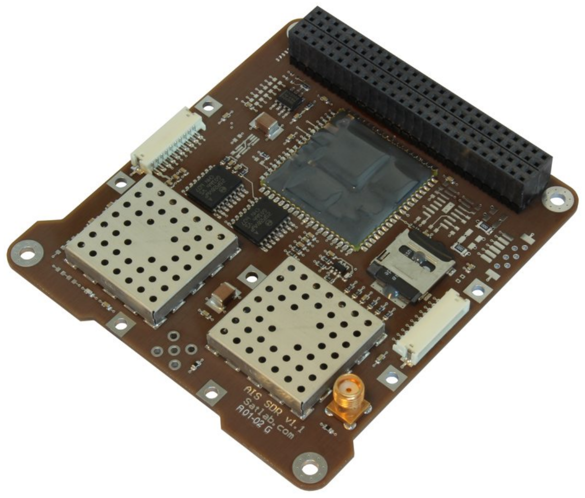
# Software-Defined AIS Receiver for CubeSats

Document ID **SLDS-AIS-1.4**, revision 1.4

## 1 Features

- Low power SDR-based AIS receiver
- Software configurable for 162 MHz channels or 156.8 MHz long-range channels
- Flight heritage from multiple CubeSat and micro-satellite missions
- Allows raw sampling of RF spectrum
- CubeSatKit compatible polyimide PCB
- On-board LNA, RF filters and data storage
- Cubesat Space Protocol (CSP) compatible over CAN-bus and UART
- Delivered with client software library for easy integration



## 2 Description

The Satlab QubeAIS is a fully self-contained SDR based AIS receiver, suitable for LEO satellite missions. Using less than 1 W during full load, this versatile SDR offers excellent performance given the typical constraints of a CubeSat - or as secondary payload on larger satellites. The receiver is software configurable for simultaneous reception of either AIS channels 1 & 2 (162 MHz) or the long-range channels 3 & 4 (156.8 MHz).

The trade-off between a reconfigurable hardware down-converter and a DSP based SDR algorithm, enables a large amount of processing power at a minimal power consumption. The demodulation algorithm uses per-packet adaptive filtering and center frequency estimation, which ensures good reception even at $>$3000 km line-of-sight and at a large Doppler frequency offset.

Support for the Cubesat Space Protocol (CSP) on CAN-bus or UART, eases integration and reduces the buy-to-fly-time significantly.

The QubeAIS is flight proven on several CubeSat and microsatellite missions, including AAUSAT3 flying in a 800 km polar orbit with a dipole antenna.

## 3 Electrical Details

| Parameter | Typ. Power Consumption |
|---|---|
| 3.3 V power | 760 $\mathrm{mW_{average}}$ |
| | 850 $\mathrm{mW_{peak}}$ |
| 5.0 V power | 125 $\mathrm{mW_{average}}$ |
| | 140 $\mathrm{mW_{peak}}$ |

Both 3.3 V and 5.0 V are required for operation.

| Interface | Options |
|---|---|
| 3.3 V supply | H1-48/50/52 |
| 5.0 V supply | H1-47/49/51 |
| Communication | CAN up to 1 Mb/s |
| | UART up to 500 kb/s |
| | SMA straight/right angle |
| Ant. connector | MCX straight/right angle |
| | Front or side placement |

## 4 Hardware Overview

The Satlab QubeAIS uses a low-IF frontend in conjunction with a 16-bit fixed point DSP to maximize performance at low power consumption. This also allows a single frontend to be used for receiving both AIS channels, on 162 or 156.8 MHz, at the same time. The following block diagram outlines the main hardware components of the receiver:

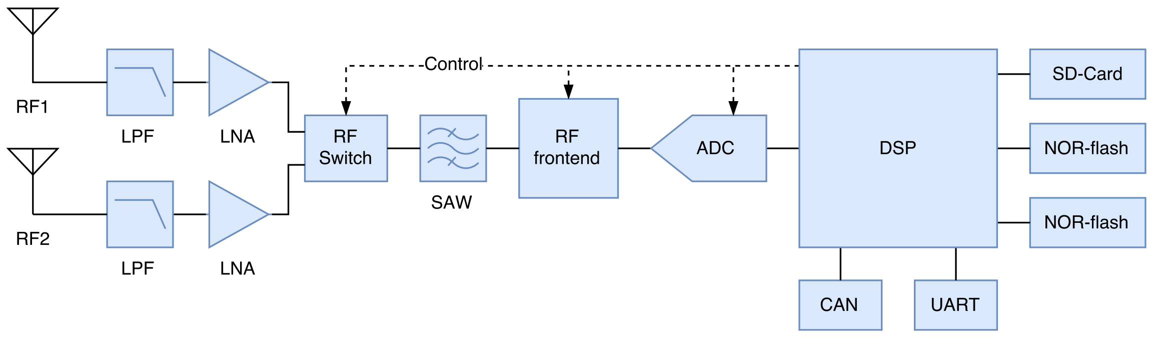


**Figure 1:** Overview of the hardware structure

The receiver allows for two placements of the RF connector, RF1 or RF2. Each connector is followed by a Low Pass Filter (LPF) to reduce the influence of strong nearby transmitters, such as satellite downlink in the UHF- and S-band. If the satellite uses downlink in the VHF band, please consult Satlab ApS before ordering. An RF switch controlled by the DSP selects which input connector to use.

The on-board LNA ensures a good noise figure for the overall system, and simplifies the final integration process as the system only requires a passive antenna and the QubeAIS. The LNA on the unused input is powered down by the DSP to conserve power.

The SAW filter covers the entire marine VHF band, to increase out-of-band rejection, while allowing the receiver to be tuned to both the 162 MHz and 156.8 MHz AIS channels - or any future allocations. As long as two AIS channels are allocated in the marine VHF band, and no more than 50-150 kHz apart, the receiver will be able to receive both channels at the same time (with a software update to re-tune the receiver).

The RF frontend performs single-conversion to a low IF, while providing good image rejection using a built-in IF filter. The filter is reprogrammable to 100, 150 or 200 kHz bandwith. The default 100 kHz is sufficient to contain both AIS channels on 162 or 156.8 MHz. Reprogrammable RF, IF and ADC gains are used to ensure high dynamic range and reconfigurability after launch.

A 12-bit ADC is used to digitize the filtered IF and is the only part running from the 5.0 V DC supply.

The DSP is the central part of the design. It contains RAM, flash and interfaces for communication and data storage. Channel filtering, GMSK demodulation and decoding of both of the AIS channels are all executed in the DSP. A packet detector estimates key parameters for each individual AIS message, and passes these on to the packet filter and demodulator, to ensure good reception in LEO. The full algorithm has been optimized for the 16 bit fixed point DSP, to allow fast processing at low power consumption. A special operating mode allows the DSP to save raw IF samples to persistent storage, for download and processing on the ground.

Connected to the DSP are two 16 MB NOR-flashes and a 1 GB industrial grade SD-card with built-in error correction capabilities. The SD-card is the primary application and data storage for the receiver, while one of the NOR-flashes acts as backup storage in case the SD-card fails. The other NOR-flash is used as configuration memory and is operated in read-only mode after launch.

## 5 Software Overview

The QubeAIS receiver software consists of 3 main parts, shown in figure 2: The platform, AIS loader and the AIS application. This section describes the operation of each component in detail.

