HeIMDALL DAQ Firmware

(Data Acquisition Subsystem for Kerberos SDR)

Document version:	0.9.20201130
Author:	Tamás Pető
Kerberos SDR version:	2.0
Kerberos hardware version:	2.0
DAQ Subsystem firmware version:	2.0rc
DSP Subsystem version:	-
Changelog:	
0.1.20191019	Initial version
0.2.20191117	Updated for firmware version:0.1.dev8, DAQ Subsystem - Calibration DAQ Subsystem - FSM
0.2.1.20191217	IQ header updated according to firmware version : 0.2.dev1
0.3.20200208	Updated for firmware version 0.2.1.dev
1.0.20201130	Updated for firmware version 2.0rc Prepare for release

1 Top Level System Overview

The main goal of this comprehensive documentation is to introduce the architecture of the Kerberos SDR and the operation of the Data Acquistion Firmware. From this document the reader can understand the basic concepts and design considerations.

In this section we are going to introduce the basic concepts of the Kerberos SDR based software radio architecture. The suggested system can be distributed to four main parts. To the antenna system, the DAQ (Data Acquisition) subsystem, the DSP (Digital Signal Processing) subsystem and to a Display unit. The antenna system is connected to the DAQ Subsystem through coaxial cables, while the DAQ-DSP and the DSP-Display connections can be standard Ethernet links or they can run on the same processing machine.

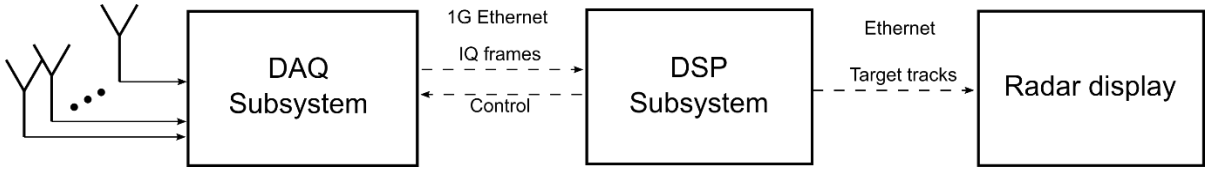

Figure 1. Block diagram of the suggested system

The main goal of the module separation is to improve the data throughput, the reliability and the testability of the full system. Since the data acquisition is a time critical task in most of the signal processing application, it has to be implemented in a dedicated processing module. It has to hand over the raw collected data in a given time period, otherwise data loss occurs, which cannot be allowed in coherent receiver systems. Similarly, the DSP tasks could be also time critical considering a quasi-real-time application. To this end, it is advisable to provide a dedicated processing unit to these tasks as well. Beside this, the radio system (DAQ and DSP) itself can also be deployed at a distant place from the operator, hence we have to ensure the capability to display the results on a separated terminal which in not part of computational critical processing units. In the next section we are discussing the responsibilities of these main submodules.

2 DAQ Subsystem

The DAQ Subsystem is responsible for the analog signal reception, digitalization and for the transfer of the digitalized data. From the hardware structure point of view, the core components of the DAQ Subsystem are the Kerberos SDRs and an SBC (Single Board Computer). A simplified block diagram of the data flow is illustrated in Figure 2.

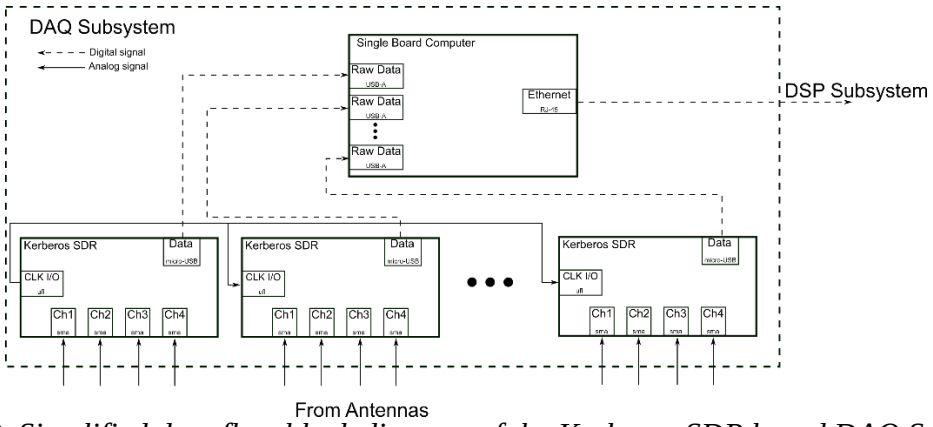

Figure 2. Simplified data flow block diagram of the Kerberos SDR based DAQ Subsystem

Each of the Kerberos SDR receivers contains four coherent RF receiver channels. At each end of the analog signal paths an ADC (Analog-to-Digital Converter) digitalize the signals and forwards the digital data through a USB connection. These data channels are merged into a single USB data lane with the use of a USB HUB. The block diagram of this data flow can be seen in Figure 3. These USB data lanes are then connected to an SBC, which realize the data acquisition.

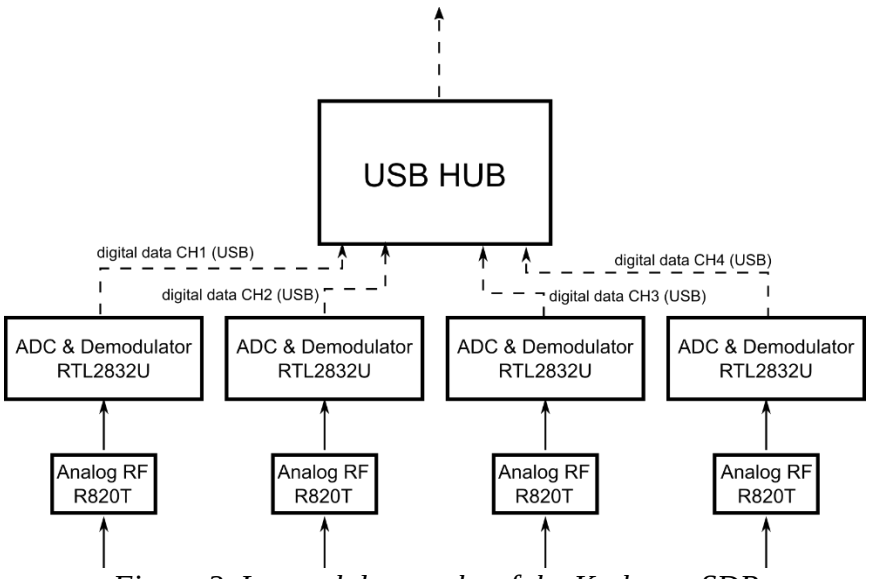

Figure 3. Internal data paths of the Kerberos SDR

On the SBC side the raw received digital data is packed into data frames and then transferred to the DSP Subsystem for further processing. The data acquisition is a time critical tasks as all the

samples have to be handed over from the USB data buffers to prevent data override and hence sample loss. Taking these considerations into acount the computational heavy processing taks should be avoided in the DAQ SBC.
At the same time this SBC should be as simple as possible to maintain the low cost producibility of the full system. The high-speed Ethernet connection capability also belongs to the requirements of the SBC, as it has to transfer the acquired data in real-time. The following table summarize some typical data-link requirements for passive radar application.

Illuminator	Channel Count	Data width	Sampling Rate	Required Link Speed
FM	8	IQ 32 bit float	200 kHz	100 Mbit/s
DAB	8	IQ 32 bit float	1500 kHz	768 Mbit/s
DVB-T	4	IQ 32 bit float	2400 kHz (Max!)	614 Mbit/s
DVB-T	8	IQ 8 bit signed integer	2400 kHz (Max!)	300 Mbit/s

Table 1. Typical data link speed requirements (Passive Radar mode)

2.1 DAQ Subsystem Firmware

This section discuss in general the build-up of the data acquisition firmware that runs on the SBC. The firmware application is developed under Linux environment and tested with the following SBCs: ASUS TinkerBoard, Raspberry Pi 3 and 4.

In Figure 4 we can see the detailed block diagram of the linked processing modules. As the demonstrator is mainly developed for research and educational purposes, the full firmware is composed from minor software parts in order to ensure high flexibility and testability. Using this build-up, developers and researches who make experiments with the system can easily understand, modify or replace any code sections (processing modules).
The modules of the firmware are written mixed in Python and C languages. The IQ data is exchanged between the consecutive processing modules through the standard I/O interface or through shared memory interfaces depending on the data type of the stream. The slow speed inter-process communication is implemented via Linux FIFOs. Using this architecture the individual processing blocks can run on different threads, hence the throughput is optimized. All the processing modules has framed data outputs! The format of the IQ frame is descried in detail in the later section.

2.1.1 Shared-memory Interface:

In order to avoid unnecessary memory copies between the consecutive modules, the firmware use shared memories. A shared memory interface is a ping-pong buffer which can be accessed by two consecutive modules. The data is written by the producer module (located backward) into the 'A' and 'B' buffers alternately. When a buffer is ready, the producer module informs the sink module (located forward) through the "forward control FIFO" that the data is ready to process. After the sink module has accessed, and processed the buffer, it inform the producer module that the given buffer is now free so it can use again.

In the following section we are going to discuss the data path, and the main role of each of the processing blocks. In the next sections the threading structure, the module configurations mechanism, the logging and finally the data framing is described.

