**Mercury Quick Manual**

**Mercury Quick Manual**

**Structure, API Functions and Configuration Options**

**Fadi Jerji, Ph.D.**

Quick Manual for Mercury version 0.2 issue 1. Copyright (C) 2022-2024 Fadi Jerji Author: Fadi Jerji

Email: fadi.jerji@ <gmail.com, caisresearch.com, ieee.org>

ORCID: 0000-0002-2076-5831

**Contents**

| **1 Introduction** | |  | **1** |
| --- | --- | --- | --- |
| 1.0.1 | What is Mercury . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 1 |
| 1.0.2 | History . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 1 |
| 1.0.3 | Older versions of Mercury . . . . . . . . . . . . . . . . . . . . . . . | | 1 |
| 1.0.4 | The current version of Mercury (0.2) . . . . . . . . . . . . . . . . . | | 2 |
| 1.1 Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | | 3 |
| 1.1.1 | The physical layer . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 3 |
| 1.1.2 | The datalink layer . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 4 |
|  | 1.1.2.1 | ARQ ............................. | 4 |
|  | 1.1.2.2 | Gearshift . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
|  | 1.1.2.3 | API.............................. | 5 |

| **2 First steps** |  |  | **7** |
| --- | --- | --- | --- |
| 2.0.1 | Get Mercury on Linux . . . . . . . . . . . . . . . . . . . . . . . . . | | 7 |
|  | 2.0.1.1 | Prerequisites . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
|  |  | ALSA Sound: . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
|  |  | GNUPLOT: .......................... | 7 |
|  |  | Documentation: . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
|  | 2.0.1.2 Mercury source code . . . . . . . . . . . . . . . . . . . . . | | 8 |
| 2.0.2 | Modes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 8 |
| 2.0.3 | Testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 9 |
|  | 2.0.3.1 | Physical layer . . . . . . . . . . . . . . . . . . . . . . . . | 9 |
|  | 2.0.3.2 | Datalink layer . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
| 2.0.4 | Configurations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 16 |
|  | 2.0.4.1 | Physical layer . . . . . . . . . . . . . . . . . . . . . . . . | 16 |

4

2.0.4.2 Datalink layer

........................

21

**3** **License**

**23**

**GLOSSARY**

**AGC** Automatic Gain Controller

**API** Application Programming Interface

**ARQ** Automatic Repeat Request

**AWGN** Additive White Gaussian Noise

1. Base Band

**BER** Bit Error Rate

**BP** Band Pass

**CFO** Carrier Frequency Offset

**CRC** Cyclic Redundancy Check

**DSP** Digital Signal Processing

**FEC** Forward Error Correction

**FFT** Fast Fourier Transfer

**FIFO** First In First Out

**FIR** Finite Impulse Response

**GBF** Gradient Bit-Flipping Algorithm

**GI** Guard Interval

**GRC** GNU Radio Companion

**HPF** High Pass Filter

**LDPC** Low-Density Parity-Check

**LPF** Low Pass Filter

**LS** Least Square

**MER** Modulation Error Rate

**OFDM** Orthogonal Frequency-Division Multiplexing

6

**P2P** Peer to Peer

**PAPR** Peak-to-Average Power Ratio

**PB** Pass Band

**PSK** Phase Shift Keying

**PTT** Push To Talk

**SNR** Signal to Noise Ratio

**SPA** Sum-Product Algorithm

**ZF** Zero Forcing

7

**Chapter 1**

**Introduction**

**1.0.1 What is Mercury**

Mercury is a configurable open-source software-defined modem. Mercury is written fully in C++ for performance.

While the current version of Mercury is developed and tested on Linux-based operating systems, it doesn’t rely on any non-C++-standard library, thus it can be easily ported to other operating systems.

**1.0.2 History**

**1.0.3 Older versions of Mercury**

The development of Mercury started on January 2022 and the first pre-release was version 0.1 on December 2022 containing 9.5K lines of code. The Mercury 0.1 featured a physical layer with Low-Density Parity-Check (LDPC) Forward Error Correction (FEC), an Orthogonal Frequency-Division Multiplexing (OFDM), a Zero Forcing (ZF) channel estimator, Finite Impulse Response (FIR) filters, a Dx Yx pilot framing, a Schmidl&Cox-based Guard Interval (GI)-based time synchronization, a Carrier Frequency Offset (CFO) compensator, a Base Band (BB) ad Pass Band (PB) simulation, and a Dummy datalink layer with a TCP/IP interface. The Mercury 0.1 could handle Signal to Noise Ratio (SNR) of 5 dB.

1

Mercury 0.1.3 was released on July 2023 containing 18K lines of code. This pre-release introduced a datalink layer with an Automatic Repeat Request (ARQ) Peer to Peer (P2P) a new TCP/IP Application Programming Interface (API), a Gearshift auto-negotiation to auto-select the modulation and FEC, an uplink and downlink SNR measurement, a low overhead structured data block and data batches division to avoid retransmission.

Mercury 0.1.5 was released on January 2024 containing 25K lines of code. This pre-release introduced several enhancements to the physical layer including a Sum-Product Algorithm (SPA) LDPC decoder, a configurable preamble, a new Schmidl&Cox-based preamble-based time synchronization, a preamble-based CFO compensation, an enhanced ZF channel estimator, a Peak-to-Average Power Ratio (PAPR) reduction stage, a Band Pass (BP) transmission FIR filter, an enhanced reception logic with an increased reception buffer, a time and frequency interleavers and an Automatic Gain Controller (AGC) stage.