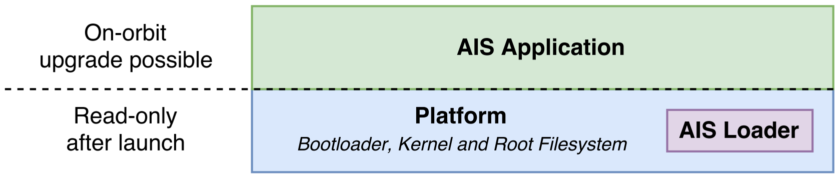

On-orbit upgrade possible
Read-only after launch
AIS Application
Platform
Bootloader, Kernel and Root Filesystem
AIS Loader

**Figure 2:** Overview of the AIS receiver software components

The platform and AIS loader both reside in write-protected internal flash on the DSP. They can be upgraded on ground via the debug interface, but remain fixed after launch so the receiver always boots in a known configuration. The AIS Application is executed by the AIS loader from SD-card or NOR-flash and can be upgraded on-orbit if necessary.

### 5.1 Platform

The platform component includes the U-Boot bootloader, Linux kernel and a minimal root filesystem including Busybox shell and common debugging tools. On boot, the entire platform is loaded from write-protected flash into RAM and executed. Access to the Linux command shell is available on the debug UART pins in the P2 connector, although this is not required for the operation of the receiver.

### 5.2 AIS Loader

After the kernel is started, the platform initialization system will mount the SD-card and NOR-flash filesystems to `/mnt/sd` and `/mnt/nor` respectively. Then, the AIS loader program is run from `/usr/bin/aisloader` in the platform root filesystem.

The AIS loader will first search for an `aisboot` file on the SD-card, which can be used to instruct the loader to boot a different receiver application than the default. If no `aisboot` file is found on the SD-card, the NOR-flash will be checked. If the NOR-flash does not contain an `aisboot` file either, the AIS loader will check for the existence of the default receiver at `/mnt/sd/p/ais` and run it if it exists. In case no default receiver is found on the SD-card, the `/mnt/nor/p/ais` receiver from NOR-flash will be started.

In the rare case that neither the SD-card nor the NOR-flash contain a valid receiver program, or if the started receiver program faults, the AIS loader will start a fallback server that includes a BTP server (see section 7.3). This can then be used to diagnose the problem via remote shell or upload a new receiver software.

### 5.3 AIS Application

The AIS application implements the primary functionality of the QubeAIS receiver. As shown on Figure 3, the 100 kHz IF output from the RF-frontend is sampled by the ADC, and processed by the demodulator block as described in section 5.3.1. The demodulator batch processes samples of approximately 700 ms of RF data. In the Message Decoder, candidate message from the demodulator are checked for correct HDLC frame structure and the CRC checksum is verified.

If the message was correctly received, it is passed to the Message Filter which can be used to only accept messages with certain MMSI numbers, message IDs, positions and/or country of origin. Finally, all single-slot accepted messages are tagged with a sequence number and stored in a persistent list on SD-card or NOR-flash. Furthermore, the most recently received position reports from each MMSI is also stored in a RAM-based unique list. This makes it possible to quickly download the most recent data. AIS messages spanning 2 slots, typically static vessel information, are stored in a dedicated static list in flash.

The receiver is always operating in one of five *modes*; IDLE, RUNNING_SINGLE, RUNNING_CONT, RUNNING_AND_-SAVING_SINGLE, or RUNNING_AND_SAVING_CONT. In IDLE mode the sampler and demodulator are disabled to save power, and only the CSP services are running. RUNNING_SINGLE and RUNNING_CONT enables the sampler and demodulator for either a single or multiple samples. The demodulator can be requested to store the raw IF data to the sample store on either SD-card or NOR-flash. This mode is enabled in the RUNNING_AND_SAVING_SINGLE and RUNNING_AND_SAVING_CONT modes. The QubeAIS can be configured to start in either IDLE or RUNNING_CONT mode. See subsection 7.1.2 for more info on switching modes.

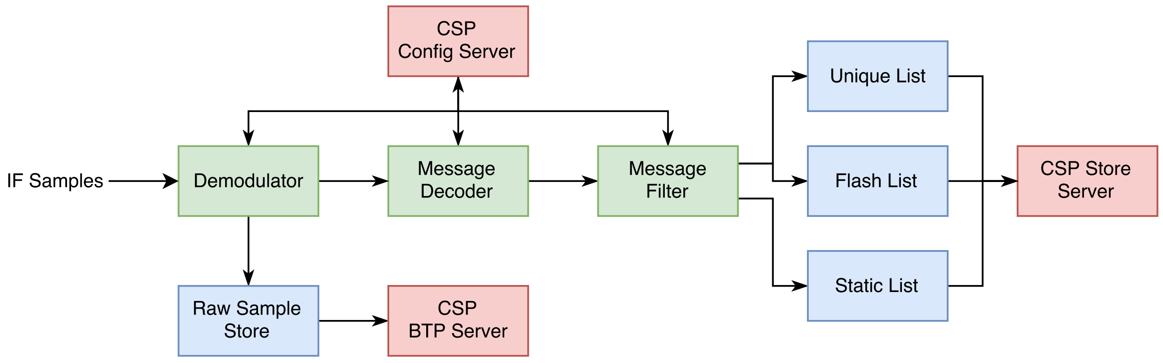


**Figure 3:** Overview of the software structure

Three CSP servers are used to configure the receiver, read back status and to download messages and raw samples. The Blob Transfer Protocol (BTP) is a lightweight file transfer protocol built on CSP, that is used to provide reliable transfer of the raw sample files. BTP is further explained in Application Note SLAN-BTP and a client implementation is available on request.

### 5.3.1 Demodulator and Frame Decoder

The main part of the receiver executable is the demodulator. The demodulator and frame decoder are responsible for converting the sampled data to valid AIS messages. Figure 4 outlines the data flow in these two blocks.

Input samples from the ADC is first filtered by two channel filters to isolate the two AIS channels, either channels 1 & 2 or channels 3 & 4 depending on the frontend setting. A packet detector is then run on the filtered data to search for possible packets on the channels. Using a packet detector has the advantage of lowering the power consumption of the receiver, since the remaining demodulator stages are only run when packets are detected. Since AIS messages received in space can be subject to up to ±4 kHz of Doppler shift, a center frequency estimator is used to adjust the subsequent stages of the demodulation algorithm.

When a packet is detected, its position in the sample and its estimated center frequency is passed to a narrow packet filter. Output from the packet filter is processed by the GMSK demodulator and passed to the clock and data recovery block for reconstruction of the 9600 bps bitstream. Finally, the bitstream is checked for correct HDLC framing structure and CRC checksum by the packet verification block.

For customers who wish to develop their own demodulator, Satlab provides a demodulator plugin kit that allows the Demodulator, Message Decoder and Message Filter to be replaced with a custom library. The receiver allows on-orbit switching between multiple plugins, e.g. for performance comparison between multiple algorithms. Plugins are implemented as a dynamic library that process the raw IF samples and has access to the normal message store and message download via CSP. Please contact Satlab for more information on this option.

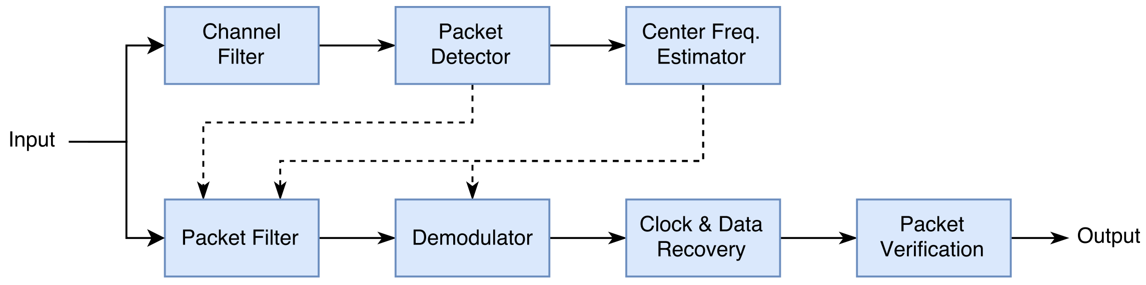


**Figure 4:** Overview of the demodulator and frame decoder

### 5.3.2 Message Store and Message Filter

As shown in the software overview on figure 3, correctly demodulated messages can be filtered by a message acceptance filter. This allows messages to be filtered on MMSI number, minimum/maximum latitude/longitude, MID (Maritime Identification Digits, the part of an MMSI that denotes the ships country of origin) and message types. As default, the filter accepts all message types, MMSIs and positions.

