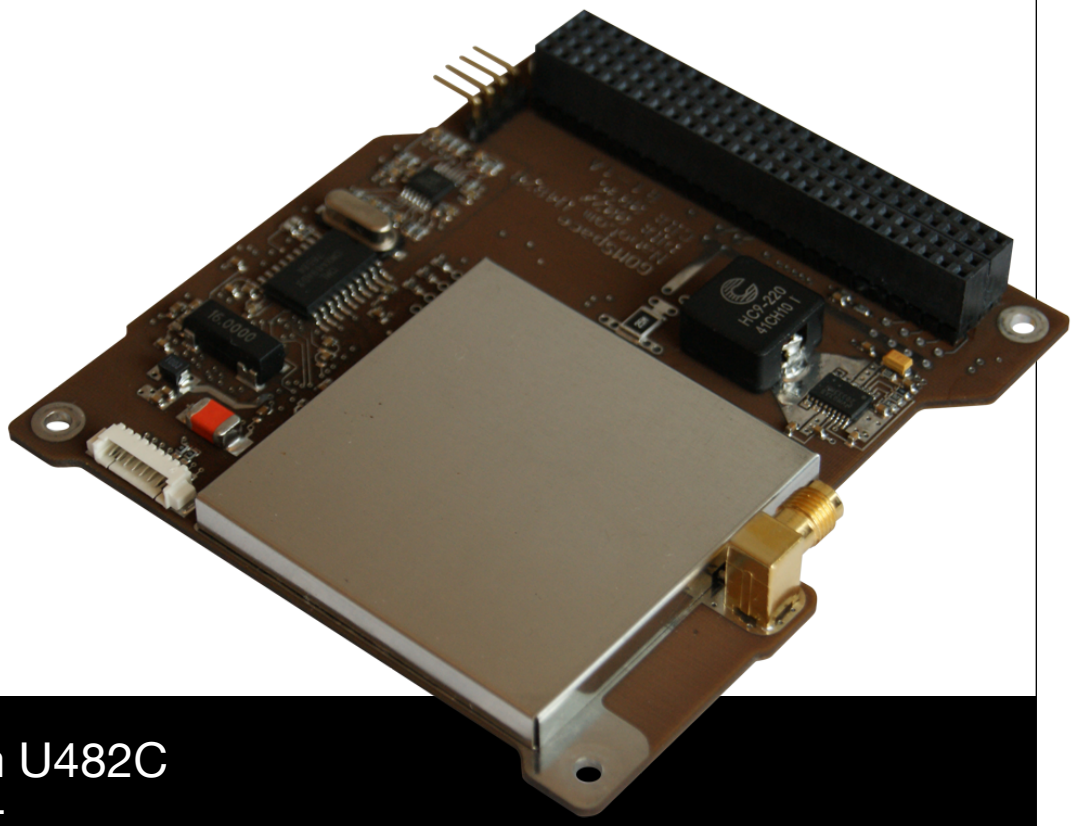
GOMSPACE



NanoCom U482C
Datasheet
v5.0

UHF packet transceiver system for space applications

NanoCom U482C

The NanoCom U482C offers a reliable space-link to small satellite platforms with its flight-proven transceiver and simple yet fully-featured architecture. Mission success highly reliant on the space-link quality, and with its sensitive receiver and error-correcting code, NanoCom U482C ensures a good link margin in any low-earth orbit.

Feature Overview

- Half-duplex UHF narrow-band FM transceiver
  - Designed for 435-437MHz operation
  - Super heterodyne receiver
  - Packet reception down to -125dBm at 1200bps
  - Transmitter with 27-33dBm at > 45 % PAE
- MSK baseband
  - Uplink: 1200-4800 baud
  - Downlink: 1200-9600 baud
- Adaptive frame format with:
  - 32 bit ASM header
  - FEC (Golay) encoded length and type field
  - Frame length 1-255 bytes (without FEC)
  - Per-Packet Optional Virterbi FEC r=1/2, K=7
  - Per-Packet Optional Reed-Solomon FEC (223,255)
  - Per-Packet Optional CCSDS Randomization
  - Per-Packet Optional CRC32
- On-board measurement of:
  - PCB and PA temperatures
  - Battery voltage
  - RSSI and RF-error on receiver
- Autonomous audio morse beacon when idle
- User-defined timeout, interval and WPM
- Configurable content (text and measurements)
- Simple CSP based I^2C interface at 400kbps
- Builtin over temperature protection
- High-effeciency buck-converter for transmitter supply
- Compatible with the Cubesat Kit "PC104" formfactor
- Operational temperature: -30 C to +60 C
- Dimensions: 95.40 mm x 90.15 mm x 18.00 mm
- PCB material:
  Glass/Polyamide IPC 6012C cl. 3/A
- IPC-A-610 Class 3 assembly

Applications

- Single CubeSat satellites
- Triple CubeSat satellites
- Nano Satellites

Compatibility

- GomSpace products
- CubeSat Kit products
- Innovative Solutions in Space products
- ClydeSpace products

Flight Heritage

- SSETI-Express (Design)
- AAUSAT-II (Baseband)
- Xatcobeo (U480)

Functional Description

The NanoCom U482 communication system is designed to provide the space-segment-part of a space-link for nano- and pico-satellites by means of a half-duplex UHF transceiver operating in the amateur radio 70cm band. The internal interface is an I^2C bus allowing command and data exchange with the NanoCom board via a simple protocol.