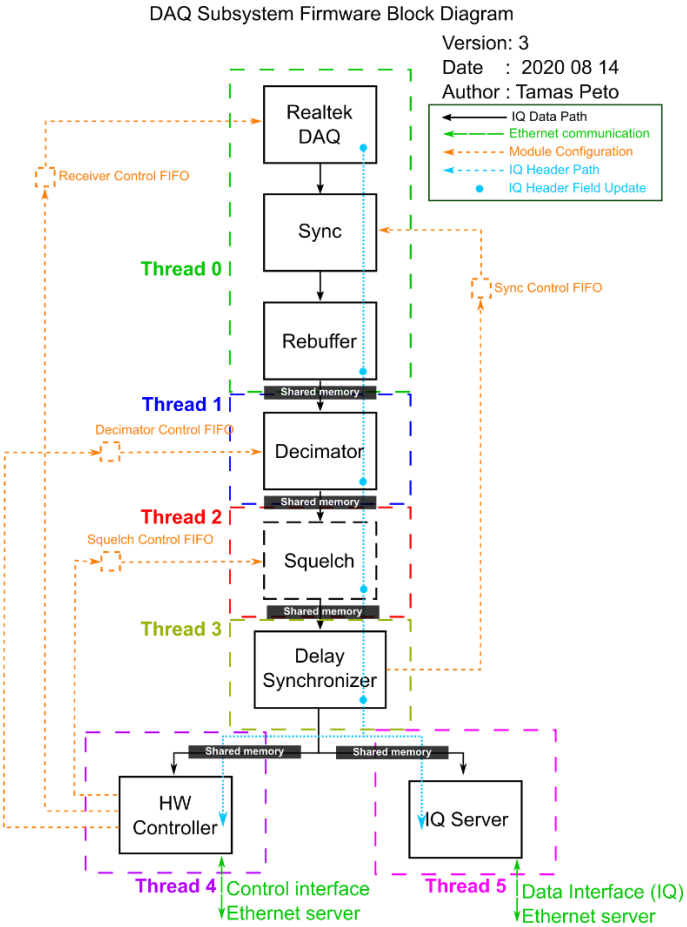

Figure 4. DAQ Subsystem Firmware block diagram

2.1.2 Realtek DAQ:

The entry point of the data acquisition system is the "Realtek DAQ" module. This block is responsible for the reception of the IQ samples from the RTL2832U chips through the USB connections. The received data has a format of 16 bits in which 8 - 8 bit represent the In-Phase (I) and the Quadrature (Q) components. The program reads one block of samples at the time from each of the USB buffers (RTL2832U). (This read block size is configurable in a configuration.ini file) When it successfully collects one block of samples from all of the receiver channels it packs the raw data into a frame and sends out at its standard output.
This module is also responsible for the low level hardware configuration. It configures the desired operation frequency, the sampling rate, the gain values and controls the internal noise source. For a detailed list of the wrapped hardware configuration options please check the firmware configuration section.

Programming Language	C
Input data source	USB buffers
Input data type	complex 8 bit int
Input data block size	channel count x daq block size
Output data sink	std. output
Output data type	complex 8 bit int
Output data block size	channel count x daq block size
Configuration	daq_chain_config.ini _data_control/rec_control_fifo
Logging	_logs/rtl_daq.log
Source file	_daq_core/rtl_daq.c

Table 2. Parameters of RTL DAQ module

2.1.3 Sync:

From the std. output of the "Realtek DAQ" module the "Sync" module reads the data frames and place the useful data bytes into circular buffers. Every channel has its own circular buffer. Every time when a new data frame arrives at the input, data from the circular buffer is written to the output. With the proper controlling of the write and read addresses of the circular buffers, the data bytes of the individual receiver channels can be shifted relative to each other, hence time delays can be introduced. Using this synchronization mechanism, at the output of the "Sync" module the data streams of the individual channels are sample aligned. At this point we have to emphasize that however these channels are aligned in the sample level it does not mean that they are perfectly aligned in time. In other words they are not coherent. In order to achieve fully synchronous operation which is mandatory for the coherent operation, the time delays at the sub-sample level should be compensated as well. This synchronization is achieved with the phase calibration which is implemented in a later processing blocks.

Programming Language	C
Input data source	std. input
Input data type	complex 8 bit int
Input data block size	channel count x daq block size
Output data sink	std. output
Output data type	complex 8 bit int
Output data block size	channel count x daq block size
Configuration	daq_chain_config.ini _data_control/sync_control_fifo
Logging	_logs/sync.log
Source file	_daq_core/sync.c

Table 3Parameters of Sync module

2.1.4 Rebuffer

The main purpose of this processing block is to provide possibility of using different sample sizes in the data acquisition part and in the DSP part. The input buffer size is set according to the optimal buffer size of the "Realtek DAQ" module (Optimal when no sample loss occurs), while the output data block size should be set according to the required coherent processing interval.
The rebuffer program accepts data blocks at its standards input and stores the input data in a circular buffer. When the required amount of data have been collected it writes a block of data into its output shared memory interface. The input and output block sizes are configurable through the daq_chain_config.ini file. In the current implementation there is limitation regarding to the block sizes. In case the output block size is less than the input block size, only the first portion will be forwarded, the remaining samples are dropped from the input frame! It is recommended to always use bigger block size at the output than at the input.

Programming Language	C
Input data source	std. input
Input data type	complex 8 bit int
Input data block size	channel count x daq block size
Output data sink	shared memory
Output data type	complex 8 bit int
Output data block size	channel count x cpi size x decimation ratio
Configuration	daq_chain_config.ini
Logging	_logs/rebuffer.log
Source file	_daq_core/rebuffer.c

Table 4.Parameters of Rebuffer module

2.1.5 Squelch

The use of this program block is optional in the daq chain. Its main goal is to capture the short, burst like signals and this way to improve the overall effectiveness of processing by rejecting the not useful portion of the full received RF signal. The module accepts raw complex data on its input through shared memory and investigates the content of the data. In case a predefined trigger condition is met, the module starts buffering the input data. When the required number of samples have already collected the module writes the collected data to its output shared memory. The trigger condition can be any kind of user defined data checking function, such as a simple amplitude gating with a threshold or even a more complex synchronization word checking in a data stream.

While the full system may not able to perform the data acquisition and processing on continuous data streams in real time, with the use of this module the stream can be continuously monitored and the more costly processing steps can be performed only on the relevant signal portions.

To use the squelch mode, enable it in the "daq_chain_config.ini" file with setting the 'en_squelch' field to 1.

Programming Language	C
Input data source	shared memory
Input data type	complex 32 bit float
Input data block size	channel count x daq block size
Output data sink	shared memory
Output data type	complex 32 bit float
Output data block size	channel count x cpi size x decimation ratio
Configuration	daq_chain_config.ini
Logging	_logs/squelch.log
Source file	_daq_core/squelch.c

Table 5. Parameters of Squelch module

2.1.6 FIR Decimator

This processing block implements anti-aliasing filtering and decimation. With the use of decimation we can decrease the sample rate of the processed signal, hence computational resources are saved in the upcoming signal processing modules**.** When the sample rate of a signal is decreased spectrum folding occurs. The filtering stage of the decimator prevents the out of band interference signals to be folded into the useful spectral region. The Decimator module of the firmware realize filtering and decimation with the use of the FIR filter architecture. In order to utilize the benefits of the oversampling and filtering the bit width of the iq data has to be increased to increase the dynamic range of the data representation. According to this the 8 bit signed integer data format is changed by the decimator to 32 bit complex floating point.
The decimation ratio, tap size and the coefficients of the FIR filter is configurable through the daq_chain_config.ini file.

Programming Language	C
Input data source	Shared memory
Input data type	complex 8 bit int
Input data block size	channel count x cpi size x decimation ratio
Output data sink	shared memory
Output data type	complex float 32
Output data block size	channel count x cpi size
Configuration	daq_chain_config.ini
Logging	_logs/decimator.log
Source file	_daq_core/fir_decimate.c

Table 6. Parameters of FIR decimator module

2.1.7 Delay Synchronizer

The "Delay Synchronizer" module is closely related to the "Sync" module which performs the time domain alignment of the coherent data streams. As we have discussed earlier the "Sync" module excepts the delay values to be set from another module. These delay values are determined by the Delay Synchronizer module.
To estimate the time delays to be set this module calculates the cross-correlation functions and determine the position of their main peak. In case all channels are perfectly alligned, then the main peaks of all the cross-correlation functions can be found at the zero delay position. Otherwise, the main peak distance from the zero delay position determines the current time delay of the channel. In case the system has M channels, then M-1 cross-correlation function have to be calculated, as one of the channels is selected to be a standard channel and the rest of the channels (M-1) are matched in time delay to the standard one. The index of the standard channel and the length of the cross-correlation function can be set in the daq_chain_config.ini file.

In order to estimate the exact time delay from the cross-correlation functions, the analog input signal (which should be common to all channels) should has noise like behavior. In most passive radar scenario this excitation is guaranteed by the received illuminator signal. When the IoO is not available, the KerberosSDR can use its internal noise source to provide the excitation signal for the sample delay calibration.

Let us emphasize that this procedure only applies to the sample delay level calibration. Coherent receiver applications requires the calibration of the amplitude and phase differences as well (sub-sample level). From the main peaks of the cross-correlation functions this information is available, but currently not utilized!

At the startup of the system the "Delay Synchronizer" module cooperates with the "Sync" module and calibrates the sample delays of the received data streams. Whenever a new block of samples pass through the processing chain, the "Delay synchronizer" module checks the synchrony of the channels, and issues corrections to the "Sync" module if the time delay is not zero. The detected delay errors are also noted in the header of the data frame, thereby the upcomming signal processing blocks are informed. In this case they can decide to drop the data frame or use it with restrictions.

The module is placed in the data chain after the decimator block in order to save computational resources.

Programming Language	Python3
Input data source	Shared memory
Input data type	complex float 32
Input data block size	channel count x cpi size
Output data sink	Shared memory
Output data type	complex float 32
Output data block size	channel count x cpi_size
Configuration	daq_chain_config.ini
Logging	_logs/delay_sync.log
Source file	_daq_core/delay_sync.py

Table 7.Parameters of Delay Synchronizer module

2.1.8 Hardware Controller

The "Hardware Controller" module can instruct the "Realtek DAQ" module to

- change center frequency
- enable or disable the internal noise source
- set the IF (Intermediate Frequency) gain values of the receiver channels,

and also to reconfigure the threshold value in the squelch module (if used).
This controlling functionality is implemented with the use of the "rec_control_fifo" and optionally on the "squelch_control_fifo". When it comes to beamforming or DOA estimation, the amplitude and phase transfers of the individual receiver channel have to be calibrated. At the same time we can assume, that changing the internal IF gain of the receiver changes the current amplitude and phase transfer functions as well. So once we have performed the amplitude and phase calibration we do not want to change the gain values again, otherwise we would need to perform the calibration again. On the other hand, for passive radar applications, the gains have to be tuned and increased as much as possible in order to achieve the highest possible dynamic range.
This module can automatically tune the gain values in each of the receiver channels to achieve maximum gain with no overdrive. After the proper gain values are set, we can fix them and perform the final calibration. This semi-automatic gain tuning is implemented in the "Hardware Controller" module and controllable with the use of corresponding fields in the daq_chain_config.ini file.

