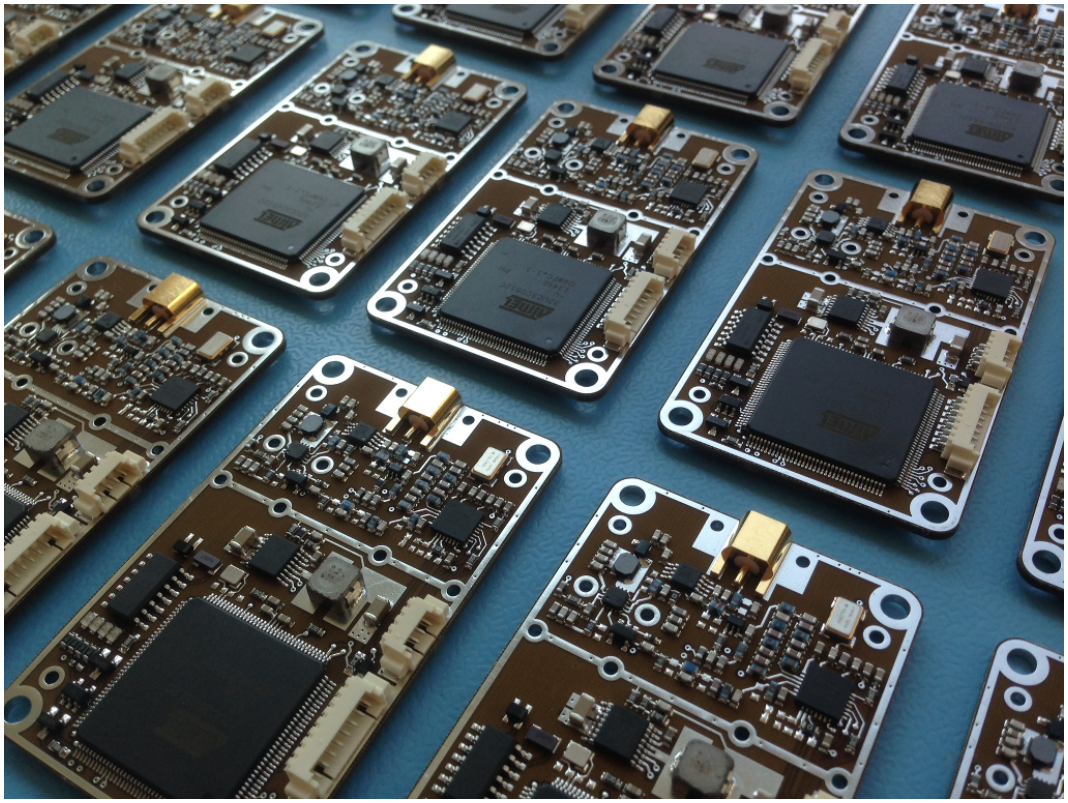


NanoCom
AX100

Manual

Long-range software configurable VHF/UHF transceiver

Release 3.13.0

GomSpace A/S | Langagervej 6 | DK-9220 Aalborg East | Denmark
gomspace.com | sales@gomspace.com | T: +45 71 741 741

Product name: NanoCom AX100

Document No.: 1013824

Revision: 3.13.0

Author: Gomspace

Approved by: Gomspace

Approval date: 2020

Confidentiality Notice

Table of Contents

1. Unpacking and handling precautions

Warning: Please observe precautions for handling electrostatic sensitive devices

The AX100 is an ESD sensitive device vulnerable especially on the following interfaces: RF-Connector all CPU I/O pins. Proper precautions must be observed during the handling of the device.

Please use an ESD mat and a wrist strap as a minimum. Please wear gloves to avoid fingerprints on the anodized aluminum shield, this part is particularly difficult to rinse off. If any cleaning of the parts are required prior to flight, use only ESD safe cleaning methods and a neutral, non-reactive, IPA solvent.

Fig. 1.1: ESD handling tools

2. Quickstart guide

2.1 RF Load

Warning: Please ensure that the RF connector is connected to a proper 50 ohm RF load capable of handling minimum 2 Watt, or a properly matched 50 ohm antenna, at all times when keying up the transmitter. Failure to do so will result in damage or degradation of the transmitter.

The first thing to connect to the AX100 is a dummy load, or antenna. The right angle MCX connector gives a solid click when connected.

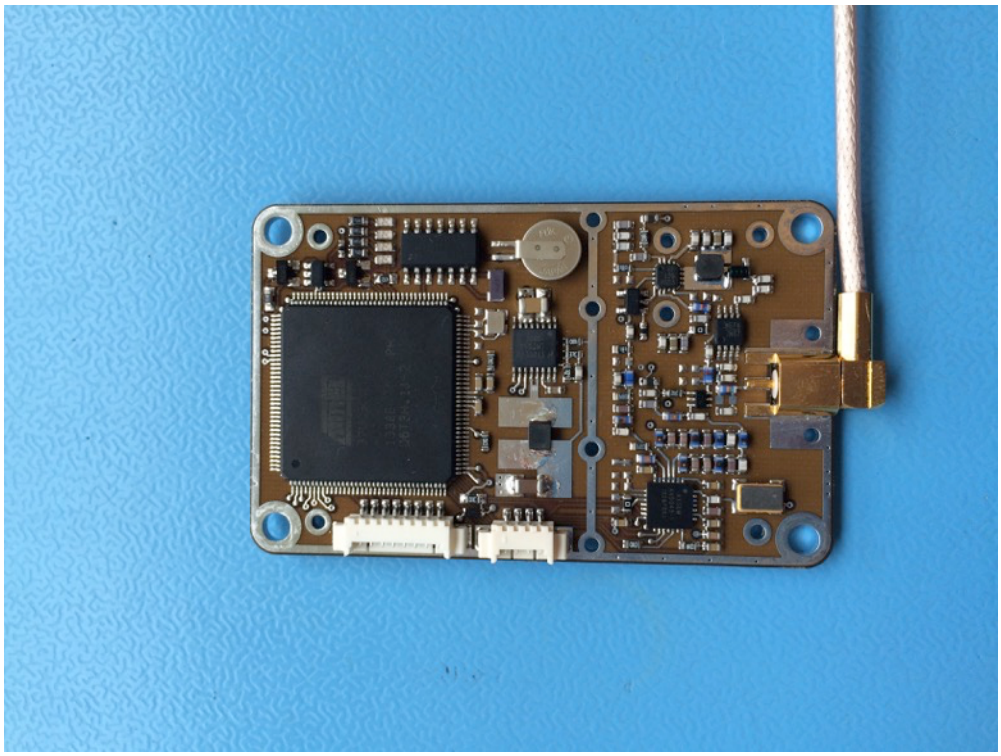


Fig. 2.1: Connecting the MCX connector

In this example a RG316 cable is used with MCX/SMA. The SMA end will be connected to a 2 W dummy load.

Note: Cable and dummy load is not included with the AX100.

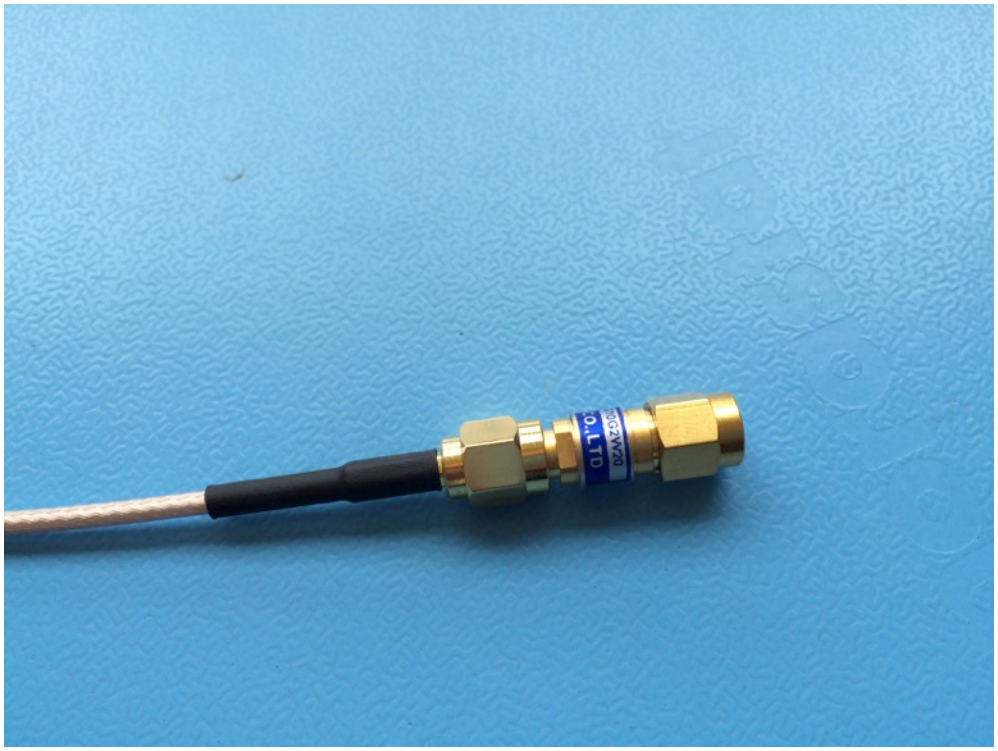


Fig. 2.2: 50 Ohm dummy load of 2 Watt

The dummy load is in this case a 20 dB attenuator. These SMA loads is not designed for continuous transmission and will get very hot if the transmitter is left operating for more than 2-3 minutes at a time. Please allow the load to cool if testing continuously, or get a bigger load with a heat sink.

2.2 Debug connector w/ power supply

The easiest way to access the AX100 is to use the accompanying 4-pin debugging connector. It has GND, VCC and UART RX/TX. A custom FTDI USB/SERIAL cable is made to fit with the 4-pin picoblade connector, and the VCC line is driven by an external power supply, instead of the FTDI 3.3 Volt interface.

Warning: Do not power the system using FTDI power and external power supply simultaneously.

Warning: Please ensure that any PC/Laptop connected to the USB cable has a properly grounded AC-plug. The external power supply ground and PC/Laptop ground must be the same. Failure to do so will result in Common Mode noise on the GND lines of up to 100+ Volts.

The external power supply must be set to 3.3 volt and a current limiter of 1000 mA. The ground connection is taken made one of the four PCB corners, but could also be made in the USB/Serial cable.

Fig. 2.3: External power supply

2.3 EMI Shield

The black anodized aluminum EMI shield and heat sink will come pre-installed on the AX100 module. The module will be factory checked out both before and after mounting the shield. Pictures will be taken of the PCB before mounting the shield, so please do not try to remove the shield to check what is inside. Furthermore the screws of the shield will be tightened to factory specifications and secured with locktite™ fastener glue. Removing the shield will void the warranty on the product.

2.4 Using a motherboard

The AX100 is designed to fit on a PC/104 GomSpace motherboard with the FSI connector. Please refer to the motherboard documentation for information about mounting and powering the AX100 module.