All packet framing plus the optional Viterbi encoding/decoding is performed by the NanoCom so only the raw data packets need to be transferred over I^2C bus allowing for swift integration both of the hardware and the communication protocol. Vital housekeeping measurement points are sampled at user-defined intervals and stored for retrieval via the I^2C bus or spacelink upon request. A CW/FM morse beacon is activated when the system is idle which may morse any combination of the housekeeping elements in human-readable format.

The unit uses one or two power supplies depending on configuration: VCC and VCC_TX. VCC powers all the digital and analog circuitry plus the receiver. VCC_TX powers the transmitter power amplifier. These supplies may be connected to various pins in the stack connector, and VCC_TX can use an on-board high-effeciency buck-converter.

The recommended configuration is to use the buck-converter for VCC_TX and supply this from a voltage higher than the required supply for the transmitter. In this way, the current through the stack connector is minimized which may be necessary as the power amplifier can draw up to several amps - more than the rating of the stack-connector.

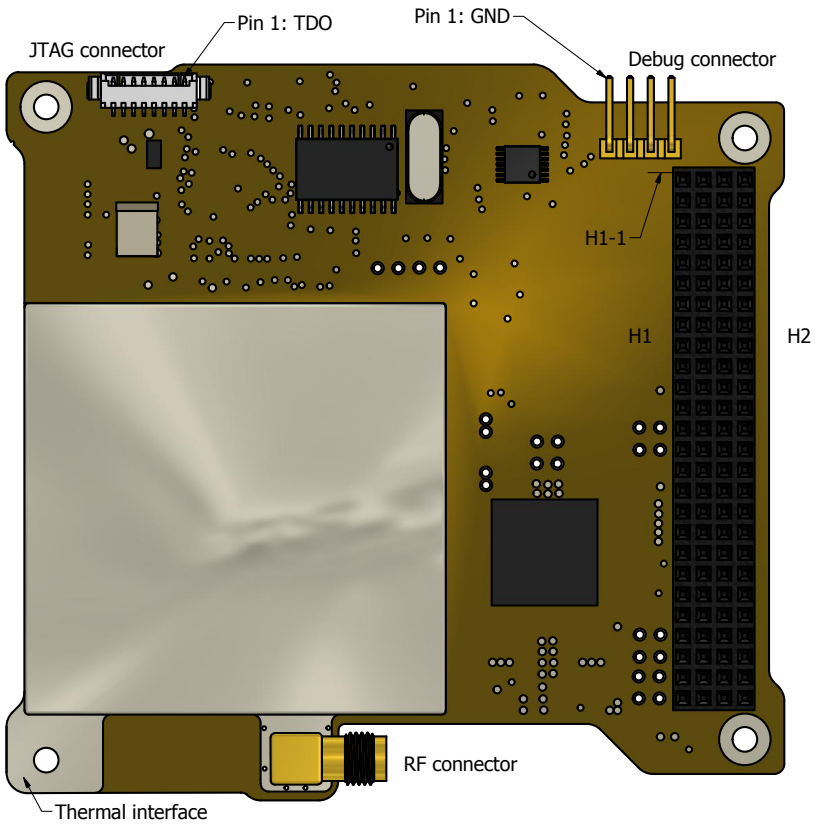
Getting Started

Be sure to use observe antistatic handling procedures at all times when handling the NanoCom U482 transceiver. Before applying power to the board, make sure that the following conditions are in order:

- The SMA RF connector is connected to a 50 Ohm load rated at >4W.
- The ground connection is properly connected to ground on the power supply.
- The thermal power dissipated by the transmitter can be conducted away from the board. In 1 atm. pressure the transmitter is operational up to 60 deg. C. While the power amplifier can be operated up to +80 deg. C. *Do not exceed these temperatures.*
- If operated in vacuum, the heat must be conducted away from the board by means of a solid thermal connection to the corner-holes or by soldering a heat-strap directly onto the ground-plane below the power amplifier. *Contact gomspace for further instructions.*
- All four corner holes are connected to GND so make sure there are no short-circuit from these holes to other electrical potentials.
- If operated at temperatures below 0 deg. C in atmospheric air make sure the board is cleaned properly with IPA before so that the unavoidable condensing upon de-freezing does not cause electrically conductive ionised water to form on the circuits.

Connectors

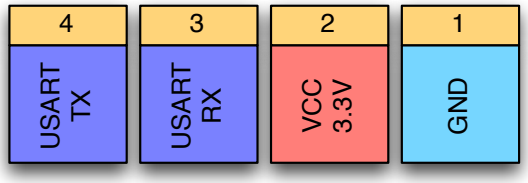
The U482C is equipped with the standard CubesatKit Stack Connector (H1, H2) plus a debugging connector, RF connector and a JTAG connector for firmware upgrades.



Debug Connector

The Debug connector can be used to power the unit and to access a console interface to control and monitor the system - mainly used by GomSpace for testing and check-out. The functionality is self-explanatory and changes a bit with different software revisions as it is mainly meant for testing. The interface is a 3.3V (CMOS level) UART (serial port) running at 500 kbaud.

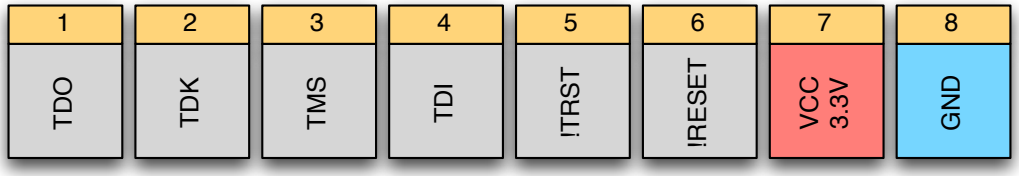
Debugging Connector Pinout



JTAG Connector

A Molex PicoBlade connector is used for firmware upload. No further documentation is provided unless for some reason it is necessary to upload firmware without the presence of a GomSpace engineer.