The hardware controller module also implements an Ethernet based control interface through which some of the parameters of the firmware can be reconfigured remotely (center frequency, gain, squelch threshold).The Ethernet based remote configuration service is running on port 5001 with TCP protocol by default.

Programming Language	Python3
Input data source	Shared memory
Input data type	IQ header only
Input data block size	-
Output data sink	-
Output data type	-
Output data block size	-
Configuration	daq_chain_config.ini
Logging	_logs/hw_controller.log
Source file	_daq_cor/hw_ctr.py

Table 8. Parameters of Hardware Controller module

2.1.9 IQ Server

The "IQ Server" module is responsible for transferring the preprocessed and framed IQ data through an Ethernet interface from the DAQ Subsystem to the DSP Subsystem. The realized server implements double buffering to be able to simultaneously receive data from the preceding module and transfer the IQ data. These two processes run in parallel on different threads inside the program. The IQ server service is running on port 5000 with TCP protocol.

Programming Language	C
Input data source	Shared memory
Input data type	complex float 32
Input data block size	channel count x cpi size
Output data sink	Ethernet socket
Output data type	complex float 32
Output data block size	channel count x cpi size
Configuration	
Logging	_logs/iq_server.log
Source file	_daq_core/iq_server.c

Table 9.Parameters of IQ Server module

2.2 Module Sample Sizes

Along the data acquisition chain the number of samples in a data frame may change when the frame passes-through certain blocks. In this short section we summarize all the actions which could affect the sample sizes.

The first program block in the data acquisition chain is the Realtek DAQ module. This module reads "daq_buffer_size" number of samples from each of the receivers simultaneously. Where one sample consist of 8 bit In-phase and 8 bit Quadrature data (IQ). This read size can be configured in the daq_chain_config.ini-file. In case this sample size is decreased then the Realtek DAQ module has to read the USB buffers more rapidly, which could lead to losing samples unintentionally due to the insufficient speed of the processor. On the other hand if the "daq_buffer_size" is too large then the consecutive processing blocks may not able to handle the received samples in the available time period. In case sample loss occurs then the user should tune this buffer size and check if it solves the problem. The default value of this buffer size is: 262144 (2^{18}) IQ samples.

The Sync module does not change the sample number. The rebuffer module is created to change the sample size in a data frame. In most applications the number of samples in the coherent processing interval differ from the optimal "daq_buffer_size" value. To this end the rebuffer module repacks the data, and changes the number of samples in a block from "daq_buffer_size" to "cpi_size ". Here we assumed that the decimation ratio is equal to 1! More precisely, at the output of the rebuffer module there will be "cpi_size x decimation ratio" number of samples per channel. This behavior is expound as follows:

When the data frame reach the decimator module the sampling rate of the signal is decreased with the decimation ratio. According to this to produce "cpi_size" number of IQ samples at the output, the decimator required "cpi_size x decimation_ratio" number of samples in total per channel. The decimator module does not accumulates samples for the decimation.

After the data frame reached the decimator module, the "cpi_size x decimation ratio" sample size is changed to "cpi_size" with filtering and decimation. The subsequent processing blocks all operate with "cpi_size" number of samples. Note that the bit depth of single sample is increased after decimation as the signal-to-noise ratio of the signal is increased with the filtering and higher dynamic range is required to represent this signal. In the current implementation the fir decimator use CF32 (Complex Float 32 bit) format at its output, while it reads the data in CINT8 (Complex Int 8 bit) representation from its input.

The" daq_buffer_size", "cpi_size" and the "decimation_ratio" parameters are all configurable in the "daq_chain_config.ini" file.

2.3 Calibration

The firmware of the DAQ Subsystem performs automatic gain, (clutter canceller - only with the presence of external ADPIS circuit), sample delay, amplitude and phase calibration at every system startup. This calibration procedure is controlled by two FSMs (Finite State Machine). These are implemented in the DS (Delay Sychnronizer) and in the HC(Hardware Controller) modules. The FSM in the HC module executes the tuning of the gain values and the ADPIS hardware (if present) and it is also responsible for controlling the calibration frames.

2.3.1 Gain Tuning:

In coarse of the gain tuning procedure the system tries to find those gains settings which provide the best sensitivity performance. In ideal conditions this operation can be guaranteed with maximizing the gain of the receiver channels, but without overdriving the ADC (Analog tot Digital Converter) of the receiver. In other words, the ADC of the receiver should be driven full scale to minimize the effect of the quantization noise. This effect is dominant in case of the RTL2832U based receivers as it has very a limited, 8 bit resolution ADCs (Analog to Digital Converter).

Since the received signal power can be different on the individual channels, the gain tuning procedure is performed independently on them. At system startup the HC FSM increase the gain values on all the channels. If no overdrive is detected it increases the gains further. Once overdrive is detected on a given channel, the gain value is set back to the previous value and further gain tuning is disabled on this particular channel. The gain tuning runs until all the channels reach at least once the overdriven state or the gain values are maximized.

The gain tuning procedure stops fully only after receiving a certain number of consecutive frames without ADC overdrive. This limit is configurable through the configuration ini file. It is also possible to ask the system to tune the gain values of the individual receiver channels commonly. In this case all the gains will be equal to the maximum possible of the "weakest"(has the lowest gain) channel. This feature can be enabled or disabled in the ini file. This tuning procedure is fully implemented in the HC module. The changing of the gain values is realized with the use of a control FIFO located at: _data_control/rec_control_fifo.

2.3.2 ADPIS Tuning (Passive Radar mode)

In case the system is extended with a dedicated Analog Direct Path Interference Supression PCB the DAQ firmware may perform control voltage tuning at startup and regulary as well. In order to compress the required dynamic range, the circuit subtracts the signal of the reference channel from the surveillance channels. To accomplish maximum supression, the signal of the reference channel

should be properly weighted (amplified and delayed) before the subtractions. This amplitude weighting and phase shifting is realized with a qudrature mixer on the PCB. The main goal of the ADPIS tunning algorithm is to properly regulate the control voltages of the IQ modulator to maintain the compressed dynamic range state.

The tuning procedure can be started only after that an initial gain tuning has been carried out as the good initial scaling is mandatory for the proper operation of the optimizer.

When the optimizer algorithm accomplished the control voltage tuning, gain tuning takes place again to achieve full scale driving on the ADCs since the input signal is now attenuated (Mainly the direct path signal component).

During normal operation when the system founds that the input signal power is increased relative to the last tuned state, it may initiate the tuning procedure again as the optimal control voltages may variating with time.

The steps of the ADPIS tuning can be summarized as follows:

1. Initial gain tuning
2. ADPIS tuning
3. Redo gain tuning
4. Power level tracking

2.3.3 Sample Delay and IQ Calibration

The system is able to automatically compensate the time differences of the individual receiver channels on the level of the discrete signal samples and on the level of the phase delays. While the sample mismatches can be attributed to the data acquisition, the phase distortions are inherent to the receiver channels. Other than phase difference compensation, coherent receivers has to deal with amplitude distortions as well, since most of the coherent algorithms assumes fully equalized channels.

The amplitude / phase calibration and correction is implemented in the Delay Synchronizer module. During the IQ calibration each sample from the multichannel signal array is multiplied by a complex correction number. In order to avoid the unnecessary calculations the firmware performs this task on the decimated data. (e.g.: when the decimation ratio is 10, only every 10th sample is preserved so 9 complex multiplication is needless)

The sample delay and IQ calibration procedure can be implemented in a number of way. One of the possibilities is to use a dedicated calibration signal, which is available at all the time and has the required parameters to perform proper calibration. To this end, the Kerberos SDR coherent receiver implements a built-in wideband noise source which output is equally distributed among the receiver channels. This calibration source is suitable both for sample delay and for IQ calibration. The calibration process itself is implemented by two distinct FSMs. Their internal states are describes in detail in the DAQ Subsystem - FSM section.

Once the calibration is achieved, depending on the current user settings the firmware can step into the calibration tracking mode, where the sample delay and the IQ calibration state is monitored and maintained. For more detailed information on the actual operation of the calibration procedure and on the possible configurations please check the Calibration track modes section bellow.

For optimal performance, it is highly recommended to use 50 Ohm terminations at the antenna inputs during the calibration. When the calibration is finished, the user should remove the terminations and insert the antennas cables instead. In case the 'calibration\require_track_lock_intervention' feature is enabled in the 'daq_chain_config.ini' file, the FSM of the Hardware Controller module does not allow the system to step into normal operation until the user do not reconnect the cables. To let the system pass through this stage the user should manually place '1' in the '_data_control/iq_track_lock' file. In case the track_lock_intervention feature is disabled, the system will automatically switch into normal operation, when the initial calibration is finished.

Kerberos SDR V2 or later: Starting from Kerberos SDR V2 the signal of the internal noise source is injected into the receiver channels via RF switches instead of the previously used directional coupler network. Hence, the termination of the receiver channels in calibration mode is automatically guaranteed, the use of the track lock feature is no longer required. When the user turns on the noise source on the PCB the hardware controls the RF switches simultaneously. The RF switches can route signal to the input of the receiver chip either from the antenna connectors or from the calibration signal network depending on the given control signal. (The unused ports are always internally terminated)

Calibration track modes: Depending on the hardware configuration and the applied use case the firmware can be operated in different calibration tracking modes.

Tracking mode ID	Description
0	Tracking mode is disabled. The calibration of the sample delays and IQ differences are performed at system startup then then the firmware no longer checks the synchrony. Sync flags in the header are set to constant true after that the system has accomplished the initial synchronization.