3. Console interface

The console interface is a powerful debug interface for the AX100 radio. The console is running at 500.000 baud, 8n1, 3.3 V, TTL levels. For Linux the program ‘minicom’ can be recommended, and for windows the program ‘putty’ is also recommended.

In order to setup ‘minicom’, please use ‘apt-get install minicom’ on a debian based distribution, or similar, to install it. After the installation open minicom as the root user and with the -s option: ‘sudo minicom -s’. This will enter the setup, where you must go to console settings and setup the baudrate to 500.000 baud, 8n1, and disable flow-control also. When this is completed, go back and select ‘save as dfl’ to store those settings as the default. Finally you may open minicom by using the following command:

```
minicom -D /dev/ttyUSB0 -con
```

The –D option specifies which device you wish to use and the –c option controls color on or off. In Figure 5 AX100 v1.8 boot screenthe boot-screen of the AX100 is shown. This may look different for different software revisions and settings.

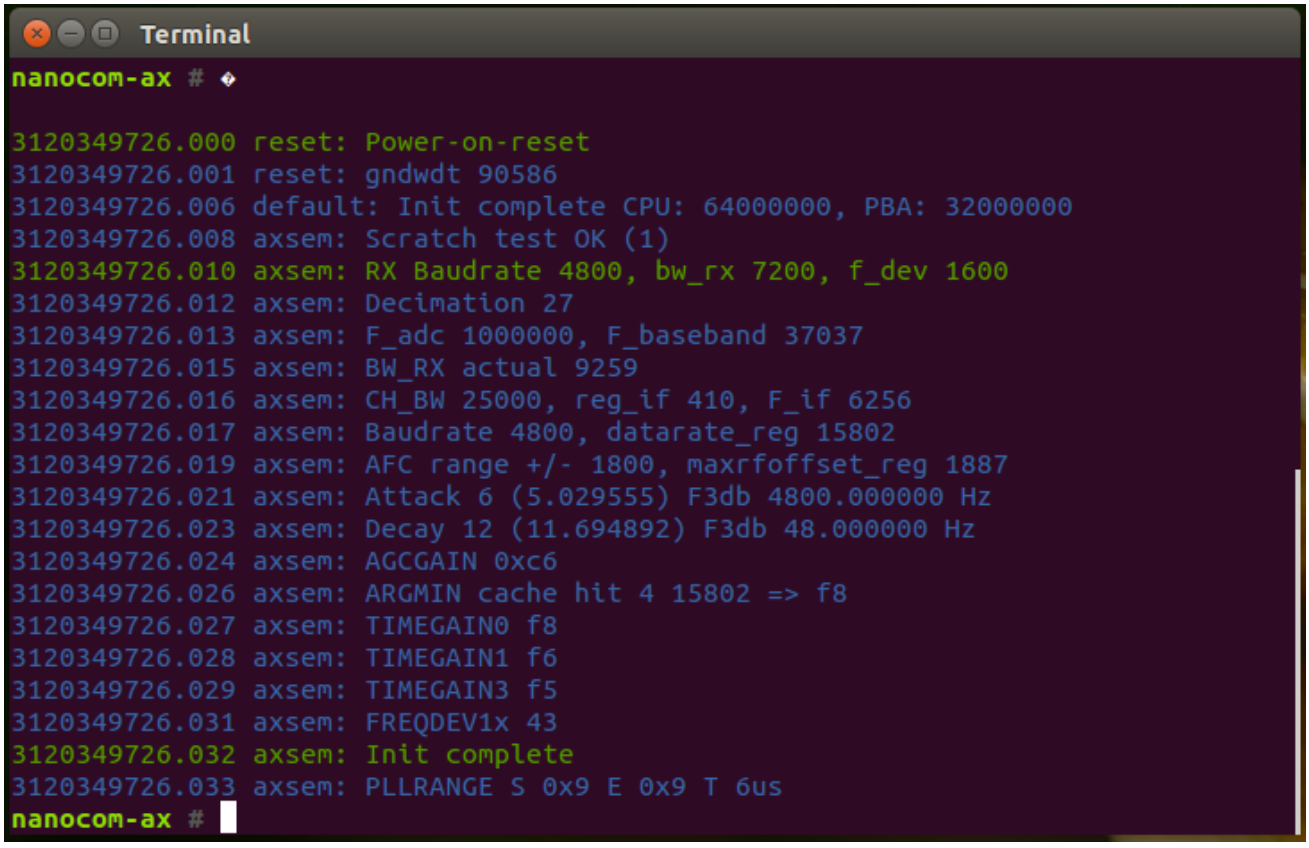


Fig. 3.1: v3.0 boot screen

The specific commands that can be entered will be described later in this manual, but a general help is to type the ‘help’ command:

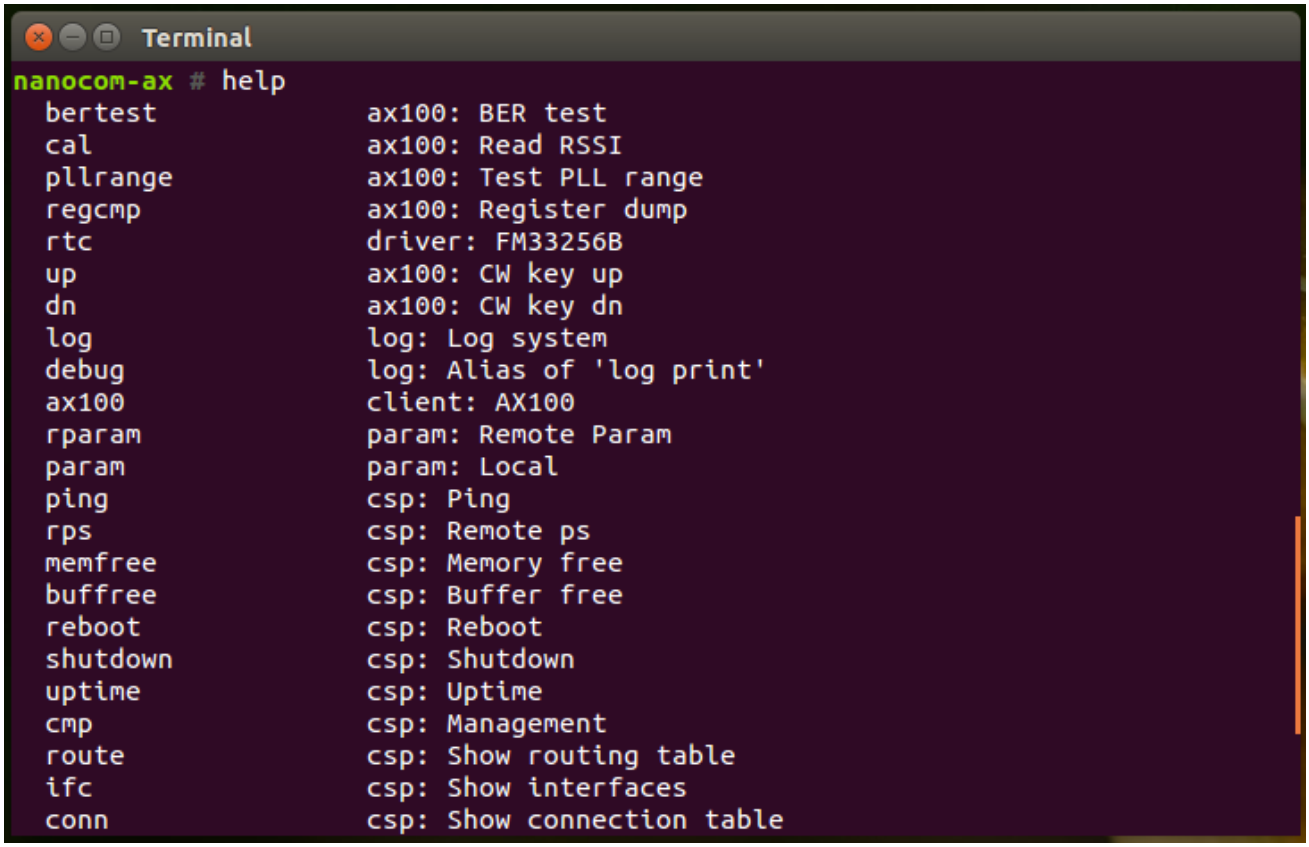


Fig. 3.2: v3.0 help screen

This provides a simple list of all the “top-level” commands that can be issued over the debug interface.

Note that any excerpt from the console used in this manual will look like this:

```
nanocom-ax # cmp ident 5 1000
Hostname: com
Model:    NanoCOM AX100
Revision: v1.8
Date:     Sep 10 2014
Time:     11:41:36
```

This was an example of the ‘cmp ident’ command, sent from the AX100 to the AX100 over the loopback interface in order to request the software identity of the AX100.

The most common commands to use the AX100 are:

```
ping <node>
param list <table>
param mem <table>
param get <name>
param set <name> <value>
ax100 hk
```

For further details on the parameters available, see section *List of parameter tables*.

4. CSP Port numbers

The AX100 listens on the following port numbers on CSP:

Table 4.1: List of port numbers

Port	Name	Description
0	CSP_CMP	Control Port
1	CSP_PING	Returns a copy of the packet received
2	CSP_PS	Returns process list
3	CSP_MEMFREE	Returns memory free
4	CSP_REBOOT	Reboots subsystem
5	CSP_BUF_FREE	Returns number of free buffers
6	CSP_UPTIME	Returns subsystem uptime
7	AX100_PORT_RPARAM	Controls AX100 with parameter system (see below)
9	AX100_PORT_GNDWDT_RESET	Resets the AX100 ground WDT

The first ports 0-6 are default port numbers handled by the 'libcsp' network stack. They are common on all gomspace CSP systems. For a description on how to use the CSP ports, please see the libcsp repository (https://github.com/libcsp/libcsp).

The *RPARAM* port is used to service remote parameter get/set requests. For a full list of parameters of the AX100 please see the next section of this manual. For a description on the parameter system and how to execute get/set commands, please refer to the libparam documentation.

For a description on the ground WDT please see *Ground WDT*.

5. Parameters

The AX100 uses a universal parameter system shared with other Gomspace products. All parameters shown in the parameter table can be modified through this system. To read more about this, please refer to the separate manual for the parameter system.

5.1 Parameter tables

Parameters reside in the RAM memory of the system, and have been split into different sections called “parameter tables”. For a list of tables, refer to the following table:

Table 5.1: List of parameter tables

Table number	Name	Description	Boot File	Default File (R/O)
0	System config	See *Table 0: System configuration*	0	24
1	Receiver config	See *Table 1: Receiver configuration*	1	25
3	Radio chip mem	Direct access to radio chip	none	none
4	Telemetry	See *Table 4: Telemetry*	none	none
5	Transmitter config	See *Table 5: Transmitter configuration*	5	29

5.2 Parameter Modifiers

Writing to a parameter can have different consequences, determined by how the parameter is accessed by the firmware. This is expressed using parameter modifiers found next to the type

5.2.1 (A) Active parameter

The types marked with an A are active parameters; this means they are checked by the firmware during execution. In practice this means that for example, the ‘preamblen’ parameter is read for each time a packet is transmitted, and therefore the value will take effect on the next transmitted frame. Another example is the ‘reboot_in’ and ‘tx_inhibit’ parameter that will automatically decrement once per second.