JTAG Connector Pinout



RF Connector

The RF connector is a right-angle SMA (TYCO ELECTRONICS 5-1814400-1).

Stack Connector

The following table shows the pinout for the CubeSat Kit Connector H1 and H2 (see page 7). Pins with red dots are optional (to be agreed upon time of order placement). Some pins are shown multiple times as they can be configured to either of multiple connections.

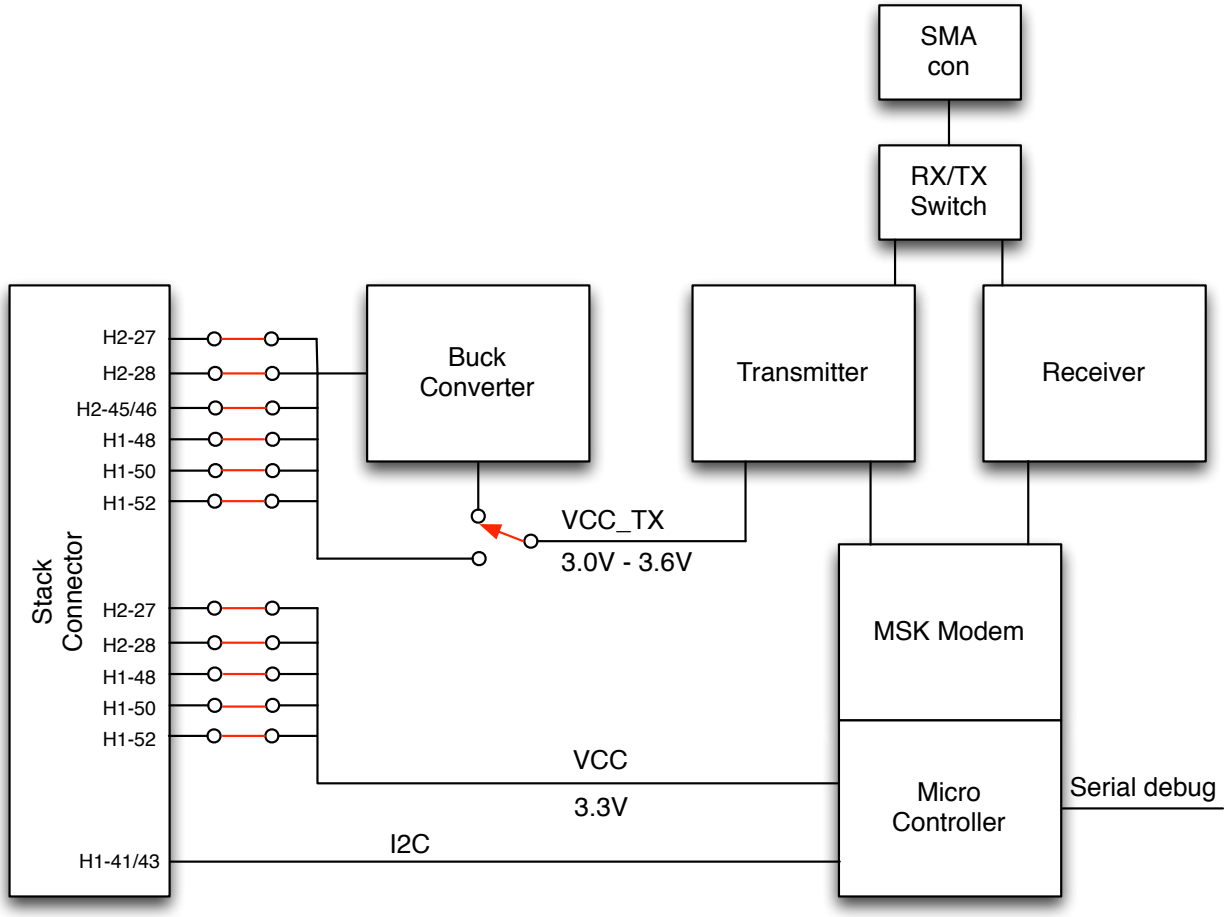
Pin#	Mnemonic	Description	Opt
H1-41	I2C-SDA	I2C serial data	
H1-43	I2C-SCL	I2C serial clock	
H1-48	VCC	3.3V supply	●
H1-50	VCC	3.3V supply	●
H1-52	VCC	3.3V supply	●
H2-27	VCC	3.3V supply	●
H2-28	VCC	3.3V supply	●
H1-48	VCC_TX	Transmitter supply	●
H1-50	VCC_TX	Transmitter supply	●
H1-52	VCC_TX	Transmitter supply	●
H2-27	VCC_TX	Transmitter supply	●
H2-28	VCC_TX	Transmitter supply	●
H2-45	VCC_TX	Transmitter supply	●
H2-46	VCC_TX	Transmitter supply	●
H2-29	GND	Power ground	
H2-30	GND	Power ground	
H2-32	GND	Power ground	
H2-45	V_BAT	Battery voltage for measurement and optional dc-dc converter	
H2-46	V_BAT	Battery voltage for measurement and optional dc-dc converter	

Block Diagram

Below are the principle block diagrams showing the architecture of the transceiver and the baseband/digital circuits.

Main Block

The main block diagram gives an overview of supply and signal lines in the U482C. The red lines are configurable during manufacture and should be specified by the customer using the options sheet provided prior to ordering.



