Notes on the Kraken Software Defined Radio (SDR) Ethernet Server

Version 1

5 Dec 2025

For corrections, comments, questions, or general complaints please contact:

Bert Shelters

Member, IEEE

b.shelters.5@gmail.com

Document Change Log

Version	Date	Changes	Contributors
0	25 Nov 25	Initial release	B. Shelters
1	5 Dec 25	Corrected error in listed center frequency limit. The correct frequency limit is 27-1750 MHz	Adrian

Table of Contents

1. Background

1.1. Overview

What follows is a compilation of notes regarding the operating principles, concepts, and interface definitions for using the Kraken Software Defined Radio (SDR) with the optional ethernet server. This information was pulled together to support the development of a software application that converts the native Kraken In-Phase and Quadrature Phase (IQ) data packet to the standard SigMF data format for a hobby SDR project. The information assembled and summarized here is taken from various sources including the available technical documentation, forums, and inspection of the Kraken data acquisition source code. Since considerable research was required to obtain the necessary information to support this application development, these notes are offered to other hobbyist and researchers with the hope that it will be helpful to their purposes.

1.2. Background

The Kraken is a five-channel, phase coherent SDR for research and hobby applications to include direction finding, passive RADAR, and interferometry. To produce coherent channel data, the Kraken SDR uses custom Data Acquisition (DAQ) firmware (named Heimdall) which is typically installed on a dedicated Single Board Computer (SBC) (Such as a Raspberry Pi). The output of the DAQ is amplitude and phase corrected IQ data for each active antenna channel packed into an “IQ Packet” for transfer to a Digital Signal Processing (DSP) application. The default Kraken firmware distribution includes a Direction-of-Arrival (DoA) DSP application that is collocated with the DAQ on the SBC to enable near-real time application feedback.

For advanced use cases, it is often desirable to access the coherent IQ data directly for detailed analysis or for applying custom DSPs [1]. The DAQ implements an ethernet server that client DSP applications can connect to stream IQ packets. Figure 1 gives an example of a firmware architecture in which an external DSP application (In this case a Kraken SigMF writer) is connected to the DAQ SBC by a wired ethernet machine-to-machine connection.

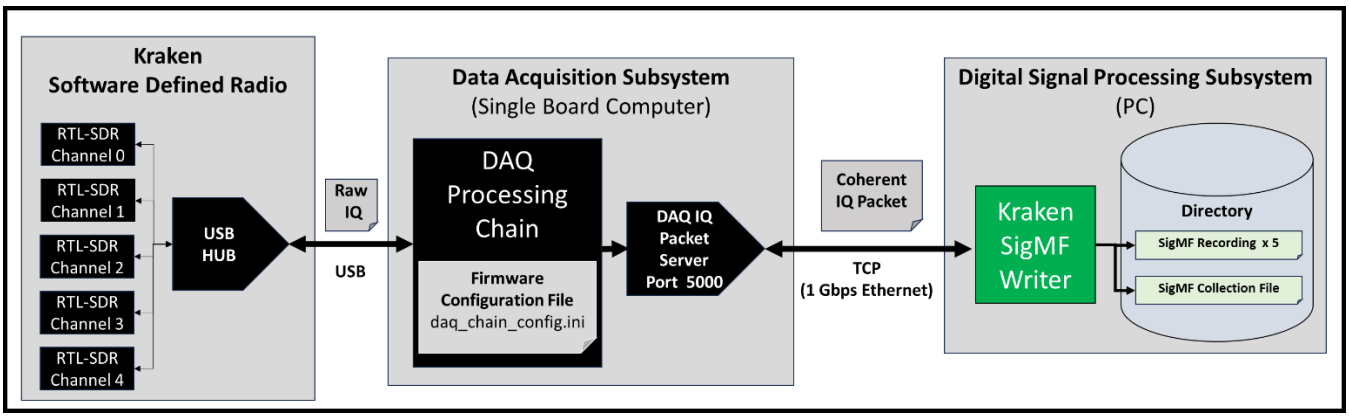

Figure 1: Firmware architecture for the external DSP use-case.

1.3. Comments on Real-Time vs Streaming vs Batch Processing

The computational and data transfer latencies incurred with the use of the ethernet server prohibit real-time processing or even reliable streaming in many cases. For DSP applications which rely on reliable streaming IQ data, such as the default DoA application delivered with the Kraken, the DAQ-internal shared-memory interface (vs the Ethernet server) should be used. The use of the shared memory interface however requires that the DSP application be deployed to the same computer as the DAQ. As SBCs such as the Raspberry Pi are typically used to run the DAQ software, DSP applications that run concurrent with the DAQ firmware on the same machine must support the custom Kraken shared memory interface and must further be optimized to support the limited computational resources available on the SBC (which also has to run the DAQ firmware at the same time). While highly optimized DSPs, such as the default DoA algorithm, can be deployed in such a way, for research and development it is often desirable to run DSPs on external computers with more computational power where advanced development environments can be more readily used.

It should be noted that the DAQ firmware does not support true real-time processing in any configuration as the DAQ processing chain uses buffers to perform batch processing on the raw data from each channel to produce coherent data (See Table 1 below). Thus, all data from the Kraken SDR is by design batch processed and the “cost” for coherence is the additional data processing latency. As the Kraken is purpose built for coherence, this “cost” is typically acceptable especially in coherent signal processing applications prioritize coherence quality over data latency.

Processing Method	Recommended with Kraken SDR?	IQ Packet Interface	Data Stream	DAQ – DSP Timing Relationship
Real-Time	No	N/A	Continuous	DAQ concurrent with immediate DSP
Streaming	Yes	Shared Memory	Accumulated	DAQ concurrent with minimal latency DSP
Batch	Yes	Ethernet Server	Accumulated	Asynchronous and separate DAQ and DSP

Table 1: Processing method support with the Kraken firmware architecture.

2. Configurations and Controls

An external DSP client can connect to the External Hardware Control interface (Port 5001) to send commands and IQ Packet Server (Port 5000) interface to retrieve streaming IQ Packet data. The streaming IQ Packet data is delivered as it becomes available according to the as-configured DAQ processing chain instance in the firmware configuration file. Once calibration is complete and data packets begin streaming, there are several packet header fields that the downstream application must monitor to ensure that only valid data is sent to the DSP application. [2,3]

The DAQ firmware is optimized for low latency streaming of coherent IQ data packets through the use of multi-threaded processing and shared memory interfaces. In order to keep the parallel processing chain in sync, the firmware configurations are not alterable after initialization. Thus to change any configuration, the DAQ firmware must be restarted with an updated configuration file. [2,3]

2.1. DAQ Firmware Configurations

The following sections provide recommendations for configuration settings for the external DSP application use case. While not a comprehensive list of all available configuration options, the configuration settings discussed below should be considered when selecting configuration options when the ethernet interface (vs the shared memory interface) is used.