Mercury 0.1.5 of Mercury was successfully tested using Sbitx radios using Raspberry Pi 4 microcomputers. With a transmission power of 20 Watts, over a 70 km distance, a received SNR= -3 dB using TX\_test, RX\_test, and Config\_0.

**1.0.4 The current version of Mercury (0.2)**

Mercury 0.2 was released on June 2024 containing 47.7K lines of code. This pre-release brought a new Least Square (LS) channel estimator with a configurable estimation win-dow, an enhanced Time and Frequency synchronization for low SNR values, an enhanced TX and RX filtering with separate filters for time synchronization and data messages, an enhanced TX and RX mechanisms for higher robustness and computing eﬀiciency, new Time and Frequency interleavers, new LDPC codes (1/16 to 14/16) optimized for multipath channel and an outer Cyclic Redundancy Check (CRC) code, an energy dis-persal for power amplification eﬀiciency, a pre-equalization to combat filters and Digital Signal Processing (DSP) imperfections, a PAPR and Modulation Error Rate (MER) mea-surements, two pilot distribution modes for different channels and bitrate requirements with further optimized parameters such as number of symbols, number of carriers, and synchronization symbols. Dynamic partial configuration of the physical layer for comput-ing performance enhancement, an enhanced API of the physical layer to allow for byte or bit transfer, as separate Data and Acknowledge message robustness configuration, a

2

new Ladder-based Gearshift mode for the datalink layer for low SNR, 17 new robustness modes for the ARQ/ Gearshift, and several bug fixes and performance enhancements.

**1.1 Structure**

The mercury software-defined modem is composed of two main layers, the physical layer, and the datalink layer.

**1.1.1 The physical layer**

The physical layer features an OFDM modulator/demodulator with an LDPC error correction code encoder/decoder with an embedded Additive white an Additive White Gaussian Noise (AWGN) channel simulator.

The physical layer is the part responsible for data protection via the use of LDPC codes and information mapping on the electromagnetic spectrum via modulation using OFDM and other digital signal processing methods.

The physical layer provides a connection-less transmission. In other words, the trans-mitter doesn’t have any information regarding the receiver status, the channel status, or packets received/lost. This task is left to the upper layers.

The physical layer can be interfaced from the upper end and the lower end. The lower end interfaces with the hardware via the sound card using the Alsa driver. The speaker and microphone can be connected to an off-the-shelf radio transceiver to transmit the signal over an electromagnetic channel.

The upper layer can interface with the physical layer via three main connection points, send(), receive(), and load\_configuration(). In addition, function get\_configuration(SNR) allows the physical layer to inform the upper layer of the appropriate configuration for a specific SNR.

3

**1.1.2 The datalink layer**

**1.1.2.1 ARQ**

The datalink can be in one of two roles, a commander or a responder. The commander is the side responsible for controlling the message flux. The datalink layer provides a connection based using an ARQ via retransmission and acknowledgment mechanisms. This is to guarantee the reception of the information even in cases of message loss. In the case of an ACK message lost, a request to resend the last ACK message can be sent.

The datalink layer packs the messages in batches to avoid a fast TX/RX switch. Nevertheless, the batch size is configurable and can be set to one. Several batches form a block. Writing to the TX data buffer and reading from the RX\_data\_sync\_buffer are done in blocks. The block size is configurable as well.

**1.1.2.2 Gearshift**

The datalink later exchanges channel state information to negotiate the highest data rate possible for the downlink channel, and in case of a change, adapts to the new channel condition. This ”Gearshift” is done via a set\_config message that a commander can send at any time and a recovery mechanism in case of a worsening channel condition. The channel evaluation and gearshift operations are done at the beginning of each data block.

The gearshift algorithm is configurable and can be an SNR based that relies on a test message to evaluate the channel condition (downlink) and set the configuration accord-ingly, or a Ladder algorithm that starts at an initial selectable robustness configuration and switches to the next configuration in the case of crossing a configurable threshold or return to the more robust configuration in case of not achieving a configurable lower transmission success threshold.

The gearshift mechanism can enter a temporary configurable block period to avoid multiple unsuccessful up-gearshifts.

The robustness configurations are numbered (0 to 16) where CONFIG\_0 is the most robust one.

4

The datalink layer connects to a possible application layer via two TCP/IP connec-tions, a data dump buffer, and a control connection with an API.

**1.1.2.3 API**

A base API was implemented to provide the basic functionality in a way that would be familiar to users of commercially available modems. The API can easily be updated and new commands can be added to provide further control to the application layer.

The base API is composed of the following commands:

MYCALL mycall\r

Sets the AQR call sign to the value of the string ”my\_call”.

Reply: OK\r

LISTEN ON\r

Sets the ARQ to a responder role and waits for an incoming connection.

Reply: OK\r

CONNECT my\_call destination\_call\r

Sets the ARQ to a commander role and initiates the connection sequence to ”destina-

tion\_call”.

Reply: OK\r

DISCONNECT\r

Terminates the current connection.

Reply: OK\r

BW2300\r

Sets the bandwidth to 2300 Hz.