Receiver

The receiver is a super heterodyne receiver with two intermediate frequencies of 21.4MHz and 455kHz. The first IF of 21.4MHz is high enough to give good image frequency suppression by the 437MHz filter after the LNA, while the second IF at 455kHz ensures good channel separation due to a steep +/-6kHz (or optionally +/- 17.5kHz) filter. The FM detector is an integrated device, SA606, with high sensitivity which provides the audio output plus an RSSI measurement. An indication of the RF error on the input is taken from the DC-level of the filtered audio-signal before its AC-coupling.

Transmitter

The direct-FM transmitter consists of a VCO built around a silicon tuning diode which allows the TX frequency to be adjusted by a few kHz by moving the DC level of the AF input up or down. After the VCO is a filter and a buffer leading the low-distortion signal into the power amplifier which outputs 30-34dBm with an efficiency around 50% depending on the temperature. The transmitter frequency is temperature-dependent in a range of maximum +- 10 kHz, so the ground station operator needs to take the frequency drift into account.

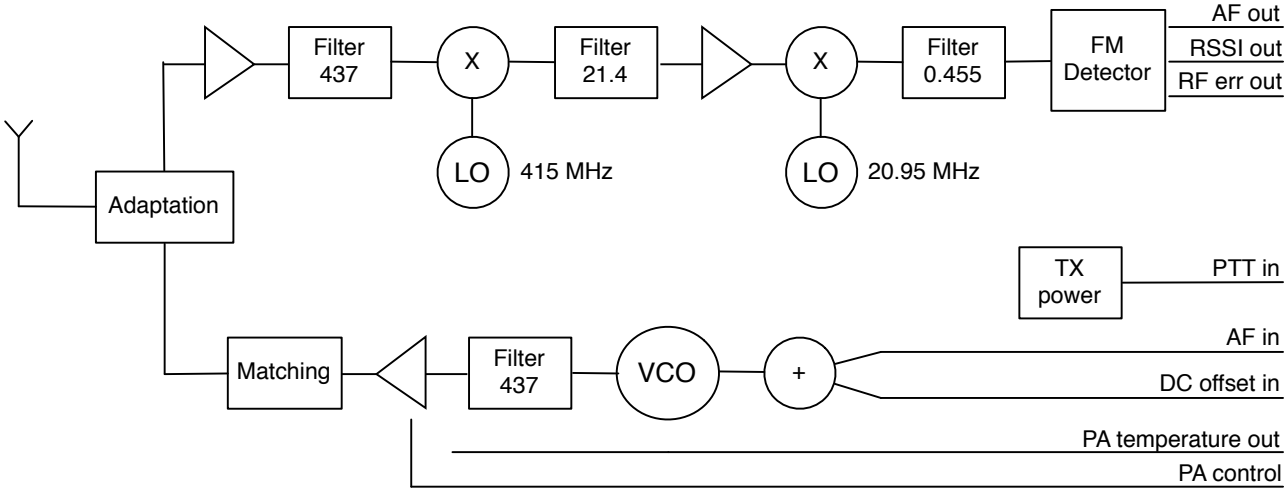


Figure: Transceiver block

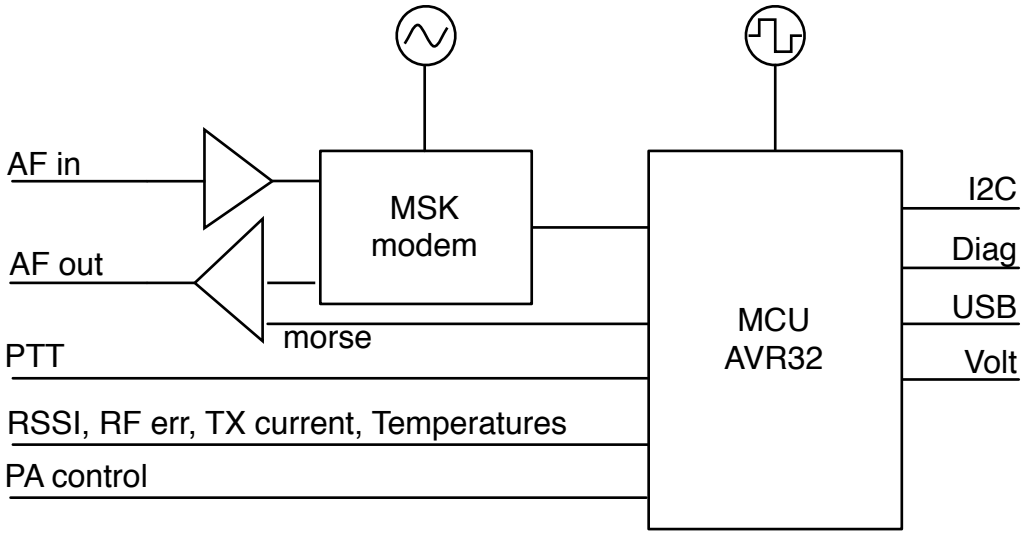


Figure: Baseband and uC

Baseband

The baseband processing is done by an MSK modem allowing 1200, 2400 and 4800 baud half-duplex operation. The transmitter supports 9600 baud. Only use this option if you are sure that your FM receiver has a large enough baseband bandwidth to carry a 15 kHz signal without distortion.

Microcontroller

The heart of the NanoCom system is a 32 bit microcontroller, Atmel AVR32, which controls the modem and the transceiver and facilitates communication with other on-board systems via its I²C interface. The uC continuously clocks in the bits that the modem detects searching for a sequence marking the beginning of a packet. In the CCSDS frame format this sequence is called the ASM (attached sync marker) and once detected, the uC receives the next 3 bytes of data, which contains a golay encoded length field and FEC type field, indicating how much data to buffer and how to decode it. The received frames may be FEC decoded if the type field indicates this option is enabled. If the CRC-32 or Reed-Solomon checksum is correct a CSP packet is extracted from the CCSDS frame and routed to the destination on the internal network.