5.2.2 (B) Boot-up parameter

The types marked with an B can only be applied during system power-on. That means that after setting the parameter, nothing will happen. In order to apply the change, the table configuration must be stored and the system rebooted.

5.2.3 (R) Read-only parameter

The parameters marked with an R are read-only. That means that the firmware of the system will automatically update the value of the parameter, and that it can be used to readout a system state or value. Parameters marked with an R can be written to, but the value will most likely be overwritten by the firmware at a later time. An exception to this is counter values, which is only incremented by the firmware. So in order to reset the counters, it is possible to write a zero to the counter parameters.

5.2.4 (P) Persistent parameter

Finally the parameters marked with a P means that they are persistent. A persistent parameter will be stored each time they are changed in a separate memory location, allocated specifically for that single parameter. An example is the tx_inhibit parameter which runs in it's own non-volatile memory area, completely separate from the parameter storage system. The advantage of this is, that they are not overwritten when saving and loading parameters to/from a parameter table.

5.3 Table 0: System configuration

The first parameter table contains all the system parameters. These parameters should be set once for your mission, and should not have to be modified during operations. Most of the parameters are boot time configuration that requires a reboot in order to take effect.

Table 5.2: Parameter table 0: System configuration

Addr	Name	Type	Default Value	Comment
0x0000	rssi_offset	I8 A	-11	Sets the RSSI indicator offset, if you have external gain please adjust here for correct RSSI readings
0x0002	max_temp	I16 A	60	Maximum temperature in degrees of the PA allowed before automatic key-down occurs
0x0004	bgndrssi_ema	FLT A	0.500000	Exponential moving average (alpha value) [0.01 – 1.00]
0x0008	csp_node	U8 B	5	CSP address of the AX100 module
0x0009	i2c_en	BL B	true	Enables I2C
0x000B	can_en	BL B	false	Enables CAN
0x000C	extptt_en	BL B	false	Enable push-to-talk driver (used in GS100 only) – This conflicts with the I2C controller, so i2c_en must be zero while EXTPTT is enabled.
0x000D	led_en	BL B	true	Set to zero to disable the on-board leds. The LEDs should be disabled for flight in order to preserve power
0x000E	kiss_usart	I8 B	4	Set which USART to use for KISS interface. Set to -1 to disable KISS interface. Refer to the datasheet for USART port numbers
0x000F	gosh_usart	I8 B	2	Set which USART to use for GOSH interface. Set to -1 to disable GOSH interface. Refer to the datasheet for USART port numbers
0x0010	i2c_addr	U8 B	5	The non-shifted I2C address of the system, set to the same as csp_node, for normal operation
0x0012	i2c_khz	U16 B	400	The speed of the I2C master, this may vary depending on the bus capacitance.
0x0014	can_khz	U16 B	1000	The speed of the CAN bus
0x0018	reboot_in	U16 A	0	Number of seconds before automatic reboot will happen. This will automatically count down and reboot the AX100 if set.
0x001C	tx_inhibit	U32 AP	429496729	Number of seconds the transmitter will be shutdown. This will automatically count down and turn on the transmitter when reaching zero. *Note: this parameter is persistent*
0x0020	log_store	U8 B	0	Enable log-system FRAM storage backend: 'log hist' gosh command (only use for debugging)
0x0021	tx_pwr	U8 A	0	TX power level: 0 = minimum, 3 = maximum. Refer to checkout documentation for actual levels.
0x002C	max_tx_time	U16 A	10	Maximum number of seconds to key up the transmitter.
0x002E	max_idle_time	U16 A	3600	Number of seconds the receiver can be idle, before the receiver is reinitialized.
0x0030	csp_rtable	STR B	""	CSP routing table in the CIDR format: Example could be: "0/0 AX100, 8/2 KISS"
0x0090	legacy_hmac	BL	false	If HMAC authentication is used (see *Table 1: Receiver configuration* or *Table 5: Transmitter configuration*) this must be enabled to be compatible with pre 3.10 versions.
0x0094	kiss_baud	U32 B	500000	The symbol rate of the KISS USART

5.4 Table 1: Receiver configuration

The configuration for the receiver is kept in table 1. These parameters are all 'active' parameters which mean that they will take effect immediately after being changed. If you need to change multiple receiver parameters over the radiolink, use the rparam query system to build a packet with multiple parameters.

Table 5.3: Parameter table 1: Receiver configuration

Addr	Name	Type	Default Value	Comment
0x0000	freq	U32 A	437250000	Frequency in [Hz]
0x0004	baud	U32 A	4800	Baudrate
0x0008	modindex	FLT A	0.000000	Same as the tx_modindex (see below) - rx and tx modindex should have matching values. Set to zero for autocalculation.
0x000C	guard	U16 A	0	RX guard in [ms] (guard period after receiving a frame)
0x000E	pllrang	U8 A	9	Startup value of the PLLRANGE register
0x000F	mode	U8 A	5	Framing mode
0x0010	csp_hmac	BL A	false	Enable HMAC (checksum and authentication)
0x0011	csp_rs	BL A	true	Enable Reed-Solomon
0x0012	csp_crc	BL A	true	Enable CRC-32
0x0013	csp_rand	BL A	true	Enable CCSDS randomization
0x0020	csp_hmac_key	DAT A	000000000	HMAC key (needs to match transmitter)
0x0030	ax25_call	STR A	” “	The callsign used as the SRC field in outgoing AX.25 frames
0x0040	bw	U32 A	0	Receiver bandwidth in Hz, must be at least 1.5 times bigger than the baudrate but can be set to zero in order to let the AX100 auto calculate the best rx_bw for the desired bitrate
0x0044	afcrange	I32 A	-4	Sets the AFC pull-in range in Hz. Set to zero to disable AFC. Set to a negative value to autocalculate based on the formula: afcrange = rx_bw/(-1 * rx_afcrange) – The automatic calculation is capped at 10 kHz

5.5 Table 5: Transmitter configuration

The first part of the transmitter configuration is similar to the receiver configuration. In general, the value stored in the receiver configuration should be mirrored into the opposite transmitters configuration. (Apart from the guard and pllrang, which only effects the local end of the link)

The parameters from 0x40 and up are transmitter only.

These parameters are all ‘active’ parameters which mean that they will take effect immediately after being changed.

Table 5.4: Parameter table 5: Transmitter configuration

Addr	Name	Type	Default Value	Comment
0x0000	freq	U32 A	437250000	same as receiver parameter
0x0004	baud	U32 A	4800	same as receiver parameter
0x0008	modindex	FLT A	0.000000	same as receiver parameter
0x000C	guard	U16 A	50	same as receiver parameter
0x000E	pllrang	U8 A	9	same as receiver parameter
0x000F	mode	U8 A	5	same as receiver parameter
0x0010	csp_hmac	BL A	false	same as receiver parameter
0x0011	csp_rs	BL A	true	same as receiver parameter
0x0012	csp_crc	BL A	true	same as receiver parameter
0x0013	csp_rand	BL A	true	same as receiver parameter
0x0020	csp_hmac_key	DAT A	000000000	same as receiver parameter
0x0030	ax25_call	STR A	” “	The callsign used in the DST field for outgoing AX.25 packets
0x0040	preamb	X8 A	0xAA	The byte to use as preamble
0x0041	preamblen	U8 A	50	The length of the preamble in bytes
0x0042	preambflags	U8 A	56	The flags to use for the preamle (do not modify)
0x0043	intfrm	X8 A	0xAA	The byte to use between two frames
0x0044	intfrmlen	U8 A	0	The number of bytes to put between two frames
0x0045	intfrmflags	U8 A	56	The flags to use for the intfrm bytes (do not modify)
0x0048	rssibusy	I16 A	-95	The transmitter will consider the channel busy when the RSSI is above this value
0x004A	kup_delay	U8 A	0	An additional delay of the first frame after the radio have been keyed up. Usefull for slow RX/TX relays
0x004C	pa_level	X16 A	0x04A9	The input level for the PA (do not modify)
0x0050	ber	FLT A	0.000000	Injects random bit-errors in transmitted frames with this probability

5.6 Table 4: Telemetry

The radio contains a separate memory area allocated for telemetry data such as the current temperature, data counters and the bootcounter. The memory map for the telemetry is as follows:

Table 5.5: Parameter table 4: System Telemetry

Addr	Name	Type	Default Value	Comment
0x0000	temp_brd	I16 R	0	Board temperature (near MCU)
0x0002	temp_pa	I16 R	0	PA temperature (near PA)
0x0004	last_rssi	I16 R	0	Last received RSSI
0x0006	last_rferr	I16 R	0	Last received RF error
0x0008	tx_count	U32 R	0	Number of tx packets since reboot
0x000C	rx_count	U32 R	0	Number of rx packets since reboot
0x0010	tx_bytes	U32 R	0	Number of tx bytes since reboot
0x0014	rx_bytes	U32 R	0	Number of rx bytes since reboot
0x0018	active_conf	U8 R	0	The currently active system configuration (table 0), cf. *Boot sequence*
0x0020	boot_count	U16 R	0	The number of reboots
0x0024	boot_cause	X32 R	0	The cause of the reboot (see below)
0x0028	last_contact	U32 R	0	The timestamp of the last valid packet (required a CSP timesync to work)
0x002C	bgnd_rssi	I16 R	-120	The current background RSSI level (Exponential moving average, see bgndrssi_ema parameter in table 0)
0x002E	tx_duty	U8 R	0	Total TX duty time since reboot
0x0030	tot_tx_count	U32 RP	0	Number of tx packets (total)
0x0034	tot_rx_count	U32 RP	0	Number of rx packets (total)
0x0038	tot_tx_bytes	U32 RP	0	Number of tx bytes (total)
0x003C	tot_rx_bytes	U32 RP	0	Number of rx bytes (total)

5.6.1 `boot_cause`

The `boot_cause` parameter is a bit mask which can be interpreted as follows:

Table 5.6: `boot cause` bit mask

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Meaning			BOD33		AWIRE			OCDRST
Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Meaning	CPUERR			JTAG	WDT	EXT	BOD	POR

And:

- BOD33: Brown-out 3.3 V reset
- AWIRE: AWIRE Reset
- OCDRST: OCD Reset
- CPUERR: CPU Error
- JTAG: JTAG Reset
- WDT: Watchdog Reset
- EXT: External Reset Pin
- BOD: Brown-out Reset
- POR: Power-on Reset

For more details, refer to the Reset Cause (RCAUSE) register in the Atmel AT32UC3C series datasheet.

