

Application Note

MSR4 - MULTIPURPOSE SATELLITE RECEIVER

Getting started with I/Q Streaming Application

Products:

- ▶ R&S®MSR4-IQ

M.Bordini | 8INH | Version 03.01 | 06.2023 | Document ID 1179.7104.01

ROHDE & SCHWARZ

Make ideas real



Contents

1	Overview.....	3
2	The MSR4 device	3
3	Requirement.....	5
3.1	Licensing	5
3.2	Receiving System Specification	5
3.2.1	Configuration and Operating system	6
4	Git Repositories	8
4.1	Preface.....	8
4.2	R&S-Jerry-Setup	8
4.3	R&S-Jerry-Driver	9
4.4	Verification flow	9
4.5	R&S-Jerry-Gnuradio.....	10
4.6	R&S wideband I/Q Recording Driver	12
5	Operation.....	12
5.1	Preface of operating options	12
5.2	Operating with WebUI interface	12
5.2.1	RadioFreq settings	13
5.2.2	Protocol settings.....	14
5.2.3	UDP/IP settings	14
5.3	Operating with gRPC interface.....	14
6	Data Format.....	15
6.1	Ethernet framing.....	15
6.2	HRZR protocol	17
6.2.1	Calculating Power Spectrum	17
7	Ordering Information	19

1 Overview

The intention of this Application Note is to provide an overview of all necessary tasks for getting along with successful I/Q Streaming.

The necessary technical requirements are described as well as the streaming parameters of the I/Q data stream.

The general operation and basic setup of the MSR4 is described in the user manual and is not part of this document.

2 The MSR4 device

The MSR4 is a four-channel Satcom receiver whose input channels each have a simultaneous bandwidth of 200 MHz, which can be tuned in the frequency range from 500 MHz to 3000 MHz.

The four receive channels can be tuned individually or on the same frequency.

Two transmission channels are also available soon, each with a simultaneous bandwidth of 200 MHz, which can be tuned in the frequency range from 900 MHz to 2500 MHz.

The MSR4 generates digital I/Q data from the received spectrum, which is transmitted via 4 assigned SFP+ 10 Gbit/s Ethernet connections.

Each receiving channel has a dedicated SFP+ connection.

The transmission channels can be used in different ways, such as the analog playback of digital I/Q data, which is fed to the MSR4 via the SFP+ connection.

The dedicated standard 1 Gbit/s Ethernet connection via RJ45 is used for control and management of the MSR4 such as configuration over Web UI or an integrated gRPC interface.

This results in numerous applications in the area of Satcom.

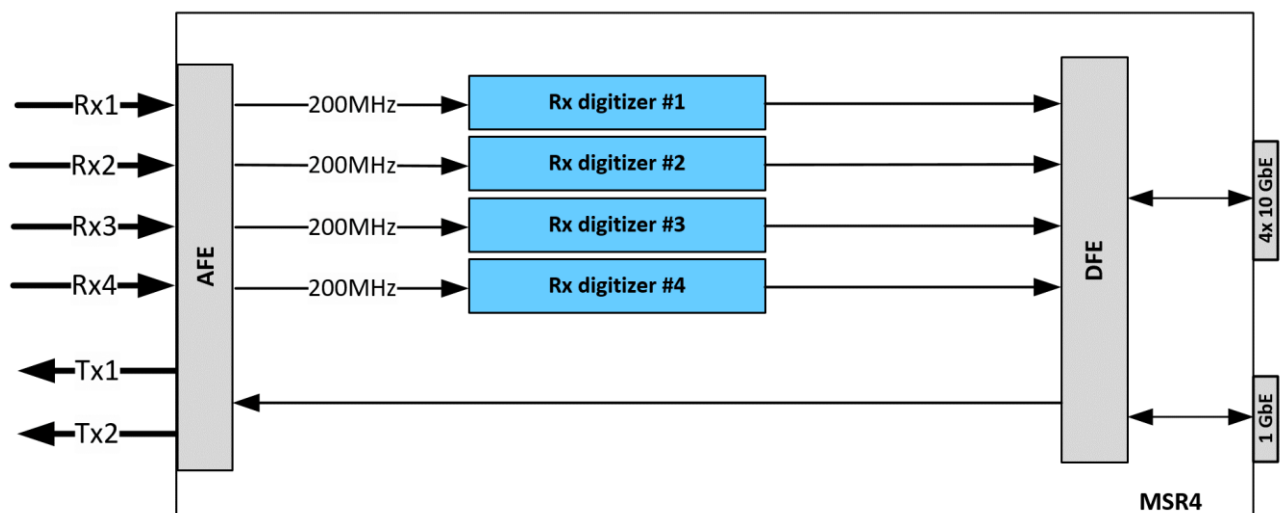


Figure 1: Block schematic of MSR4

There are a number of software packages available for the MSR4, including I/Q Streaming and the TX option.