Temperatures, RSSI and RF error are sampled by the internal 10 bit ADC in the uC and stored such that the user can retrieve the values upon request.

The Morse beacon signal is generated by sinusoidal modulation of a PWM channel by the uC which give an 800Hz tone which is fed directly to the amplifier/mixer along with the MSK output signal. In morse mode, the modem output is always disabled.

Connections

For compatibility with the majority of the products on the pice-satellite market, the system uses a 104-pin stack-connector composed by two SAMTEC ESQ-126-39-G-D connectors (optionally other types can be mounted) and all connections to other on-board systems (except the antenna) is done through this connector.

The NanoCom system is supplied by either a single 3.3Vdc supply or by two supplies: 3.3Vdc for the uC, baseband and receiver plus another regulated supply for the transmitter of up to 3.6Vdc. The system also supports running the power amplifier from an unregulated 5-28Vdc supply on the Vbatt connector, using an on-board switch-mode dc-dc converter. A common ground connection is used. The I²C bus uses two wires and has optional pull-up resistors on-board. A total pull-up resistance of max 1kOhm is recommended in order to ensure reliable communication at 400kbps.

The RF connection is an sturdy right-angle SMA from TYCO ELECTRONICS providing a reliable connection to the antenna.

Morse Beacon

The NanoCom transceiver has a built-in morse beacon which can be used for tracking and rudimentary housekeeping. When the radio link (both up and down) has been idle for a defined amount of time (morse_interval), the radio can be configured to start transmitting a morse beacon.

The beacon itself is pre-configured to send out a specific morse string (a callsign typically), along with some housekeeping data such as the battery voltage or system temperature. The morse system is run-time configurable and can be enabled or disabled from the ground station.

When the radio link is utilized by either an satellite sub-system transmitting data, or by receiving data from the ground station. The NanoCom will pospone the beacon transmission temporarily, in order to avoid colliding with up/downstream data. The pospone period is typically 5 minutes, but also configurable.

The beacon system has 3 different modes:

0. CW mode

1. CW + FM mode

2. FM mode

CW mode controls the supply to the power amplifier. The morse becon can therefore be received with a radio in either USB/LSB mode, or a dedicated CW mode.

CW + FM mode uses a modulated carrier (a 600hz tone), but is still controlling the supply to the power amplfifier. This has the advantage that the beacon can be received both in CW and in FM mode. However it is not pure CW since the energy of the signal is spread in the spectrum by the modulation, and it is not pure FM either, because the turning on and off of the PA produces a 'cliking' noise.

FM mode is with the power amplifier constantly on, turning the modulated tone on and off. This uses more power, since the PA is turned on even when the morse code is not signalling. However it can be advantageous if a tracking signal is what is wanted.

Electrical Characteristics

Parameter	Condition	Min	Typ	Max	Unit
VCC	Supply voltage	3.20	3.30	3.40	V
VCC Tx	Supply voltage to transmitter	3.00	3.30	3.60	V
Current Consumption, VCC	Power consumption VCC, 3.3V				
- standby	Listening			70	mA
- rx	Receiving			70	mA
- tx	Transmitting			40	mA
Power Consumption, VCC_TX	Power consumption VCC_TX				
- standby	Listening			0.1	mA
- rx	Receiving			0.1	mA
- tx	Transmitting	2000	5000	5500	mW

RF Specifications

[1] Sensitivity is understood as follows. Min: point where the receiver starts decoding frames with PER ~ 95%, Max: Point where receiver consistently decodes packets with PER <= 1%. Typ: PER ~ 10%

Parameter	Condition	Min	Typ	Max	Unit
Transmitter					
Frequency		437.000	437.425	438.000	MHz
Output power		27	33	34	dBm
TX-on delay:				5	ms
Temperature range		-20		70	deg. C.
Temperature stability			12	15	ppm
Receiver					
Frequency		437.000	437.425	438.000	MHz
Sensitivity (See note 1)	1200 Baud, FEC	-125	-123	-122	dBm

Parameter	Condition	Min	Typ	Max	Unit
	2400 Baud, FEC	-123	-122	-120	dBm
	4800 Baud, FEC	-120	-119	-117	dBm
	9600 Baud, FEC (17.5 kHz)	-115	-110	-100	dBm
RX-on delay				5	ms
Temperature range		-20		70	deg. C.
Temperature stability			10	15	ppm

Baseband specifications

Parameter	Condition	Min	Typ	Max	Unit
Modulation	FFSK/MSK				
Frequencies	1200 baud 1		1200		Hz
	1200 baud 0		1800		Hz
	2400 baud 1		1200		Hz
	2400 baud 0		2400		Hz
	4800 baud 1		2400		Hz
	4800 baud 0		4800		Hz
	9600 baud 1		4800		Hz
	9600 baud 0		9600		Hz
Required SNR			10		dB

Tone frequencies are phase continuous; transitions occur at the zero crossing point.

Modem device: CMX469AD3.

Temperature protection

During the housekeeping cycle the radio will check temperature sensor A, which sits close to the power amplifier, for an over temperature condition. If the radio is transmitting the power to the TX is shut off immediately and the radio will go into RX mode. The modem will still continue to be in TX mode until the transmission has ended.