Received position messages are stored in a file called `aisdata.bin` on either the SD-card or the NOR-flash, depending on where the receiver program is running from. The `aisdata.bin` file has a default size of 100000 messages of 40 bytes each. A 2000-entry unique list is maintained in RAM and will store the last received message from up to 2000 MMSIs. Additionally, a 15000 messages buffer is used to store two-slot AIS messages.

Messages are stored in the flash and unique list with a 32-bit sequence number and a 32-bit timestamp (UNIX time, i.e. number of seconds since midnight on January 1st, 1970). Each message is stored with up to 26 bytes AIS message data and 14 bytes overhead for a total size of 40 bytes. In the message store, free messages are marked with a sequence number of `0xffffffff`.

Two CSP ports are available for download of messages. One port, `AIS_PORT_AIS_STORE`, provides access to the full 40 byte messages while `AIS_PORT_AIS_STORE_SHORT` can be used to download position reports in a condensed 20 byte format, referred to as a *short frame*. The short frames contain MMSI, message ID, position, timestamp and sequence number. In addition, the `AIS_PORT_STORE_REQUEST` can be used to request messages based on a selection bitfield.

Tables 1 and 2 lists the regular and short AIS frames data structures. The reserved fields contain debugging output from the demodulator and may be changed in a future version.

**Table 1:** The common `ais_frame_t` data structure

| **Field Name** | **Field Type** | **Description** |
|---|---|---|
| `seq_nr` | `uint32_t` | Message sequence number |
| `time` | `uint32_t` | UNIX timestamp of message reception in seconds |
| `rssi` | `uint8_t` | Estimated message RSSI |
| `res` | `uint8_t`[3] | Reserved |
| `flags` | `uint8_t` | Flag bits, see description |
| `size` | `uint8_t` | Size of data in bytes |
| `data` | `uint8_t[26]` | Data array |

In the `flags` field, bit 0 (the LSB) is set for messages with correct CRC checksum. Bits 1 & 2 mark the AIS channel the message was received on, i.e. `0b00` to `0b11` for channels 1 to 4 respectively. Bit 3 to 7 are used to mark the buffer the message was downloaded from. It will be equal to AIS_BACKEND_FILE (0) for messages from the file store and equeal to AIS_BACKEND_UNIQUE (1) for messages from the unique list.

The ais_short_frame_t data structure listed in table 2 is used for condensed data from AIS position reports.

rssi is a pseudo-RSSI figure, meaning that it will indicate the signal magnitude in the ADC range. It is thus not the signal power in dBm.

**Table 2:** The common ais_short_frame_t data structure

| Field Name | Field Type | Description |
|---|---|---|
| mmsi | uint32_t | AIS message MMSI number |
| time | uint32_t | UNIX timestamp of message reception |
| rssi | uint8_t | Estimated message RSSI |
| msgid | uint8_t | AIS message ID |
| flags | uint8_t | Flag bits, see description |
| lat | int32_t | Message latitude in 1/10000 min. |
| lon | int32_t | Message longitude in 1/10000 min. |

The ais_frame_static_t data structure listed in table 3 is used for static AIS data such as message ID 5 and 24. These include vessel name, destination and IMO number.

**Table 3:** The common ais_frame_static_t data structure

| Field Name | Field Type | Description |
|---|---|---|
| seq_nr | uint32_t | Message sequence number |
| time | uint32_t | UNIX timestamp of message reception |
| rssi | uint8_t | Estimated message RSSI |
| res | uint8_t[3] | Reserved |
| flags | uint8_t | Flag bits, see description |
| size | uint8_t | Size of data in bytes |
| data | uint8_t[55] | Data array |

## 5.4 Setup

One of the NOR-flash devices shown in Figure 1 is used as a configuration partition, mounted read-only under /mnt/rom in the root filesystem. The config directory on the partition contains a number of configuration files, listed in table 4. The files are parsed by both the AIS loader and the AIS application during startup. If a file is not found, or contains invalid data, the default value is used.

It is strongly recommended not to alter these parameters on-orbit, since setting them to a invalid value could render the receiver unresponsive.

**Table 4:** Nonvolatile configuration parameters and default settings

| Parameter | Range | Default Value | Description |
|---|---|---|---|
| canrate | 125000 - 1000000 | 1000000 | CAN bitrate |
| uartrate | 9600 - 500000 | - | UART bitrate |
| cspaddress | 0-30 | 3 | CSP address of receiver |
| cspgateway | 0-30 | 4 | CSP gateway of receiver |
| csproutes | Text | - | CSP routes, see format in description |
| cspcryptokey | Text | - | CSP XTEA/HMAC key |
| inputant | 1 or 2 | 1 | Input antenna connector |

The canrate and uartrate files contain the bitrate for the CAN-bus and UART interfaces. The UART rate should be set to 0 (or the file deleted) if UART is not used. CSP configuration is handled with the csp* files. By default, the receiver is configured to use CSP address 3 (cspaddress) and to forward all traffic through node 4 (cspgateway), which is typically the spacecraft radio or OBC. The default route is set to send all traffic via the CAN interface, but this can be overwritten using the csproutes file. The syntax of the file is:

```
address[:mac]=interface[,address[:mac]=interface].
```

The address can be either a numerical CSP destination address from 0-31 or ”default” to set the default route. The optional mac field selects the ”MAC layer” address, i.e. the destination address in the CAN identifiers. If omitted, the destination address will also be used as MAC address. On the point-to-point UART interface the MAC field is ignored. interface must be either can to use the CAN-bus or kiss to use KISS frames on UART.

As an example, to set all traffic to go via the UART interface use:

```
default=kiss
```

CSP message authentication (HMAC) and encryption (XTEA) is supported if the cspcryptokey is set. Note that setting this value does not mandate HMAC/XTEA on incoming connections, but the receiver will reply with matching authentication settings as incoming frames.

To change a configuration variable, use the Linux shell interface and remount the /mnt/rom partition writable and echo the new value into the file:

```
root:/> mount -o remount,rw /mnt/rom
root:/> echo 500000 > /mnt/rom/config/uartrate
root:/> echo default=kiss > /mnt/rom/config/csproutes
```

A reboot of the receiver is required for changes of the configuration variables to take effect.

## 6 Receiver Performance

In figure 5 the typical reception performance for the AIS receiver is shown. The bold line shows the average reception performance over the full temperature range from -40℃ to +70℃ and for the two AIS frequencies on 161.975 MHz and 162.025 MHz ±3 kHz Doppler shift. The three thinner lines shows the average over the frequency span, but for maximum, minimum and typical operating temperatures.