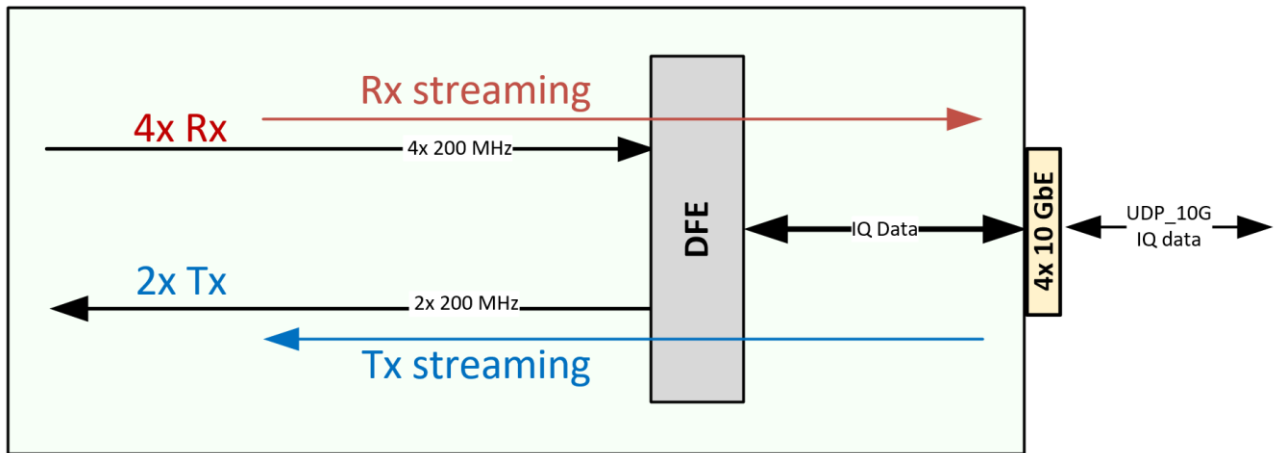


Figure 2: Streaming options Rx and Tx



3 Requirement

3.1 Licensing

Make sure your MSR4 has all MSR4-IQ licenses necessary for your application or task. Each receiving channel as well as each streaming type needs its own license.

For example, four channels streaming simultaneously over UDP/IP need four MSR4-IQ licenses. The streaming application to the analog transmit channels of MSR4 is the MSR4-TX option. Both transmit channels can be used for individual tasks. The transmit option is planned for future use.

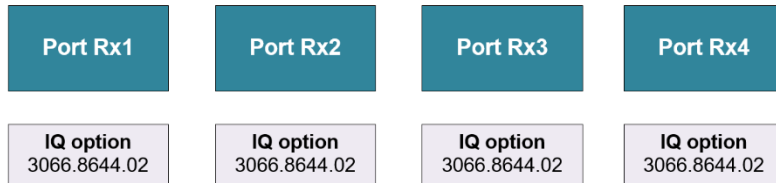


Figure 3: Licensing model MSR4 (Receiver part)

Take care that the external hardware components such as SFP modules and optical cables are supported by R&S. Otherwise data corruption or link losses can occur.

3.2 Receiving System Specification

The recommendation of the required Hardware System receiving data is to have the minimum hardware requirements below.

- ▶ 12 Core CPU System or better for a four channel system
- ▶ 32 GB+ RAM
- ▶ 2048+ MB Virtual RAM
- ▶ 2 GB+ Free Disk Space depending on individual requirement
- ▶ 10 Gbit/s Network Interface Card (Intel E810 preferred)
- ▶ NAS system with capable transfer rate
- ▶ OS with hugepage support

Especially the supported Hardware for the Intel DPDK has to be verified, listed here:

<http://core.dpdk.org/supported/>

Reference Hardware

- ▶ Lenovo ThinkSystem SR635 Rack Server
- ▶ 4x Intel DC P4610 Series 1,6 TB
- ▶ 128 GB DDR4-RAM
- ▶ Intel x520 Network Interface

3.2.1 Configuration and Operating system

It is strongly recommended the one CPU core per channel is mapped for receiving data and another CPU core per channel for transmitting data to your application. Prevent Kernel IRQs on CPU cores running on DPDK application for a non-disturbed streaming without packet losses.

Another CPU core should take care about housekeeping functionalities.

3.2.1.1 SR-IOV

Single root I/O virtualization (SR-IOV) is an extension to the PCI Express (PCIe) specification and allows a device, such as a network adapter, to separate access to its resources among various PCIe hardware functions.

These functions are:

- ▶ A PCIe Physical Function (PF). This is the primary function of the device. Physical Functions are discovered, managed, and configured as normal PCI devices. Physical Functions configure and manage the SR-IOV functionality by assigning Virtual Functions.
- ▶ One or more PCIe Virtual Functions (VFs). Each VF is associated with the device's PF. A VF shares one or more physical resources of the devices, such as a memory and a network port, with the PF and other VFs on the device. The VFs have access to a subset of the PF's functionality, enforced by the PF's configuration.