2.1.1. Enabling the IQ Server

The default configurations were selected for use with the Kraken default DoA algorithm which relies on the DAQ shared-memory interface to send IQ packets from the DAQ firmware to the DoA algorithm on the same computer. The “out_data_iface_type” field in the “data_interface” group entry set in the daq_chain_config.ini firmware configuration file must be changed from “shmem” to “eth” to enable the IQ server as the output data interface for the DAQ. [2,3]

2.1.2. Setting the Payload Size

As the data transfer latency when using the ethernet server is typically longer than that of the shared memory interface, the IQ packet payload data size might require adjustment in order to accommodate data rate limitations. The data size of the IQ packet payload can be modified thru the Coherent Processing Interval (CPI) size and decimation ratio configurations. The CPI size is the number of samples per channel for a given IQ packet. The length of the CPI is set by the "cpi_size" field in the "pre-processing" group entry set in the daq_chain_config.ini firmware configuration file. The configuration selections and methodology by which the DAQ processing chain achieves the specified "cpi_size" has implications for downstream DSP applications and thus the evolution of the cpi_length parameter throughout the end-to-end DAQ processing chain is described below. [2,3]

For the purposes of illustration how CPI size changes through the DAQ processing chain, it is helpful to artificially split the DAQ processing chain up into three regimes according to when the number of complex samples in a CPI per channel (henceforth just referred to as number of samples for brevity) may be modified.

1. Initial ADC capture prior to Rebuffering
2. After Rebuffering prior to Decimation
3. After Decimation (Final value reported in packet header)

In the first regime the number of samples is set by the RTL DAQ module to be the integer "daq_buffer_size" field in the "daq" group entry set in the daq_chain_config.ini firmware configuration file. The default firmware configuration file value is 262144. The default value was selected to be optimal for the RTL DAQ module to minimize sample loss from the initial signal capture by the ADC hardware, therefore it is not recommended that this value be changed in nominal operations. [2,3]

In the second regime the number of samples may be changed by buffering. Since the initial RTL DAQ module is performed with buffers of a predetermined size purely for reasons of data processing efficiency and stability, the rebuffering module is required to accumulate and package up a sufficient number of samples to achieve the desired number of samples specified by the "cpi_size" field in the firmware configuration file. To achieve this the rebuffer module implements a circular buffer which also serves to insulate the RTL DAQ processes from the downstream DAQ processing to the handle the asynchronous data flows associated with real time processing. The number of samples for each channel is equal to the required cpi_size multiplied by the decimator factor such that after decimation (in the following processing module), the final number of samples will be equal to the required value. [2,3]

In third regime, decimation may be performed if the decimation ratio set by the firmware configuration file is not equal to one. At this point the number of samples is equal to the required value set by “cpi_size” in the firmware configuration file and the value is set as the cpi_length in the packet. [2,3]

It is often necessary to adjust the CPI size in order to satisfy downstream DSP applications or overcome processing latency limitations. While there are no DAQ buffering limits on maximum CPI size, practical processing throughput limits will soon be realized if too large of a CPI size is requested. For hobbyist applications, it is a best practice to minimize CPI size much as practical to overcome limited computational resources.

It is a best practice to set the CPI size to be an integer multiple of the ADC buffer size set by the “daq_buffer_size” field in the “daq” group entry set in the daq_chain_config.ini firmware configuration file. The reason for this is that the Rebuffer module sets up a ring buffer between the RTL DAQ and the Decimator module that acts to accumulate ADC buffer sized data segments until sufficient data is available to satisfy the required CPI Size after decimation. The Rebuffer module only takes the number of samples it requires to meet the CPI Size requirement, therefore data will be lost if only a fractional ADC buffer segment is used. Note that the “daq_buffer_size” field in the firmware configuration file should not be changed in nominal operations. [2,3]

The Decimation module down-samples a signal by the “decimation_ratio” field in the “pre_processing” group entry set in the daq_chain_config.ini firmware configuration file. Note that a decimation ratio of one means no data is discarded. The decimation approach maintains the time-domain duration of the CPI however the sampling rate (and therefore the instantaneous bandwidth) of the signal is reduced. Conversely, simply reducing the number of samples (reducing the CPI buffer size) reduces the time-domain duration of the received signal, but preserves the same sampling rate and thus maintains the same instantaneous bandwidth. The degree to which signal time duration reduction or frequency bandwidth reduction can be tolerated depends on the downstream DSP application. [2,3]

Both decimation and CPI buffer reduction are lossy processes meaning that the data is not recoverable. Since Decimation requires the DAQ module to perform additional signal processing, client DSP applications should trade whether the data volume and processing latency savings from the DAQ decimation application is warranted with the additional processing latency times incurred.

2.1.3. Setting the RF Center Frequency

The RF center frequency can be set by “center_freq” field in the “daq” group entry set in the daq_chain_config.ini firmware configuration file. The SDR can be tasked to any integer

frequency in Hz in the range 270,000,000 Hz to 1,750,000,000 Hz. The RF center frequency can also be changed at any time via the external control interface. [2,3]

2.1.4. Disabling Squelch Mode

The Kraken SDR supports a squelch mode, the operation of which are described in the Heimdall DAQ firmware documentation. If squelch processing is not required or desired, the en_squelch ” field in the “squelch” group entry set in the daq_chain_config.ini firmware configuration file should be set to zero to disable squelch mode. [2,3]

2.1.5. Managing Coherence Calibration

The requirement for the DAQ firmware is to provide coherent multi-channel data to an external DSP application. The DAQ firmware must be properly calibrated to account for channel mismatched timing delays inherent to real hardware and data acquisition processing latencies. The DAQ firmware automatically performs coarse (sample level) and fine (phase level) coherence calibration upon startup, using the internal noise source to derive correction factors to apply to the collected data. Once coherence is achieved according to the requirements set in the firmware configuration file, the DAQ firmware continues to measure coherence across each active channel for each IQ Packet produced. Over time updated correction factors may be needed as coherence quality may degrade across one or more channels. [2,3]

The DAQ firmware supports three options for tracking coherence quality and initiating calibration. Mode selection is made using the “cal_track_mode” field in the “calibration” group entry set in the daq_chain_config.ini firmware configuration file. [2,3]

In Mode 0 (cal_track_mode = 0) the DAQ firmware does not monitor the sample level (delay_sync flag) or the phase level (iq_sync_flag) coherence quality and coherence calibration is not performed after initialization. [2,3]

In Mode 1, the DAQ firmware monitors both sample and phase level calibration on every frame (both Data Frame and Calibration Frame). [2,3]

In Mode 2, the DAQ firmware sends a set number of Data frames followed by a set number (or burst) of calibration frames. Coherence quality is only measured for the calibration frames. The number of Data frames that passes before calibration occurs is set by the “cal_frame_interval” field in the “calibration” group entry set in the daq_chain_config.ini firmware configuration file and the number of Calibration frames in the Calibration burst is set by the “cal_frame_burst_size” field in the “calibration” group entry set in the daq_chain_config.ini firmware configuration file. [2,3]

Mode 2 is the default mode of operation and is recommended for most use cases. Mode 2 ensures that only Calibration frames with the internal noise source used as the reference illuminator are used to determine if calibration is required or not. If Data frames are used (as in Mode 1), real signal Time-Of-Arrival (TOA) differences at the antennas may trigger unnecessary calibrations. Figure 2 gives the logic flow for Mode 2 Calibration tracking. [2,3]