1	In this tracking mode the synchronization is maintained continuously, with checking each and every IQ data frames. Since all the data packets are checked, suitable signal for the calibration have to be available continuously. In passive radar operation mode the illuminator signal generally fulfills this requirement. Such an illuminators are the DVB-T, DAB or the FM signals. The system maintains the sample level synchrony with simply calculating the amplitudes of the cross correlation functions. However the IQ calibration tracking has to take into account the applied calibration technique and also the configuration of antenna system. The KerberosSDR system has a built-in noise source which can inject calibration signal into the inputs of the receiver channels through directional couplers or RF switches. When the usage of its internal noise source is enabled ('daq\en_noise_source_ctr' field in the daq_chain_config.ini) the firmware calibrates the system with this built-in source. As a result, the receiver channels will be calibrated up to the antenna inputs. When the system switches back to normal operation, it turns off its internal noise source and thereafter the KerberosSDR will receive signals from the antenna terminals. (For optimal performance the user should disconnect the 50 Ohm terminations at this time and connect the antenna cables in place of them) From this point, the phase relations of the different receiver channels are dependent on the available signals and on the used antenna configuration. When the first data frame is received (after the system switched to normal operation) the firmware saves the currently seen amplitude and phase differences and will use it as a reference in the later tracking stage. In the tracking stage the firmware continuously the evaluates the amplitude and phase relations and compares the current values with the reference. In case the deviation is greater than the allowed maximum, the checking algorithm indicates that the IQ synchrony may lost. When at least 3 (user controllable) consecutive frames indicates that the IQ relations may have changed, the system considers that the IQ calibration is no longer appropriate and re-initiates the calibration procedure. The maximum allowed amplitude and phase deviations are controllable via the 'calibration\amplitude_tolareance' and the 'calibration\phase_tolerance' fields respectively in the daq_chain_config.ini file. To change the maximum number of the allowed faulty frames edit the 'calibration\maximum_sync_fails' field in the config file.

2	Burst tracking mode. In case the continuous availability of a suitable calibration signal cannot be guaranteed, the firmware can regularly switch back to calibration mode from normal mode to check whether the sample delay and the IQ calibration is still valid or not.I n contrast to the continuous tracking mode the test signal is now the built-in noise source, hence the system records the reference IQ difference vector when the noise source is enabled. During the tracking phase, the FSM of the HC module regularly turn on the noise source for a couple of frames and then turns it off. The duration of the noise source burst is controllable via the 'calibration\cal_frame_burst_size' parameter, while the break between two consecutive noise bursts can be set with the 'calibration\cal_frame_interval' parameter in the config file. In case the system detects sample or IQ sync loss at least 3 times (user controllable, 'calibration\maximum_sync_fail') consecutively it considers that the IQ calibration is no longer appropriate and re-initiates the calibration procedure. Note that this tracking mode is effective only if the user does not have to replace the antenna cables with 50 Ohm terminations for each calibration or for each calibration checking.

Table 10. Description of the calibration track modes

2.4 Finite State Machines

2.4.1 Delay Synchronizer FSM

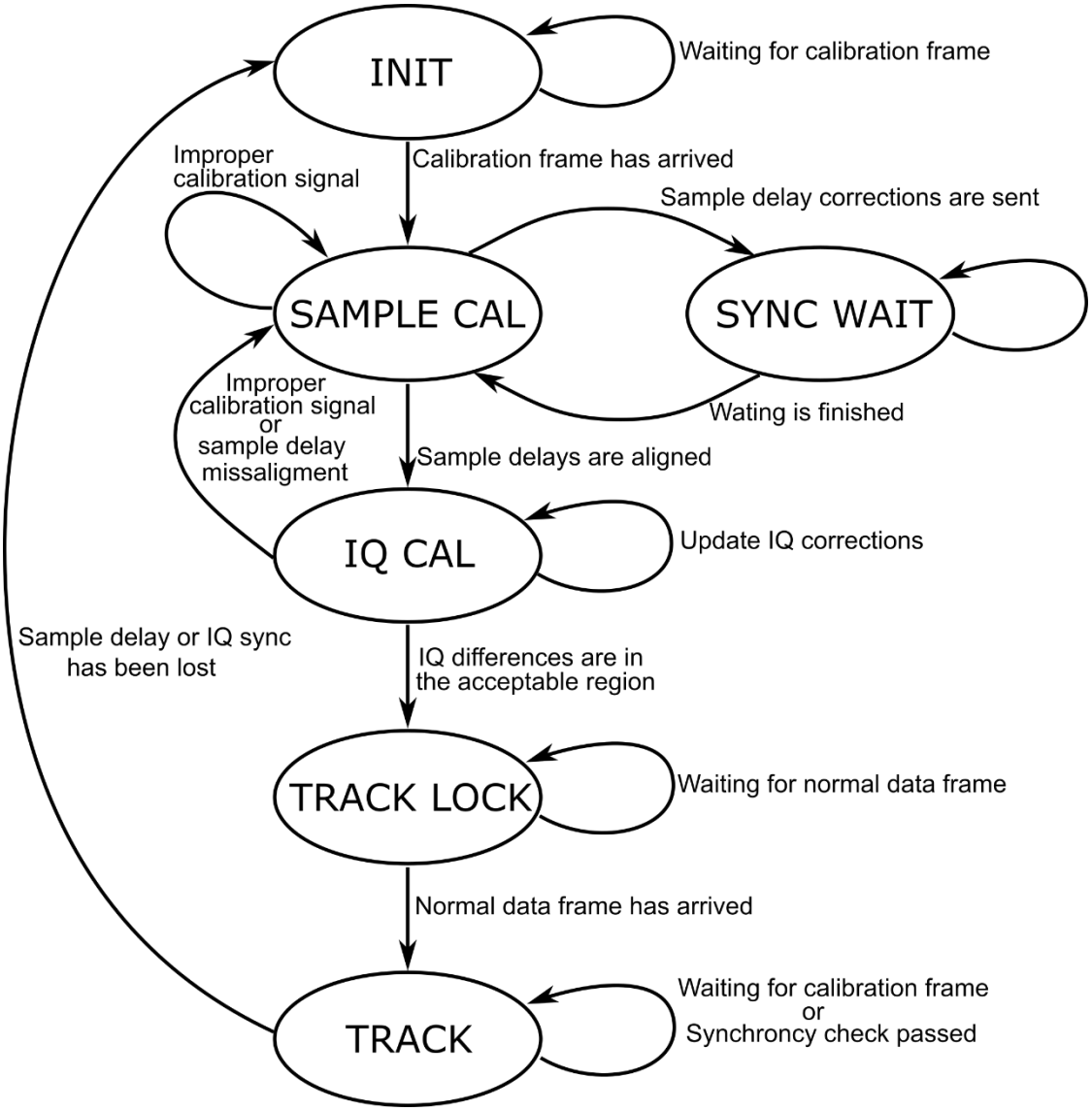

Figure 5. FSM of the Delay Synchronizer module

STATE_INIT: This is the first state of the FSM at system startup. In this state the Delay Sync. module is waiting for the calibration signal to arrive. This information is encoded into the header of the IQ data frame (frame_type field).When the first calibration frame is received by the module it goes to the SAMPLE_CAL state.

STATE_SAMPLE_CAL: In the sample delay calibration state the module continuously calculates the cross-correlation functions of all the channels to determine their exact sample

mismatches. It does not only determines the position of the highest correlation peak but also estimates its dynamic range. In case the amplitude of the correlation peak is low, the delay calibration cannot be performed due to the insufficient signal-to-noise ratio of the calibration signal. In this case the module stays in this state and waits for a more applicable calibration data frame to arrive.

In case the correlation peak dynamic range checking is passed, the module extracts the estimated sample delays. If the sample delays are zero, the system is in a calibrated state and goes next to the IQ_CAL state to perform the amplitude and phase calibration. Otherwise it sends the delay values to the Sync module which sets the desired delays on the different channels. While the new delay values takes effect the system steps into the SYNC_WAIT state.

STATE_SYNC_WAIT: After that the Delay Synchronizer module has sent the delay correction values to the sync module in the SAMPLE_CAL state, it waits for a few frame in the SYNC_WAIT state. During these few data frames the data pipeline is filled up with the correctly delayed data and the system goes back to the SAMPLE_CAL state to verify the synchrony.

STATE_IQ_CAL: As soon as the system has achieved the sample level synchrony it becomes ready to perform the IQ calibration. In the first step it verifies the sufficiency of the dynamic range of the main correlation peaks. (At this point the correlation peak offset should be zero as the sample delays have been already corrected) In case this check fails, the state machine goes back to the SAMPLE_CAL state as it could mean sample delay loss.

Otherwise the IQ calibration procedure can be continued. The algorithm extracts the amplitude and phase differences and check whether it lies in the acceptable tolerance region or it must be calibrated. In the latter case the module updates the correction values and stays in this state to perform the checking of the updated correction values again. Once the IQ differences are in the acceptable region the FSM enters to the TRACK_LOCK state.

STATE_TRACK_LOCK: When the state machine reaches the TRACK_LOCK state, the sample delay mismatches and the IQ differences are corrected. The system signals this state with setting the corresponding flags in the header of the IQ data frame (sample_sync_flag, iq_sync_flag). The calibration part is finished and the system now enters to the tracking phase in which it can continuously follow and maintain the synchronized state. To start this tracking phase the DAQ module has to enter to the normal operation, in which the calibration signal is turned off and it sends normal data packets through the data tunnel instead of the calibration packets. The Delay Synchronizer module waits in this state for the normal data packets to arrive. After arriving the first normal data packet (IQ header frame_type field) the FSM steps into the TRACK state.

Note that the switching between the normal and calibration mode is implemented in the FSM of the Hardware Controller module.

STATE_TRACK:

In the tracking state the Delay Synchronizer module tries to check every data packets whether the sample and the iq synchrony is lost or not. The sample synchrony is checked by calculating the dynamic range of the correlation peak at the zero offset position. In case the dynamic range is

insufficient the sample sync may lost. Other than this, the IQ differences are also estimated from this correlation peak. In case the differences are out of the tolerance it could mean the loss of IQ synchrony. Either the sample or the IQ sync check fails the system increase an error counter. When the error counter reaches the allowed maximum, the synchrony is considered to be lost and the FSM goes back to the INIT state to recalibrate the system.

In contrast, when the current data frame passes the both checks, this error counter is decreased (if it is not zero). With this design the smaller bust like errors can be filtered and the system lose the synced state only if a number of consecutive frames are erroneous.

It is important that the checking of the synced state is controlled differently in the different calibration track modes and calibration options. For further information please check the corresponding sections.