6. Stored configurations

6.1 Boot sequence

The AX100 will first check the non-volatile memory for the boot-file. If the data and table CRC are correct, it will load that into running memory and continue booting. If the boot-file does not exist, or the CRC fails, it will continue and try the special write-protected boot file. Again if the data and table CRC are correct, it will use that. Otherwise the system will assume that no configuration has been ever stored by the customer, and revert to an internal backup configuration in the FLASH of the MCU. This configuration is set by GomSpace and cannot be changed.

Which configuration is currently in use is reflected by the `active_conf` parameter in the telemetry table. It is an enum that can be interpreted as follows:

Table 6.1: Active configuration enum

Active configuration	`active_conf`
Fallback configuration (internal backup configuration)	0
Default configuration (from write-protected boot file)	1
Stored configuration (from non-volatile memory)	2

In the *List of parameter tables* it is possible to see which files will be read during bootup. Each configuration have a 'boot-file' and a 'default-file'. The boot file is the first attempted file which can be saved to in-orbit. The default file is the second choice if the boot-file is not valid. This is stored in read-only memory and cannot be changed in-orbit.

Important: Please make sure that the default-file has been stored correctly before launching. Otherwise the system will startup using the internal backup configuration in the MCU Flash. The internal backup configuration in the MCU Flash will have the tx_inhibit parameter set, so the radio will effectively stop transmitting.

6.2 Saving a parameter table

In order to store a configuration from memory to a file-number, the commands `param save` or `param load` can be used. The commands `param save` or `param load` will save to FRAM and load from FRAM respectively.

Below is an example of modifying a parameter in parameter table 1 and storing parameter table 1 to FRAM:

```
nanocom-ax # param mem 1
nanocom-ax # param get mode
  GET mode = 2
nanocom-ax # param set mode 5
  SET mode = 5 (1)
nanocom-ax # param save 1 1
Table CRC 46704
Data CRC 63250
nanocom-ax # reset
...
AXSEM: Init complete
nanocom-ax # param mem 1
nanocom-ax # param get mode
GET mode = 3
```

6.2.1 Setting the default configuration

In order to store to the special write protected boot configuration the FRAM chip must first be unlocked. Here is an example of doing it manually:

```
nanocom-ax # rtc debug clear_wp
nanocom-ax # param save 0 24
Table CRC 46704
Data CRC 1180
nanocom-ax # param save 1 25
Table CRC 46704
Data CRC 1180
nanocom-ax # param save 5 29
Table CRC 46704
Data CRC 1180
nanocom-ax # rtc debug set_wp
```

Note: This can only be done over the debug interface. So when the radio is in space, there is no way to alter the alternate boot file configuration.

The default configuration can also be updated and verified using the `config` command. To update the default configuration for all three parameter tables use `config update_default all`:

```
nanocom-ax # config update_default all
Table CRC 9508
Data CRC 1900
Updated settings in FRAM for table 0, saving to file 0: OK
Table CRC 9508
Data CRC 1900
Updated default factory settings in FRAM for table 0, saving to file 24: OK
Table CRC 57481
Data CRC 53000
Updated settings in FRAM for table 1, saving to file 1: OK
Table CRC 57481
Data CRC 53000
Updated default factory settings in FRAM for table 1, saving to file 25: OK
Table CRC 56172
Data CRC 13489
Updated settings in FRAM for table 5, saving to file 5: OK
Table CRC 56172
Data CRC 13489
Updated default factory settings in FRAM for table 5, saving to file 29: OK
```

To verify or check the default configuration for all three parameter tables use `config verify_default all`:

```
nanocom-ax # config verify_default all
File 0 CRC values: Table 9508 Data 1900
File 24 CRC values: Table 9508 Data 1900
Verification of default factory settings in FRAM for table 0, file 24: OK
File 1 CRC values: Table 57481 Data 53000
File 25 CRC values: Table 57481 Data 53000
Verification of default factory settings in FRAM for table 1, file 25: OK
File 5 CRC values: Table 56172 Data 13489
File 29 CRC values: Table 56172 Data 13489
Verification of default factory settings in FRAM for table 5, file 29: OK
```

The default configuration can also be updated/verified on individual tables, e.g. for parameter table 1:

```
nanocom-ax # config update_default 1
Table CRC 57481
```

(continues on next page)

(continued from previous page)

```
Data CRC 53000
Updated settings in FRAM for table 1, saving to file 1: OK
Table CRC 57481
Data CRC 53000
Updated default factory settings in FRAM for table 1, saving to file 25: OK
```

And verify:

```
nanocom-ax # config verify_default 1
File 1 CRC values: Table 57481 Data 53000
File 25 CRC values: Table 57481 Data 53000
Verification of default factory settings in FRAM for table 1, file 25: OK
```

Important: Please note that the default configuration must always be updated, verified and tested before launch.

To test the default configuration without waiting 48 hours, a ground watchdog timeout can be simulated by manually setting the ground watchdog timer to something smaller, using the `config gnd_wdt` command. To view the current ground watchdog time value, run the command `config gnd_wdt`:

```
nanocom-ax # config gnd_wdt
gnd wdt value is 171155 seconds
```

To set the ground watchdog timer to e.g. 60 seconds, run the command `config gnd_wdt 60`:

```
nanocom-ax # config gnd_wdt 60
gnd wdt value is 60 seconds
```

After this, the ground watchdog will expire after 60 seconds and the normal configuration will be cleared and the default configuration will be used instead.

Note: Remember to store the normal configrauion again if/when the ground watchdog expires.

6.3 Loading a parameter table

The parameter table will be automatically loaded on a reboot of the system. However it is possible to use the *param load* or *rparam load* commands to load a file to memory.

7. Safety functions

The AX100 comes with several safety functions that try to take several cubesat fault scenarios into consideration.

7.1 Temperature protection

The simplest of the AX100 safety features is the temperature protection. There is a system inside the MCU that will monitor the temperature of the power amplifier and automatically force a transmitter key-down in the event of an over temperature situation. The 'max_temp' parameter is used to set which value that will be at. If the radio goes into over temperature protection, but never comes out, for example if the environment temperature never goes below the set value, or the value was accidentally set too low, the receiver will still be running, and it should therefore be possible to set the 'max_temp' parameter to a higher value. The sample frequency on the temperature protection is 20 Hz with no hysteresis.

7.2 TX max time

The 'max_tx_time' parameter defines for how many seconds the transmitter can be keyed up, without a key-down. When the radio is transmitting a lot of data, the radio will automatically key down after this number of seconds, to listen for any uplink data, before keying up and continuing the transfer. The minimum delay between the key-down and the key-up is controlled by the TX 'guard' setting. For the default settings this is set to have a key-down period of 50 milliseconds for each 10 seconds, so the overhead of having this occasional stop and listen feature is very small. The benefit is that it should always be possible to get an uplink through to the satellite within a maximum delay of 10 seconds.

7.3 RX idle time

The RX idle timer will count receiver inactivity. If the receiver has been inactive, that means without receiving any data, for 'max_idle_time' seconds, the receiver configuration will be reinitialized. This feature is designed to prevent against receiver configuration SEU's (Single Event Upset). Initialization takes less than 100 milliseconds.

7.4 TX inhibit

The TX inhibit counter is special because it's the only configuration parameter that survives a reboot. It has been designed to count down each second until it reaches zero where it will stop. As long as the counter is not equal to zero, the transmitter will not be allowed to be keyed up. This is a safety feature used by many cubesats to ensure a certain period of radio silence after first power-up. It can also be used later to make the AX100 be silent up for a long duration in case that the satellite is decommissioned.

Note: When the AX100 is delivered, it will be running on the factory configuration (i.e. no boot-config or alternate-boot-config will be stored) – where the tx_inhibit value will be set to UINT32_MAX. This means that a system will not allow the user to transmit before the tx_inhibit value have been set to zero. This is a safety feature to ensure correct configuration, before transmitting.

7.5 Ground WDT

The most important safety feature of the AX100 must be the ground WDT. This is a counter residing at a pre-set FRAM location, that independently from the parameter system will count down to zero. If it reaches zero, it will revert to the default configuration stored in the default files:

- Default configuration in file 24 will be loaded to table 0
- Default configuration in file 25 will be loaded to table 1
- Default configuration in file 29 will be loaded to table 5

This is a safety feature to protect against accidentally storing and setting an invalid configuration that would otherwise permanently disable the contact from the ground to the satellite.

Resetting the ground WDT has a separate CSP service on port 9 and can be cleared by sending an empty request to this port. Here is the example code for doing this:

```
csp_transaction(CSP_PRIO_HIGH, node_com, AX100_PORT_GNDWDT_RESET, 1000,
NULL, 0, NULL, 0);
```

You can also reset the GNDWDT with the following gosh command:

```
nanocom-ax # ax100 gndwdt_clear
```

The ground WDT timeout is 172800 seconds (48 hours). This means that the the ground WDT must be reset before the 48 hours elapses, otherwise the ground WDT will revert the AX100 back to the default configuration and reboot the AX100.

Please note that the ground WDT will be decremented by 1800 seconds everytime the AX100 is rebooted. This is a safety feature to recover faster from an endless reboot situation caused by invalid configuration.

8. Telemetry

Please see *Table 4: Telemetry* for a list of parameters in the telemetry table.

8.1 Receiving Telemetry with parameter system

The parameter system can be used to receive the telemetry data. The special memory area '4' of the parameter system can be used. In order to change which parameter table the local parameter client is using, use the 'param mem 4' command.

Here is an example of getting the telemetry data using the parameter system:

```
nanocom-ax # param mem 4
nanocom-ax # param list
Parameter list 4:
  0x0000 temp_brd          I16 257
  0x0002 temp_pa           I16 263
  0x0004 last_rssi         I16 -125
  0x0006 last_rferr        I16 1702
  0x0008 tx_count          U32 0
  0x000C rx_count          U32 0
  0x0010 tx_bytes          U32 0
  0x0014 rx_bytes          U32 0
  0x0018 active_conf       U8  2
  0x0020 boot_count        U16 7620
  0x0024 boot_cause        X32 0x00000001
  0x0028 last_contact      U32 3120366763
  0x002C bgnd_rssi         I16 -125
  0x002E tx_duty           U8  0
  0x0030 tot_tx_count      U32 797682
  0x0034 tot_rx_count      U32 856783686
  0x0038 tot_tx_bytes      U32 73266462
  0x003C tot_rx_bytes      U32 565161177
```

Note that the 'param mem 4' command was used to change over to the telemetry data. If 'param mem 0' is selected, the 'param list' will show the system configuration again.