On Linux the SR-IOV drivers are implemented in the kernel. The core implementation is contained in the PCI subsystem, but there must also be driver support for both the PF and VF devices. VFs have near-native performance and provide better performance than paravirtualized drivers and emulated access.

[Source: [Overview of Single Root I/O Virtualization \(SR-IOV\) - Windows drivers | Microsoft Learn](#), [Chapter 13. SR-IOV Red Hat Enterprise Linux 6 | Red Hat Customer Portal](#)]

Instructions on how to activate SR-IOV in the BIOS settings can be found in the corresponding git repository. [r&s-jerry-setup]

The activation of SR-IOV has to be configured in the BIOS of the receiving server. An example is given in the following figures.



Figure 4: BIOS main title, go to Device settings



Figure 5: BIOS, choose the Network adapter

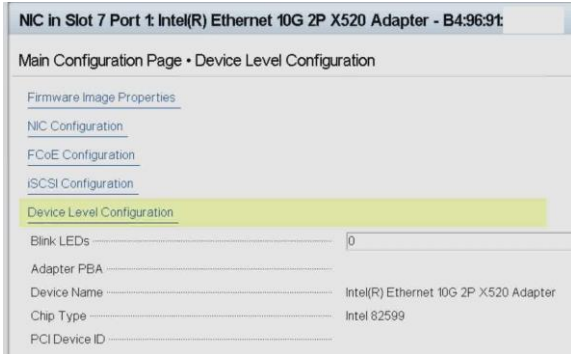


Figure 6: BIOS, choose the device configuration

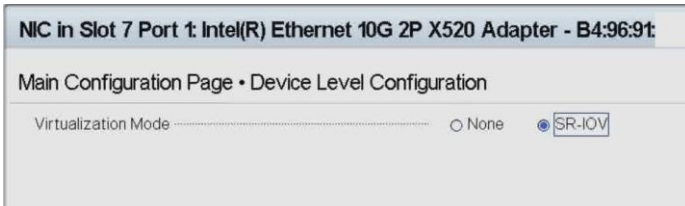


Figure 7: BIOS, activate SR-IOV for Network Interface card

3.2.1.2 DPDK

The Data Plane Development Kit (DPDK) is an open source software project managed by the Linux Foundation. It provides a set of data plane libraries and network interface controller polling-mode driver for offloading TCP packet processing from the operating system kernel to processes running in user space.

[Source: <https://www.dpdk.org/>]

This allows us in combination with SR-IOV to further accelerate network traffic.

Information which DPDK version to use can be found in the corresponding git repository [rs-jerry-setup & rs-jerry-driver].

4 Git Repositories

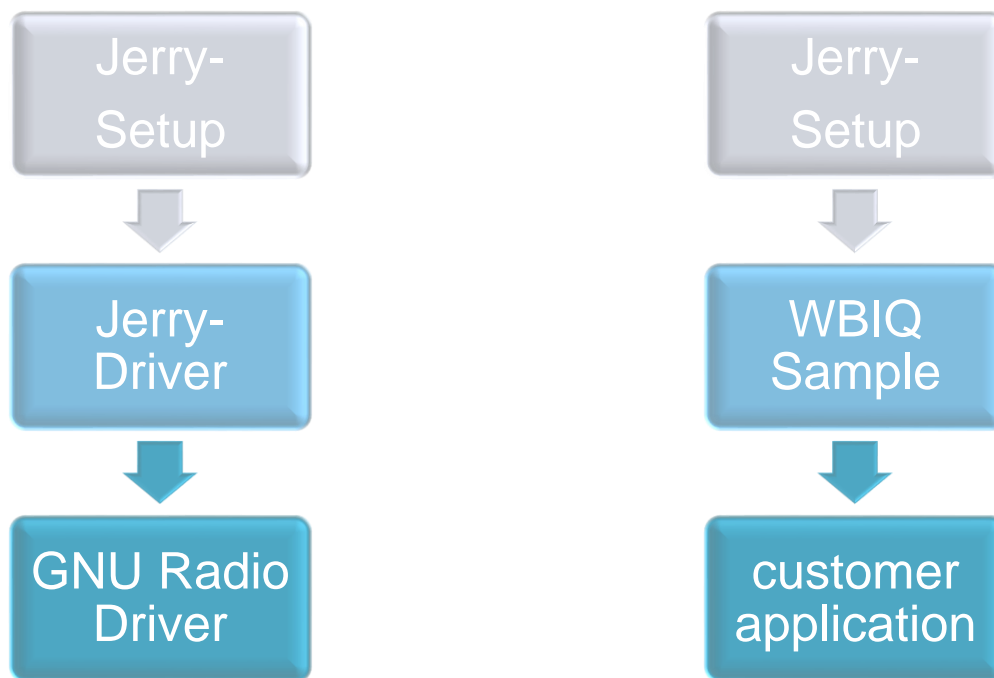
4.1 Preface

There are two main Repositories for the MSR4 Streaming application. First the Jerry-setup on the receiving server is necessary, so that DPDK as well as the I/O virtualization is configured.