Reply: OK\r

BW2500\r

5

Sets the bandwidth to 2500 Hz.

Reply: OK\r

BUFFER TX\r

Reads the number of data bytes in the TX buffer yet to be sent.

Reply: BUFFER bytes\r

In the case of an unrecognized command, a reply: NOK\r is provided.

6

**Chapter 2**

**First steps**

**2.0.1 Get Mercury on Linux**

**2.0.1.1 Prerequisites**

First, you need to install Mercury dependencies:

**ALSA Sound:**To use the sound card via ALSA Sound library, the library should be installed and linked to the code at compilation. To install ALSA Sound run the following command:

sudo apt−g e t i n s t a l l l i b a s o u n d 2 −dev

**GNU PLOT:**In case of switching the configuration of the physical layer plot\_active to YES, the Gnuplot is required to be installed on the machine. To install GNU plot run the following command:

sudo apt−g e t i n s t a l l g n u p l o t −x11

**Documentation:**To generate the documentation, the Doxygen and GraphViz packets should be installed. To install Doxygen and GraphViz run the following commands:

sudo apt−g e t i n s t a l l doxygen

sudo apt−g e t i n s t a l l g r a p h v i z

7

**2.0.1.2 Mercury source code**

To get the latest version of Mercury, you need to clone the source code from the

GitHub repository:

g i t c l o n e h t t p s : / / g i t h u b . com/ R h i z o m a t i c a / mercury . g i t

then navigate to the Mercury directory

cd mercury

finally, compile the source code using “make”

make mercury

or generate the documentation

make doc

optionally you can install the binary files

make i n s t a l l

**2.0.2 Modes**

Mercury can operate in one of five different modes, the mode can be selected in “source/main.cc”

t e l e c o m \_ s y s t e m . operation\_mode = ‘ ‘ s e l e c t e d mode ” ;

ARQ\_MODE: this mode engages the datalink layer with ARQ and the physical layer, this is the operational mode.

BER\_PLOT\_baseband: this mode runs a physical layer-only baseband Bit Error Rate (BER) simulation over an AWGN channel with/without plotting. In this mode, filters, PAPR reduction and up/down frequency converters are not included in the simu-lation

BER\_PLOT\_passband: this mode runs a physical layer-only passband BER simu-lation over an AWGN channel with/without plotting. In this mode, the full TX and RX functions of the physical layers are included in the simulation.

8

TX\_TEST: this mode generates an output signal with random data content.

RX\_TEST: this mode demodulates and decodes a received signal with/without plot-ting.

**2.0.3** **Testing**

In addition to running the BB or the PB simulation for a specific configuration, the physical layer of Mercury can be tested on its own or the full Mercury stack can be tested as a whole.

**2.0.3.1 Physical layer**

To test Mercury’s physical layer, two computers are required with a crossover connec-tion of the speaker and microphone (the speaker of the first computer is connected to the microphone of the second one and vice versa).

In this case (or in the case of connecting Mercury to a physical radio), physical layer parameters such as the signal frequency, power, and filter values need to be adjusted adequately. This can be done from the physical layer configuration file.

Alternatively, an ALSA loopback with GNU Radio Companion (GRC) can be used.

To configure the ALSA loopback run the following command:

sudo modprobe snd−a l o o p

To install the GRC software run the following command:

sudo apt−g e t i n s t a l l g n u r a d i o

Clone and compile two separate copies of Mercury.

considering the loop-back ALSA card has the number ’3’, configure the first Mercury (Mercury 1) speaker to

speaker\_dev\_name=”plughw : 3 , 0 , 3 ” ;

and the Mercury 1 microphone to

9

speaker\_dev\_name=”plughw : 3 , 0 , 2 ” ;

and configure the Mercury 2 speaker to

speaker\_dev\_name=”plughw : 3 , 0 , 1 ” ;

and the Mercury 2 microphone to

speaker\_dev\_name=”plughw : 3 , 0 , 4 ” ;

By starting GRC with the “Mercury\_channel\_test.grc” project, the connection be-tween the two Mercuries is established.