2.4.2 Hardware Controller FSM

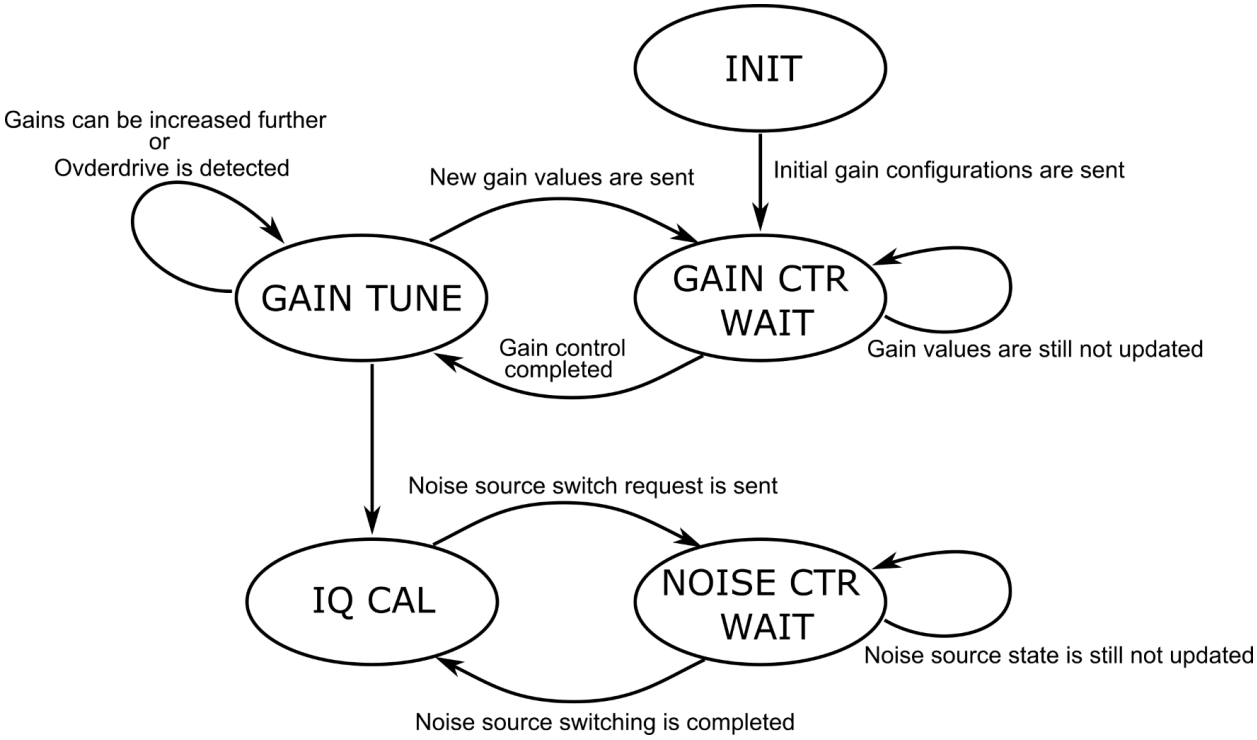

Figure 6. FSM of the Hardware Controller module

INIT: The FSM of the HC module starts its operation at the INIT state. The main purpose of this state is to set the hardware into a default configuration. It configures the initial gain values, the ADPIS control values (if the dedicated hardware is present) and turns off the calibration signal source. In the current version only the initial gain value configuration is implemented! Consequently, the next state after the initialization has completed is the GAIN_CTR_WAIT.

GAIN_CTR_WAIT: In this state the FSM waits for the new gain settings to take effect. It is checked by reading the current gain values from the header of the IQ data frame, which is updated by the DAQ module.

GAIN_TUNE: The FSM in this state tries to find the best gain settings. In case the gain tuning is enabled it increase the gain values on all the channels. If no overdrive is detected it increase the gains further. Once overdrive is detected on a given channel, the gain value is set back to the previous value and further gain tunings are disabled on this particular channel. The gain tuning runs until all the channels reach at least once the overdriven state or the gain values are maximized. After the gain tuning is completed in this way the FSM no longer stays in the GAIN_TUNE state and steps forward to the IQ_CAL state.

IQ_CAL: The HC FSM in the IQ_CAL state works in close co-operation with the FSM of the Delay Synchronizer module. Through the header of the IQ data frame the HC FSM is continuously informed about the current state of the Delay Synchronizer FSM. In case the Delay Synchronizer

module signals out of sync state the HC FSM turns on the internal noise source and thereby provides calibration frames for the sample delay and IQ calibration procedures. When the Delay Synchronizer module achieved the synced state it turn off the internal calibration signal source and thereby switches back to normal operation. In case the burst calibration track mode is enabled it also turn on and off the internal noise source periodically. For further information on this mode please check the calibration track mode section. Whenever the noise source is controlled, the FSM steps into the NOISE_CTR_WAIT state from the IQ_CAL state.

NOISE_CTR_WAIT: When the noise source is controlled, the HC FSM has to wait until its request takes effect. At first dummy frames are arriving and then normal or calibration frames (depending on the noise source state). The IQ header contains information about the current state of the internal noise source, and this field in monitored by the HC module in the NOISE_CTR_WAIT state. When the desired switching has been completed the FSM goes back to the IQ_CAL state.

2.5 Control Interface

The DAQ Subsystem implements an Ethernet based remote control interface in the Hardware Controller Module. Through this interface, some of the firmware parameters can be reconfigured. The control interface server works fully independently from the normal IQ data path, it receives and processes the incoming request on a dedicated thread. To maintain system consistency and stability the server accepts connection and only from one client at a time, and does not receives further request until the previous one is fully processed. When the client side sends a configuration request the server receives the command, informs the main (HWC) module that a new request is pending and waits for the completion. After the request has been successfully processed by the main module the server sends back a request completion message to the client to inform that it is ready to accept new requests.

A standard message is 128 bytes long. The first 4 bytes are interpreted as the command word, while the reaming 124 bytes are designated to store the parameters of the request.

Valid command words are the followings:

Command word	Parameters	Request description
INIT	None	Initialize the communication on the control interface
EXIT	None	Used to inform the server that client will disconnect
STHU	threshold value: 1 float	(Squelch ThresHold Update) Sets the threshold value in the squelch module
FREQ	Frequency in Hz: 1 uint 64	Changes the center frequency of the receivers
GAIN	gain values of the individual channels: 4 uint 32	Changes the IF gain values of the receivers

Table 11. Valid commands of the control interface

The default port of the DAQ configuration service is:5001

2.6 Squelch Mode

The squelch mode of the DAQ firmware is created to support the interception of short, burst like signals. In this mode the DAQ chain drops all the received data frames and forwards data only when a signal burst is detected. The burst signal processing is implemented in two different processing modules. In the DAQ Firmware and the in the DSP.

The DAQ firmware is responsible mainly to detect the burst signal and forward it to the DSP. Actually it finds the beginning of the burst and forwards a block of samples (which eventually contains the full burst as well if the parameters are set properly) This feature is realized in the squelch module. To start the DAQ chain in squelch mode, the users has to enable it in the daq_chain_config.ini file with setting the en_squelch field to 1. (In case it is set to zero, squelch module is bypassed in the chain)

Configuration:

In this section , the configuration of the main parameters, required to utilize the squelch module properly are explained through different practical examples.

What the user has to know preliminary is the maximum length of the burst signal. Lets us assume for example it is Tburst=23 ms, and let us assume that the bandwidth of the signal is 2.4 MHz (we are not using decimation, we will cover this problem later in the discussion). The sample period is Ts=1/(2.4*10^6)= 416 us, thus the burst is Tburst/Ts=55200 samples long. Thinking in power of 2 block sizes, we can set the CPI length parameter to 2^16 = 65536 (daq_chain_config.ini -> cpi_size=65536), so the burst will fit into this frame length (Theoretically you could set it to any number greater than 55200). In general, the daq_buffer_size parameter is less or equal to the cpi_size.

During operation the squelch module has to buffer the input data, since it does not know when the burst will start inside the currently received data frame. When the trigger condition is met on the current frame (somewhere inside the frame), the module starts buffering cpi_size samples and packs it into new data frame.

Example:

Let say the daq_buffer_size is equal to the cpi_size and we have the frame structure depiced in Figure 1.

- Frame 1: The squelch module reads the first data frame, since it does not found any burst in the frame (the trigger condition is not met) it will drop the incoming data frame and forward a trigger wait frame. (Trigger wait frames has no payload)

- Frame 2: After reading the second data frame, it checks whether the trigger condition is met. The trigger is activated and the modules start buffering enough data to fill completely on data frame with data (cpi_size number of samples). At this point the module does not send out any frame, the system is triggered.

- Frame 3: When the third data frame is read, the module fill up the payload of the prepared frame completely and sends out a data frame (which contains the captured signal burst.) Started from the last sample of the sent data frame the module looks for another burst in the remaining portion of the third incoming data frame. The trigger condition is met again, it detects the beginning of another burst, and then start collecting samples.

- Frame 4: Reading the 4-th data frames enables the system to send out a new data frame (which contains the second burst). After sending the data frame the module scans the remaining portion of the incoming data frame but does not found new burst. The system wait for a new incoming frame.

- Frame 5: When the 5-th data frame is read, the trigger will not be activated and the module will send out a trigger wait frame.

The squelch module does not search the end of the burst! It will be done in DSP. With this solution the data frames always can have the same length, and there is no need to handle arbitrary sized data frames. (however the firmware is ready for that)

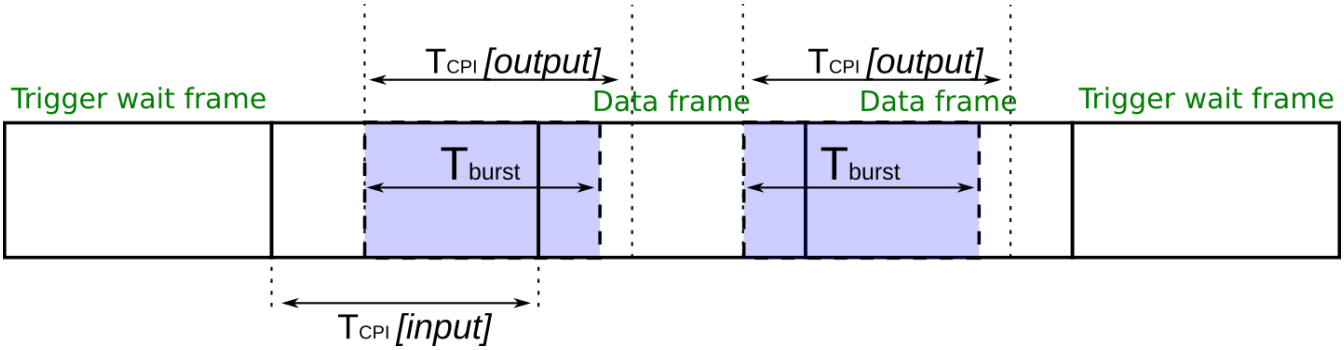

Figure 7. Example frame structure

Decimation: In case decimation is enabled, the rebuffer module accumulates enough data to perform the decimation by the decimator module prior to the squelch module Let's say the bandwidth of our signal is now only 1.2 MHz instead of 2.4MHz. Now the sampling period Ts=1/(1.2*10^6)= 832 us, and the burst is Tb/Ts=27600 samples long. We will use the same daq_buffer_size=65536 value, but now the cpi_size can be halved, 32768.