- ▶ Jerry-Setup - basic initialization

From there, there are two solutions for different tasks:

- ▶ Jerry Driver - used for single channel streaming applications and the GNU Radio Driver
- ▶ WBIQ Driver - used for up to 4 channel I/Q Streaming to internal mass storage devices for a persistent I/Q data storage



4.2 R&S-Jerry-Setup

This repository is used to prepare the network interface card as well as configuring other relevant settings to use DPDK / SR-IOV.

The repository contains everything necessary to be able to run the R&S-Jerry-Driver on your system / machine.

Additional info can be found in the Repository:

<https://github.com/Rohde-Schwarz/rs-jerry-setup>

4.3 R&S-Jerry-Driver

Being the main Repository, this repository holds the functionality to receive and forward I/Q-data from the MSR4 and offers tools to communicate and configure the MSR4 remotely via gRPC.

Goal of this repository is to install a single static library as well as necessary headers on your system. Doing so allows you to include the static library in any of your projects to receive data from the MSR4 and/or configure the MSR4.

Additional info can be found in the repository:

<https://github.com/Rohde-Schwarz/rs-jerry-driver>

4.4 Verification flow

Listed below is a Python code example for the extraction of the HRZR protocol on the receiving server.

```
uint16_t nb_rx;
int prev_seq = 0;
int seq = 0;
while (!dppkSource->shouldQuit())
{
    // Check if new packets are available
    nb_rx = rte_eth_rx_burst(dppkSource->getPortID(),
definitions::SELECTED_QUEUE, mbufs, 32);
    if (nb_rx)
    {
        for (j = 0; j < nb_rx; j++)
        {
            // Get the next packet
            struct rte_mbuf *m = mbufs[j];
            packet = rte_pktmbuf_mtod(m, struct
hrzr_packet_all_headers *);

            // Checking for correct sequence number
            prev_seq = seq;
            seq = HrzrHeaderParser::getSequenceNumberFromHeader(packet->hrzr);
            dppkSource->total_packets++;
            if(seq != prev_seq+1 && prev_seq != 4095 && seq != 0)
            {
                dppkSource->total_packets_lost += std::abs(seq-prev_seq);
                dppkSource->total_packets += std::abs(seq-prev_seq);
            }
        }
    }
}
```

```

        int diff = std::abs(seq-prev_seq);
        if(seq < prev_seq){
            diff = seq + (4095-prev_seq);
        }

        float delta = (float)dpdkSource->total_packets_lost /
dpdkSource->total_packets;

        std::cout << "skipping " << diff << " packets.\t" << " rate: " <<
delta << std::endl;
    }

    // If packet is correct: enqueue using DPDK. Else: discard it
    if (packet->ipv4_hdr.next_proto_id == IPPROTO_UDP &&
packet->ether_hdr.ether_type == htons(RTE_ETHER_TYPE_IPV4)
        && packet->udp.dst_port == htons(dpdkSource->getUdpRxPort()) &&
m->pkt_len == definitions::PAYLOAD_SIZE + sizeof(struct hrzr_packet_all_headers)
        &&
dpdkSource->isValidPacketType(HrzrHeaderParser::getControlFromHeader(packet->hrz
r)))
    {
        rte_ring_enqueue(dpdkSource->getRteRing(), (void *) m);
    }
    else
    {
        rte_pktmbuf_free(m);
    }
}
}
}

```

4.5 R&S-Jerry-Gnuradio

As an example on how to use the driver, this repository provides a unique OutOfTreeModule for GNU Radio which receives data from the aforementioned driver and sending it to other GNU Radio blocks to modify or display the data.

The GNU Radio block allows configuration of the MSR4 and provides a single output port to stream the data to other blocks in GNU Radio.

To use the GNU Radio block, the R&S-Jerry-Driver needs to be installed on your system / machine.

