что поможет нам в будущем при организации настоящих каналов передач.

Список литературы

[Вик] Википедия. *QPSK*. URL: [https://en.wikipedia.org/wiki/Phase-shift_keying#Quadrature_phase-shift_keying_\(QPSK\)](https://en.wikipedia.org/wiki/Phase-shift_keying#Quadrature_phase-shift_keying_(QPSK)).