It is also possible to get the telemetry table using rparam:

```
nanocom-ax # rparam download 5 4
Downloading RPARAM table 4 for node 5 on port 7
Checksum is: 0xF207
  0x0000 temp_brd
  0x0002 temp_pa
  0x0004 last_rssi
  0x0006 last_rferr
  0x0008 tx_count
  0x000C rx_count
  0x0010 tx_bytes
  0x0014 rx_bytes
  0x0018 active_conf
  0x0020 boot_count
  0x0024 boot_cause
  0x0028 last_contact
  0x002C bgnd_rssi
  0x002E tx_duty
  0x0030 tot_tx_count
```

(continues on next page)

(continued from previous page)

```
  0x0034 tot_rx_count
  0x0038 tot_tx_bytes
  0x003C tot_rx_bytes
nanocom-ax # rparam getall
  0x0000 temp_brd          I16 257
  0x0002 temp_pa           I16 263
  0x0004 last_rssi         I16 -125
  0x0006 last_rferr        I16 416
  0x0008 tx_count          U32 0
  0x000C rx_count          U32 0
  0x0010 tx_bytes          U32 0
  0x0014 rx_bytes          U32 0
  0x0018 active_conf       U8  2
  0x0020 boot_count        U16 7620
  0x0024 boot_cause        X32 0x00000001
  0x0028 last_contact      U32 3120366763
  0x002C bgnd_rssi         I16 -124
  0x002E tx_duty           U8  0
  0x0030 tot_tx_count      U32 797682
  0x0034 tot_rx_count      U32 856783686
  0x0038 tot_tx_bytes      U32 73266462
  0x003C tot_rx_bytes      U32 565161177
```

Note: *rparam init* cal also be used instead of rparam download (if your client support saving remote parameter tables)

8.2 Receiving Telemetry with the GOSH command

The AX100 client have a GOSH command that can be used to get telemetry. The command example is:

```
nanocom-ax # ax100 hk
  0x0000 temp_brd          I16 257
  0x0002 temp_pa           I16 263
  0x0004 last_rssi         I16 -127
  0x0006 last_rferr        I16 194
  0x0008 tx_count          U32 0
  0x000C rx_count          U32 0
  0x0010 tx_bytes          U32 0
  0x0014 rx_bytes          U32 0
  0x0018 active_conf       U8  2
  0x0020 boot_count        U16 7620
  0x0024 boot_cause        X32 0x00000001
  0x0028 last_contact      U32 3120366763
  0x002C bgnd_rssi         I16 -125
  0x002E tx_duty           U8  0
  0x0030 tot_tx_count      U32 797682
  0x0034 tot_rx_count      U32 856783686
  0x0038 tot_tx_bytes      U32 73266462
  0x003C tot_rx_bytes      U32 565161177
```

Notice that the output looks the same, because the parameter system does the work for the ‘ax100 hk’ command’.

9. Layer 3: Network-layer (Cubesat Space Protocol)

The AX100 works as a CSP router that is capable of receiving packets on one of its four interfaces: Radio-link, I2C-bus, CAN-bus or KISS/Serial interface, and route that message to one of the other interfaces. The basic RX/TX operation of the AX100 radio does therefore not require any special commands in order to make the radio send and receive. Everything regarding the data-link layer, medium access control and physical setup is handled by the AX100 automatically, but can be adjusted by the operator using the parameter system.

In order to transmit a packet from any subsystem on the satellite to a ground station node, a valid CSP packet must be generated and placed on the satellite bus. This could for example be the on-board computer (OBC) sending a beacon-message to the ground station. If the OBC has libcsp (a free open-source CSP library) built-in, all it needs is to call the function:

```
/* send out packet */
csp_sendto(CSP_PRIO_LOW, 10, 31, 0, CSP_O_NONE, beacon, 0);
```

In this example a low priority message will be generated with the destination address 10, destination port 31 and a source-port of zero (meaning you cannot reply to this message). This function will transmit the beacon over the satellite bus to the default router for the satellite. The AX100 uses the destination field of the packet to route the message onto the radio-link interface, and on the ground station the message will be routed onto the serial port using the KISS protocol and then finally to the ground station PC.

For more information on libcsp, go to http://libcsp.org

9.1 Understanding routing entries

The routing table of the AX100 can be listed by using the ‘route’ command. The routing table is in the CIDR format. Each line in the table is read as ‘<addr>/<netmask bits> <interface> <nexthop>’

The CSP 1.0 address field is 5 bits. The netmask defines how many of these bits signify the network address, and implicitly how many signify the node address. Here are a few examples:

8/2 – Here the address is 8, and the netmask bits is 2. Translating the bits into a netmask it becomes a binary 11000b. This means that the hosts mask is 00111b which means that the host’s can range from 000b (0) to 111b (7) – So adding the host range to the network address, the routing entry 8/2 signify all routes from 8 to 8 + 7 = 15. Another way of seing this is if you have a packet destined for node 14, you take the destination address and apply the netmask 14 & 11000b = 8, so we have a match.

0/0 – This is the default route. Applying a mask of 00000b always gives the address of zero, therefor any packet destination will match this route.

The router gives priority to the routing entry with the highest netmask bits. So a /5 route takes precedence over a /2 and /0 route.

9.1.1 Example 1: Spacecraft routing table

Here is an example from an AX100 in a satellite, with address 5.:

```
nanocom-ax # route
5/5 LOOP
0/0 AX100
0/2 I2C
8/5 KISS
```

In this case traffic for 0-7 is routed to I2C, 8 to KISS, 5 to LOOP and every thing else to the AX100 radio interface.

9.1.2 Example 2: Ground station routing table

Here is an example with a ground station node with address 14:

```
nanocom-ax # route
14/5 LOOP
0/0 AX100
8/2 KISS
```

This routing table will send all traffic destined for the 8/2 network on to the KISS interface, and the default route 0/0 will be send to the AX100 (ie. the radio link). The LOOP interface 14/5 is for local traffic only.

9.2 Altering the routing table (temporarily)

If you only wish to change the routing table present in RAM on a system, you can use the 'cmp route_set' command.:

```
nanocom-ax # cmp route_set
usage: route_set <node> <timeout> <addr> <mac> <ifstr>

nanocom-ax # cmp route_set 5 1000 6 255 AX100
Sending route_set to node 5 timeout 1000
Dest_node: 6, next_hop_mac: 255, interface AX100
Success
nanocom-ax # route
5/5 LOOP
0/0 AX100
0/2 I2C
8/5 KISS
6/5 AX100
```

This tells the router to send all traffic to node 6 over the AX100 radio-link.

This routing table change will be reset to default after a reboot.

9.3 Altering the routing table (persistent)

In order to set the routing table, and remember it after a reboot, the table can be entered in a serialized string format using the 'csp_rtable' system parameter.

Here is an example of a routing table that is serialized:

9.3.1 Example 1: Serialized routing table

```
nanocom-ax # param get csp_rtable
  GET csp_rtable = "0/0 AX100, 0/2 I2C, 6/5 CAN"
```

Here the table uses the same format as the route command outputs.

9.3.2 Example 2: Serialized routing table with MAC address

In some cases, the MAC address of the next hop is required. This is the case for the I2C interface. So in this example we have an OBC that needs to send its default route to the AX100 radio. So it's default route looks a bit different:

```
nanocom-ax # param get csp_rtable
  GET csp_rtable = "0/0 I2C 5, 0/2 I2C"
```

This will send all traffic to 0/2 (nodes 0-7) to the I2C interface with no specified MAC address. This traffic will therefore have the same I2C address as the CSP address. However the default route is using another entry: '0/0 I2C 5' here the final 5 number is the I2C address for the next hop. So traffic using the default routing entry will have its MAC address set to address 5, and therefore be sent to the AX100 radio from the OBC.

9.4 Interface statistics / Conn stats

Each interface has its own RX and TX counters which can be accessed with the following command:

```
nanocom-ax # ifc
LOOP    tx: 00002 rx: 00002 txe: 00000 rxe: 00000
                drop: 00000 autherr: 00000 frame: 00000
                txb: 38 (38.0B) rxb: 30 (30.0B)

AX100   tx: 00000 rx: 00000 txe: 00000 rxe: 00000
                drop: 00007 autherr: 00000 frame: 00000
                txb: 0 (0.0B) rxb: 0 (0.0B)

I2C     tx: 00000 rx: 00000 txe: 00000 rxe: 00000
                drop: 00000 autherr: 00000 frame: 00000
                txb: 0 (0.0B) rxb: 0 (0.0B)
```

Not all counters are used by each interface, and it will be different what each interface is using their counters for.

CSP also has a list of active connection that can be displayed with the 'conn' command:

```
nanocom-ax # conn
[00 0x1c34] S:0, 0 -> 0, 0 -> 0, sock: 0x0
[01 0x1c58] S:1, 0 -> 0, 0 -> 0, sock: 0xf888
[02 0x1c7c] S:0, 5 -> 5, 17 -> 0, sock: 0x0
[03 0x1ca0] S:0, 5 -> 5, 0 -> 17, sock: 0x0
[04 0x1cc4] S:0, 0 -> 0, 0 -> 0, sock: 0x0
```

This output shows 4 connections in the idle state = 0, and one connection in the active = 1, state. The active connection has a socket pointer, indicating that this is a listening socket. The other closed connections still retain their last used source and destination addresses and ports so it is possible to trace the connection usage even when they are closed.

9.5 MTU

The MTU of the different CSP interfaces varies a bit. Especially the radio-link is important, since this is the bottleneck.

Interface	MTU
I2C	255
KISS	255
CAN	255
RF	251
RF (mode2)	239

The MTU over the radio link will decrease depending on which Layer2 features have been added:

Feature	Cost (bytes)
Reed-Solomon	32
CRC-32	4
HMAC	4
Randomization	0
AX25 (Mode 6)	16
Golay (Mode 5)	3
Len (Mode 2)	1

10. Layer 2: Data-link layer

The data link layer of the AX100 has three purposes:

1. To encapsulate the network-layer packet into a data-link-layer frame.
2. To provide frame-level error correction and control.
3. To control the medium access using a MAC protocol.

These topics will be addressed separately:

10.1 Framing formats

The AX100 supports 6 different framing formats. The framing-mode can be controlled using the 'mode' parameter. Each mode has its own strengths and weaknesses.

Table 10.1: Overview of framing formats

Mode	Advantages	Disadvantages	Recommended
1 RAW	Testing only	No data	No
2 ASM	Good Sync	Corrupt length	No
3 HDLC	Standard	Bad Sync	No
4 Viterbi	Best sensitivity	Bad Sync, lower bitrate	No
5 GOLAY	Good sync, Good Length	Not as sensitive as Viterbi	Yes
6 AX.25	Standard	Sync errors, no FEC	No

10.1.1 Mode 1: RAW

The RAW mode will pass all received bits unsynchronized to the microprocessor. This is helpful for testing purposes when measuring the BER for example, but not very useful for receiving data. The RAW mode could be used to implement legacy framing formats by writing a software synchronizer for that format, but currently there are no such formats supported by the AX100 firmware. The only reason to use the RAW mode is therefore to run the BER test that is described later in the test section.

10.1.2 Mode 2: ASM

Mode 2 is known as the Attached Sync Marker (ASM) mode. In this mode the RF module will search for a unique 32-bit ASM word in the bit-stream. It does so using a certain confidence, meaning that if more than 28 out of the 32 bits match, it will be considered a match. When a match has been found, the next byte of the message is considered the length field. This length field will be used by the RF module to offload the MCU, by buffering the entire RX message until the frame has been completely received. This gives a low total power usage of the system when in this mode.