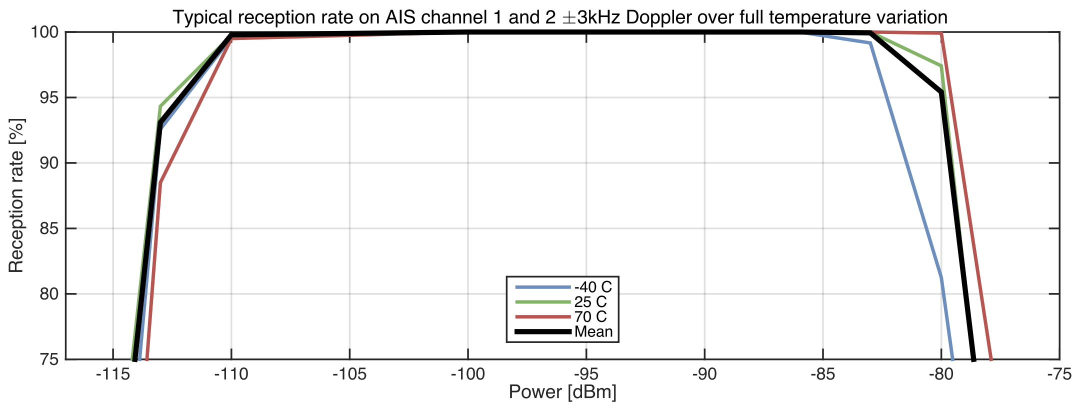


**Figure 5:** Typical receiver performance as a function of signal power averaged over AIS channel 1 and 2 $\pm$3 kHz Doppler.

This can be related to a simplified link budget simulation. The basis of this simplified signal level budget is a Class A vessel with vertical antenna, illustrated in figure 6. The ship transponder output is assumed to be 12.5 W, with 1 dB loss from transponder to the antenna. The 3 ship antenna types are assumed over ideal ground plane and on the satellite side a 0 dBi antenna and 1 dB cable loss is assumed. This results in the signal levels into the receiver as shown in figure 7 and 8 for a 400 km and 600 km orbit respectively. The two dashed lines shows the reception limits for the receiver, showning that with the default settings of the receiver, the AIS signal strength falls nicely within the operating range of the AIS receiver.

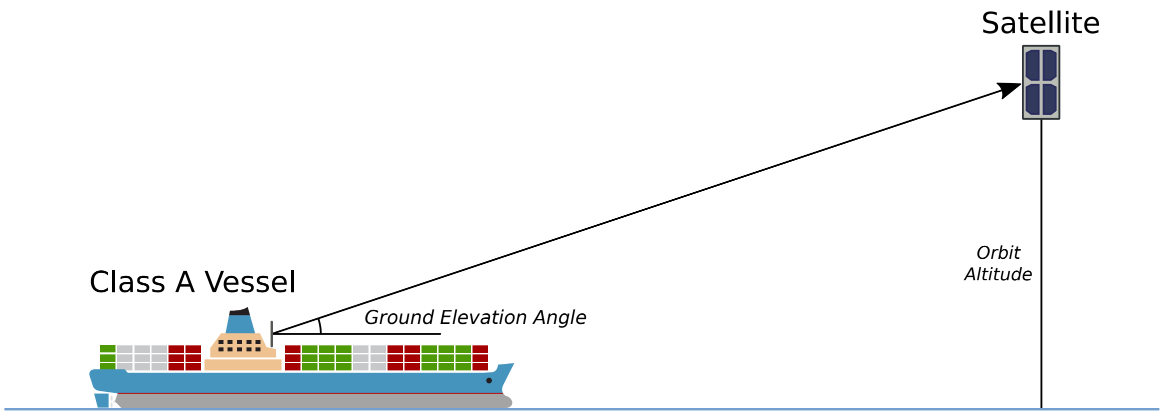


**Figure 6:** Simplified receiver sensitivity simulation

*It should be noted that this is a very simplified calculation and a detailed link budget analysis should be performed for each mission individually.*

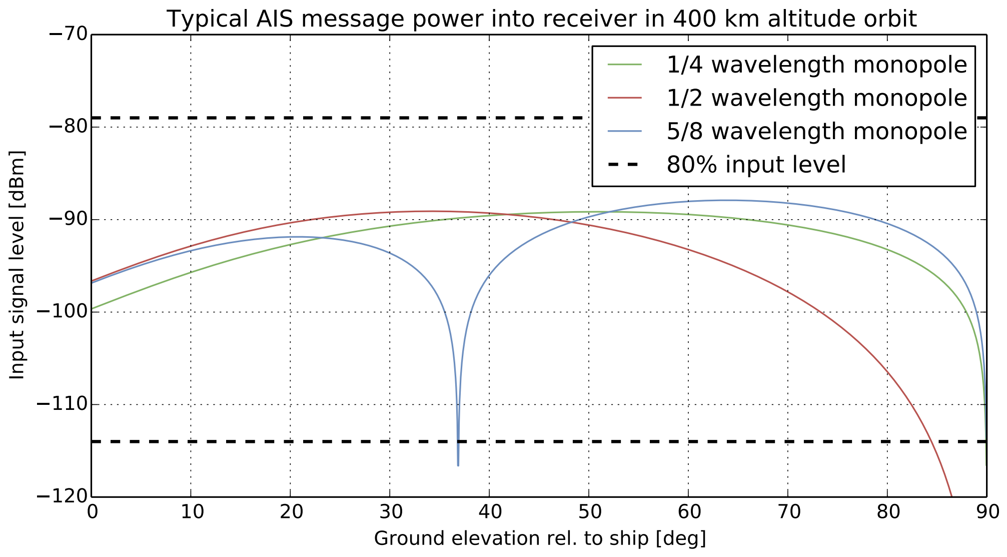


**Figure 7:** Receiver performance 400 km orbit

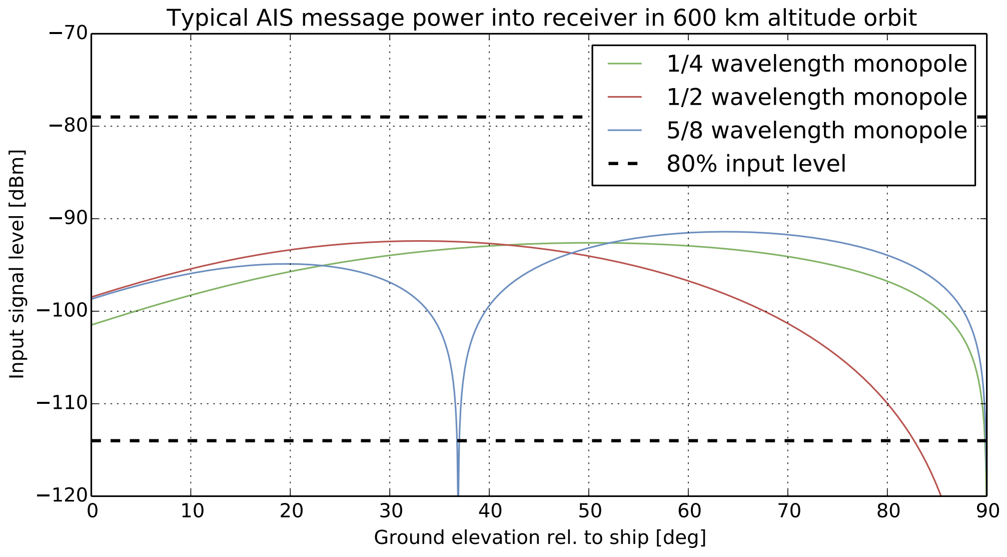


**Figure 8:** Receiver performance 600 km orbit

# 7 Software Interface Description

The QubeAIS receiver is controlled using the Cubesat Space Protocol (CSP) via CAN-bus or UART/KISS. This chapter lists the available service ports and data structures used to communicate with the device. The CSP address of the receiver is set using the configuration variables as explained in section 5.4. Table 5 lists the available service ports with a short description of each service.

When the device is booted, it will automatically start listening for requests on the service ports. Each service can be categorized as either *Configuration and Status*, *Message Store* or *Blob Transfer Protocol*.

All data structures are packed, i.e. no alignment padding is used between the fields, and unless otherwise noted all fields are transferred in network byte order (big endian).

This chapter is meant to serve as a reference. Satlab supplies a support library, `libsatlab`, containing an `ais.h` header file with all data structures and wrapper functions around the CSP calls to simplify integration of the receiver.