When the radio has been put into temperature protection mode, the TX will not receive power until the temperature has decreased below the defined safe level. It is the housekeeping task that updates the temperature sensor, so the minimum duration in this mode is defined by the sampling interval of the housekeeping task.

NOTE: It is not recommended to set the housekeeping sample interval higher than 30 seconds. If it is set much higher than this, the power amplifier may reach too high temperatures without the housekeeping task noticing. The default interval is 10 seconds.

To set the desired over temperature protection level, use the ‘max_temp’ variable in the configuration structure. It may be set to any positive value between 1 and 255 degrees. The special value 0, turns off the protection completely.

NOTE: It is not recommended to turn over temperature protection off, unless it is believed that the temperature sensor returns unreliable data. If this is the case, the radio my appear to be in constant over temperature protection. In this event it is still possible to send a new configuration to the radio and store it in the flash memory, which has temperature protection disabled.

Command and Data Interface

The I²C interface is used for commanding NanoCom and for receiving housekeeping and status messages. NanoCom operates on the I²C bus in multi-master node, that is either as "slave receiver" or as "master transmitter". A “frame” is defined as a single I²C transmission of any number of bytes between a START and STOP condition (REPEATED_START may be used also).

In a multi-master I²C network, a controller should only enter master-mode (set the clock output) when it wants to send something to another node. Otherwise it should be in slave-receive mode and wait for incoming messages, for example from the radio. This avoids constantly polling the radio for incoming data in single master-mode only I²C networks, thus lowering the total I2C bus usage, message latency and jitter.

This mode of operation resembles a typical computer network, and therefore each message is prepended with a packet-header, in order to see its source and destination.

CSP Packet Header

NanoCom implements a network-layer protocol called Cubesat Space Protocol. This protocol defines a simple 32-bit network header, in which the address and destination port must be set correctly for NanoCom to accept the message.

When transmitting any I²C frame, the NanoCom will send a two bytes (big-endian, msb) length field, and then a correct encoded CSP-Header of 4-bytes. A detailed description is found on www.libcsp.org and on http://en.wikipedia.org/wiki/Cubesat_Space_Protocol

The most important fields to notice is the source and destination, which is used as an OSI layer 3 network header for correct routing of messages, and the port numbers which are used as OSI layer 4 transport header.

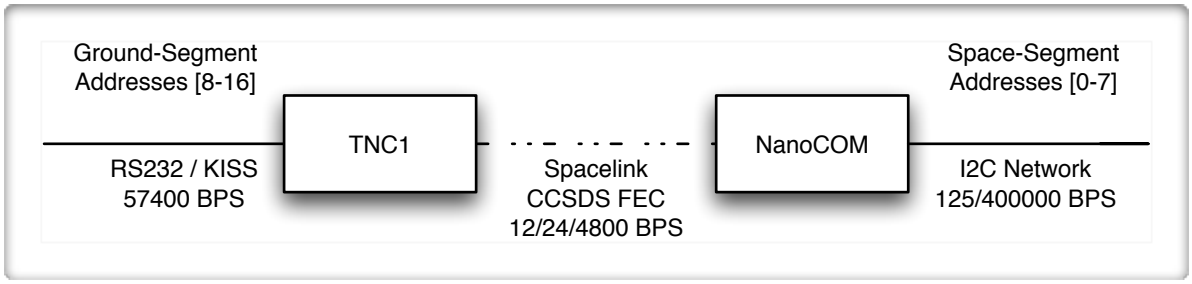
Each node will have an address and a set of port numbers it is listening on.

Routing traffic with the NanoCom radio

The CSP header defines a 5 bit network address which gives up to 32 nodes on the network with addresses from 0 to 31. These addresses has been divided into a space-segment and a ground-segment. These segments are bridged together by the NanoCom radio and Gomspace TNC devices. The default routing configuration is to divide the network such that all addresses 0 to 7 is space-segment, and addresses 8-15 are ground-segment. (The upper 16 addresses are unused)

So in order to transmit a message from the satellite to the ground station, any I²C node can send a CSP packet with a destination address from 8-15 to the NanoCom I²C address. The NanoCom will then forward this packet to the radio interface, based on the routing table.

In order to transmit a message from the ground-segment and back to the satellite, a CSP packet with a destination address of 0-7 must be sent to the TNC1 device over the KISS serial interface. This is also shown in the figure below:



The advantage of using the NanoCOM device as a router, is that any node on the I²C bus can accept a message from the ground-segment directly, thereby bypassing the on-board computer, for direct sub-system to sub-system communication. This facilitates both the use of a centralized and decentralized architecture.

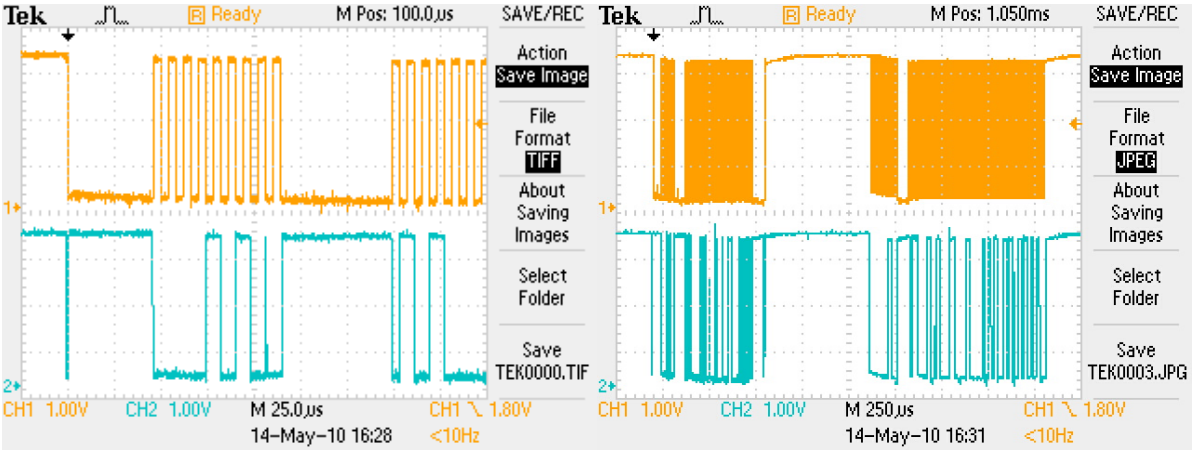
CSP header and NanoCom data is Big-Endian