Additional info can be found in the repository:

<https://github.com/Rohde-Schwarz/rs-jerry-gnuradio>

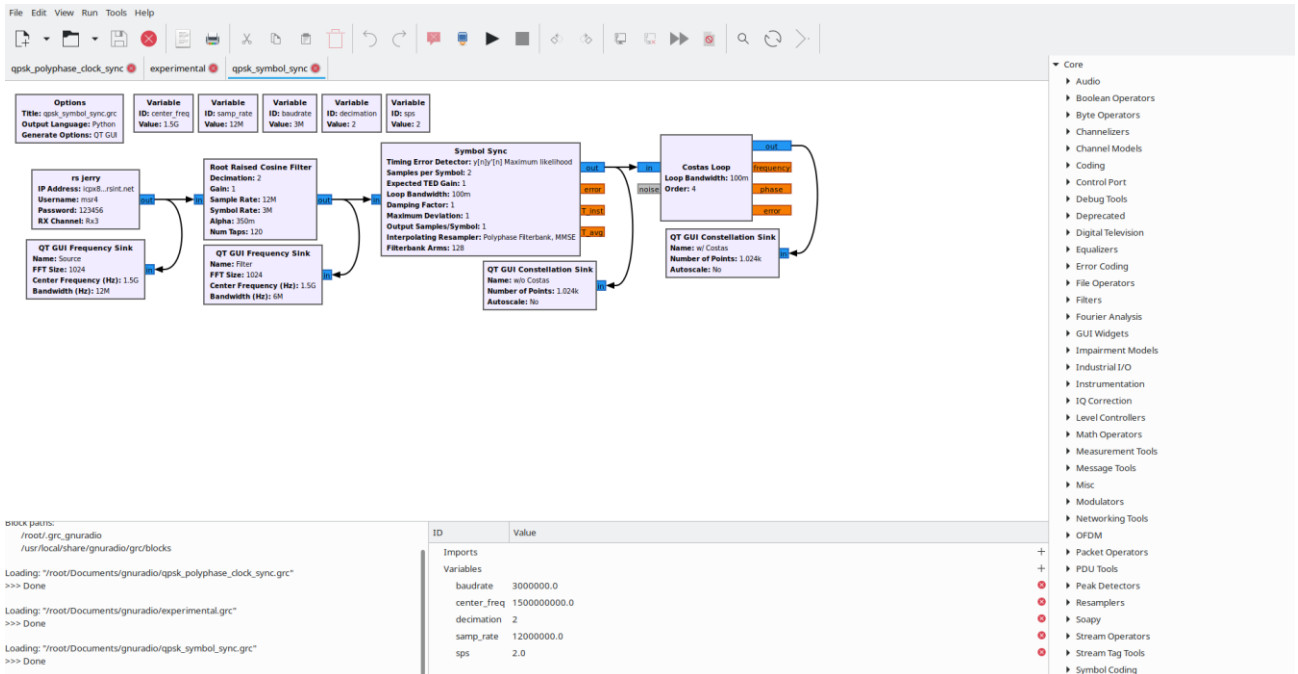


Figure 8: Example of a simple QPSK receiver

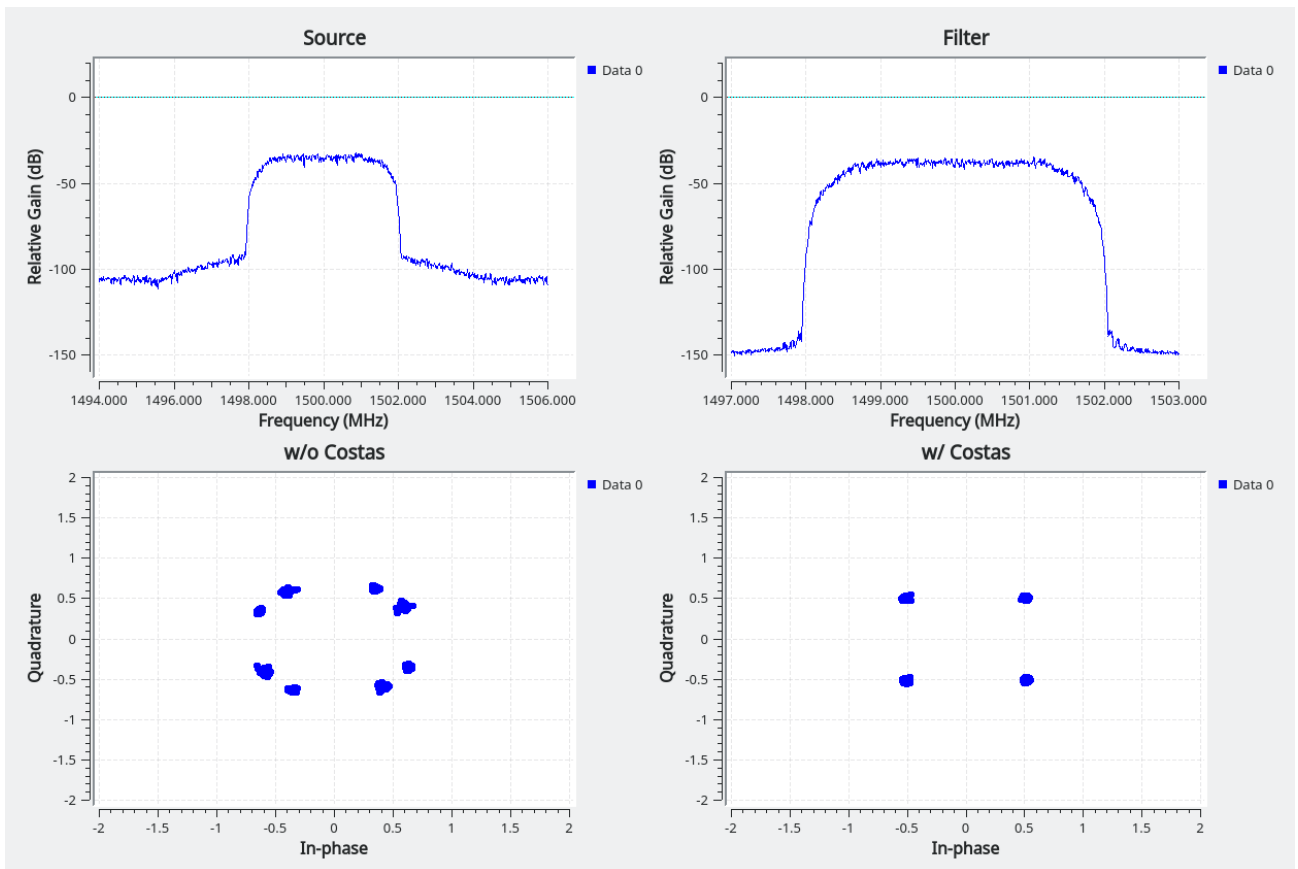


Figure 9: Diagrams of QPSK receiver

4.6 R&S wideband I/Q Recording Driver

This proof-of-concept Repository is used for storing up to four I/Q data streams (10 Gbit/s Ethernet) from MSR4 directly to the NVMe mass storage of the receiving server. The only supported format is the HRZR protocol.

<https://github.com/Rohde-Schwarz/rs-wbiq>

5 Operation

5.1 Preface of operating options

There are three different ways of operation to stream and process I/Q samples with the MSR4.

- ▶ Web UI interface
- ▶ gRPC interface
- ▶ GNU Radio

5.2 Operating with WebUI interface