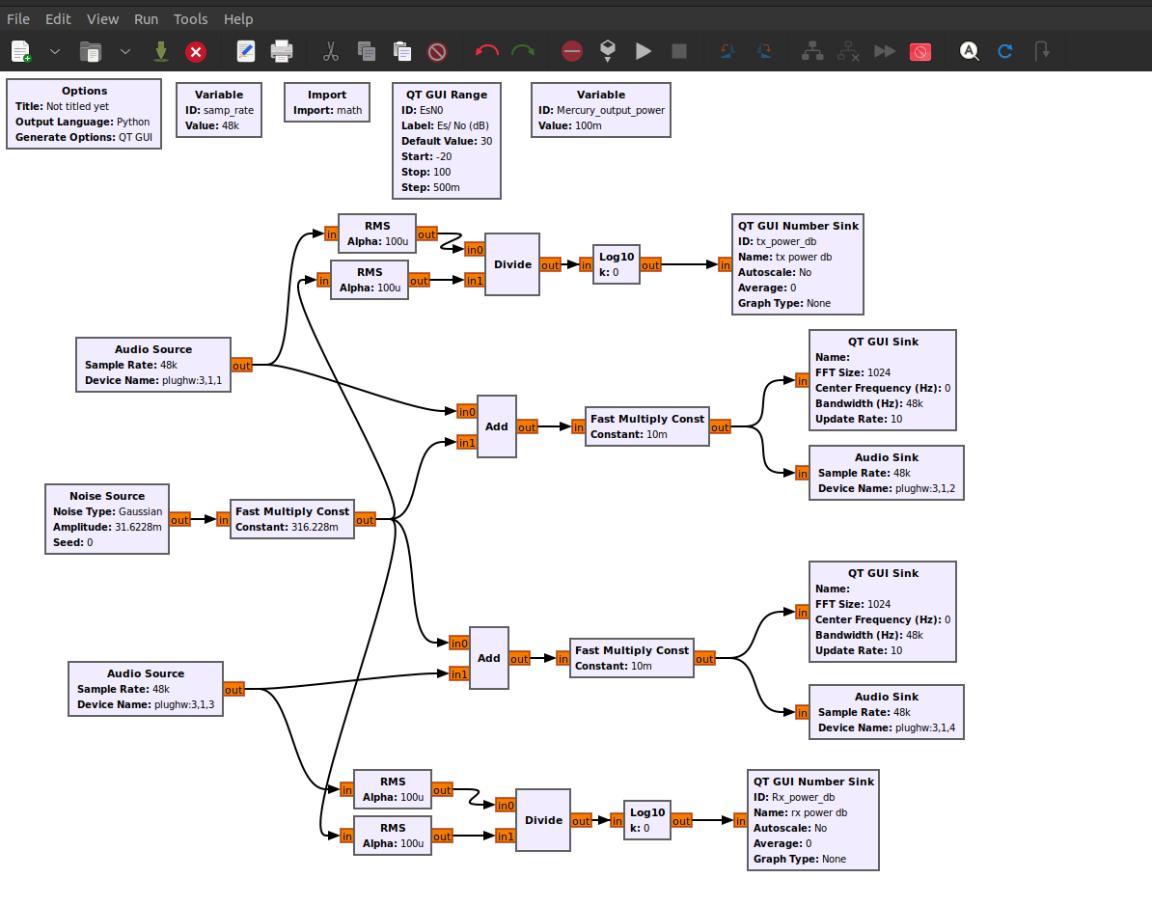


Figure 2.1: Mercury channel test.

10

By starting the GNU Radio project, both the spectrum analyzer and the waterfall can be accessed.

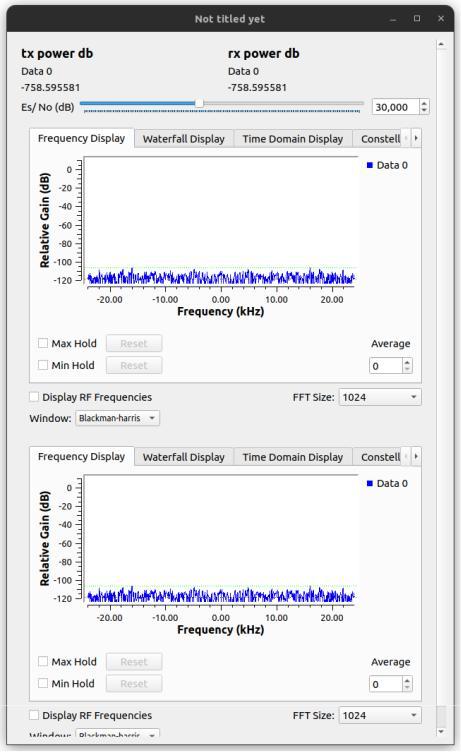


Figure 2.2: Spectrum analyzer.

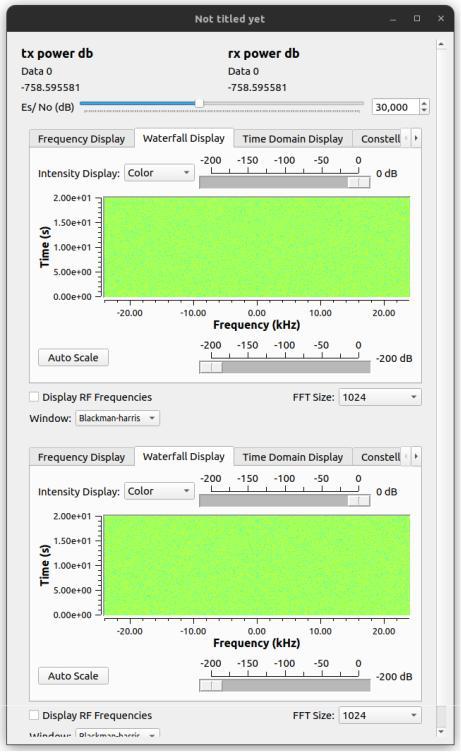


Figure 2.3: Waterfall.

11

By configuring Mercury 1 to TX\_TEST mode and starting the program, the pilot pattern and current configurations are printed in the terminal.