When measuring coherence, the Delay Synchronizer module measures coarse coherence at the signal sample level (sample synchrony check) and fine coherence at the signal phase level (IQ synchrony check). Successful coarse calibration is required to begin fine calibration. [2,3]

In the sample synchrony check, the cross correlation between the IQ data collected from two different channels is evaluated at two points: at zero offset and a non-zero offset (nominally 100 samples from zero). If the difference between these values is less than or equal to 20 dB, the sample synchrony check fails and the delay_sync_flag is set to integer zero. If the difference between theses values is greater than 20 dB, the sample synchrony check passes and the delay_sync_flag is set to integer one. The following fields in the “calibration” group entry set in the daq_chain_config.ini firmware configuration file are used in the sample synchrony check:

In the IQ synchrony check, the spatial correlation matrix is calculated between different channels to determine IQ amplitude and phase correction values. If the amplitude and phase correction values are below the configured threshold tolerance values, the IQ synchrony check passes and the iq_sync_flag is set to integer one. If either of these values are above the configured threshold value, the IQ synchrony check fails and the iq_sync_flag is set to integer zero. The amplitude and phase tolerances may be set by the “amplitude_tolerance” and “phase_tolerance” field in the “calibration” group entry set in the daq_chain_config.ini firmware configuration file. [2,3]

The hardware controller keeps count of the number of consecutive frames that fail either synchrony checks. If the number of consecutive frames exceeds the threshold set by the “maximum_sync_fails” field in the “calibration” group entry set in the daq_chain_config.ini firmware configuration file, the Hardware Controller sets the DAQ Firmware into coherence calibration. Five (this is a hard coded “NO_DUMMY_FRAMES” value in the source code) Dummy frames are sent through the DAQ processing chain at the beginning of calibration. [2,3]

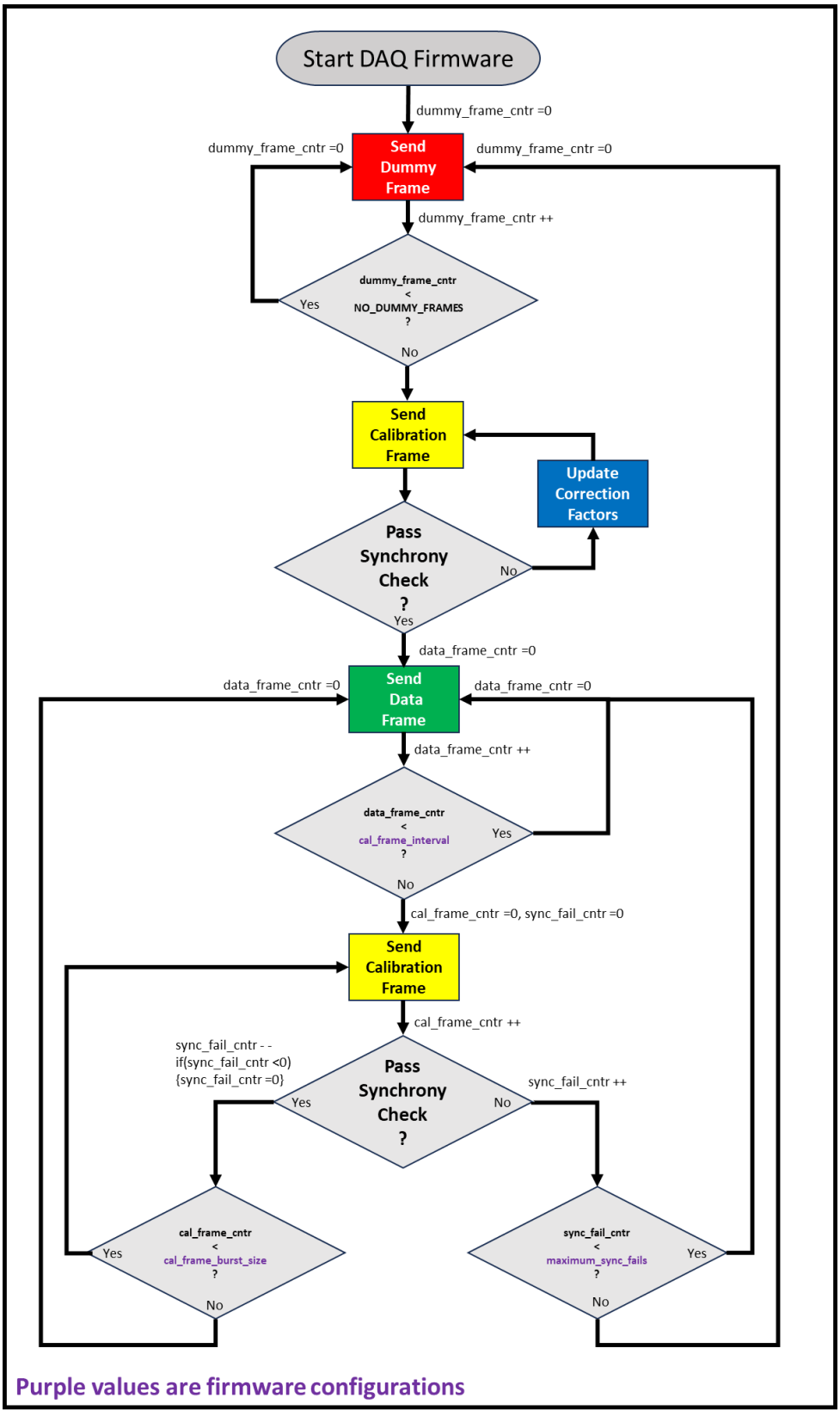

Figure 2: Calibration Tracking Mode 2 Logic Flow.

2.2. IQ Packet Ethernet Server

The IQ Packet Server consists of a TCP socket that can be accessed at port 5000 of the DAQ computer. Once the DSP client connects to the DSP server, the DSP server will send a single IQ packet for each request sent by the client. The messaging sequence between the DSP client and the DAQ server is given in Figure 3.