For a detailed instruction of getting started with the MSR4 please refer to the User manual.

The default instrument name is <Type><variant>-<serial_number>, for example, msr4-101053. The unique device identifier is provided as a sticker on the rear panel of the R&S MSR4. It consists of the device order number and a serial number.

Enter your user name and password.

The initial username is msr4. The initial password is the device serial number.



Figure 10: MSR4 unique device identifier on the rear panel

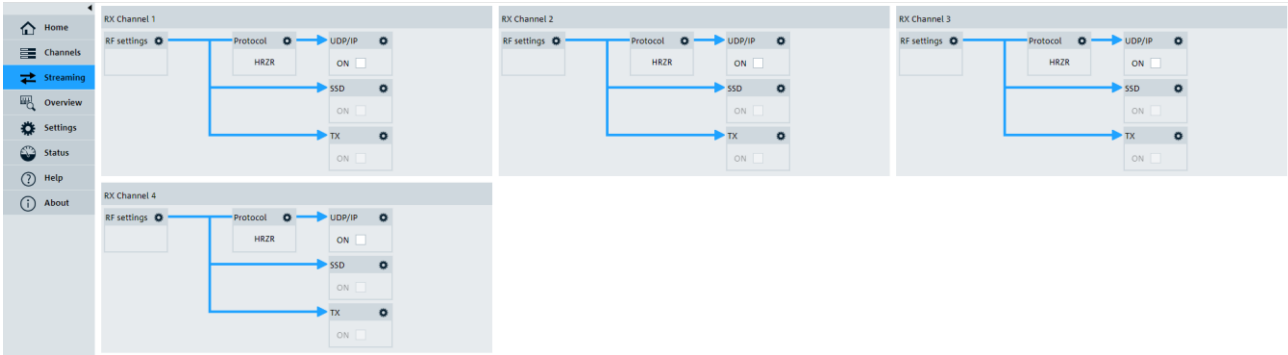


Figure 11: Streaming tab of MSR4 Web UI

Open the Streaming tab of MSR4 to see a streaming overview of all active Receiving channels. As listed in Figure 11 there are four active channels each with UDP/IP streaming but without Transmit channel and SSD streaming.