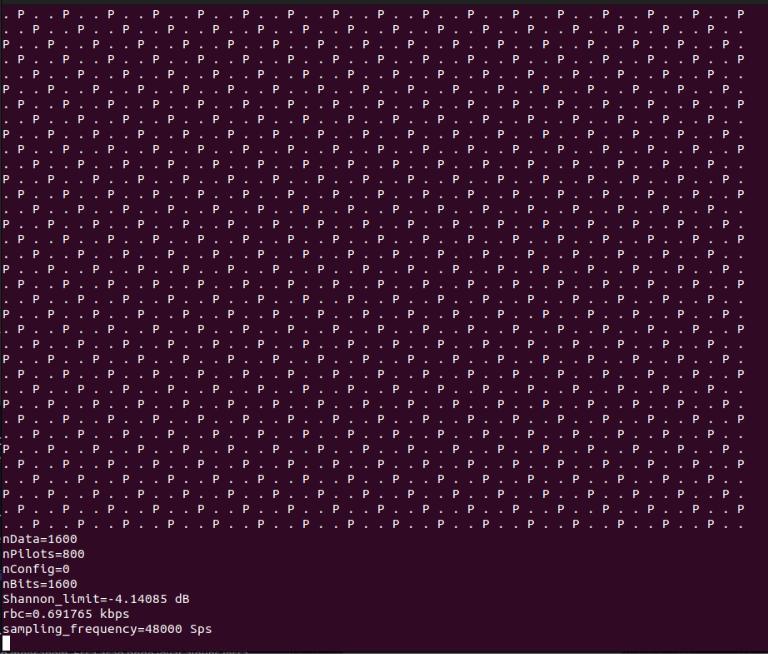


Figure 2.4: Mercury pilot pattern and current configurations.

12

The transmitted signal appears on both the spectrum analyzer and the waterfall.

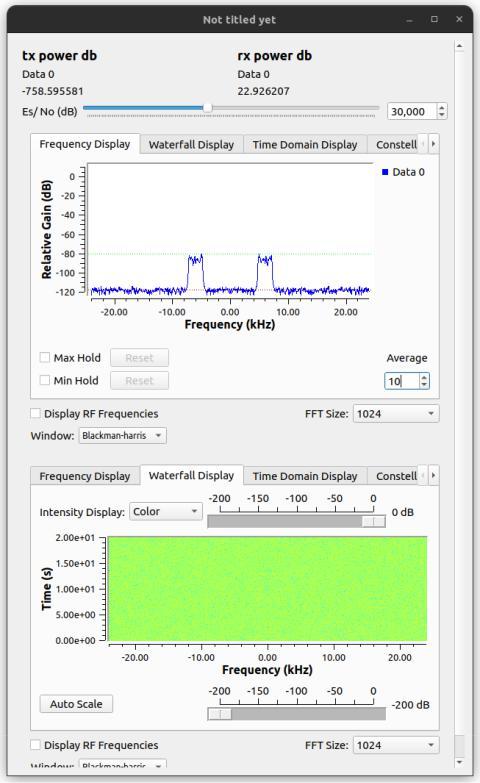


Figure 2.5: Spectrum analyzer.

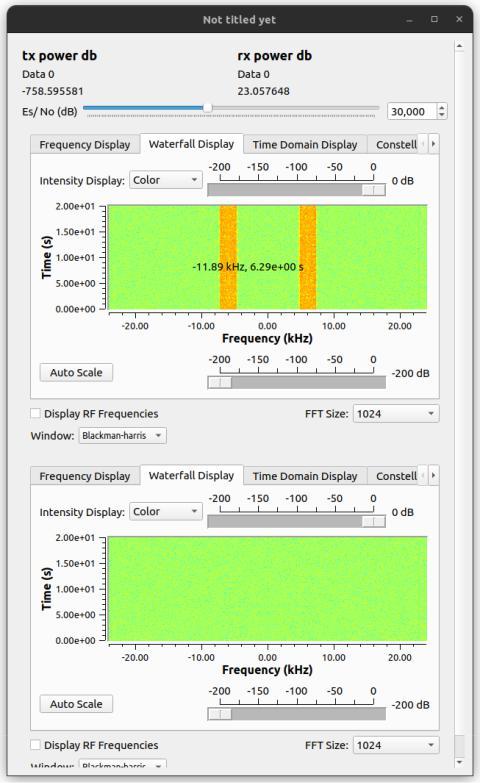


Figure 2.6: Waterfall.

13

By configuring Mercury r to RX\_TEST mode and starting the program, the pilot pattern and current configurations are printed in the terminal.

It is important to verify that the plotting folder exists and Mercury has access to it, or that the plotting is inactive. This can be done from the physical layer configuration file.

Once the signal is received and decoded, the decoded message’s initial bytes and synchronization information, and signal strength (dbm) are printed and the received con-stellation is plotted.

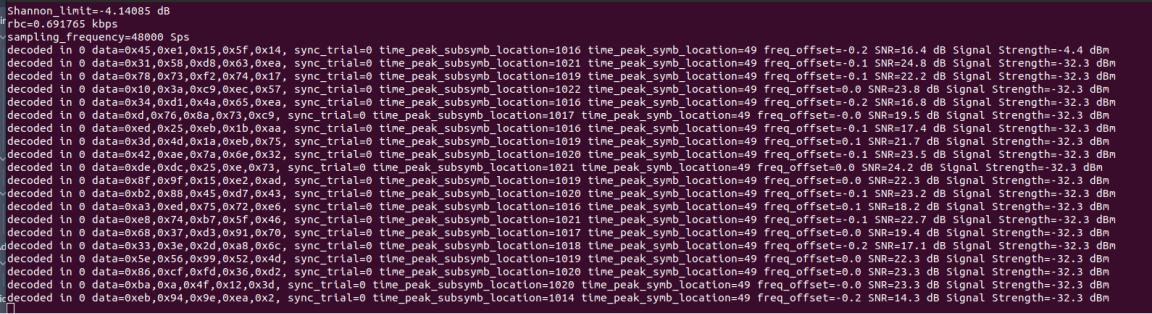


Figure 2.7: Decoded message information.