The chance for missing a signal burst could be minimized with setting the cpi_size parameter as close as possible to the length of the burst.

DoA DSP:

Until this point we have detected the start of the burst and forwarded a block of samples to the DSP. In case the squelch mode is enabled in the DSP it will try to cut out the useful section of the burst form the full received data frame. To do this it calculates the average power of the received signal with a sliding window and applies the previously set threshold value to cut out the useful

portion (useful when the average power is greater than the threshold). Finally the direction estimation is applied to this gated signal.

2.7 Unit Testing

The ultimate goal of the testing service is to validate all the essential and specific functionalities and operations of all the processing modules in the DAQ firmware. This is particularly important during the development period. The firmware is considered to be ready for release only after passing all the tests cases. Each test is identified by two number, the first one identifies the tested module, while the second denotes the test case.

E.g.: Test_1_3, stands for sync module testing, test case 3.

The following tables summarize the test cases

Sync Module

Test ID	Test Name	Test Description
1_0	Frame forward test with normal data frames	In this simple test the IQ frame process and forward capabilities of the module is tested. The module is excited with 10 normal data frames and its output is captured. In the recorded output data stream the structure of the IQ frames is checked with inspecting the sync word field. Expected to output 9 data frames.
1_1	Frame forward test with dummy frames	Similar to test_1_0, but the module is excited with 10 dummy frames. Expected to output 10 frames.
1_2	Frame forward test with calibration frames	Similar to test_1_0, but the module is excited with 10 calibration frames. Expected to output 9 frames.
1_10	Ramp test with normal data frames	The module is excited with a ramp signal. This test can reveal potential indexing problems and sample losses. The maximum value of the ramp is a prime number (29) to exclude potential testing problems. The data channels in the full data stream are shifted relative to each other, hence the independency of the data channels can be checked as well.

		Antenna channel 1, I samples goes from 0 to 29 Antenna channel 1, Q samples goes from 30 to 59 … Antenna channel 4, I samples goes from 203 to 232 Test parameters: Number of input frames: 10 Delay values:[0,0,0,0]
1_11	Ramp test with calibration frames	Similar to test 1_10, but performed with calibration type frames
1_20	Delay set test with ramp excitation	Checks the delay control capabilities of the module. In the first step the delay values are set after starting module. Then a ramp test is performed with the same excitation as in the test 1_10. We are expecting that the ramp counters on the individual data channels are shifted relative to each outer (and relative to the reference ramp counter) according to the previously set delay values. Test parameters: Number of input frames: 10 Delay values:[2,5,1,7]
1_21	Delay set and reset test with ramp excitation	Checks the delay reset capabilities of the module. In the first step, delay values are set and then a reset is requested. After resetting the module a ramp test is performed with the same excitation as in the test 1_10. We are expecting that the ramp counter of the individual channels at the output of the module are aligned with the reference ramp counter. **CURRENTLY NOT IMPLEMENTED!**
1_30	Dynamic delay set test with ramp excitation	Delay value configuration command is sent after exciting the module with few normal data frames. **CURRENTLY NOT IMPLEMENTED!**

1_31	Dynamic delay reset test with ramp excitation	Delay reset command is sent after configuring some delay values and exciting the module with few normal data frames. **CURRENTLY NOT IMPLEMENTED!**
1_32	Dummy frame arrive test	Upon reception of a dummy frame the internal memory of the sync module should be reset. A dummy frame is sent to the input of the module after exciting with a few normal data frames. **CURRENTLY NOT IMPLEMENTED!**

Rebuffer

Test ID	Test Name	Test Description
2_0	Frame forward test with normal data frames	Similar to test 1_0. In this test the daq_buffer_size and the cpi_size is equal and the decimation ratio is zero hence no rebuffering will be performed. Test parameters: Number of input frames: 10 Input buffer size: 2^18 Output buffer size: 2^18 Decimation ratio: 1
2_1	Frame forward test with dummy frames	Similar to test_2_0, but the module is excited with 10 dummy frames In this test the daq_buffer_size and the cpi_size is equal and the decimation ratio is zero hence no rebuffering will be performed.
2_2	Frame forward test with calibration frames	Similar to test_2_0, but the module is excited with 10 calibration frames

		In this test the daq_buffer_size and the cpi_size is equal and the decimation ratio is zero hence no rebuffering will be performed. Test parameters: Number of input frames: 10 Input buffer size: 2^18 Output buffer size: 2^18 Decimation ratio: 1
2_3	Sample number increase test with normal data frames	Similar to test 2_0, but the cpi size is double the daq buffer size (N > N_daq). In this case rebuffering will be performed. The output buffer size is the integer multiple of the input buffer size Test parameters: Number of input frames: 10 Input buffer size: 2^18 Output buffer size: 2^19 Decimation ratio: 1
2_4	Sample number increase test with calibration frames	Similar to test 2_3, but performed with calibration type frames
2_5	Sample number increase with fractional rebuffering	Similar to test 2_3, but the input and output buffer size are prime numbers. Test parameters: Number of input frames: 10 Input buffer size: 37 Output buffer size: 101 Decimation ratio: 1 Expected: 3 output frames

2_6	Decimation ratio handling test with normal data frames	Similar to test 2_0, but the decimation ratio is R=3. Test parameters: Number of input frames: 10 Input buffer size: 2^18 Output buffer size: 2^18 Decimation ratio: 3
2_7	Sample number decrease test with normal data frames	Similar to test 2_0, but the cpi size is the fraction of the daq buffer size(N < N_daq). Test parameters: Number of input frames: 10 Input buffer size: 101 Output buffer size: 37 Decimation ratio: 1 Expecting multiple frame output with inputting a single frame **NOT IMPLEMENTED**
2_10	Ramp test with normal data frames	Similar to test 1_10. The input and output buffer sizes are the same, hence no rebuffering will be performed. Test parameters: Number of input frames: 10 Input buffer size: 2^18 Output buffer size: 2^18 Decimation ratio: 1
2_11	Ramp test with calibration frames	Similar to test 2_10, but performed with calibration type frames. Test parameters: Number of input frames: 10

		Input buffer size: 2^18 Output buffer size: 2^18 Decimation ratio: 1
2_12	Ramp test with sample number increase (fractional rebuffering)	Similar to test 1_10, but the output buffer size is greater than the input buffer size. Test parameters: Number of input frames: 10 Input buffer size: 37 Output buffer size: 101 Decimation ratio: 2
2_13	Ramp test with sample number decrease (fractional rebuffering)	Similar to test 1_10, but the output buffer size is smaller than the input buffer size. Test parameters: Number of input frames: 10 Input buffer size: 101 Output buffer size: 37 Decimation ratio: 2 **CURRENTLY NOT IMPLEMENTED!**
2_20	ADC overdrive accumulation test	The rebuffer module should accumulate ADC overdrive flags within the data collection period and reset the accumulated flag values upon sending out the rebuffered data frame. **CURRENTLY NOT IMPLEMENTED!**
2_32	Dummy frame arrive test	Similar to test 1_32. **CURRENTLY NOT IMPLEMENTED!**

Squelch

Test ID	Test Name	Test Description
3_0	Frame forward test with normal data frames	Similar to test 2_0. Normal data frames are forwarded only when the module is triggered. In this test triggering is forced with setting the trigger level to zero. Test parameters: Number of input frames: 10 Trigger threshold:0 Expected to arrive 10 normal data frames.
3_1	Frame forward test with dummy frames	Similar to test 2_1. Dummy frames ignore trigger conditions. Expected to arrive 10 dummy frames.
3_2	Frame forward test with calibration frames	Similar to test 2_3. Calibration frames ignore trigger conditions. Expected to arrive 10 normal data frames.
3_3	Trigger wait frames	The module should generate trigger wait type frames when receiving normal data frames but the trigger condition is not met. Test parameters: Number of input frames: 10 Trigger threshold:1
3_4	Frame forward test with calibration frames and non zero trigger	Similar to test 3_2 but the trigger threshold is non zero. Calibration frames should be forwarded regardless of the trigger condition. Test parameters: Number of input frames: 10

		Trigger threshold:1 Expected to arrive 10 calibration frames.
3_5	Trigger test	Test the operation of the squelch mechanism including the triggering and the signal stream cropping. The test is carried out with the following data frame structure. 5 normal data frames, in which the payload is constant 0 on the first channel. (Trigger condition will not met) 3 data frame in which the signal amplitude will trigger the squelch module. 5 normal data frame The checker is expecting to receive the following data frame structure: 5 Trigger wait frame 2 normal data frame (One data frame is consumed for buffering) 4 trigger wait frame
3_10	Ramp test	Tests the cropping mechanism with ramp excitation. Excitation includes 3 normal data frames. This 3 normal data frame will include 2 burst. Each burst starts with a large peak to trigger the squelch module. Excitation parameters: CPI length: 256 sample First pulse signal sample position: 100 Ramp starts at sample index: 101 First pulse length consist of 155 sample (101-255) First pulse ends in the second CPI at sample position 55. (56 samples) Second pulse signal sample position: 100

		Ramp starts at sample index: 101 First pulse length consist of 155 sample (101-255) First pulse ends in the second CPI at sample position 55. (56 samples)
3_32	Dummy frame arrive test	Similar to test 2_32. **CURRENTLY NOT IMPLEMENTED!**

FIR Decimator

Test ID	Test Name	Test Description
5_0	Frame forward test with normal data frames	Similar to test 1_0, but expected to output 10 data type frames.
5_1	Frame forward test with dummy frames	Same as test 1_1. Expected to output 10 dummy type frames.
5_2	Frame forward test with calibration frames	Similar to test 1_2, Expected to output 10 calibration type frames.
5_3	Frame forward test with trigger wait frames	Similar to test 5_1, but carried out with 'trigger wait' type frames.
5_10	Transfer function test with swept CW signal	The module is excited with several series of normal data frames. Each series contains a sinusoidal excitation with enough number of frames to generate one output frame full of the decimated signal. The frequency of the excitation signal is increased from one frame group to the next one, until the entire frequency band is covered. The checker script extracts the amplitude of the

		decimated signals, evaluates the measured transfer function and checks against the theoretical transfer.
5_11	Phase continuity testing with CW signal	The module is excited with 5 normal data frames, which contains a continuous single sinusoidal signal on each antenna channels. At the output the phase continuity of the decimated sinusoidal is checked.