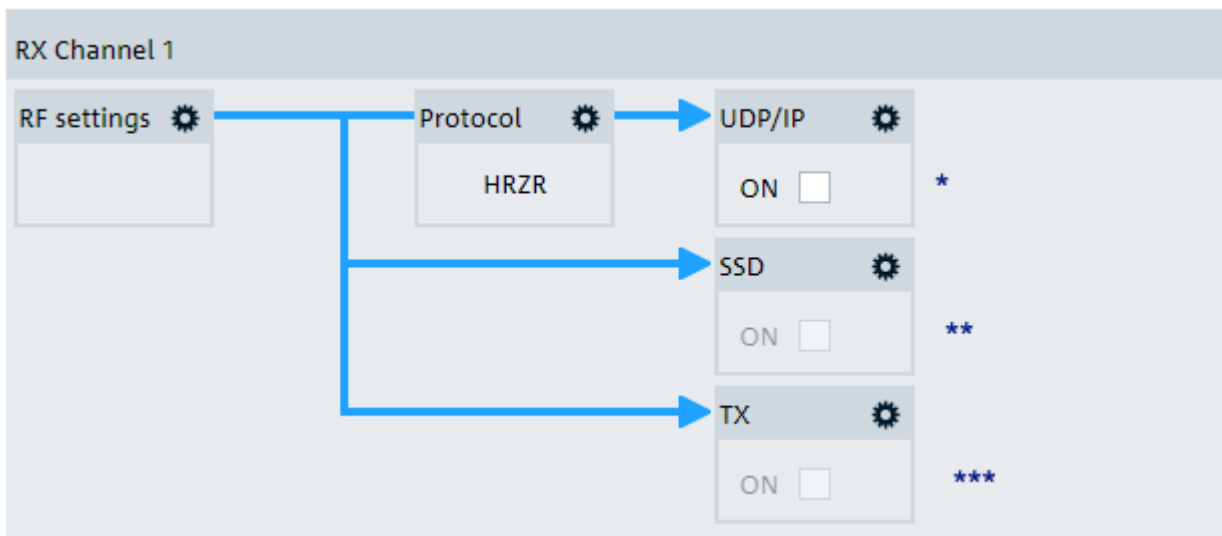


Figure 12: Streaming options

In Figure 12 all implementations of Streaming are shown, with inactive streaming highlighted in grey.

The UDP/IP (*)

The SSD¹ (**)

The TX² (***)

5.2.1 RadioFreq settings

The RF settings configure the RF input signal that provides the I/Q data.

Parameter	Explanation
Satellite frequency	Center frequency of the carrier at the satellite.
Downconverter frequency	Conversion frequency of the downconverter. Required to calculate the L-band frequency in the receiver from the carrier frequency at the satellite.

¹ For future use

² For future use

Parameter	Explanation
Analysis bandwidth	Displayed frequency range of the RF input channel, centered around the "L-Band frequency". The maximum bandwidth is the full span of the channel, which is 200 MHz.
L-Band frequency	For reference only: the input frequency at the R&S MSR4; calculated as the center frequency of the carrier, downconverted to the L-band. The allowed frequency range for reception is from 500 MHz to 3000 MHz, with a resolution of 1 Hz. The allowed frequency range for transmission is from 900 MHz to 2500 MHz, with a resolution of 1 Hz. The channel bandwidth is 200 MHz.

5.2.2 Protocol settings

The Protocol settings define the communication format used to stream the data from the R&S MSR4 to the connected device. Currently, only "HRZR" protocol is supported for I/Q data streaming. For more information on the protocol, see section 6.

5.2.3 UDP/IP settings

The UDP/IP settings determine the data receiver. Note that the connection settings to the receivers are configured in the Network Settings.

Parameter	Explanation
Destination address	The address on the receiver the data is streamed to, e.g. an IP address.
Port	Port on the receiver the data is streamed to
Network interface	The interface connector on the R&S MSR4 that the receiver is connected to. Currently, the RX channels are permanently assigned to the available connectors.

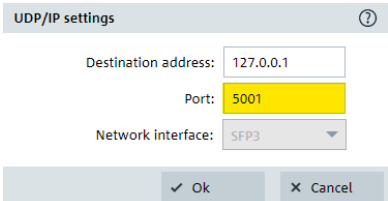
5.3 Operating with gRPC interface

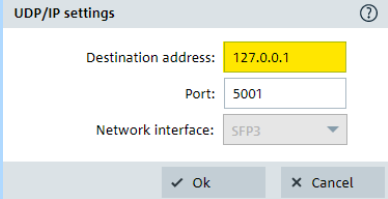
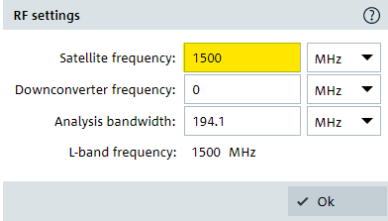
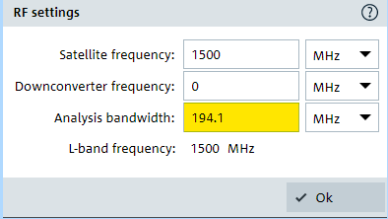
gRPC is a modern open source high performance Remote Procedure Call (RPC) library and framework developed by Google.

The gRPC interface is used to fully interact with the MSR4 settings via code without having to manually configure the WebUI.