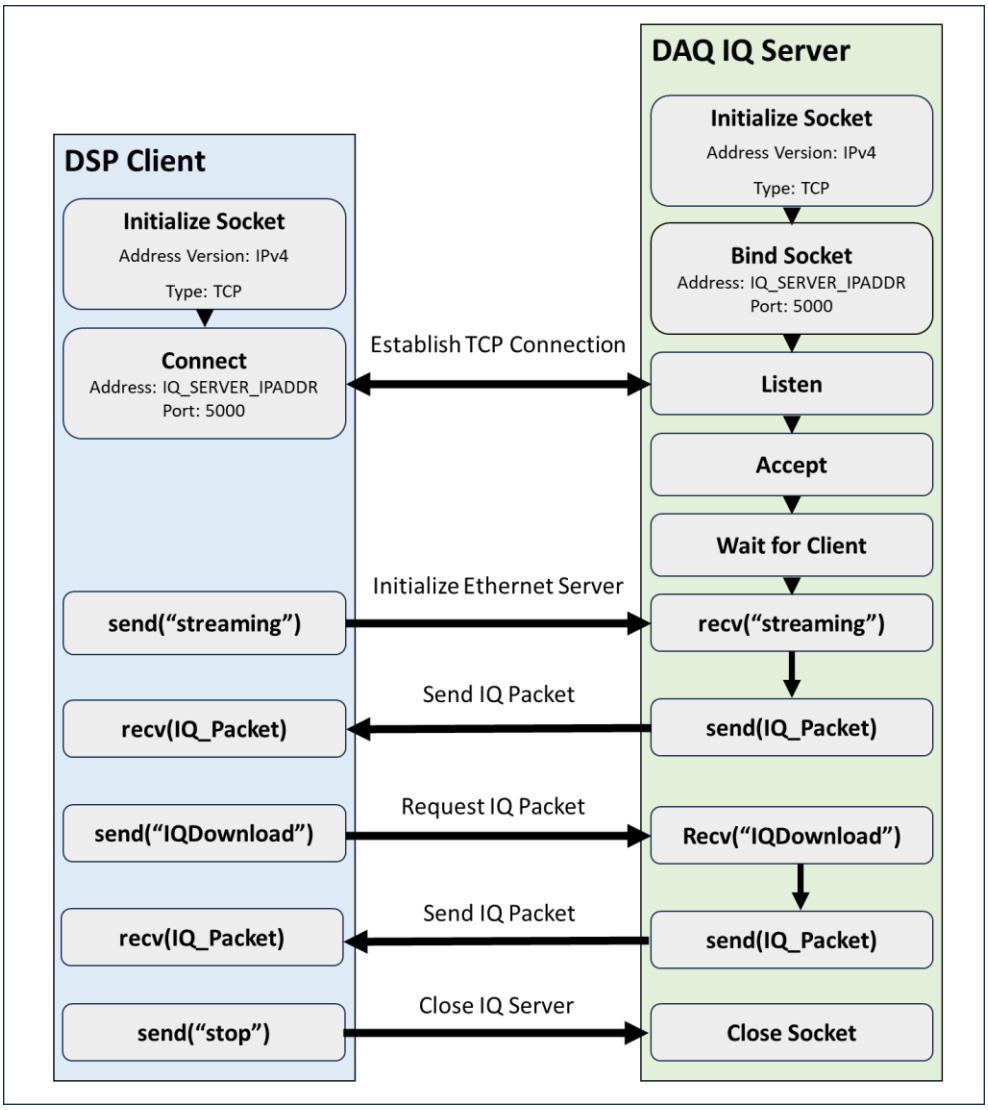

Figure 3: IQ Packet Ethernet Server messaging sequence.

Once the ethernet server is initialized, the server will shut down if any command other than "IQDownload" is received.

2.3. External Command Interface

The DAQ External Command Interface consists of a TCP socket that can be accessed at port 5001 of the DAQ computer. Valid command messages are 128 bytes total comprising of a 4 byte command word and a 124 byte parameter field as indicated in Figure 4 below. [2,3]

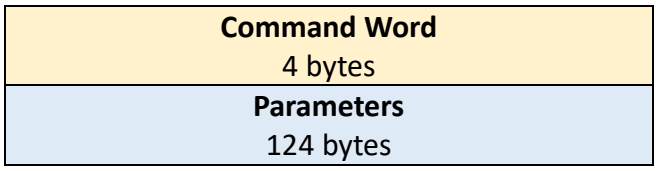

Figure 4: DAQ external command interface structure.

The messaging sequence between the DSP client and the DAQ Control Interface is given in Figure 5.

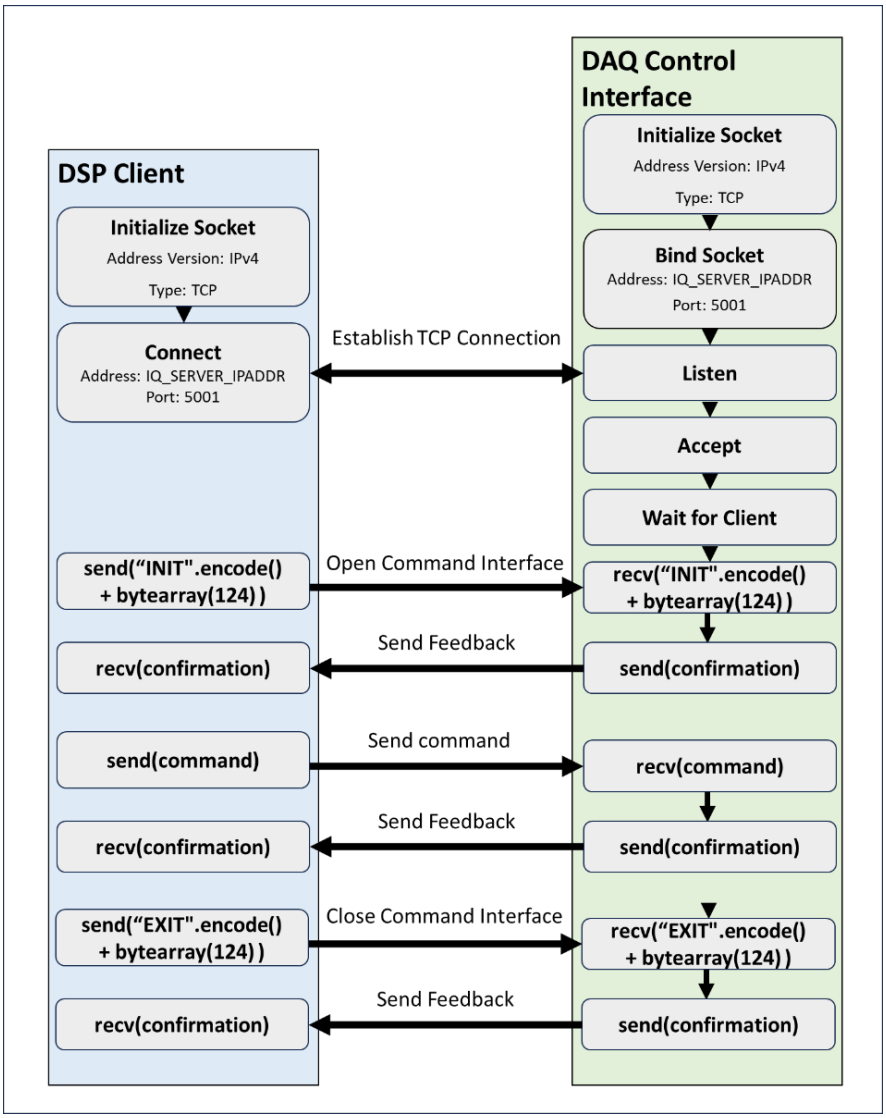

Figure 5: External Command Interface.

Valid commands are given in the table below. See the DAQ firmware documentation for more information regarding the squelch mode. The server acknowledges receipt of a valid command by returning to the client confirmation message consisting of the command word “FNSD” with an empty payload. [2,3]

Command Word	Parameters	Description
INIT	None (Empty)	Open the command interface
EXIT	None (Empty)	Close the command interface
STHU	Float	Sets threshold squelch value if squelch is used
FREQ	1x Uint64_t	Sets the center frequency in Hz. Valid settings are integers in the range 270,000,000 Hz to 1,750,000,000 Hz.
GAIN	5x Uint32_t	Sets the gain for each channel. Valid gain settings for the R820T Tuner are 0 (default), 9, 14, 27, 37, 77, 87, 125, 144, 157, 166, 197, 207, 229, 254, 280, 297, 328, 338, 364, 372, 386, 402, 421, 434, 439, 445, 480, and 496.