Since CSP has been designed to work with a variety of different processors and architectures, all nodes on the network has to agree on a byte-order. The byte order chosen is Big-Endian, which is also used in TCP/IP and other networks. It is often just referred to as Network-byte-order.

If your microprocessor is a Little-Endian platform, you need to byte-swap all incoming data-types larger than one byte. This means that a uint16_t, which is two-bytes, needs to be swapped from big-endian to little-endian before it can be used as a regular 16-bit integer. This of course introduces a bit of processing overhead on the network-code, but it ensures that all nodes can communicate.

I2C Examples:

Two examples of the I2C communication is shown below, left: Start of I2C frame to the NanoCom radio. Right: Full CSP ping request and reply.



Notice in the picture to the left, that the I2C address is binary 0000101 which is only 7 bits, the 8'th bit is a 0 which means I2C Write. The following 9'th bit is the acknowledge bit from the NanoCom. Also notice that there are some delays between the start condition and the first byte, and between the address and the data-bytes. This is normal and is caused by the execution speed of the software that must prepare the data for transmission. (This is called clock stretching and may be done both by master and slave when they are not ready).

Command Overview

The following table shows a list of commands accepted by the NanoCom radio. The table should be read as follows: First a mnemonic, or command name is given. This corresponds directly to a specific CSP port number to which the request must be transmitted. For each port, a request and a reply is specified. The row is split into two, the upper-row is the request, and the lower is the reply. For both, a data-type, a data-size and a description is given. Some ports take arguments as simple data-types, some accept complex structures. Below the table, a list of data-structures is given.

This interface is subject to change, and may be further extended and/or improved in the future. Always refer to the specific documentation version delivered with your NanoCom radio.

Mnemonic	Port	Request/Reply Data	Bytes	Description
GET_CONF	7	empty	0	Empty request
		com_config_t;	struct	Full radio configuration (see below)
SET_CONF	7	com_config_t;	struct	Full radio configuration (see below)
		none	n/a	No reply, can be verified with a GET_CONF
GET_STATUS	8	empty	0	Empty request
		com_status_t;	struct	Full radio status (see below)
GET_LOG_HK	9	empty	0	Empty request
		log_hk_t[8];	10*8	Array of housekeeping data elements
GET_LOG_RSSI	10	empty	0	Empty request
		log_rssi_t[10];	8*10	Array of RSSI log elements (see below)
SET_CONF_RESTORE	14	empty	0	Empty request, will restore factory config
		none	n/a	No reply, can be verified with a GET_CONF
SET_TX_INHIBIT	15	uint8_t	1	Single byte 1 = TX inhibited, 0 = TX allowed
		uint8_t	1	Echo of request if OK.

Detailed Command Specification

The different NanoCom data-structures are explained in detail below. The data-types are defined in the accompanying nanocom.h header.

GET/SET_CONF

The NanoCom device listens on port 7 for incoming requests to either GET or SET the configuration. If the request is empty, it is interpreted as a GET command, if it contains a valid com_config_t data-structure, it is seen as a SET command. The data-structure is formatted like this:

```
/** Shared data-structures */
typedef struct __attribute__((packed)) {
        uint8_t do_rs;                  // 1 = turn on reed-solomon FEC, 0 = off
        uint8_t do_random;              // 1 = turn on CCSDS randomization, 0 = off
        uint8_t do_viterbi;             // 1 = turn on viterbi K=7 FEC, 0 = off
        uint8_t tx_baud;                // TX baud, [12,24,48]
        uint8_t rx_baud;                // RX baud, [12,24,48]
        int16_t tx_max_temp;            // Automatic shutoff TX at temp deg C [0 = dont check temp]
        uint16_t preamble_length;       // Time in miliseconds
        uint8_t morse_enable;           // Turn morse ON/OFF, 0 = off, other = on
        uint8_t morse_mode;             // CW = 0, CW+FM=1, FM = 2;
        uint8_t morse_cycle;            // Turn parameter cycling 1 = on, 0 = off
        uint8_t morse_en_voltage;       // Enable voltage output
        uint8_t morse_en_rx_count;      // Enable RX count output
        uint8_t morse_en_tx_count;      // Enable TX count output
        uint8_t morse_en_temp_a;        // Enable Temp A output
        uint8_t morse_en_temp_b;        // Enable Temp B output
        uint8_t morse_en_rssi;          // Enable last RSSI output
        uint8_t morse_en_rf_err;        // Enable last RF error output
        uint16_t morse_inter_delay;     // Delay between beacons, in seconds.
        uint16_t morse_pospone;         // Delay before first beacon after activity
        uint8_t morse_wpm;              // WPM after paris standard, dfl = 20
        uint8_t morse_text[20];         // Text to morse, terminate with '\0'
        uint16_t morse_bat_level;       // Minimum battery level, int 500 = 5.00V
        uint16_t hk_interval;           // Interval between HK sampling in seconds
} nanocom_conf_t;
```

Notice that some of the datatypes are two bytes and must be transmitted big-endian byte order

When sending the CONFIG_SET command the values are written to the run-time configuration and saved to the non-volatile FLASH memory, and restored upon reboot.