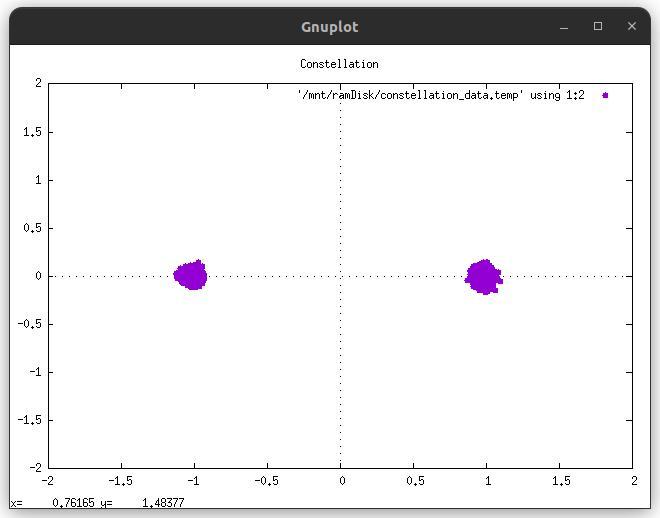


Figure 2.8: Received constellation.

The same steps can be repeated to test the connection of Mercury 2 to Mercury 1.

in addition, the robustness configuration for both Mercury 1 and Mercury 2 can be changed from the physical layer configuration file to test any of the 17 pre-set config-

14

urations. This can be done by setting the variable “init\_configuration” to the desired configuration.

The robustness of each of those configurations can be tested by changing the value of *Es/N*0 in the GNU radio window.

**2.0.3.2 Datalink layer**

Once the physical layer connection is validated in both directions, the mode of both Mercury 1 and Mercury 2 can be switched back to ARQ\_MODE and the data link layer API can be used to set a CALL for each of them, initiate the connection and transfer data from one to another.

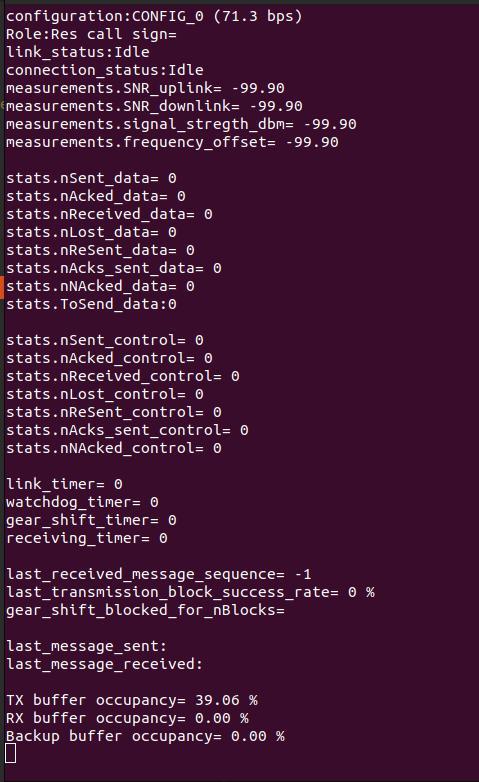


Figure 2.9: Datalink.

The datalink layer can be tested with Gearshift active or inactive, the initial configu-ration and the ACK configuration along with other configurations such as the API ports and timeout values can be configured fro the datalink layer configuration file.

15

**2.0.4** **Configurations**

The physical layer and the datalink layer of Mercury can be configured independently.

**2.0.4.1 Physical layer**

The physical layer parameters can be defined in “physical\_layer/physical\_config.cc”

The physical layer has the following parameters to be configured (more fine-tuning parameters are in telecom\_system.cc/telecom\_system.h):

**Roubustness:**

init\_configuration: The initial configuration to be used for transmission (Modulation, code rate, etc.)

value can be set to (CONFIG\_0, CONFIG\_1 ... CONFIG\_16).

Table 2.1: Roubustness configurations.