The benefit of using this mode is that due to its long 32-bit sync work, it gives a low number of false positives (synchronizations). This means that the Medium Access Control layer will have an easier task of getting access to the link, if it knows weather or not the receiver is busy.

The disadvantage of this mode is that it relies on a simple 8-bit length field, which can be corrupted. Meaning that there is a slight risk of having a false packet length. It also puts a limit to the maximum frame length of 240 bytes. Furthermore, this mode cannot use the hardware Viterbi module of the RF chip, so it relies solely on the in-frame FCS (either CRC32 or Reed-Solomon).

ASM Frame layout

The 32-bit sync word is: 0xC9D08A7B MSB (Unencoded), the LEN field valid from 0-240.

Encoding: NRZ, G3RUH/K9NG scrambled, Most significant bit first.

10.1.3 Mode 3: HDLC

High-Level Data Link Control (HDLC) is a bit-oriented code-transparent synchronous data link layer protocol developed by the International Organization for Standardization (ISO).

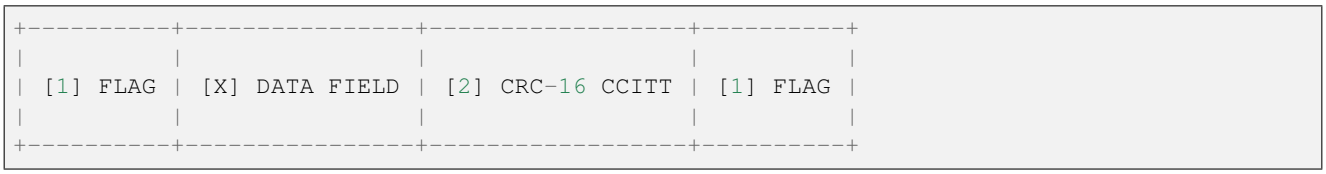
The AX100 radio link is asynchronous and therefore has no mechanism to mark the beginning or end of a frame, so the beginning and end of each frame has to be identified. This is done by using a frame delimiter, or flag, which is a unique sequence of bits that is guaranteed not to be seen inside a frame. This sequence is '01111110', or, in hexadecimal notation, 0x7E. Each frame begins and ends with a frame delimiter. A frame delimiter at the end of a frame may also mark the start of the next frame. A sequence of 7 or more consecutive 1-bits within a frame will cause the frame to be aborted. When using the HDLC mode, the transmitter and receiver will automatically insert bit stuffing to ensure that the unique sequence 0x7E does not occur inside the data field of the frame.

The advantage of HDLC is that is a very common standard supported by many modems.

The disadvantage is that the start and stop flags 0x7E will occur very often in random noise, this means that the MAC layer will be notified of an incoming frame very often, and therefore there is a risk that a TX message could be delayed due to a false synchronization. Another disadvantage is that with such a simple start and stop flag, a single bit-flip in a single byte within the data field could signal a frame-end to the receiver and corrupt the entire frame.

The HDLC mode is generally not recommended because better modes exist.

HDLC Frame layout



Encoding: NRZI, G3RUH/K9NG scrambled, Least significant bit first (apart from 2 byte CRC-16).

10.1.4 Mode 4: HDLC + Viterbi FEC

Some of the disadvantages of the HDLC mode can be removed by adding a Viterbi encoder/decoder around the entire HDLC frame. This is done using a hardware module, before the data is sent and before the synchronizer receives it. The Viterbi encoder uses a special sequence to synchronize the trellis decider, therefore the preamble of the packets in this mode must be 0x7E and there must be a separation of at least four 0x7E's between two frames in order to maximize the probability of a valid packet synchronization on the second frame.

The advantages of the HDLC + FEC mode is that the Viterbi coding gives an increase in the sensitivity. Also the framer is now much more certain not to cut off a message due to a bit-error.

The disadvantage is that the data-rate will be halved when passing through a the Viterbi encoder.

The Viterbi FEC can be combined with the Reed-Solomon FEC to give the very popular CCSDS mode “conv r=1/2 k=7 + rs(223,255)”.

Viterbi Frame layout

Encoding: Viterbi R=1/2 K=7. Most significant bit first.

10.1.5 Mode 5: ASM + GOLAY

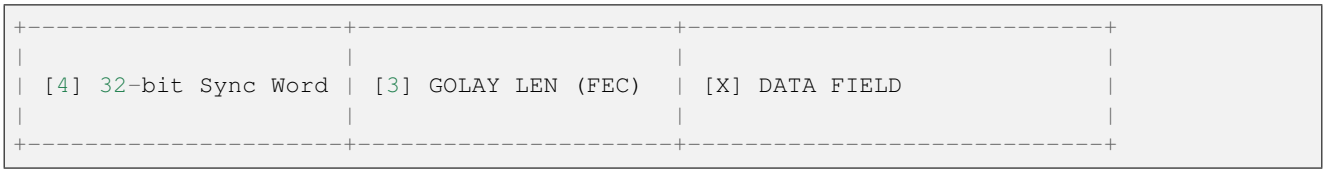
The ASM + Golay is the same as Mode 2, but it solves the problem with the uncoded length field by using a software framing synchronizer. This synchronizer will look for a sync word and a 3 byte Golay field. The Golay field contains 12-bits length and 12-bits FEC + parity. This can be used to correct up to three biterrors in the length field.

The advantage of this mode is that the 28/32-bit ASM sync, together with the robust Golay length field gives a very high certainty of a valid frame and length. In the rare case where a frame has too many bit errors in the length field, it would be very likely to also fail the following FEC/CRC checks. This gives a very close to zero chance of receiving a corrupted frame.

The disadvantage is the additional required processor time in order to verify the Golay field. This have been mitigated somewhat by a more efficient interrupt usage, a hardware accelerated SPI driver and offloading sync word detection to a hardware correlator for the first frame in a transmission. That means that the AX100 does not use any more power in idle than in mode 2.

When a frame have been received, the use of Reed-Solomon FEC will correct bit-errors in the data-field. The CCSDS randomization is needed when using this mode, sine there is no hardware encoding performed by the RF module.

Golay Frame layout



Encoding: NRZ, No scrambling (CCSDS randomization recommended), Most significant bit first.

10.1.6 Mode 6: HDLC + AX.25

The AX.25 mode will attach a 16 byte AX.25 header to each outgoing frame containing a configurable source and destination callsign. The AX.25 relies on the HDLC frame format, so it has all the disadvantages of HDLC and is therefore not recommended.

Note that the AX.25 mode will simply encapsulate CSP packets into an AX.25 frame before transmission and strip it off during reception before routing the CSP packet. The AX.25 frames are transmitted as unnumbered information frames with 112 bit address, 1 byte control field (always 0x03) and 1 byte PID field (always 0xF0). The destination and source SSID fields are 0x60 and 0x61.

AX25 Frame layout

```
+----------+----------------+----------------+------------------+----------+
|          |                |                |                  |          |
| [1] FLAG | [16] AX.25 HDR | [X] DATA FIELD | [2] CRC-16 CCITT | [1] FLAG |
|          |                |                |                  |          |
+----------+----------------+----------------+------------------+----------+
```

Encoding: NRZI, G3RUH/K9NG scrambled, Least significant bit first (apart from 2 byte CRC-16).

10.2 Error detection and correction

Regardless of which framing mode is used, the AX100 Data-link layer supports the following additional modifications to the data field:

- *csp_crc*: 4 byte CRC32 checksum to the frames
- *csp_rs*: 32 byte Reed-Solomon (223, 255) block code
- *csp_rand*: CCSDS randomization of frame
- *csp_hmac*: Frame-Level HMAC

These features are applied for transmission in the following order:

```
HMAC -> CRC -> Reed Solomon FEC -> Randomization
```

It is recommended to always use randomization, Reed-Solomon and either HMAC or CRC. For uplink it makes sense to use HMAC for security. For downlink ti makes sense to use CRC only for simplicity.

10.2.1 CRC32 frame validation

When using the CRC32 FCS mode, a 32-bit cyclic redundancy check will be added to each outgoing frame. This is a simple check to detect any bit-flips in the frame.

The polynomial used is CRC-32C (Castagnoli). You can find the implementation on http://libcsp.org in `csp_crc32.c`.

10.2.2 Reed Solomon Coding

The other option for the FCS field is a 32-byte Reed-Solomon block code. This is known as a RS(223,255) coder and is capable of correcting up to 16 bytes per frame. It also provides error detection, so corrupted frames will be discarded.

The RS coding will increase the sensitivity of the receiver significantly for framing mode 2 and 3. If framing mode 4 is used, it can also provide an extra gain, however not as big.

The disadvantage of RS coding is its rather large overhead, which remains constant even for smaller frames. That means that a simple 1 byte ping message would become 33 bytes with RS coding. However the advantage of the FEC is generally preferred in exchange for some additional overhead.

10.2.3 Randomization

In order to ensure low run-lengths of both ones and zeroes in the transmitted stream the data can be randomized by xor'ing with a pre-shared sequence. In frame-mode 2, 3, 4 and 6 there is already a hardware scrambler (G3RUH), so scrambling the frame in software before transmitting is not necessary, however it also does not give any disadvantages.

The pseudo-random sequence used is based on the CCSDS recommended polynomial:

$$h(x) = x8 + x7 + x5 + x3 + 1$$

10.2.4 Hash-based message authentication (HMAC)

If the HMAC is activated in the receiver it will only accept packets from transmitters which have the matching `csp_hmac_key` parameter. This is useful for protecting a satellite from uplink packets sent from unauthorized ground stations.

10.3 Medium access control

The final topic for the data-link layer that will be addressed is the Medium Access Control. The MAC is an important part of a half-duplex radio where functions like listen-before-talk and CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) are commonly used.

10.3.1 Carrier Sense (RSSI)

Whenever a frame is queued for transmission, the link is constantly monitored for any signal that another transmitter would be running. There are two indicators that are checked:

1. RSSI
2. Receiver de-framer

First of all if the RSSI level seen is above a certain threshold set by the TX 'rssibusy' parameter, the MAC layer assumes that the link is being taken by another transmitter. Secondly the receiver is asked weather or not it has detected a frame-start condition.

If any of these conditions are true, the MAC layer will prevent the transmission and hold it in its outgoing buffer for later transmission.

The rssibusy level should be fine tuned after launch to optimize it to the real levels seen by the radio in space. This can be done by looking at the background RSSI over time and also look at the RSSI for the received frames. Then a good level will be below the RSSI for the received frames but also a few dB above the background RSSI to not prevent the radio from sending when the channel is free. Be sure to set rssibusy to a high value at launch to ensure that the radio will not be blocked from sending.

10.3.2 Collision avoidance

The MAC algorithm on the AX100 takes advantage of the presumption that there will only be two radios sharing the link. This gives a unique method for collision avoidance by inserting a guard period after each transmission has ended.