It is important note that all unit test should run with the provided default configuration file. The unit test scripts are prepared only to check the operation of the modules in this scenario and the proper operation of the test scripts are not guaranteed with other configurations. In case the user would like to test the system in different configurations, the condition of the unit tests and the implementation should be revisited

2.8 Firmware Configuration

The modules of the DAQ firmware can be configured at start time using the daq_chain_config.ini file. This following table list the available configuration options. The short description of the configuration field can be found in the third column.

<table>
<tr><th>Group</th><th>Field name</th><th>Data type</th><th>Designation</th></tr>
<tr><td>hw</td><td>name</td><td>String, ASCII</td><td>Short denomination of the Data Acquisition System. E.g.: Kerberos4</td></tr>
<tr><td>hw</td><td>unit_id</td><td>integer</td><td>Unique serial number of the DAQ system.</td></tr>
<tr><td>hw</td><td>Ioo_type</td><td>integer</td><td>Type of the used Illuminator of Opportunity (Used in Passive Radar mode)</td></tr>
<tr><td>hw</td><td>num_ch</td><td>integer</td><td>Configures the number of receiver channels.</td></tr>
<tr><td>daq</td><td>log_level</td><td>integer</td><td>Sets the logging level in the firmware modules. The logging levels and their corresponding selection numbers are listed as follow:
<table>
<tr><th>Mode</th><th>Value</th><th>Mode description</th></tr>
<tr><td>TRACE</td><td>0</td><td>Shows regularly updating status of the module in the process of data flow.</td></tr>
<tr><td>DEBUG</td><td>1</td><td>Shows the internal operation of the module</td></tr>
<tr><td>INFO</td><td>2</td><td>Shows the external actions that have impact on the</td></tr>
</table>
</td></tr>
</table>

			<table><tr><td></td><td></td><td>operation of the mule</td></tr><tr><td>WARNING</td><td>3</td><td>Messages of operation that may have effect on the desired operation</td></tr><tr><td>ERROR</td><td>4</td><td>Error messages that are not critical, the module can be still operational</td></tr><tr><td>FATAL</td><td>5</td><td>These messages summarize the conditions for module shutdown</td></tr></table>
daq	daq_buffer_size	integer	This number declares the number of samples in a block read by the Realtek DAQ code from the devices. This buffer size is used by the "Sync" module as well.
daq	center frequency	integer	RF operating frequency of the coherent receiver. This field is interpreted in Hz.
daq	sample_rate	integer	ADC sampling frequency. This field is interpreted in Hz
daq	gain	integer	IF gain values in the coherent receiver. Gain specified in this field is set by the Realtek DAQ module at the startup of the system. In case the semi-automatic gain tuning is enabled this gain values may overwritten by the Hardware Controller module. Valid values are the followings: 0, 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, 496
daq	en_noise_source_ctr	integer	If enabled the system will control the internal noise source when calibration frame type is requested. Can be disabled on non-Kerberos systems.
daq	ctr_channel_serial_no	integer	Serial number of the Realtek receiver which drives the noise source in Kerberos systems.
pre_processing	cpi_size	intger	Number of samples in a coherent processing interval. Please check the "Module Sample Sizes" section for further information.
pre_processing	decimation_ratio	integer	Decimation ratio used by the CIC decimator module
pre_processing	fir_relative_bandwidth	float	Relative bandwidth of the decimator FIR filter. Must be in the range of [0..1]
pre_processing	fir_tap_size	int	Number of taps used by the decimator FIR filter.
pre_processing	fir_window	string	Window function used for the design of the decimator FIR filter. For the possible used options check the *scipy.signal.get_window* function at scipy.org
pre_processing	en_filter_reset	integer	Enables the resetting of the FIR decimator. In this mode the internal memory of the filter is set to zero when a new block is processed

calibration	corr_size	integer	Number of samples used for the cross-correlation calculation. It must be less or equal to the cpi_size.
calibration	std_ch_ind	integer	Index of the standard channel to which the rest of channels will be matched in time
calibration	en_frac_cal	integer	Enables the fractional sample delay compensation in the Delay Synchronizer module. (This feature is currently not implemented)
calibration	en_iq_cal	integer	If enabled(=1), the amplitude and phase differences of the individual receiver channels are calibrated.
calibration	cal_track_mode	integer	0: Calibration tracking disabled (sample sync and iq sync flags are set true) 1: Conitnuous tracking 2: Tracking with calibration frame bursts
calibration	require_track_lock_intervention	integer	If enabled, the automatic calibration procedure will not step into the tracking phase until the user will not set correspond flag. (For more detail see the DAQ Subsystem\Calibration section) 0: disable 1: enable
calibration	cal_frame_interval	integer	Number of normal data frames between two calibration bursts. Used in cal_track_mode 2.
calibration	cal_frame_burst_size	integer	Number of frames in a calibration burst. Used in cal_track_mode 2.
calibration	amplitude_tolerance	integer	Maximum allowed amlitude deviation between the coherent receiver channels.

calibration	phase_tolerance	integer	Maximum allowed phase deviation between the coherent receiver channels.
calibration	maximum_sync_fails	integer	Maximum allowed succesive synchronization check fails.
calibration	gain_lock_interval	integer	Minimum required number of consequtive frames without overdrive to stop gain tuning.
calibration	unified_gain_control	integer	If enabled (=1) the gain of the receiver channels are forced to take the same value
adpis	en_adpis	integer	Enables the Analog Direct Path Interference Suppression mode. This feature required dedicated hardware support.
adpis	en_gain_tune_init	integer	Enables the semi-automatic gain tuning at the start of the system
adpis	adpis_proc_size	integer	Number of samples used by ADPIS optimizer algorithm
adpis	adpis_gains_init	List of integers	The hardware controller module will set these gain values at the start of the system. This list must has the same amount of values as the number of channels.
squelch	en_squelch	integer	Set this to 1 to enable the usage of the squelch module. This flag decides whether the squelch or the rebuffer module will be instantiated

squelch	amplitude_threshold	float	Normalized amplitude threshold value used by the squelch module. Must be in the range of [0..1]
data_interface	out_data_iface_type	string	Output interface of the DAQ firmware. Can be “eth” or “shmem”

Table 12. Fields of the DAQ chain configuration ini file.

2.9 Developer Notes

2.9.1 IQ/ sample sync check may fails at the first few calibration frames in cal_track_mode=2.

When the system turns into calibration mode, the rtl_daq module sends out a few 'dummy' typ' frames to clean up the chain, then after it turns on the noise source and sends 'calibration' type frames. During this switch the system has a transient response which may deteriorate the stationary operation and the delay sync module will detect sync loss at the first couple of frames.

Due to this reason it always recommended to set *max_sync_fails* in the config file to more than 10, and correspondingly the *cal_frame_burst_size* must be at least 10 as it should be greater than the *max_sync_fails* param. Otherwise the system never be able to aggregate 10 sync fails, so it will not notice sync loss.