**Table 5:** CSP service names and ports.

| Port Name | Port Number | Service Description |
|---|---|---|
| AIS_PORT_STATUS | 8 | AIS receiver status |
| AIS_PORT_MODE | 9 | Receiver mode configuration |
| AIS_PORT_AIS_STORE | 11 | Message store for long packets |
| AIS_PORT_AIS_STORE_SHORT | 12 | Message store for short packets |
| AIS_PORT_BTP | 13 | Blob Transfer Protocol |
| AIS_PORT_AIS_RX_STATUS | 17 | Detailed AIS receiver and message filter status |
| AIS_PORT_CONFIG | 19 | AIS general configuration |
| AIS_PORT_AIS_STORE_STATIC | 20 | Message store for static packets |
| AIS_PORT_STORE_CONFIG | 21 | Message store config |
| AIS_PORT_STORE_REQUEST | 22 | Message store bitmap request |
| AIS_PORT_TIME_CONFIG | 23 | Time configuration |
| AIS_PORT_CHANNEL_CONFIG | 24 | Channel configuration |
| AIS_PORT_DEMOD_CONFIG | 30 | Custom demodulator configuration |
| AIS_PORT_DEMOD_USER | 31 | Custom demodulator user port |

## 7.1 Configuration and Status Interface

### 7.1.1 AIS_PORT_STATUS

This port is used for reading the current status of the receiver, and to verify that it is operating nominally. When the QubeAIS receives a CSP packet on the AIS_PORT_STATUS port, it will return the data structure listed in table 6.

**Table 6:** The `ais2_port_status_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `critical_errors` | `uint8_t` | Number of critical errors since boot |
| `warnings` | `uint8_t` | Number of warnings since boot |
| `runs` | `uint16_t` | Number of samples processed since boot |
| `packs_detected` | `uint16_t` | Number of packets detected since boot |
| `running_from` | `uint8_t` | Storage ID where the program is running |
| `bootcount` | `uint16_t` | Total number of times the system has booted |
| `crc_ok` | `uint32_t` | Number of received messages with correct CRC since boot |
| `unique_mmsi` | `uint16_t` | Number of unique MMSIs received since boot |
| `latest_mmsi` | `uint32_t` | MMSI of the last received message |
| `latest_long` | `int32_t` | Longitude of last received message in 1/10000 min. |
| `latest_lat` | `int32_t` | Latitude of last received message in 1/10000 min. |
| `rssi` | `uint8_t` | RSSI of 100 kHz channel bandwidth |
| `flags` | `uint8_t` | Status flags |

The `running_from` field will return AIS_RUNNING_FROM_SD (numerical value 0x00) when running from the SD-card, and AIS_RUNNING_FROM_NOR (numerical value 0x01) when running from the NOR-flash.

When the status is requested, the receiver will measure the RSSI in the receiver frontend and return it in the `rssi` field. The field step size is 0.5 dBm, and the range is shifted so an `rssi` field value of 0 corresponds to -140 dBm.

The `flags` byte is currently unused and will always be read as 0.

### 7.1.2 AIS_PORT_MODE

This port sets the receive mode of the QubeAIS as described in section 5. Depending on configuration, the system will start in either idle or continuous mode. As default the receiver will start in continuous mode.

Table 7 shows the fields in the `ais2_port_mode_t` data structure.

**Table 7:** The `ais2_port_mode_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `command_code` | `uint8_t` | Sets the new receiver mode |
| `storage_id` | `uint8_t` | Storage where RAW samples should be stored. 0=SD-card, 1=NOR-flash |
| `file_name_no` | `uint8_t` | First sample file number |
| `num_of_samples` | `uint8_t` | Number of samples to process |

The `command_code` field should be one of AIS2_MODE_IDLE (value 0), AIS2_MODE_RUNNING_SINGLE (value 1), AIS2_MODE_RUNNING_CONT (value 2), AIS2_MODE_RUNNING_AND_SAVING_SINGLE (value 3), AIS2_MODE_RUNNING_AND_SAVING_CONT (value 4).

`num_of_samples` is only used in AIS2_MODE_RUNNING_CONT and AIS2_MODE_RUNNING_AND_SAVING_CONT. If the value is larger than 0, the receiver will go to IDLE mode when the specified number of samples has been processed. If `num_of_samples` is 0, the receiver will continue to process samples until manually set in IDLE mode.

The `file_name_no` field is used to generate the file name for the save samples in AIS2_MODE_RUNNING_AND_-SAVING_SINGLE and AIS2_MODE_RUNNING_AND_SAVING_CONT modes. Sample files are named `hex(file name number + sample number).i`. If e.g. 10 samples are requested (`num_of_samples` = 10) with `file_name_no` = 10, the files will be named `0a.i` to `13.i`.

`storage_id` will select where the samples are store in AIS2_MODE_RUNNING_AND_SAVING_SINGLE and AIS2_-MODE_RUNNING_AND_SAVING_CONT modes. Data is stored on `/mnt/sd/d` and `/mnt/nor/d` respectively.

An `ais2_port_mode_t` with the new settings will be returned when a mode request is received.

### 7.1.3 AIS_PORT_AIS_RX_STATUS

The AIS_PORT_AIS_RX_STATUS returns a detailed receiver and message filter status. When a CSP packet is received on the AIS_PORT_AIS_RX_STATUS port, the `ais_rx_status_t` data structure listed in table 8 is returned.

**Table 8:** The `ais_rx_status_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `id` | `uint32_t` | Current message filter acceptance mask |
| `latmin` | `int32_t` | Current message filter minimum latitude |
| `latmax` | `int32_t` | Current message filter maximum latitude |
| `lonmin` | `int32_t` | Current message filter minimum longitude |
| `lonmax` | `int32_t` | Current message filter maximum longitude |
| `crc_ok` | `uint32_t` | Number of messages received with correct CRC |
| `crc_error` | `uint32_t` | Number of messages received with incorrect CRC but correct packet length |
| `filter_match_ok` | `uint32_t` | Number of messages that matched message acceptance filter |
| `rej_id` | `uint32_t` | Number of messages dropped because of message ID |
| `rej_pos` | `uint32_t` | Number of messages dropped because of position |
| `rej_mmsi` | `uint32_t` | Number of messages dropped because of MMSI |
| `rej_cc` | `uint32_t` | Number of messages dropped because of country code |
| `rej_time` | `uint32_t` | Number of messages dropped because of timestamp |

The first 5 fields are related to the message acceptance filter. The `id` field is a bitfield with the currently accepted message types marked by set bits, e.g. if bit 18 is set, message type 18 is accepted. `latmin`/`latmax` and `lonmin`/`lonmax` are the current message filter area in 1/10000 min.

`crc_ok`, `crc_error` and `filter_match_ok` counts the number of messages with correct and erroneous CRC checksum, as well as the number of messages that passed the message filter constraints.

The `rej_id`, `rej_pos`, `rej_mmsi`, and `rej_cc` fields counts the drop reasons in the message filter. The `rej_time` field is currently unused. Only the first field that result in the packet being rejected is incremented.

### 7.1.4 AIS_PORT_CONFIG

This port is used for configuring the RF gain stages, and to enable or disable autostart (start in running continous mode). All fields of the `ais2_port_config_t` structure are shown in table 9.

The `boots` field sets the number of boots the configuration should be valid for. If the configuration should only be valid until next reboot, use `boots` = 0. To use a configuration for all future boots, use `boots` = 0xffff.

The receiver will reply with the updated configuration.