The TX guard is a couple of milliseconds guard period after each transmission has ended. It is very important to the synchronization of two AX100's when they are running at full link utilization. The idea is that whenever a transmission has just stopped, there will be a guaranteed period at which there can be no collisions. This is known as Collision Avoidance.

10.3.3 Key up delay

The AX100 Is designed for direct operation with another AX100 using it's own built-in power-amplifier and low-noise-amplifier. For this purpose, the key-up and key-down period is so quick that no additional wait time is needed. However, when using large external power-amplifiers and especially when using coaxial relays that have higher switching times, a delay must be inserted in the transmitter.

To take an example, the AX100 is used in the GS100 ground station solution with an external VOX de-activated LNA. After receiving power, this VOX takes 10 ms before the decision to switch the LNA off. After this decision is made the mechanical RF relay takes 15 ms before they are completely switched over. So in this example the 'kup_delay' parameter should be set to 25 ms as a minimum. It's a good idea to add another ~5 ms in order to take mechanical wear and temperature variations into account.

When a transmission has ended, the receiver again takes 10 + 15 ms before the LNA is switched fully back on again. This means that when sending a ping packet to the satellite, the satellite must not respond within the first ~30 ms after receiving the uplink packet. Therefore the RX 'guard' parameter should be set also to 30 ms.

The key-up delay is implemented by transmitting extra preamble bytes before the burst preamble. The number of bytes that are inserted is given by:

$$N = \left\lfloor \frac{T_{\text{kup}} R_{\text{TX}}}{8000} \right\rfloor$$

where

- N is the number of inserted bytes
- T_{kup} is the requested key-up delay in milliseconds
- R_{TX} is the transmission baud rate

This means that for low baud rates, the key-up delay resolution is lower than for higher baud rates.

11. Layer 1: Physical layer (RF)

The physical layer consists of a PLL, a modem and a low-pass channel filter. The configuration of each of these is described in the following sections.

11.1 Modulation

The physical layer of the AX100 radio-link is a Gaussian shaped FSK modulation, known as GFSK. The modulation index is set to the minimum, which is 0.5, the modulation type is also known as MSK or GMSK for the Gaussian shaped version.

11.1.1 Modulation index

The modulation index can be changed using the TX 'modindex' parameter. For higher speeds a modulation index of 0.5 is recommended in order to minimize the transmission bandwidth, but at lower speeds an increase of the modulation index is recommended in order to increase the transmission bandwidth. The increase in transmission bandwidth can be an advantage when there is Doppler shift and other factors that influence the transmitter and receiver frequency control. The default modulation index for 4800 baud is 0.666667. But setting the modulation index parameter to zero will make it calculate it automatically based on the following formula:

```
/* Calculate optional modfactor */
if (modfactor == 0) {
   if (baudrate < 1300) {
      modfactor = 1 / 1.2;
   } else if (baudrate < 60000) {
      modfactor = 1 / 1.5;
   } else {
      modfactor = 1 / 2;
   }
}
```

11.2 Frequency

The frequency can be controlled using an on-board PLL. The AX100 have two parameters: TX 'freq' and RX 'freq' which controls the RX and TX frequencies separately. Whenever the MAC state changes from RX to TX or vice versa, the PLL will be reconfigured.

The first time the PLL is setup at a certain frequency it will do and automatic ranging which takes a couple of milliseconds. The result of the ranging is written to the debug output during initialization. Here is an example:

```
AXSEM: PLLRANGE S 0x9 E 0x9 T 0us
```

This shows the PLL VCO range start value, 0x9 in this case, and the end value 0x9, which is similar, and finally the time used (0 microseconds).

The start value of the PLL autoranging is set to 0x9 as default, which gives an almost instantaneous PLL ranging at most frequencies in the UHF range. When changing to other frequencies it might be that the VCO range result is different, and it could be an advantage to change the start value.

11.3 Bandwidth selection

Put very short: The receiver bandwidth cut-off should be at 1.5 times the bitrate. Setting this lower could cause attenuation of actually wanted signals. Especially if the transmitter and receiver has a tx/rx frequency offset. When setting the RX ‘bw’ parameter to zero it will simply calculate it automatically as 1.5 times the bitrate.

Increasing the bandwidth will increase the AFC pull-in range, and make the system less affected by rx/tx frequency offset. But it will also decrease the receiver sensitivity, due to an increase in the noise floor. The bandwidth can be selected in steps of 1 Hz.

An increase in the bandwidth by a factor two, decreases the receiver sensitivity by 3 dB.

12. Self-test Features

The AX100 contains some advanced debugging functions available when using the 4-pin serial debugging interface. These functions can be used for verification of the operation on the physical layer, data-link layer and network layer.

12.1 BER Test: RX Test routine

When testing the receiver, a key figure is the Bit Error Rate (BER). This can be measured by generating a modulated '01010101' sequence from a signal generator using the correct modulation parameters. In order to get the modulation parameters for the current settings, please refer to the debug information on the boot-screen of the AX100:

```
AXSEM: BW_RX actual 19230
AXSEM: CH_BW 50000, reg_if 819, F_if 12496
AXSEM: Baudrate 4800, datarate_reg 32820
AXSEM: F_offset 3750, maxrfoffset_reg 3932
```

Here the actual achieved rx bandwidth, the IF frequency for the mixer, the baudrate and frequency offset is displayed.

This means that the modulator in the signal generator must be configured for a 2-FSK signal with a bitrate of 4800, and a Fdev = (4800 * RX_MODINDEX) / 2 = 1600 Hz (for modindex = 1/1.5).

When this is setup, the AX100 has a command to setup the BER test. This will switch the receiver into mode 1 (RAW), and start counting biterrors. Here is an example:

```
nanocom-ax # bertest begin
3120364498.351 axsem: Scratch test OK (6)
3120364498.353 axsem: RX Baudrate 4800, bw_rx 7200, f_dev 1600
3120364498.355 axsem: Decimation 27
3120364498.356 axsem: F_adc 1000000, F_baseband 37037
3120364498.358 axsem: BW_RX actual 9259
3120364498.359 axsem: CH_BW 25000, reg_if 410, F_if 6256
3120364498.360 axsem: Baudrate 4800, datarate_reg 15802
3120364498.362 axsem: AFC range +/- 1800, maxrfoffset_reg 1887
3120364498.364 axsem: Attack 6 (5.029555) F3db 4800.000000 Hz
3120364498.366 axsem: Decay 12 (11.694892) F3db 48.000000 Hz
3120364498.367 axsem: AGCGAIN 0xc6
3120364498.369 axsem: ARGMIN cache hit 4 15802 => f8
3120364498.370 axsem: TIMEGAIN0 f8
3120364498.371 axsem: TIMEGAIN1 f6
3120364498.372 axsem: TIMEGAIN3 f5
3120364498.374 axsem: FREQDEV1x 43
3120364498.375 axsem: Init complete
3120364499.004 default: BER: 0.498333, err55: 1505, errAA: 1495, count: 3000
3120364499.630 default: BER: 0.494167, err55: 3035, errAA: 2965, count: 6000
3120364500.254 default: BER: 0.499111, err55: 4508, errAA: 4492, count: 9000
3120364500.879 default: BER: 0.497667, err55: 5972, errAA: 6028, count: 12000
```

The system will continue running the bertest until the 'bertest pause' command is issued. In order to end the bertets run the 'bertest pause' command, or reset the system.

The BER is performed from an unsynchronized bit-stream by using the fact that the pattern '01010101' (0x55) can be either that, or the shifted by one version '10101010' (0xAA). It then counts the total number of bits and calculates two different error counters, one assuming correct sync on 0x55 and other based on the sync of 0xAA. It then selects the lowest of these numbers and divides that with the total number of bits to get the BER.

12.2 BER Test: TX pattern

The command `bertest txpattern` will key up the radio and send a continous '01010101' (0x55) test pattern. Remember to set the `max_tx_time` parameter to zero in order to avoid an automatic key-down of the transmitter during the tests. Also please observer the temperature of the system during any extended TX test.

12.3 Single Carrier

The commands `up` and `dn` will turn on and off an unmodulated carrier.

13. GOSH Debugging Interface

The AX100 uses the *liblog* library from the GomSpace SDK.

Table 13.1: *List of logging groups*

Group Name	Print	Store	Description
axsem	EWIDT	EW...	Radio driver
event	EW...	EW...	Radio events handler
frame	EW...	EW...	Hexdump of RX/TX frames
reset	EWIDT	EW...	Printout during boot
mac	EWID.	EW...	MAC layer
user	EWIDT	EW...	Log insert cmd
default	EWIDT	EW...	Default level
csp	EW...	EW...	CSP debug
twim	EW...	EW...	I2C driver (master)
can	EWI..	EW...	CAN driver
gsspi	EWIDT	EW...	SPI driver
i2c	EW...	EW...	I2C driver (slave)

The following section contains a short introduction to the client commands for the liblog library:

13.1 Client Commands

Here is a list of client commands for liblog:

```
gosh # log <tab>
  print               Set printmask
  store               Set storemask
  list                Show groups
  insert              Insert log message
  hist                Show history
```

Furthermore there is an alias for 'log print':

```
nanocom-ax # help debug
  debug               log: Alias of 'log print'
```

13.1.1 print & store

The 'log print', 'debug' and 'log store' commands will change the logging levels for different groups, either for the console (print) target or the ram/fram (store) target. Here is an example of setting the debug flag for the 'mac' group:

```
gosh # log print mac
usage: print <group>[,group] <Err|Wrn|Inf|Dbg|Trc|Std|All|No>
gosh # log print mac debug
Set mask of group mac from 0x18 to 0x1E
```

13.1.2 list

The 'list' commands shows all groups and their 'print' and 'store' targets:

```
nanocom-ax # log list
Group           Print Store
axsem           EWIDT EW...
event           ..... .....
spidma          EWIDT EW...
...
```

13.1.3 insert

The insert command will create a new logging entry in the ‘user’ group of the logging system. This can be used to test the logging system and during testing to mark the begining or the result of a certain test:

```
gosh # log insert
usage: insert <level> <message>
gosh # log insert info "Beginning of some test"
3120366290.557 user: Beginning of some test
```

The accepted log level names are: ‘error’, ‘warn’, ‘info’, ‘debug’, trace’

13.1.4 hist

The hist command will show the latest stored log entries:

```
nanocom-ax # log insert error "Test error msg"
3120367702.941 user: Test error msg
nanocom-ax # log insert warn "Test warning"
3120367708.950 user: Test warning
nanocom-ax # log hist
3120367708.950395 user: Test warning
3120367702.941871 user: Test error msg
```

14. Software changelog

3.13.0 (2020-03-25)

- Bug: Fix broken kup_delay feature
- Improvement: Adapt latest document template, minor fixes in documentation

3.12.0 (2019-09-20)

- Feature: Make KISS USART baud configurable
- Bug: Drop partially working auto beacon feature
- Improvement: Describe AX.25 frame format in documentation
- Improvement: Minor documentation fixes

3.11.0 (2019-03-05)

- Bug: Improved HMAC authentication. Legacy mode available through configuration.