This allows quicker and easier usage by just changing a single value instead of fiddling around with the WebUI as well as dynamically changing values based on changing circumstances.

Some example calls using gRPC are listed below.

gRPC call	Affected WebUI setting
SetPort(5001);	

gRPC call	Affected WebUI setting
<pre>SetDestinationAddress("127.0.0.1");</pre>	
<pre>SetSatFrequencyHz(150000000);</pre> <p>*Note: Values for the function in [Hz]</p>	
<pre>SetBandwidthByAnalysisBandwidth(19410000);</pre> <p>*Note: Values for the function in [Hz]</p>	

Further examples on how to use the gRPC can be found in the Unit Tests in addition to detailed explanations in the README.md of the corresponding git Repository [rs-jerry-driver].

6 Data Format

6.1 Ethernet framing

There are two types of Ethernet framing inside the UDP/IP which you can choose.

The most useful Protocol HRZR is described in 6.2.

The general workflow using the R&S-Jerry-Driver is to:

- ▶ include the library header in your project
- ▶ configure your CMakeLists.txt
- ▶ login and configure the MSR4
- ▶ start to receive samples.

Receiving samples is as simple as calling the function

- ▶ `GetSamples(int number_of_samples, std::complex<float> *samples)`

on the client. This retrieves a maximum of `number_of_samples` items into `samples` and returns the actual number of items successfully stored in `samples`.

This function cannot be called endlessly as long as you wish to receive more data.

The R&S-Jerry-Driver repository also holds more detailed explanations and examples of this process in the form of documentation as well as UnitTests.

6.2 HRZR protocol

See User Manual MSR4.

6.2.1 Calculating Power Spectrum

The I/Q sample power can be calculated by the following:

```
scaled_fft_dbfs = get_scaled_fft(iq_samples, deci_data.F_SAMPLE_RATE,
center_gain_dbm)

dt = 1 / sample_rate_hz
n_fft = iq_samples.size
f = np.linspace(-1 / (2 * dt), 1 / (2 * dt), n_fft, endpoint=False)
plt.plot(f, scaled_fft_dbfs)
peak = np.max(scaled_fft_dbfs)

/* get_scaled_fft */
def get_scaled_fft(iq_samples: np.array, sample_rate_hz, center_gain_dbm):
    n_fft = iq_samples.size
    window = np.hanning(n_fft)
    fft_dbfs = calc_fft_dbfs(iq_samples, window, sample_rate_hz, True)
    scaled_fft_dbfs = (fft_dbfs - center_gain_dbm)
    return scaled_fft_dbfs

/* calc_fft_dbfs */
def calc_fft_dbfs(iq_samples: np.array, w: np.array, fs, shifted=False,
full_scale_bits=16-1):
    """ Calculates the fast Fourier transform (FFT) of I/Q samples relative to
RF-ADC/RF-DAC full-scale level (dBFS)

    :param iq_samples: I/Q samples (as np.array)
    :param w: Time-domain window (as np.array)
    :param shifted: Shift zero-frequency component to center of FFT spectrum
when set to true
    :return: Calculated FFT (in dBFS) and frequency vector
    """

    # Get number of I/Q samples
    nof_iq_samples: int = iq_samples.size
```

```

# Calculate FFT
n_fft = nof_iq_samples
dt = 1 / fs
fft = np.fft.fft(iq_samples * w) / n_fft

# Shift zero-frequency component to center of FFT spectrum
if shifted is True:
    f = np.linspace(-1 / (2 * dt), 1 / (2 * dt), n_fft, endpoint=False)
    fft = np.fft.fftshift(fft)
else:
    f = np.linspace(0, 1 / dt, n_fft, endpoint=False)

# Calculate power of time-domain window
w_pwr = float(20 * np.log10(sum(w) / w.size))

# Scale calculated FFT relative to RF-ADC/RF-DAC full-scale level (dBFS)
fft_dbfs = 20 * np.log10(abs(fft) / 2**full_scale_bits) - w_pwr

# Return
return fft_dbfs

```

7 Ordering Information

Designation	Type	Order No.
MSR4 - Satellite Monitoring Receiver	R&S®MSR4 (internal / external licensing dongle)	3066.8550.03 3066.8550.04
MSR4-IQ MSR4 IQ Streamer	R&S®MSR4-IQ	3066.8644.02

Rohde & Schwarz

The Rohde & Schwarz electronics group offers innovative solutions in the following business fields: test and measurement, broadcast and media, secure communications, cybersecurity, monitoring and network testing. Founded more than 80 years ago, the independent company which is headquartered in Munich, Germany, has an extensive sales and service network with locations in more than 70 countries.

www.rohde-schwarz.com



Rohde & Schwarz training

www.training.rohde-schwarz.com

Rohde & Schwarz customer support

www.rohde-schwarz.com/support