**Table 9:** The `ais2_port_config_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `boots` | `uint16_t` | How many boots the configuration should be valid |
| `adc_gain` | `uint8_t` | ADC gain value. Use 0xff to keep current value |
| `filter_gain` | `uint8_t` | RF frontend filter gain value. Use 0xff to keep current value |
| `lna_gain` | `uint8_t` | RF frontend LNA gain value. Use 0xff to keep current value |
| `autostart` | `uint8_t` | If nonzero, enable receiver autostart |

### 7.1.5 AIS_PORT_TIME_CONFIG

This port allows the system time of the QubeAIS to be adjusted and read back. The receiver does not have an RTC so the system time is undefined after startup. Since all received AIS frames are timestamped with the system time, it is thus recommended to set the time when the system is powered on. The time format is UNIX time, i.e. number of seconds since midnight on January 1st, 1970, with each timestamp is split into seconds and nanoseconds components.

To set the system time, send an `ais_port_time_config_set_time_req` with the new time. The receiver will reply with an `ais_port_time_config_set_time_rep` structure containing an error/success return value and the updated system time.

To set the receiver time, send an `ais_port_time_config_set_time_req` with the `sec` and `nsec` fields set to the new time in seconds and nanoseconds since 00:00:00 on January 1st 1970 (also known as “Epoch time”).

**Table 10:** The `ais_port_time_config_set_time_req_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `type` | `uint8_t` | Request type, must be `AIS_PORT_TIME_CONFIG_SET_TIME` |
| `sec` | `uint32_t` | New seconds |
| `nsec` | `uint32_t` | New nanoseconds |

The receiver will reply with an `ais_port_time_config_set_time_rep` packet containing the same type field and an error code (0 for success, error code on error) and the updated time. The error codes are listed in `libsatlab`.

**Table 11:** The `ais_port_time_config_set_time_rep_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `type` | `uint8_t` | Request type, must be `AIS_PORT_TIME_CONFIG_SET_TIME` |
| `error` | `uint8_t` | Error code |
| `sec` | `uint32_t` | New seconds |
| `nsec` | `uint32_t` | New nanoseconds |

To request the current system time, send a single byte `ais_port_time_config_get_time_req`. The receiver will reply with an `ais_port_time_config_get_time_rep` which is identical to the set time reply, except the type will be `AIS_PORT_TIME_CONFIG_GET_TIME`.

**Table 12:** The `ais_port_time_config_get_time_req_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `type` | `uint8_t` | Request type, must be `AIS_PORT_TIME_CONFIG_GET_TIME` |

### 7.1.6 AIS_PORT_CHANNEL_CONFIG

To select between AIS channels 1 & 2 (162 MHz center) and long-range channel 3 & 4 (156.8 MHz), send a `channel_config` structure to the `AIS_PORT_CHANNEL_CONFIG` port, with the `channel` field set to one of these defines:

```
#define AIS_PORT_CHANNEL_CH1_2 1
#define AIS_PORT_CHANNEL_CH3_4 2
```

The receiver will reply with a similar structure with the `error` field set to 0 for success or a non-zero error code on failure. The receiver must be rebooted for the change to take effect.

**Table 13:** The `ais_port_channel_config` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `type` | `uint8_t` | Request/reply type, must be AIS_PORT_CHANNEL_SET/GET |
| `error` | `uint8_t` | Error code, 0 for success |
| `channels` | `uint8_t` | Channel selection, see description for valid values |
| `flags` | `uint32_t` | Flags, currently unused |

### 7.1.7 AIS_PORT_DEMOD and AIS_PORT_DEMOD_USER

These two ports are used for the plugin demodulator functionality described in section 5.3.1. `AIS_PORT_DEMOD` is used to select between loadable modulators, while `AIS_PORT_DEMOD_USER` is reserved for use by customer code to setup proprietary configuration.

Please contact Satlab if you plan on using this feature.

## 7.2 Message Store Interface

### 7.2.1 AIS_PORT_STORE

The AIS_PORT_STORE provides access to the AIS message store. As described in the software overview, received messages are stored in a fixed size ring buffer with a sequence number and a timestamp. Table 14 lists the message store request structure. The `from_seq_nr` field selects the first sequence number to request. Note that messages are requested *back* in time, i.e. if 10 messages are requested from sequence number 100, messages 100 to 91 are returned.

The `max_num_of_frames` field selects the maximum number of frames the receiver should reply with, from 1 to 1000. If more frames are requested than are currently available, empty frames (sequence number 0xffffffff) will be added until a full packet can be sent, i.e. no packets are sent with only empty frames. The backend selects the storage backend to request messages from. It must be either AIS_BACKEND_FILE to request from the flash buffer or AIS_BACKEND_UNIQUE to request from the RAM-based unique list. The static interface is only compatible with AIS_BACKEND_STATIC and is stored in a separate file.

The `num_of_frames_per_reply` selects the number of `ais_frame_t` frames that should be returned in each `ais_port_ais_store_rep_t`. The `delay_between_replys` field can be used to insert an optional delay between each reply packet, e.g. if limited buffering is available on the spacelink.

**Table 14:** The `ais_port_ais_store_req_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `from_seq_nr` | `uint32_t` | First sequence number to request. Use 0 to request newest |
| `max_num_of_frames` | `uint16_t` | Maximum number of message to transmit (1-1000) |
| `backend` | `uint8_t` | Storage backend to request message from |
| `num_of_frames_per_reply` | `uint8_t` | Number of messages to store in each reply packet (1-4) |
| `delay_between_replys` | `uint8_t` | Delay in between packets in 10 ms steps |

The receiver will answer the message request with a number of `ais_port_ais_store_rep_t` structures as described in table 15. Note that the `seq_nr` field will contain the newest sequence number available in the buffer regardless of the sequence number that were requested. The `frame` field is an array of `num_of_frames_per_reply` AIS messages.

**Table 15:** The `ais_port_ais_store_rep_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `seq_nr` | `uint32_t` | Sequence number of last received message in buffer |
| `frame` | `ais_frame_t[]` | Array of 1-MAX_STORE_REPLY_FRAMES AIS frame data structures |

### 7.2.2 AIS_PORT_STORE_SHORT

The AIS_PORT_STORE_SHORT takes the same `ais_port_ais_store_req_t` data structure requests as AIS_ PORT_-STORE, but replies with short frames instead. The short frames are encapsulated in an `ais_port_ais_short_rep_t` that is listed in table 16. Note that the message sequence number is not included in the short frame. Instead, the requested sequence number is returned in the `seq_nr` field in the `ais_port_ais_short_rep_t` structure.

**Table 16:** The `ais_port_ais_short_rep_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `seq_nr` | `uint32_t` | Sequence number of first message |
| `frame` | `ais_short_frame_t[]` | Array of 1-MAX_STORE_SHORT_REPLY_FRAMES short frames |

### 7.2.3 AIS_PORT_STORE_STATIC

The AIS_PORT_STORE_STATIC takes the same `ais_port_ais_store_req_t` data structure requests as AIS_ PORT_-STORE, but replies with static frames instead. The static frames are encapsulated in an `ais_port_ais_static_rep_t` that is listed in table 17. Note that the static store always uses `AIS_BACKEND_STATIC`.

**Table 17:** The `ais_port_ais_static_rep_t` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `seq_nr` | `uint32_t` | Sequence number of last received message in static buffer |
| `frame` | `ais_frame_static_t[]` | Array of 1-MAX_STORE_STATIC_REPLY_FRAMES static frames |

### 7.2.4 AIS_PORT_STORE_CONFIG

This port allows modification of the message stores parameters and readback of the current store status. Two message types are accepted at this point: AIS_PORT_STORE_CONFIG_SET_SIZE and AIS_PORT_STORE_CONFIG_GET_-STATUS.

To change the size of one the message stores, send an `ais_port_store_config_size_req` to the store config port. Note that the store size will not be updated immediately, and the receiver must be rebooted for the change to take effect.