3.9 (2018-02-23)

- Bug: prevent FIFO overrun in AX.25 mode
- Bug: more graceful handling of CSP out of buffers condition

3.8 (2017-12-19)

- Feature: added config command for updating and verifying default configuration
- Improvement: decrement ground watchdog by 1800 seconds at boot
- Bug: correct handling of PA temperature above limit for TX
- Bug: AX.25 frames is now formatted according to standard
- Bug: check and correct GOSH and KISS USART config if incorrect
- Bug: enhanced tx procedure to prevent FIFO underrun
- Bug: added interrupt timeout handling

3.7 (2017-02-13)

- Feature: disabled DMA in ax5043 driver

3.6 (2017-01-11)

- Bug: Fix issue with mob_tx_wake semaphore on error interrupts
- Bug: Fix issue that cs for temp2 was not initialized

3.5 (2017-01-03)

- Bug: new libasf with fix that discards non-extended can frames

3.4 (2016-11-21)

- Bug: clear table 1 and 5 on ground watchdog timeout

3.3 (2016-09-09)

- Bug: remove potential buffer leak in golay path
- Bug: add handling of SPI DMA timeout
- Bug: fix CAN frame reordering

3.2 (2016-04-21)

- Bug: show warning if SPI chip select fails
- Improvement: update submodules for ASF, CSP, GOSH, log, param, util

3.1 (2015-03-12)

- Bug: Number of buffers
- Feature: Added manual in .rst format

3.0 (2015-11-27)

- Breaking: Include CSP header in HMAC and CRC-32 calculation (this breaks with some v2.x settings)
- Breaking: Seperate RX/TX parameter tables (New client code needs to be built)
- Feature: Support for multiple RF bands (VHF + UHF)
- Feature: Support for different RX/TX framing modes
- Feature: Increased MTU from 199 to 219 bytes (depending on FEC choices)
- Feature: CCSDS additive randomization
- Feature: AX.25 mode (mode 6)
- Feature: ASM+GOLAY mode (mode 5) (recommended)
- Feature: Common TX framer
- Feature: 'ber txpattern' will transmit a 01010101 pattern
- Feature: 'pllrange' command to test the PLL
- Feature: PARAM_BOOL support (set to true/false, on/off or 1/0)
- Feature: Added 'ber' parameter to insert random biterrors into transmitted frames (for testing)
- Bug: RFERR calculation in 'cal get' command
- Improvement: Increase MCU speed from 32 to 64 MHz
- Bug: Fix AGCGAIN calculation with multiple XTALs
- Feature: Use nsec to calculate ping times
- Improvement: Update SPI driver to use DMA (reduces power consumtion and increases speed)

2.5 (2015-07-10)

- Feature: Added support for variable power output for hw-rev: ax100-3

2.4 (2015-02-06)

- Improvement: Updated TWIM driver to use PDC
- Bug: TWIM driver could hang on bus-error
- Bug: SPI driver had a rare deadlock
- Bug: Non voluntary interrupt nesting
- Feature: log system uses absolute timestamp again

2.3.1 (2015-05-18)

- Bug: Running system out of memory from I2C caused bus stall

2.3 (2015-05-07)

- Feature: Added support for hw-rev: ax100-3
- Feature: Add rx_guard parameter (to avoid transmitting just after receiving)
- Feature: Add support for GS100 (extptt driver updated)
- Feature: Add key_up_delay parameter (to allow LNA to turn off with VOX in GS100)

- Feature: Added ‘param tables’ overview command
- Feature: Decrement GNDWDT with 1 second during boot-up
- Bug: Memory leak (added in v2.2) if loop interface was used for outgoing node (blackhole)
- Bug: Bug in TWIS driver causing bus to stall
- Feature: Remove csp ‘debug’ command (to debug csp use ‘debug csp i’ or ‘debug csp d’) with liblog
- Feature: Reduce tick timer to 100 Hz again
- Feature: Decrease number of CSP buffers to have enough memory for rparam commands
- Improvement: Better gosh tab completion
- Improvement: Decrease TWI interrupt priority (should help avoid FIFO underrun during I2C load)
- Improvement: Change ping response time counter to show microseconds
- Improvement: Much faster CSP routing due to more efficient log messaging
- Improvement: Increase MAC layer responsiveness from 50 ms to 10 ms
- Feature: Newest liblog with better colors and userinterface

2.2 (2015-04-27)

- Feature: CAN support
- Feature: Timing system without locking issues
- Improvement: Latest version of liblog with a few usability improvements
- Improvement: Remove old driver debug system
- Bug: Issue in parameter maxbuflen check (should have no influence on ax100 params)
- Bug: Allow for off-by one in the FRAM GNDWDT check (avoids some false warnings)
- Bug: TWIS driver could miss frame-end
- Improvement: Faster TWIS buffer handling (two bytes per ISR)
- Improvement: Improved memory handling of rparam service

2.1 (2015-03-24)

- Breaking: protocol change: libparam now supports partial serialization
- Bug: FIFO underrun when sending several packets back-to-back
- Bug: Add locking to FRAM driver
- Improvement: Stop transmitting if PLL lock is lost
- Feature: Add background RSSI to HK
- Feature: Add TX duty to HK
- Feature: Set default AFC range to -4 and cap autocalc output at 10 kHz

2.0 (2015-03-09)

- Breaking: protocol change: Syncword and preamble for mode 2 (ASM)
- Improvement: Internal tick timer from 100 to 1000 Hz
- Bug: beacon function was deactivated in v1.10
- Feature: High precision [1 us] clock resolution
- Feature: VMEM interface to access FRAM and AXSEM directly
- Feature: Early version of liblog with FRAM backend included (no server)
- Feature: configuration option to enable/disable liblog FRAM storage

1.10 (2015-02-10)

- Feature: rx_afcrange and rx_modindex parameters
- Feature: configurable USART interface for GOSH
- Feature: configurable USART interface for KISS/CSP
- Feature: configurable LED option
- Feature: CSP CIDR routing with storage in libparam
- Feature: new EXTPTT driver for kiruna GS
- Feature: new rssi_offset calibration parameter
- Feature: DAC settings to axsem parameters
- Feature: register compare command for easy comparison with axradiolab
- Feature: new up/dn commands to make a test carrier
- Feature: pretty reset cause printout
- Feature: I2C addr/speed configuration parameters
- Feature: CSP HMAC on Layer-2
- Feature: active_config parameter to housekeeping
- Feature: DAC output as high impedance input on the MCU
- Feature: Lower PLL CPI once again
- Feature: Negative rx_afcrange auto-selectes according to RXBW
- Feature: default RXBW to 0 instead of 15000
- Feature: default configuration file from 111 to 24
- Feature: set TX inhibit to maximum value as default, thus requirering a config save before TX
- Feature: Force AXSEM poll every second, in case IRQ was not triggered
- Feature: Increase clock speed from 16 to 32 MHz in order to handle higher speeds
- Improvement: Move commands to CONST and do linkerbased gosh setup
- Improvement: 3x improved ARGMIN calculation time
- Improvement: Update AX100 client to use fletcher checksum and empty request
- Improvement: Upgrade FreeRTOS 8.0.0 to 8.0.1
- Improvement: Upgrade libasf from 3.16 to 3.21
- Improvement: Setup LED's during boot to indicate board status
- Improvement: Increase stack size for console and ensure LED's are turned off after boot
- Improvement: Only change RX frequency setting when receiver is idle
- Improvement: libparam tab completion and a few safety features
- Bug: bug causeing tx_modindex to be set to zero for rx_baud > 60000
- Bug: channel BW calculation for BW's over 200 kHz
- Bug: USART init bug in libasf
- Bug: f3dB calculation
- Bug: bug in the RX fDev calculation for high baudrates
- Bug: negative packet length bug in csp_if_ax100
- Improvement: Remove FTP and LibIO

- Improvement: Remove dependency on libgomspace, and install libutil/libgosh instead

1.9 (2014-10-03)

- Bug: Fix issue with SDRAM being tristated
- Bug: Fix issue with CAN controller being tristated
- Improvement: Save 10 mA on setting SDRAM into deep power-down mode

1.8 (2014-09-10)

- Improvement: Discard packet on TX overtemp to prevent out of buffers
- Improvement: Increase stack sizes and decrease buffer count from 50 to 42 to prevent stack overflow

1.7 (2014-09-04)

- Improvement: Added EPS in COM beacon and EPS commands to console

1.6 (2014-09-03)

- Improvement: Set default RX BW to 15 KHz and disable LED
- Improvement: Added possibility to edit rtc data with hex input

1.5 (2014-09-02)

- Feature: Added TX inhibit feature
- Feature: Added configuration parameters for beacon interval and beacon holdoff
- Feature: Added com-beacon tx to wdt task
- Improvement: Disable SDRAM init when SDRAM is not used for heap, in order to save power

1.4 (2014-08-21)

- Improvement: Use driver debug in TWIM/S drivers
- Improvement: Added support for sending full lists in rparam service handler
- Bug: Fix slave bus error twi hw reset
- Bug: Fix multi-master i2c driver problem
- Improvement: Newest libparam

1.3 (2014-08-18)

- Bug: I2C stability fixes:
- Bug: Buffer corruption in I2C driver fixed
- Improvement: Reduced I2C speed from 400 to 100 kbps
- Improvement: Add CRC32 as default to all transactions

1.2 (2014-08-13)

- Improvement: Added new GNDCFG stored config area to file_id 111
- Breaking: Move csp_addr from param table 5 to table 0
- Improvement: Upgrade to latest libparam
- Improvement: Make GNDWDT clear stored config file_id 0 after invoking
- Feature: Remove ASM_MIN and ASM_MTU exceeded error messages
- Improvement: Added rparam clear command to delete stored config

1.1 (2014-06-26)

- Feature: Merge main.c/init.c with nanomind master
- Feature: Added ax100-client

- Improvement: System is independant of SDRAM
- Feature: Add FTP upload to RAM
- Feature: Add write protection to FRAM
- Feature: Add CSP CMP clock functions
- Bug: Fix task priority issue
- Bug: Fix PLLCPI being set too high
- Bug: Fix txpwr override on key up

1.0 (2014-05-27)

- Feature: Support for AX100-2
- Feature: AX5043 driver
- Feature: FRAM storage backend
- Feature: RAW, RAW_SYNC, HDLC and FEC modes
- Feature: GUPS parameter system
- Feature: CSP router
- Feature: Temperature sensors and over temperature protection
- Feature: RSSI based CSMA
- Feature: Builtin BER-test application

15. Disclaimer

Information contained in this document is up-to-date and correct as at the date of issue. As GomSpace A/S cannot control or anticipate the conditions under which this information may be used, each user should review the information in specific context of the planned use. To the maximum extent permitted by law, GomSpace A/S will not be responsible for damages of any nature resulting from the use or reliance upon the information contained in this document. No express or implied warranties are given other than those implied mandatory by law.