| Config | Modulation | FEC | EsN0 | Bit-rate | Bit-rate |
| --- | --- | --- | --- | --- | --- |
|  |  | code rate | (FER<0.1) | (HD pilots) | (LD pilots) |
|  |  |  |  |  |  |
| 0 | BPSK | 1/16 | -10.0 | 71.3 | 84.2 |
|  |  |  |  |  |  |
| 1 | BPSK | 2/16 | -7.5 | 156.1 | 184.5 |
|  |  |  |  |  |  |
| 2 | BPSK | 3/16 | -6.0 | 241.0 | 284.8 |
|  |  |  |  |  |  |
| 3 | BPSK | 4/16 | -4.5 | 325.8 | 385.0 |
|  |  |  |  |  |  |
| 4 | BPSK | 5/16 | -3.5 | 410.6 | 485.3 |
|  |  |  |  |  |  |
| 5 | BPSK | 6/16 | -2.5 | 495.5 | 585.6 |
|  |  |  |  |  |  |
| 6 | BPSK | 8/16 | -1.5 | 665.2 | 786.1 |
|  |  |  |  |  |  |
| 7 | QPSK | 5/16 | -0.5 | 762.6 | 889.7 |
|  |  |  |  |  |  |
| 8 | QPSK | 6/16 | 0.5 | 920.2 | 1073.5 |
|  |  |  |  |  |  |
| 9 | QPSK | 8/16 | 1.5 | 1235.3 | 1441.2 |
|  |  |  |  |  |  |
| 10 | 8PSK | 6/16 | 3.0 | 1353.7 | 1353.7 |
|  |  |  |  |  |  |
| 11 | 8PSK | 8/16 | 4.0 | 1818.1 | 1818.1 |
|  |  |  |  |  |  |
| 12 | QPSK | 14/16 | 6.5 | 2261.4 | 2654.7 |
|  |  |  |  |  |  |
| 13 | 16QAM | 8/16 | 7.5 | 2470.6 | 2882.4 |
|  |  |  |  |  |  |
| 14 | 8PSK | 14/16 | 9.0 | 3389.7 | 3389.7 |
|  |  |  |  |  |  |
| 15 | 16QAM | 14/16 | 12.5 | 4361.3 | 5088.2 |
|  |  |  |  |  |  |
| 16 | 32QAM | 14/16 | 13.5 | 5664.7 | 5664.7 |
|  |  |  |  |  |  |

16

To create a new configuration, the name of the configuration needs to be added to comon\_defines.h and the configuration parameters need to be added to the load\_configuration function in the telecom\_system.cc file of the physical layer.

Four essential parameters are needed for a new configuration, the modulation, ldpc\_rate, ofdm\_preamble\_configurator\_Nsymb and ofdm\_channel\_estimator.

**OFDM:**

The OFDM has the following main parameters (more fine-tuning parameters are in ofdm.cc/ofdm.h):

bandwidth: the used bandwidth for TX and RX.

frequency\_interpolation\_rate: sampling frequency interpolation rate to match the sound card sampling rate.

carrier\_frequency: the OFDM signal carrier frequency in TX and RX.

output\_power\_Watt: the output signal power at TX.

Nfft: the Fast Fourier Transfer (FFT) size.

Nc: the number of used sub-channels.

Dx: frequency distance between pilots.

Dy: time distance between pilots.

Nsymb: number of symbols per OFDM frame.

gi: Guard interval to remove the multipath effect.

ofdm\_start\_shift: Number of non-used subcarriers starting at 0 Hz.

bit\_energy\_dispersal\_seed: The seed used in the pseudorandom data generator of the bit energy dispersal.

**Pilots:**

pilot\_boost: to define pilot carriers’ power in relation to data carriers’ power.

ofdm\_pilot\_configurator\_first\_row: Set the first row to data or pilot.

ofdm\_pilot\_configurator\_last\_row: Set the last row to data or pilot.

17

ofdm\_pilot\_configurator\_first\_col: Set the first column to data or pilot.

ofdm\_pilot\_configurator\_second\_col: Set the second column to data or pilot.

ofdm\_pilot\_configurator\_last\_col: Set the last column to data or pilot.

ofdm\_pilot\_configurator\_seed: The seed used in the pseudorandom data generator of the pilot sequence.

ofdm\_pilot\_density: Pilot density (HIGH\_DENSITY or LOW\_DENSITY).

**Channel Estmation:**

ofdm\_channel\_estimator: The channel estimation Method (ZF or LS).

ofdm\_channel\_estimator\_amplitude\_restoration: Activate amplitude restoration (Yes, No). (only with M-Phase Shift Keying (PSK) modulation)

ofdm\_LS\_window\_width: The LS window width.

ofdm\_LS\_window\_hight: The LS window height.

**Syncronization:**

time\_sync\_trials\_max: number of time synchronization detection peaks to be con-sidered.

use\_last\_good\_time\_sync: whether to use the last-time synchronization result or try to synchronize again in each frame.

use\_last\_good\_freq\_offset: whether to use the last time synchronization result or try to synchronize again in each frame.

ofdm\_time\_sync\_Nsymb: Number of OFDM symbols when the GI is used for time synchronization.

freq\_offset\_ignore\_limit: the minimum value of sampling frequency and carrier fre-quency offset to be compensated.

**Filters:**

ofdm\_FIR\_rx\_time\_sync\_filter\_window: the FIR window at RX for the time sync part of the signal.