Table 2: Valid external command interface commands.

3. IQ Frame Description

3.1. Overview

The Heimdall DAQ firmware implements an IQ packet Ethernet server using the TCP protocol (See previous interface section). The server provides IQ packets in a device specific format consisting of the coherent IQ sample data from each active channel along with a common header for all channels containing relative metadata. The IQ packet data structure is both used internally within the shared memory “intra-face” of the DAQ modules as well as by the IQ server in external interfaces between the DAQ and a potentially remote DSP application. Therefore some of the IQ packet metadata fields used internally by the DAQ may be of little relevance for a downstream DSP application. This section provides a description of the IQ packet structure and relevant data fields. [2-4]

3.2. IQ Packet Structure

A visual description of the IQ packet structure is given in the Table 3 below. Note that this description is for the Heimdall DAQ IQ packet Version 7. [2-4]

IQ Packet Header HEADER_SIZE = 1024 bytes	sync word: Uint32_t x 1
	frame_type: Uint32_t x 1
	hardware_id: char [16]
	unit_id: Uint32_t x 1
	active_ant_chs: Uint32_t x 1
	ioo_type: Uint32_t x 1
	rf_center_freq: Uint64_t x 1
	adc_sampling_freq: Uint64_t x 1
	sampling_freq: Uint64_t x 1
	cpi_length: Uint32_t x 1
	time_stamp: Uint64_t x 1
	daq_block_index: Uint32_t x 1
	cpi_index: Uint32_t x 1
	ext_int_cnt: Uint64_t x 1
	data_type: Uint32_t x 1
	sample_bit_depth: Uint32_t x 1
	adc_overdrive_flag: Uint32_t x 1
	if_gains: Uint32_t x 32
	delay_sync: Uint32_t x 1
	iq_sync_flag: Uint32_t x 1
	sync_state: Uint32_t x 1
	noise_source_state: Uint32_t x 1
	reserved: Uint32_t x 192
	header_version: Uint32_t x 1
IQ Packet Payload PAYLOAD_SIZE [bytes] = CPI_SIZE [# Samples] x 2 [IQ sample size] x NO_CHS [# Channels] x SAMPLE_BIT_DEPTH [bits/sample] / 8 [bits/byte]	Channel 0 IQ Data: 32-bit complex float x CPI_SIZE
	Channel 1 IQ Data: 32-bit complex float x CPI_SIZE
	Channel 2 IQ Data: 32-bit complex float x CPI_SIZE
	Channel 3 IQ Data: 32-bit complex float x CPI_SIZE
	Channel 4 IQ Data: 32-bit complex float x CPI_SIZE

Table 3: IQ Packet Structure.

The size of the IQ packet header is 1024 bytes and the data type and interpretation of each field is provided in the section below. The size of the IQ packet payload depends on the size of the Coherent Processing Interval (CPI_SIZE) and number of active channels (NO_CHS). CPI_SIZE and NO_CHS are user configurations which can be set in the daq_chain_config.ini firmware configuration file. For typical operations in which case all five channels of the Kraken SDR are used, the payload size as a function of CPI_SIZE can be calculated as follows: [2-4]

PAYLOAD_SIZE [bytes] = CPI_SIZE [# Samples] x 40 [bytes/sample]

The total packet size is then calculated as

PACKET_SIZE [bytes] = HEADER_SIZE [bytes] + PAYLOAD_SIZE [bytes]

PACKET_SIZE [bytes] = 1024 [bytes] + CPI_SIZE [# Samples] x 40 [bytes/sample]

With the default CPI_SIZE (1048576 samples), the packet size would be approximately 42 MB.

3.3. IQ Packet Header

The IQ Packet Header consists of a total of 24 metadata fields. All coherent channels in the packet payload share a single header (IE there is not a separate header for each file). The table below gives the byte indices for each metadata field. Note that the packet header includes padding since the packet header is defined as a struct in C code which adds padding to maintain memory alignment. [2-4]

Start Byte	Field (Size)	End Byte
0	sync_word – uint32_t – 4 bytes	3
4	frame_type – uint32_t – 4 bytes	8
8	hardware_id – char x [16] – 16 bytes	23
24	unit_id – uint32_t – 4 bytes	27
28	active_ant_chs -uint32_t – 4 bytes	31
32	ioo_type – uint32_t – 4 bytes	35
36	Padding – 4 bytes	39
40	rf_center_freq – uint64_t – 8 bytes	47
48	adc_sampling_freq – uint64_t – 8 bytes	55
56	sampling_freq – uint64_t – 8 bytes	63
64	cpi_length – uint32_t -4 bytes	67
68	Padding – 4 bytes	71
72	time_stamp – uint64_t – 8 bytes	79
80	daq_block_index – uint32_t – 4 bytes	83
84	cpi_index – uint32_t – 4 bytes	87
88	ext_int_cnt – uint64_t – 8 bytes	95
96	data_type – uint32_t – 4 bytes	99
100	sample_bit_depth – uint32_t – 4 bytes	103
104	adc_overdrive_flag – uint32_t – 4 bytes	107
108	if_gains – uint32_t x [32] – 128 bytes	235
236	delay_sync_flag – uint32_t – 4 bytes	239
240	iq_sync_flag – uint32_t – 4 bytes	243
244	sync_state_flag -uint32_t – 4 bytes	247
248	noise_source_state – uint32_t – 4 bytes	251
252	reserved – uint32_t x [192] – 768 bytes	1019
1020	header_version – uint32_t – 4 bytes	1023

Table 4: Byte addresses for IQ packet header fields.

3.3.1. sync_word

Byte Index	Field	C Data Type	Data Size (bytes)
0-3	sync_word	Uint32_t	4

Description

sync_word is a static field used to indicate the start of an IQ Header packet. The RTL DAQ module assigns sync_word the hexadecimal value 0x2bf7b95a (737,655,130 decimal) during packet initialization. sync_word is not modified by any module. [2-4]

Usage

sync_word is used to identify the start of a valid packet by a receiving application.

3.3.2. frame_type

Byte Index	Field	C Data Type	Data Size (bytes)
4-7	frame_type	Uint32_t	4

Description

frame_type field encodes the type of the packet. The encoding scheme is given in the table below. The RTL DAQ module assigns this field a value of 0 for normal data frame at packet initialization. The RTL DAQ module then updates the field to update according to the type of packet requested by the hardware controller. [2-4]

Integer Value	Frame Type	Description
0	Data	Normal data acquisition
1	Dummy	Dummy frames are sent by the RTL DAQ module to clean up the processing chain ahead prior to calibration. Dummy frames include valid headers with default values for the IQ payload.
2	Ramp	This frame type is not used by the RTL DAQ module in nominal operations. This value is likely reserved by the developer of the RTL DAQ module for development testing with a ramp waveform excitation.
3	Cal	Calibration frames are sent when the RTL DAQ module is in calibration mode with noise source activated. Calibration is a

		dynamic process that involves recursive adjustments with multiple calibration frames to achieve sample and IQ calibration. Note that after initial calibration, the FSM may enter calibration again if sample and IQ calibration is determined to be lost.
4	Trigw	This frame type is not used by the RTL DAQ module in nominal operations. This value is likely reserved by the developer of the RTL DAQ module for development testing with a triangle waveform excitation.