This can be inspected in the delay_sync.log file in DEBUG mode.

```
DEBUG:__main__:Type:0, CPI: 104, State:STATE_TRACK
DEBUG:__main__:Sync flags are set
INFO:__main__:Delay track statistic [sync fails ,sample, iq, total][0,1,1/104]
WARNING:__main__:Dropping frame - IQ server, Total: 66
WARNING:shmemIface:Dropping frame.. Total: [67]
DEBUG:__main__:Type:1, CPI: 105, State:STATE_TRACK
WARNING:__main__:Dropping frame - IQ server, Total: 67
WARNING:shmemIface:Dropping frame.. Total: [68]
DEBUG:__main__:Type:1, CPI: 106, State:STATE_TRACK
WARNING:__main__:Dropping frame - IQ server, Total: 68
WARNING:__main__:Frame index missmatch. Expected 107 <--> 108 Received
WARNING:shmemIface:Dropping frame.. Total: [69]
DEBUG:__main__:Type:1, CPI: 108, State:STATE_TRACK
WARNING:__main__:Dropping frame - IQ server, Total: 69
WARNING:shmemIface:Dropping frame.. Total: [70]
DEBUG:__main__:Type:1, CPI: 109, State:STATE_TRACK
WARNING:__main__:Dropping frame - IQ server, Total: 70
WARNING:shmemIface:Dropping frame.. Total: [71]
DEBUG:__main__:Type:1, CPI: 110, State:STATE_TRACK
WARNING:__main__:Dropping frame - IQ server, Total: 71
WARNING:shmemIface:Dropping frame.. Total: [72]
DEBUG:__main__:Type:1, CPI: 111, State:STATE_TRACK
WARNING:__main__:Dropping frame - IQ server, Total: 72
WARNING:__main__:Frame index missmatch. Expected 112 <--> 113 Received
WARNING:shmemIface:Dropping frame.. Total: [73]
DEBUG:__main__:Type:1, CPI: 113, State:STATE_TRACK
WARNING:__main__:Dropping frame - IQ server, Total: 73
WARNING:shmemIface:Dropping frame.. Total: [74]
DEBUG:__main__:Type:3, CPI: 114, State:STATE_TRACK
DEBUG:__main__:Channel: 1, Peak dyn. range: -3.48[min: 20.00], Amp.:-35.03, Phase:-
37.92
DEBUG:__main__:Channel: 2, Peak dyn. range: -2.70[min: 20.00], Amp.:-
51.31, Phase:131.59
```

```
DEBUG:__main__:Channel: 3, Peak dyn. range: -0.70[min: 20.00], Amp.:-40.01, Phase:-
85.02
DEBUG:__main__:Channel: 4, Peak dyn. range: -7.09[min: 20.00], Amp.:-
39.58, Phase:23.74
DEBUG:__main__:Channel: 5, Peak dyn. range: -1.54[min: 20.00], Amp.:-35.73, Phase:-
5.00
DEBUG:__main__:Channel: 6, Peak dyn. range: -1.17[min: 20.00], Amp.:-
41.45, Phase:110.76
DEBUG:__main__:Channel: 7, Peak dyn. range: -0.52[min: 20.00], Amp.:-48.50, Phase:-
58.76
WARNING:__main__:Sample sync may lost
WARNING:__main__:IQ sync may lost
DEBUG:__main__:Differences: Amplitude 0.00, Phase: 0.00
DEBUG:__main__:Differences: Amplitude 35.08, Phase: 37.81
DEBUG:__main__:Differences: Amplitude 51.34, Phase: 131.95
DEBUG:__main__:Differences: Amplitude 40.09, Phase: 85.63
DEBUG:__main__:Differences: Amplitude 39.54, Phase: 24.62
DEBUG:__main__:Differences: Amplitude 35.76, Phase: 4.30
DEBUG:__main__:Differences: Amplitude 41.42, Phase: 111.04
DEBUG:__main__:Differences: Amplitude 48.49, Phase: 57.97
INFO:__main__:Delay track statistic [sync fails ,sample, iq, total][1,1,1/114]
WARNING:__main__:Dropping frame - IQ server, Total: 74
WARNING:shmemIface:Dropping frame.. Total: [75]
DEBUG:__main__:Type:3, CPI: 115, State:STATE_TRACK
DEBUG:__main__:Channel: 1, Peak dyn. range: 2.79[min: 20.00], Amp.:-
32.85, Phase:22.50
DEBUG:__main__:Channel: 2, Peak dyn. range: 1.29[min: 20.00], Amp.:-38.26, Phase:-
102.40
DEBUG:__main__:Channel: 3, Peak dyn. range: 6.58[min: 20.00], Amp.:-35.13, Phase:-
22.18
DEBUG:__main__:Channel: 4, Peak dyn. range: 14.76[min: 20.00], Amp.:-33.92, Phase:-
74.92
DEBUG:__main__:Channel: 5, Peak dyn. range: 1.27[min: 20.00], Amp.:-
38.85, Phase:36.57
DEBUG:__main__:Channel: 6, Peak dyn. range: 10.29[min: 20.00], Amp.:-
41.30, Phase:170.12
DEBUG:__main__:Channel: 7, Peak dyn. range: 1.96[min: 20.00], Amp.:-
42.59, Phase:141.16
WARNING:__main__:Sample sync may lost
WARNING:__main__:IQ sync may lost
DEBUG:__main__:Differences: Amplitude 0.00, Phase: 0.00
DEBUG:__main__:Differences: Amplitude 32.90, Phase: 22.61
DEBUG:__main__:Differences: Amplitude 38.28, Phase: 102.04
DEBUG:__main__:Differences: Amplitude 35.21, Phase: 22.79
DEBUG:__main__:Differences: Amplitude 33.88, Phase: 74.04
DEBUG:__main__:Differences: Amplitude 38.87, Phase: 37.26
DEBUG:__main__:Differences: Amplitude 41.28, Phase: 170.40
DEBUG:__main__:Differences: Amplitude 42.58, Phase: 141.95
INFO:__main__:Delay track statistic [sync fails ,sample, iq, total][2,1,1/115]
WARNING:__main__:Dropping frame - IQ server, Total: 75
WARNING:shmemIface:Dropping frame.. Total: [76]
DEBUG:__main__:Type:3, CPI: 116, State:STATE_TRACK
DEBUG:__main__:Channel: 1, Peak dyn. range: -2.31[min: 20.00], Amp.:-40.04, Phase:-
9.58
```

```
DEBUG:__main__:Channel: 2, Peak dyn. range: 6.73[min: 20.00], Amp.:-35.16, Phase:-
27.35
DEBUG:__main__:Channel: 3, Peak dyn. range: -6.84[min: 20.00], Amp.:-
41.30, Phase:11.25
DEBUG:__main__:Channel: 4, Peak dyn. range: 26.20[min: 20.00], Amp.:-
33.65, Phase:34.08
DEBUG:__main__:Channel: 5, Peak dyn. range: -0.70[min: 20.00], Amp.:-39.01, Phase:-
15.32
DEBUG:__main__:Channel: 6, Peak dyn. range: 0.99[min: 20.00], Amp.:-51.67, Phase:-
76.98
DEBUG:__main__:Channel: 7, Peak dyn. range: -0.37[min: 20.00], Amp.:-
43.53, Phase:44.76
WARNING:__main__:Sample sync may lost
WARNING:__main__:IQ sync may lost
DEBUG:__main__:Differences: Amplitude 0.00, Phase: 0.00
DEBUG:__main__:Differences: Amplitude 40.08, Phase: 9.47
DEBUG:__main__:Differences: Amplitude 35.18, Phase: 27.00
DEBUG:__main__:Differences: Amplitude 41.38, Phase: 10.64
DEBUG:__main__:Differences: Amplitude 33.60, Phase: 34.96
DEBUG:__main__:Differences: Amplitude 39.04, Phase: 14.62
DEBUG:__main__:Differences: Amplitude 51.65, Phase: 76.69
DEBUG:__main__:Differences: Amplitude 43.52, Phase: 45.55
INFO:__main__:Delay track statistic [sync fails ,sample, iq, total][3,1,1/116]
WARNING:__main__:Dropping frame - IQ server, Total: 76
WARNING:__main__:Frame index missmatch. Expected 117 <--> 118 Received
WARNING:shmemIface:Dropping frame.. Total: [77]
DEBUG:__main__:Type:3, CPI: 118, State:STATE_TRACK
DEBUG:__main__:Channel: 1, Peak dyn. range: 51.63[min: 20.00], Amp.:0.20, Phase:0.38
DEBUG:__main__:Channel: 2, Peak dyn. range: 50.29[min: 20.00], Amp.:0.10, Phase:0.04
DEBUG:__main__:Channel: 3, Peak dyn. range: 41.87[min: 20.00], Amp.:0.18, Phase:0.64
DEBUG:__main__:Channel: 4, Peak dyn. range: 40.73[min: 20.00], Amp.:0.16, Phase:0.10
DEBUG:__main__:Channel: 5, Peak dyn. range: 40.84[min: 20.00], Amp.:0.08, Phase:0.15
DEBUG:__main__:Channel: 6, Peak dyn. range: 45.59[min: 20.00], Amp.:0.07, Phase:0.34
DEBUG:__main__:Channel: 7, Peak dyn. range: 42.99[min: 20.00], Amp.:0.11, Phase:0.11
INFO:__main__:Delay track statistic [sync fails ,sample, iq, total][3,1,1/118]
WARNING:__main__:Dropping frame - IQ server, Total: 77
WARNING:shmemIface:Dropping frame.. Total: [78]
DEBUG:__main__:Type:3, CPI: 119, State:STATE_TRACK
DEBUG:__main__:Channel: 1, Peak dyn. range: 38.94[min: 20.00], Amp.:0.13, Phase:-
0.29
DEBUG:__main__:Channel: 2, Peak dyn. range: 40.02[min: 20.00], Amp.:0.11, Phase:-
0.35
DEBUG:__main__:Channel: 3, Peak dyn. range: 38.08[min: 20.00], Amp.:0.17, Phase:0.01
DEBUG:__main__:Channel: 4, Peak dyn. range: 48.27[min: 20.00], Amp.:0.11, Phase:-
0.64
DEBUG:__main__:Channel: 5, Peak dyn. range: 38.49[min: 20.00], Amp.:0.05, Phase:-
1.03
DEBUG:__main__:Channel: 6, Peak dyn. range: 39.39[min: 20.00], Amp.:0.08, Phase:-
0.64
DEBUG:__main__:Channel: 7, Peak dyn. range: 49.90[min: 20.00], Amp.:0.12, Phase:-
0.55
INFO:__main__:Delay track statistic [sync fails ,sample, iq, total][3,1,1/119]
WARNING:__main__:Dropping frame - IQ server, Total: 78
WARNING:shmemIface:Dropping frame.. Total: [79]
```

Config file used for this run:

```
[hw]
name = k8
unit_id = 0
ioo_type = 0
num_ch = 8
[daq]
log_level = 1
daq_buffer_size = 32768
center_freq = 600000000
sample_rate = 1000000
gain = 0
en_noise_source_ctr = 1
ctr_channel_serial_no = 1003
[squelch]
en_squelch = 0
amplitude_threshold = 0.5
[pre_processing]
cpi_size = 32768
decimation_ratio = 1
fir_relative_bandwidth = 1.0
fir_tap_size = 1
fir_window = hann
en_filter_reset = 0
[calibration]
corr_size = 32768
std_ch_ind = 0
en_frac_cal = 0
en_iq_cal = 1
gain_lock_interval = 20
unified_gain_control = 0
require_track_lock_intervention = 0
cal_track_mode = 2
cal_frame_interval = 10
cal_frame_burst_size = 500
amplitude_tolerance = 10
phase_tolerance = 5
maximum_sync_fails = 10
[adpis]
en_adpis = 0
en_gain_tune_init = 0
adpis_proc_size = 8192
adpis_gains_init = 0,0,0,0
[data_interface]
out_data_iface_type = eth
```

2.9.2 IQ Frame drop statistics

If you want to inspect the frame drop statistics you can enable display frame drop warnings in the shared memory interface and in the delay sync modules as well.

In shmemIface.py set

```
self.ignore_frame_drop_warning = False
```

Also in delay_sync.py set:

```
self.ignore_frame_drop_warning = False
```

In sh_mem_util.c set:

```
#define INGORE_FRAME_DROP_WARNINGS 0
```
and remake the sources

Frame drop is reported by the following modules:

Rebuffer

Decimator

….