**Table 18:** The `ais_port_store_config_size_req` data structure

| Field Name | Field Type | Description |
|---|---|---|
| `type` | `uint8_t` | Request type, must be AIS_PORT_STORE_CONFIG_SET_SIZE |
| `backend` | `uint8_t` | The backend to configure |
| `size` | `uint32_t` | New store size in messages |

The type field must be set to AIS_PORT_STORE_CONFIG_SET_SIZE. The backend selects the backend to configure (FILE, UNIQUE or STATIC), and the size field sets the new store size in messages. The receiver replies with a config size reply packet that includes an error code and the new configured size. The error field will be 0 if the new size was accepted or an error code for invalid request (invalid backend, invalid size). The reply also contains the new configured size for validation.

This port can also be used to read back the current message store size, current sequence number and number of stored messages. The `ais_port_store_status_req` packet is used to request this information. The status request only identifies the message type and the backend to read status from.

Table 19: The ais_port_store_config_size_rep data structure

| Field Name | Field Type | Description |
|---|---|---|
| type | uint8_t | Request type, must be AIS_PORT_STORE_CONFIG_SET_SIZE |
| error | uint8_t | Error code |
| backend | uint8_t | The backend to configure |
| size | uint32_t | New store size in messages |

Table 20: The ais_port_store_status_req data structure

| Field Name | Field Type | Description |
|---|---|---|
| type | uint8_t | Request type, must be AIS_PORT_STORE_CONFIG_GET_STATUS |
| backend | uint8_t | The backend to request status from |

The receiver replies with an ais_port_store_status_rep packet which contains the store size, sequence number and message count.

Table 21: The ais_port_store_status_rep data structure

| Field Name | Field Type | Description |
|---|---|---|
| type | uint8_t | Request type, must be AIS_PORT_STORE_CONFIG_GET_STATUS |
| error | uint8_t | Error code |
| backend | uint8_t | The backend that replies |
| size | uint32_t | Current store size |
| seq | uint32_t | Current sequence number |
| count | uint32_t | Current message count |

### 7.2.5 AIS_PORT_STORE_REQUEST

This port is an extension to the regular AIS_PORT_AIS_{STORE, STORE_SHORT, STORE_STATIC} ports. It allows requests of a range of AIS messages specified by a start sequence number plus a bitmap. A single subtype request message is used.

The receiver will reply with the regular ais_port_ais_store_rep_t, ais_port_ais_short_rep_t, and ais_port_ais_static_rep_t packets depending on the specified backend.

Table 22: The ais_port_store_request_bitmap_req data structure

| Field Name | Field Type | Description |
|---|---|---|
| type | uint8_t | Request type, must be AIS_PORT_STORE_REQUEST_BITMAP |
| backend | uint8_t | The backend to request messages from |
| flags | uint8_t | Optional request flags |
| frames_per_reply | uint8_t | Number of frames to include per reply |
| delay | uint8_t | Delay in 10 millisecond steps between each packet |
| start | uint32_t | Request packets from sequence number |
| bitmap | uint8_t[] | Bitmap of frames, bit 0 = start, bit n = start - n |

The type field must be AIS_PORT_STORE_REQUEST_BITMAP, and the backend field specifies the backend to request messages from. The start field specifies the first sequence number to request. To stay consistent with the regular store ports, the messages are requested backwards from start, i.e. the start fields selects the *maximum* number to request. Use start equal to 0 to request the newest available messages.

The flags byte can be used to request short frames (from backends that support it), by setting the AIS_PORT_STORE_-REQUEST_FLAG_SHORT flag.

The frames_per_reply field sets the number of AIS messages to include in each reply packet. The delay field can be used to insert a small delay between each packet (e.g. to lower the buffering requirement on routing nodes).

The bitmap field is used to request selected messages backwards from the start sequence number. If bit n is set, the receiver will send the message with sequence number start - n, i.e. bit 0 must be set to request the message with sequence number start. Bit 0 is the bit immediately after the delay field and bit 255 is the last bit in the packet.

The size of the bitfield is AIS_PORT_STORE_REQUEST_BITMAP_SIZE which is currently set to 32 bytes (so up to 256 AIS messages can be requested at a time).

## 7.3 Blob Transfer Protocol

The Blob Transfer Protocol (BTP) is a lightweight file transfer and remote shell protocol based on CSP. BTP supports upload and download of files, as well as basic file operations such as list, delete, copy and move.

On the QubeAIS BTP is mainly used for downloading raw samples, and to upload new receiver software. The remote shell interface can be used for advanced debugging if necessary.

### 7.3.1 AIS_PORT_BTP

BTP runs as a single threaded server on a single CSP port, AIS_PORT_BTP, and can thus only serve one connection at a time. The BTP protocol is explained in Application Note SLAN-BTP which is available on request.

# 8 Hardware Interfaces

The default interface recommended for QubeAIS is the CAN-bus at 1 Mb/s using Cubesat Space Protocol (CSP), available in the stack connector and P1. CSP over UART is also an possible option, available as TTL level UART in P1 or RS-232 levels in P3.

If you prefer not to use the stacking connector (e.g. due to size constraints), P1 provides connections to CAN, UART (TTL) and power. P1 is also useful for monitoring CAN and subsystem voltage, when the satellite is integrated. The CAN (H1-01 + H1-03) and power (H1-47 - H1-52) can be disconnected completely from the stack connector, which is to be specified upon ordering.

The connectors P1, P2 and P3 mates with the ZH-series female connector from JST Inc.

Typically the JTAG/serial debug connector (P2) is only used for test and firmware upgrade on ground.

## 8.1 Pinout

| **Stack connector** | |
|---|---|
| **Pin** | **Description** |
| H1-01 | CAN-L |
| H1-03 | CAN-H |
| H1-48 | Select one for 3.3V DC |
| H1-50 | |
| H1-52 | |
| H1-47 | Select one for 5.0V DC |
| H1-49 | |
| H1-51 | |
| H2-29 | GND |
| H2-30 | GND |
| H2-32 | GND |

| **P3 - RS-232 connector** | |
|---|---|
| **Pin** | **Description** |
| 1 | GND |
| 2 | 3.3V DC |
| 3 | DNC |
| 4 | DNC |
| 5 | Data UART RX (RS-232) |
| 6 | Data UART TX (RS-232) |

| **P1 - Power/CAN connector** | |
|---|---|
| **Pin** | **Description** |
| 1 | Data UART TX (TTL) |
| 2 | GND |
| 3 | Data UART RX (TTL) |
| 4 | DNC |
| 5 | DNC |
| 6 | CAN-H |
| 7 | CAN-L |
| 8 | GND |
| 9 | 3.3V DC |
| 10 | 5.0 V DC |

| **P2 - JTAG/Serial Debug connector** | |
|---|---|
| **Pin** | **Description** |
| 1 | GND |
| 2 | Vtarget |
| 3 | $\overline{\text{TRST}}$ |
| 4 | TCK |
| 5 | TMS |
| 6 | TDO |
| 7 | TDI |
| 8 | Debug UART RX |
| 9 | GND |
| 10 | Debug UART TX |

DNC = Do Not Connect

## 8.2 Electrical Levels on Pins

The table below lists the minimum and maximum allowable levels on the connector pins. Exceeding these may damage the product permanently.