Usage

Downstream DSP applications should only accept Data (normal) frame types for their purposes. Since the state of the FSM is not predictable, DSP applications should check this field with each packet received to ensure only valid data is processed.

3.3.3. hardware_id

Byte Index	Field	C Data Type	Data Size (bytes)
8-23	hardware_id	char [16] array	16

Description

hardware_id is a text field that provides the short denomination of the DAQ system. The RTL DAQ module replicates the ASCII “name” field in the “hw” group entry set in the daq_chain_config.ini firmware configuration file. The default field for the Kraken SDR is “kraken5.” hardware_id is not modified by any other module. [2-4]

Usage

Certain metadata standards for SDR data formats include a field for system name.

3.3.4. unit_id

Byte Index	Field	C Data Type	Data Size (bytes)
24-27	unit_id	Uint32_t	4

Description

unit_id is the unique serial number of the DAQ system. The RTL DAQ module replicates the integer “unit_id” field in the “hw” group entry set in the daq_chain_config.ini firmware configuration file. The default value is 0. unit_id is not modified by any other module. [2-4]

Usage

This field is potentially useful if one is using multiple identical DAQ systems simultaneously to disambiguate the packets produced from each individual device.

3.3.5. active_ant_chs

Byte Index	Field	C Data Type	Data Size (bytes)
28-31	active_ant_chs	Uint32_t	4

Description

active_ant_chs is the number of receive channels processed. The RTL DAQ module replicates the integer “num_ch” field in the “hw” group entry set in the daq_chain_config.ini firmware configuration file. The default value is 5 which is the maximum number of channels supported by the Kraken SDR hardware. active_ant_chs is not modified by any other module. [2-4]

Usage

Understanding of the number of channels is critical for multichannel signal processing applications.

3.3.6. ioo_type

Byte Index	Field	C Data Type	Data Size (bytes)
32-35	ioo_type	Uint32_t	4

Description

ioo_type is the type of the Illuminator of Opportunity (IOO) used in passive radar mode. The RTL DAQ module replicates the integer “ioo_type” field in the “hw” group entry set in the daq_chain_config.ini firmware configuration file. The default value is 0. Ioo_type is not modified by any other module. [2-4]

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field. The passive radar application is no supported due to International Traffic in Arms Regulations (ITAR).

3.3.7. rf_center_freq

Byte Index	Field	C Data Type	Data Size (bytes)
40-47	rf_center_freq	Uint_64t	8

Description

rf_center_freq is the Radio Frequency (RF) center frequency of the receiver channel given as integer hertz. The RTL DAQ module replicates the integer “center_freq” field in the “daq” group entry set in the daq_chain_config.ini firmware configuration file at packet initialization. The default firmware configuration file value is 700000000. The RF center frequency may be changed by the external command interface at any time and the RTL DAQ module updates the rf_center_freq value if a valid command is received by the RTL DAQ module. [2-4]

Usage

Knowledge of the RF center frequency is required to properly index intermediate frequency bins.

3.3.8. adc_sampling_freq

Byte Index	Field	C Data Type	Data Size (bytes)
48-55	adc_sampling_freq	Uint_64t	8

adc_sampling_freq is the Analog-to-Digital (ADC) sampling frequency given as integer hertz. The RTL DAQ module replicates the integer “sample_rate” field in the “daq” group entry set in the daq_chain_config.ini firmware configuration file at packet initialization. The default firmware configuration file value is 2400000. adc_sampling_freq is not modified by any other module. [2-4]

Usage

adc_sampling_freq is the sample rate used by the RTL2832U hardware at signal capture. The adc_sampling_freq must be set sufficiently high-enough to avoid aliasing depending on the frequency content of the signal of interest. The adc_sampling_freq is not the final sampling rate of the IQ data in the packet payload however, as further processing in the Decimate module performs decimation which reduces the sampling rate of the data. The sampling rate of the IQ data (after decimation) is given by the field "sampling_freq."

3.3.9. sampling_freq

Byte Index	Field	C Data Type	Data Size (bytes)
56-63	sampling_freq	Uint_64t	8

Description

sampling_freq is the sampling frequency given as integer hertz. The RTL DAQ module replicates the integer "sample_rate" field in the "daq" group entry set in the daq_chain_config.ini firmware configuration file at packet initialization. The default firmware configuration file value is 2400000. The initial sampling_freq value is the raw ADC sampling frequency at signal capture. [2-4]

For data type frames, sampling_freq is updated by the decimator module after decimation is performed. The decimation operation reduces the number of samples to reduce computational resources. The amount of decimation performed is determined by the "decimation_ratio" field in the "pre-processing" group entry set in the daq_chain_config.ini firmware configuration file. The updated sampling rate is determined by the following formula. [2-4]

$$SAMPLING\_FREQ = ADC\_SAMPLING\_FREQ / DECIMATION\_RATIO$$

Decimation is not performed on calibration type frames thus for calibration type frames sampling_freq is not updated past default. [2-4]

Usage

Knowledge of the sampling frequency is required to properly index intermediate frequency bins.

3.3.10. cpi_length

Byte Index	Field	C Data Type	Data Size (bytes)
64-67	cpi_length	Uint32_t	4

Description

cpi_length is the number of complex IQ samples in a Coherent Processing Interval (CPI). The RTL DAQ, rebuffer, and decimator modules are designed such that the number of complex IQ samples for each channel in each packet is equal to the number specified by the "cpi_size" field in the "pre-processing" group entry sent in the daq_chain_config.ini firmware configuration file. See the section above entitled "Setting the Coherent Processing Interval (CPI)" for an overview on how cpi_length is set. [2-4]

Usage

Downstream DSP applications need to be able to predict the number of complex IQ samples in order to set up appropriately sized data structures. cpi_length can be adjusted to reduce data volume throughput if processing latencies are insufficient for an application.

3.3.11. time_stamp

Byte Index	Field	C Data Type	Data Size (bytes)
72-79	time_stamp	Uint64_t	8

Description

time_stamp is the number of integer milliseconds since the Unix epoch at packet creation. The Rebuffer module estimates the start of the CPI by subtracting the time duration of the CPI from the last timestamp updated from the RTL DAQ module at data capture. Note that is estimation is required since the final CPI usually consists of one or more non-continuous chunks of data captured that are accumulated (buffered) to meet the desired CPI size. The longer the CPI, the less accurate this approximation is as more accumulation processing is required. [2-4]

Usage

The value of time_stamp indicates the approximate start time of the CPI from which all other samples may be indexed using the sampling period.