18

ofdm\_FIR\_rx\_time\_sync\_filter\_transition\_bandwidth: the FIR transition width at RX for the time sync part of the signal.

ofdm\_FIR\_rx\_time\_sync\_filter\_cut\_frequency: the FIR cut frequency at RX for the time sync part of the signal.

ofdm\_FIR\_rx\_time\_sync\_filter\_type: the FIR type (Low Pass Filter (LPF), High Pass Filter (HPF), etc..) for the time sync part of the signal.

ofdm\_FIR\_rx\_data\_filter\_window: the FIR window at RX for the data part of the signal.

ofdm\_FIR\_rx\_data\_filter\_transition\_bandwidth: the FIR transition width at RX for the data part of the signal.

ofdm\_FIR\_rx\_data\_filter\_cut\_frequency: the FIR cut frequency at RX for the data part of the signal.

ofdm\_FIR\_rx\_data\_filter\_type: the FIR type (LPF, HPF, etc..) for the data part of the signal.

ofdm\_FIR\_tx1\_filter\_window: the FIR window at TX.

ofdm\_FIR\_tx1\_filter\_transition\_bandwidth: the FIR transition width at TX.

ofdm\_FIR\_tx1\_filter\_cut\_frequency: the FIR cut frequency at TX.

ofdm\_FIR\_tx1\_filter\_type: the FIR type (LPF, HPF, etc..) at TX.

ofdm\_FIR\_tx2\_filter\_window: the FIR window at TX.

ofdm\_FIR\_tx2\_filter\_transition\_bandwidth: the FIR transition width at TX.

ofdm\_FIR\_tx2\_filter\_cut\_frequency: the FIR cut frequency at TX.

ofdm\_FIR\_tx2\_filter\_type: the FIR type (LPF, HPF, etc..).

**Preamble:**

ofdm\_preamble\_configurator\_Nsymb: number of preamble symbols.

ofdm\_preamble\_configurator\_nIdentical\_sections: number of identical parts per pream-ble symbol.

19

ofdm\_preamble\_configurator\_modulation: preamble modulation;

ofdm\_preamble\_configurator\_boost: preamble boost.

ofdm\_preamble\_configurator\_seed: preamble pseudo-random bits seed.

**PAPR reduction:**

ofdm\_preamble\_papr\_cut: preamble PAPR cut limit (before filtering)

ofdm\_data\_papr\_cut: data PAPR cut limit (before filtering)

**FEC:**

decoding algorithm: the LDPC decoding algorithm can be either the SPA or the Gradient Bit-Flipping Algorithm (GBF).

standard and framesize: the LDPC code deploys specially designed LDPC matrices of the size 1600 bits with several different code rates.

code rate: the code rate that defines the protection level vs data rate (1/16, 2/16, 3/16, 4/16, 5/16, 6/16 8/16, 14/16).

GBF\_eta: the GBF LDPC decoder correction rate.

nIteration\_max: the number of maximum decoding iterations.

The outer code has the following options:

outer\_code: (CRC16\_MODBUS\_RTU or NONE)

**ALSA:**

microphone\_dev\_name: Alsa capture device name (”plughw:1,0” for example).

speaker\_dev\_name: Alsa play device name (”plughw:1,0” for example).

microphone\_type: Capture channel.

microphone\_channels: Capture channels (Mono, Stereo, Left, Right).

speaker\_type: Play channel.

speaker\_channels: Play channels (Mono, Stereo, Left, Right).

speaker\_frames\_to\_leave\_transmit\_fct: Number of frames to be played before leav-

20

ing the Alsa transmission function, to avoid Alsa underrun.

**Plot:**

plot\_folder: Folder to be used as a temporary for the plot function.

plot\_plot\_active: Plot function activation (Yes, No).

**2.0.4.2 Datalink layer**

The datalink layer parameters can be defined in data\_link\_layer/datalink\_config.cc

The datalink layer has the following parameters to be configured:

fifo\_buffer\_tx\_size: The First In First Out (FIFO) transmission buffer size (data to be transmitted).

fifo\_buffer\_rx\_size: The FIFO reception buffer size.

fifo\_buffer\_backup\_size: Backup FIFO buffer for configuration switch.

link\_timeout: Maximum time without any activity on the link (TX or RX) before link drops (ms).

tcp\_socket\_control\_port: Control TCP/IP port.

tcp\_socket\_control\_timeout\_ms: Control TCP/IP timeout (ms).

tcp\_socket\_data\_port: Data TCP/IP port.

tcp\_socket\_data\_timeout\_ms: Data TCP/IP timeout (ms).

init\_configuration: The initial data configuration to be used.

ack\_configuration: The configuration to be used for acknowledgment messages.

gear\_shift\_on: Gearshit setting (Yes, No).

gear\_shift\_algorithm: Gearshit algorithm (SNR-based, Ladder based)

gear\_shift\_up\_success\_rate\_limit\_precentage: Threshold for up-gearshifting.

gear\_shift\_down\_success\_rate\_limit\_precentage: Threshold for down-gearshifting.

gear\_shift\_block\_for\_nBlocks\_total: Number of transmission blocks to be done

21

without a gearshift try in case of an up-gearshift failure.

batch\_size: Number of messages per data batch.

nMessages: Number of messages per block.

nResends: Number of trials for each data/control message.

ack\_batch\_size: Number of messages per acknowledgment batch.

control\_batch\_size: Number of messages per control batch.

ptt\_on\_delay\_ms: time in milliseconds to wait after switching on the Push To Talk (PTT) before sending data.

ptt\_off\_delay\_ms: time in milliseconds to wait after switching off the PTT after sending data.

switch\_role\_timeout\_ms: time in milliseconds to wait before switching roles with the responder in case of an empty data buffer.

22

**Chapter 3**

**License**

Mercury: A configurable open-source software-defined modem.

Copyright (C) 2022-2024 Fadi Jerji

Author: Fadi Jerji

Email: fadi.jerji@ <gmail.com, caisresearch.com, ieee.org>

ORCID: 0000-0002-2076-5831

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, version 3 of the License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

23