| Maximum Allowable Levels | | |
|---|---|---|
| **Pin Description** | **Parameter** | **Value** |
| 3.3 V DC | $V_{min}$ | 3.1 V |
| | $V_{max}$ | 3.5 V |
| | $V_{P\text{-}P}$ | $<$100 mV † |
| 5.0V DC | $V_{min}$ | 4.9 V |
| | $V_{max}$ | 5.1 V |
| | $V_{P\text{-}P}$ | $<$100 mV † |
| RF in | $P_{max}$ | -20 dBm |
| | $V_{DC_{max}}$ | $\pm$5 V |
| CAN-L/H | $V_{in}$ | -4 V - +16 V |
| | $V_{in_{\Delta}}$ | -6 V - +6 V |
| | $V_{out_L}$ | 0.5 V - 1.25 V |
| | $V_{out_H}$ | 2.45 V - 3.3 V |
| | $V_{out_{\Delta}}$ | 1.2 V - 3.0 V |
| UART-TTL | $V_{in_H}$ | 2.0 V - 3.6 V |
| | $V_{in_L}$ | -0.2 V - 0.5 V |
| | $V_{out_H}$ | ($V_{3.3V}$ - 0.5 V) - $V_{3.3V}$ |
| | $V_{out_L}$ | 0.0 V - 0.5 V |
| UART-RS232 | $V_{in_{max}}$ | -25 V - +25 V |
| | $V_{in_{min}}$ | -3.0 V - +3.0 V |
| | $V_{out_{min}}$ | -5.0 V - +5.0V |
| | $V_{out_{typ}}$ | -5.4 V - +5.4 V |

† Although 100 mV noise is allowed on the power lines, noise here can affect the reception performance. It is advised to keep the supply noise as low as possible.

## 9 Qualification

The QubeAIS receiver has been through a number of test campaigns to verify its performance over temperature, vacuum, vibration and radiation. An overview of the testing performed on the receiver is shown in table 23. As this list is non-exhaustive, please contact Satlab ApS for further information if needed.

**Table 23:** Qualification Parameters

| Parameter | Value |
|---|---|
| Thermal soak | -40℃ to +85℃ |
| Thermal vacuum | -10℃ to +55℃, $10^{-5}$ mbar |
| Vibration | 9.5 $G_{rms}$ |
| TID | 20 kRad(Si) |

It should be noted that the levels which are listed in table 23 is a superset of the different tests the receiver has been through during various test campaigns. The receiver has passed qualification for multiple launchers, including PSLV, Vega, Falcon 9, Atlas-5, LongMarch-11 and the HTV.

### 9.1 Acceptance Testing

Each board delivered will, as part of the acceptance testing, have been subject to a full RF performance testing covering the temperature interval from -30℃ to +70℃, the power interval from -113 dBm to -86 dBm and on the 4 nominal AIS frequencies: 161.975 MHz, 162.025 MHz, 156.775 MHz, 156.825 MHz along with ±3 kHz Doppler for each frequency.

## 10 Board Outline

Figure 9 shows the board with R/A SMA connectors, and pin1 markings of the ZH-series connectors.

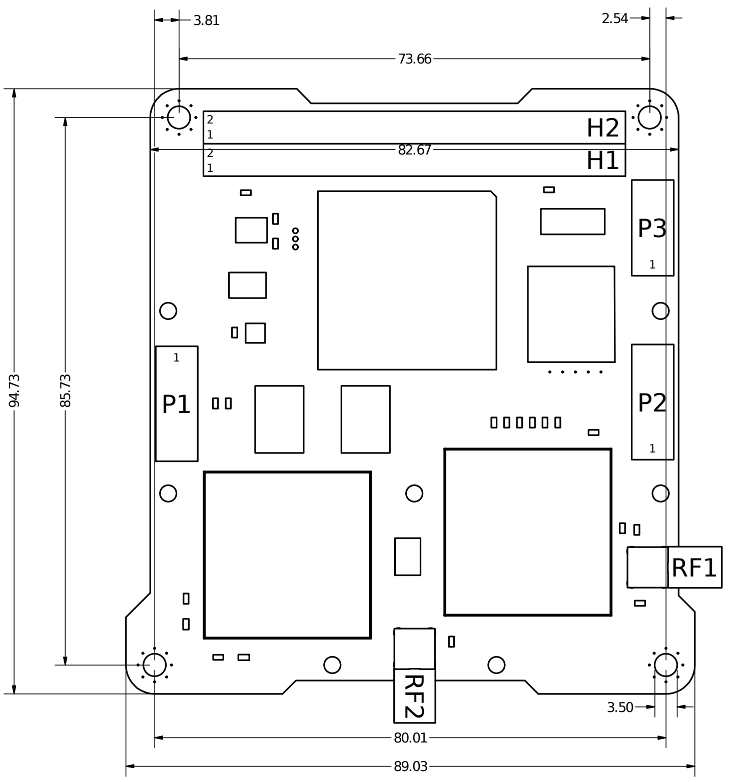


**Figure 9:** Board outline and pin1 markings. Dimensions in mm.

The picture in figure 10 shows the QubeAIS receiver with straight SMA connector for the RF1 option. Note that the straight SMA connector does not extend outside of the board outline like the right angle SMA.

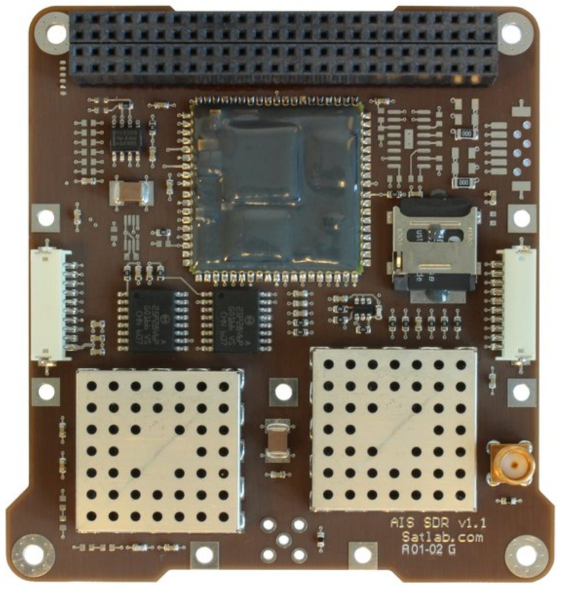


**Figure 10:** Top view of the QubeAIS with SMA straight in the RF1 connector placement

## 11 Ordering Information

Please fill out the below form when ready to place a order:

| QubeAIS Order Form | | |
|---|---|---|
| Conformal Coating | No | ☐ |
| | Yes (additional charge) | ☐ |
| Stacking Connector | Not mounted | ☐ |
| | ESQ-126-38-G-D | ☐ |
| 3.3V DC option | H1-48 | ☐ |
| | H1-50 | ☐ |
| | H1-52 | ☐ |
| 5.0V DC option | H1-47 | ☐ |
| | H1-49 | ☐ |
| | H1-51 | ☐ |
| Interface | CAN | ☐ |
| | UART TTL levels | ☐ |
| | UART RS-232 levels | ☐ |
| CSP Address (optional) | Please specify [0-30] | |
| RF Connector | RF1 MCX R/A | ☐ |
| Choose one | RF1 MCX Straight | ☐ |
| | RF1 SMA R/A | ☐ |
| | RF1 SMA Straight | ☐ |
| | RF2 MCX R/A | ☐ |
| | RF2 MCX Straight | ☐ |
| | RF2 SMA R/A | ☐ |
| | RF2 SMA Straight | ☐ |

## 12 Revision History

The document ID of this datasheet is **SLDS-AIS-1.4** and the revision number is **1.4**.

| Revision | Date | Description |
|---|---|---|
| 1.4 | 2017-03-06 | Add RF performance, Setup and Qualification sections. Revise text and update figures |
| 1.3 | 2015-05-19 | Updated with new images of board v.1.1 and general cleanup |
| 1.2 | 2015-01-03 | Updated with new store config and status ports |
| 1.1 | 2014-03-01 | Added static data store for msg ID 5 and 24 |
| 1.0 | 2013-08-23 | First released version |

## Table of Contents