3.3.12. daq_block_index

Byte Index	Field	C Data Type	Data Size (bytes)
80-83	daq_block_index	Uint32_t	4

Description

daq_block_index is the circular buffer index for IQ packets prior to rebuffering. It used internally by the RTL DAQ and Rebuffer modules. The daq_block_index is set by the RTL DAQ module in the "daq_buffer_size" IQ packet headers at initial signal capture. One or more "daq_buffer" IQ packets are typically combined by the Rebuffer module to create a new "cpi_size" IQ Packet to satisfy the required "cpi_size." Since the same IQ packet header structure is used for "daq_buffer_size" and "cpi_size", the last value for daq_block_index is passed along in the "cpi_size" IQ packet header to create a valid IQ packet. [2-4]

Usage

daq_block_index is an internal field used in the DAQ signal processing chain. While a required field in the IQ header, downstream DSP applications should ignore this field.

3.3.13. cpi_index

Byte Index	Field	C Data Type	Data Size (bytes)
84-87	cpi_index	Uint32_t	4

Description

cpi_index is the sequential index of an IQ packet created by an instantiation of the DAQ processing chain. The first cpi_index is set to zero for the first packet by the Decimator module and the value is increased by one for each subsequent packet. The Delay Synchronizer FSM uses cpi_index to log calibration events. Note that cpi_index is updated regardless of frame type (Data, Calibration, or Dummy) therefore a series of valid Data frames may not have continuous cpi_index increasing values as calibration may occur in between acquisition of Data frames. [2-4]

Usage

DSP applications can use cpi_index to quickly determine the relative chronology of a set of IQ frames.

3.3.14. ext_int_cnt

Byte Index	Field	C Data Type	Data Size (bytes)
88-95	ext_int_cnt	Uint64_t	8

Description

ext_int_cnt is an “extended integration counter”. ext_int_cnt is set to zero by the RTL DAQ module with the code comment “Extended integration counter is not used by RTL-DAQs” and is not used by any other modules in the DAQ processing chain. The use of this field is not mentioned in the Heimdall DAQ firmware documentation. [2-4]

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field.

3.3.15. data_type

Byte Index	Field	C Data Type	Data Size (bytes)
96-99	data_type	Uint32_t	4

Description

data_type encodes the type of IQ data contained in IQ packet according to the coding scheme given in the table below. The RTL DAQ module sets data_type to 1 at packet initialization and then reassigns the value to either 0 for dummy data or 1 for normal IQ data used in calibration or data frames. The Decimation module sets data_type to 3 after it executes. Note that a data_type value of 3 does not necessarily mean that the resulting IQ data has been decimated, but rather a value of 3 only indicates that the Decimation module was executed. For example, if the decimation ratio in the firmware configuration file is set to a value of 1 (No decimation), decimation processing and filtering will not be performed, and the value for the data_type field in the resulting IQ header will still be set to 3. [2-4]

Integer Value	Data Type
0	Dummy Data
1 or 2	IQ Data Before Decimation Processing
3	IQ Data After Decimation Processing

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field.

3.3.16. sample_bit_depth

Byte Index	Field	C Data Type	Data Size (bytes)
100-103	sample_bit_depth	Uint32_t	4

Description

sample_bit_depth gives the number bits per sample in the IQ packet. The RTL DAQ Module initializes sample_bit_depth to 8 since the RTL2832U chip natively produces 8 bits In-Phase and 8 bist Quadrature-Phase samples. The Decimator Module increases the sample bit depth up to be 32 bits In-Phase and 32 bits Quadrature-Phase. Note that the increase in bet depth occurs irrespective of whether or not decimation processing and filtering is actually performed. Sample bit depth is also not a configurable or tasking parameter since the DAQ firmware uses standard DSP libraries which accept a standard 32-bit complex data format.

Usage

DSP applications require knowledge of the bit depth used in the IQ packet in order to successful ingest the wideband data.

3.3.17. adc_overdrive_flags

Byte Index	Field	C Data Type	Data Size (bytes)
104-107	adc_overdrive_flags	Uint32_t	4

Description

The adc_overdrive_flags is a flag that indicates if an ADC module becomes saturated during the course of a data collection. Saturation occurs when the voltage received by the ADC exceeds the maximum input voltage. Saturation is typically detected when the ADC returns a maximum sample value (255 is the maximum value for the 8 bit RTL2832U chip). The RTL DAQ module checks every sample of the received IQ data for saturation values (maximum value). If a saturation value is discovered, a binary encoding scheme is used to convey which of the independent channels is experiencing saturation. The encoding scheme is given in the table below. Starting with an 8-bit integer initialized to zero, the corresponding bit is toggled from zero (no saturation detected) to one (saturated value detected) to indicate that a saturation value was detected on that channel. Once a bit is toggled during a CPI, it remains in that state. The 8-bit value is converted to a 32 bit integer for the packet header. The Rebuffer module accumulates the adc_overdrive_flags from each of the “daq_buffer” IQ packets which are

combined to formed the final “cpi_size” IQ packets. Indications of ADC saturation are critical for calibration and receiver gain tuning by the Hardware Control module which attempts to set the maximize gain in order to minimize quantization noise. In receiver gain tuning, the Hardware Control module gradually decreases the gain settings for a given channel until only an acceptable number of consecutive frames (set by the “gain_lock_interval” field in the “calibration” group entry set in the daq_chain_config.ini firmware configuration file) contain saturation values. [2-4]

Bit	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Value	0 (Unused)	0 (Unused)	0 (Unused)	Channel 4	Channel 3	Channel 2	Channel 1	Channel 0

Usage

Downstream DSP applications should reject IQ packets with saturated data. After initialization, the Hardware Control FSM does not actively monitor for saturation values therefore it is incumbent on the DSP application to determine if gain tuning is required. If gain tuning is required, the DAQ processing chain must be re-initialized.

3.3.18. if_gains

Byte Index	Field	C Data Type	Data Size (bytes)
108-235	if_gains	Uint32_t [32]	128

Description

if_gains are the Intermediate Frequency (IF) gain values for the receiver channels. This field is an array of size 32 where each element of the array contains the integer gain value for the corresponding index channel (IE array index 0 is the gain value for channel 0). Note that since the Kraken SDR has five channels, only five elements of the array are used. The RTL DAQ module identically initializes the if_gains for each channel as the “gain” field in the “daq” group entry set in the daq_chain_config.ini firmware configuration file. The default gain value in the firmware configuration file is zero. Valid gain settings for the R820T Tuner are 0 (default), 9, 14, 27, 37, 77, 87, 125, 144, 157, 166, 197, 207, 229, 254, 280, 297, 328, 338, 364, 372, 386, 402, 421, 434, 439, 445, 480, and 496. if_gains is updated after initial gain tuning by the Hardware Control module but remains static after that. [2-4]

Usage

DSP applications can ignore this field so long as the Semi-Automatic Gain Control (Semi-AGC) is successful in limiting the number of frames with saturated data. Should the Semi-AGC approach yield an unacceptable number of frames with saturated data, manual approaches to setting receiver gain may be attempted via the external command interface.

3.3.19. delay_sync_flag

Byte Index	Field	C Data Type	Data Size (bytes)
236-239	delay_sync_flag	Uint32_t	4

Description

delay_sync_flag is a flag that indicates whether or not the IQ data in the packet has passed the sample synchrony check. The sample synchrony check is one of two checks that the Delay Synchronizer module performs to determine if calibration is required to maintain coherent operation. The sample synchrony check in particular measures coarse coherence at the discrete signal sample level (vs fine phase coherence – see iq_sync_flag). If automatic calibration is enabled, the Delay Synchronization FSM will count the number of packets that fail the sample synchrony check and start calibration when the count of consecutive packets that failed the synchrony check surpasses a set threshold. [2-4]

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field.

3.3.20. iq_sync_flag

Byte Index	Field	C Data Type	Data Size (bytes)
240-243	iq_sync_flag	Uint32_t	4

Description

iq_sync_flag is a flag that indicates whether or not the IQ data in the packet has passed the IQ synchrony check. The IQ synchrony check is one of two checks that the Delay Synchronizer module performs to determine if calibration is required to maintain coherent operation. The IQ synchrony check in particular measures fine coherence at the signal phase level (vs coarse coherence at the discrete sample level – see delay_sync_flag). If automatic calibration is enabled, the Delay Synchronization FSM will count the number of packets that fail the IQ

synchrony check and start calibration when the count of consecutive packets that failed the synchrony check surpasses a set threshold. Note that IQ synchrony check is only performed after sample synchrony is confirmed. [2-4]

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field.

3.3.21. sync_state

Byte Index	Field	C Data Type	Data Size (bytes)
244-247	sync_state	Uint32_t	4

Description

sync_type field encodes the state of the Delay Synchronizer Finite State Machine (FSM) at the time of packet creation. The FSM logic is described in the "HeIMDALL DAQ Firmware" documentation (See references). The encoding scheme is given in the table below. The RTL DAQ module assigns this field a value of zero at packet initialization. The Delay Synchronizer Module then updates the field to update according to the state of the FSM. [2-4]

Integer Value	FSM State	Short Description
0	None	sync_state is initialized to zero at packet creation
1	STATE_INIT	Wait for calibration signal (First state)
2	STATE_SAMPLE_CAL	Perform sample synchrony test
3	STATE_SYNC_WAIT or STATE_FRAC_SAMPLE_CAL	Wait for Sync Module to update delay samples according to the sample synchrony test result
4	STATE_IQ_CAL	Perform IQ sample synchrony test and adjust IQ data with correction values
5	STATE_TRACK_LOCK	Calibration is complete, delay_sync_flag and iq_sync_flag are set, Calibration signal is turned off, DAQ module starts to produce DATA (vs calibration) packets
6	STATE_TRACK	Nominal data flow, sample and IQ synchrony test are performed, monitors number of consecutive synchrony test failures to trigger calibration again (STATE_INIT)

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field. DSP applications should only accept Data (normal) frame types identified with the frame_type field (see frame_type field) above.

3.3.22. noise_source_state

Byte Index	Field	C Data Type	Data Size (bytes)
248-251	noise_source_state	Uint32_t	4

Description

noise_source_state indicates whether or not the internal calibration noise source is active (integer value one) or inactive (integer value zero). When the Hardware Control module sends commands to RTL DAQ module to activate the noise source, the frame type is set to Calibration (see frame_type field). Calibration frames are used in delay synchrony and IQ synchrony calibration. [2-4]

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field. DSP applications should only accept Data (normal) frame types identified with the frame_type field (see frame_type field) above.

3.3.23. reserved

Byte Index	Field	C Data Type	Data Size (bytes)
252-1019	reserved	Uint32_t [192]	768

Description

The reserved field is allocated in the IQ packet header that is set aside for future use. This field is not used for any purpose in IQ packet header version 7. [2-4]

Usage

While a required field in the IQ header, downstream DSP applications should ignore this field.

3.3.24. header_version

Byte Index	Field	C Data Type	Data Size (bytes)
1020-1023	header_version	Uint32_t	4

Description

header_version provides the integer valued version number for the IQ Packet Header specification used in a packet. Version 7 is the IQ Packet Header specification described in this user guide. [2-4]

Usage

Downstream DSP applications should confirm that their respective IQ packet capture application is properly configured to the same IQ Packet Header Specification as the DAQ processing chain.

3.4. IQ Packet Payload

Each complex IQ sample in the IQ Packet payload is stored as a 32-bit complex float format where both the In-Phase (I) and Quadrature-Phase (Q) components are separate little endian 32-bit IEEE 754 standard floating-point numbers. The I and Q component for each sample are interleaved (starting with the in-phase component) and data from each channel is stored sequentially starting with channel 0 as shown in the Table 5 below. [2-4]

Channel 0, Sample 0 In-Phase Component: 32-bit float
Channel 0, Sample 0 Quadrature Phase Component: 32-bit float
Channel 0, Sample 1 In-Phase Component: 32-bit float
Channel 0, Sample 1 Quadrature Phase Component: 32-bit float
Channel 0, Sample 2 In-Phase Component: 32-bit float
Channel 0, Sample 2 Quadrature Phase Component: 32-bit float
. . .
Channel 0, Sample CPI_SIZE-1 In-Phase Component: 32-bit float
Channel 0, Sample CPI_SIZE-1 Quadrature Phase Component: 32-bit float
Channel 1, Sample 0 In-Phase Component: 32-bit float
Channel 1, Sample 0 Quadrature Phase Component: 32-bit float

Channel 1, Sample 1 In-Phase Component: 32-bit float
Channel 1, Sample 1 Quadrature Phase Component: 32-bit float
. . .
Channel 4, Sample CPI_SIZE-1 In-Phase Component: 32-bit float
Channel 4, Sample CPI_SIZE-1 Quadrature Phase Component: 32-bit float

Table 5: IQ Packet Payload Structure.

4. Disclaimer

The author of these notes is not associated with the Kraken SDR project or development team and any statements, thoughts, and opinions are the author's and do not reflect the Kraken SDR project or the development team. This information is provided in the hope that it will be helpful for other researchers and hobbyists in the spirit of open source development. These notes are provided without any guarantees, and corrections if identified should be reported to the author via email. Should there be any inconsistencies between the information contained in this report and the official Kraken SDR documentation, the official Kraken SDR documentation shall prevail.

5. Sources

1. *How to install and run gr-krakensdr natively on Window.* (2023, April 7). Redgo. Retrieved November 25, 2025, from Microsoft Word - gr-krakensdr_natively_on_Windows_(V1.0_230407).docx.
2. Pető, T. (2025, November 11). *HeIMDALL DAQ Firmware*. heimdall_daq_firmware Github Repository. heimdall_daq_fw/Documentation/HDAQ_firmware_ver1.0.20201130.pdf at main · krakenrf/heimdall_daq_fw.
3. Pető, T. and others. *Heimdall DAQ Firmware Source Code (IQ Packet Version 7).* heimdall_daq_firmware Github Repository. krakenrf/heimdall_daq_fw: Coherent data acquisition signal processing chain for multichannel SDRs.
4. Pető, T. (2025, November 11). *IQ Data Framing*. krakensdr_docs Github Repository. krakensdr_docs/misc_docs/IQFraming_v1p1.pdf at main · krakenrf/krakensdr_docs.