GET_STATUS

Transmitting any message to port 8 is replied with a data structure which looks like this:

```
typedef struct __attribute__((packed)) {
        uint32_t bit_corr_tot;          // Total bits corrected by viterbi
        uint32_t byte_corr_tot;         // Total bytes corrected by reed-solomon
        uint32_t rx;                    // Total packets detected
        uint32_t rx_err;                // Total packets with error
        uint32_t tx;                    // Total packets transmitted
        int16_t last_temp_a;            // Last temperature A in [C]
        int16_t last_temp_b;            // Last temperature B in [C]
        int16_t last_rssi;              // Last detected RSSI [dBm]
        int16_t last_rferr;             // Last detected RF-error [Hz]
```

```
        uint16_t last_batt_volt;        // Last sampled battery voltage [mV/10]
        uint16_t last_txcurrent;        // Last TX current [mA]
        uint32_t bootcount;             // Total bootcount
} nanocom_data_t;
```

Again, all fields of the data-structure is converted to Big-endian before transmitting on the network.

The battery voltage is in mV divided by 10, so 7.60 Volt = 760. The signed integers are two's complement.

GET_LOG_HK

The NanoCom Radio records Housekeeping information at a configurable interval. This means that in addition to the "current" status information obtained though the GET_STATUS command, a set of historical samples can be obtained from the GET_LOG_HK command. The following datatype is repeated 8 times in a GET_LOG_HK reply.

```
typedef struct __attribute__((packed)) {
        uint32_t time;                  // CPU Timestamp (processor Ticks since boot)
        int16_t temp_a;                 // Temperature of A in [C]
        int16_t temp_b;                 // Temperature of B in [C]
        uint16_t batt_volt;             // Battery voltage, int 712 = 7.12 V
} nanocom_hk_t;
```

This can be used to get an impression of the temperatures of the NanoCom and the Battery voltage over time, without the need of an on-board computer to do data-logging.

GET_LOG_RSSI

Every time the receiver detects the beginning of a frame, it logs the RSSI and RF-error to a ring-buffer. The contents of this buffer can be obtained by the GET_LOG_RSSI command. The answer will be 10 log_rssi_t types:

```
typedef struct __attribute__((packed)) {
        uint32_t time;                  // CPU Timestamp (processor Ticks since boot)
        int16_t rssi;                   // Measured RSSI [dBm]
        int16_t rferr;                  // Measured RFerr [Hz]
} nanocom_rssi_t;
```

SET_CONF_RESTORE

Send a packet to this port in order to restore the NanoCom configuration to "factory" preset. The packet content is ignored, so an empty packet is ok.

SET_TX_INHIBIT

Send a single byte to this port. Sending a 1 will turn the transmitter off and prevent it from turning on. Sending any other value than 1 will allow the TX to be turned on again. Note: This value is NOT stored in FLASH memory, so the TX inhibition is only in effect until disabled by command or a reboot of the NanoCom. The default value at boot is zero.

Using the NanoCom Client library

The above data-types are defined in the nanocom.h, they are used directly to pack the data for the space-link frames, but also as a part of the C-library. The function prototypes for the library are as follows:

```
/**
 * Send a SET_CONF message to a NanoCom or TNC
 * @param config pointer to config structure
 * @param node address of nanocom/tnc
 * @return result of csp_transaction
 */
int com_set_conf(nanocom_conf_t * config, uint8_t node);

/**
 * Send a GET_CONF message to a NanoCom or TNC,
 * and wait for a response before returning.
 * The timeout is 5 seconds.
 * @param config pointer to where retrieved config will be stored
 * @param node address of nanocom/tnc
 * @return result of csp_transaction
 */
int com_get_conf(nanocom_conf_t * config, uint8_t node);

/**
 * Send a SET_CONF_RESTORE message
 * This will delete the current stored config from the NanoCom/TNC
 * and return the device to factory settings. This is useful in the
 * event that an invalid configuration was sent to the node.
 * @param node address of nanocom/tnc
 * @return result of csp_transaction
 */
int com_restore_conf(uint8_t node);

/**
 * Send a GET_STATUS message
 * And wait for a response. Timeout is 5 seconds.
 * @param data pointer to where data will be stored
 * @param node address of nanocom/tnc
 * @return result of csp_transaction
 */
int com_get_status(nanocom_data_t * data, uint8_t node);

/**
 * Send a GET_RSSI message
 * And wait for a response. Timeout is 5 seconds
 * @param data pointer to where data will be stored, should be array of nanocom_rssi_t (with size >= 10)
 * @param count pointer to where the number of com_rssi_t structs received is stored
 * @param node address of nanocom/tnc
 * @return result of csp_transaction
 */
int com_get_log_rssi(nanocom_rssi_t * data, uint8_t * count, uint8_t node);

/**
 * Send a GET_HK message
 * And wait for a response. Timeout is 5 seconds
 * @param data pointer to where data will be stored (array size >= 8)
 * @param count poitner to number of hk entries received
```

```
 * @param node address of nanocom/tnc
 * @return result of csp_transaction
 */
int com_get_hk(nanocom_hk_t * data, uint8_t * count, uint8_t node);
```

In order to compile this code, you'll need the following:

1) nanocom.h - Header file

2) nanocom.c - Function implementation

3) LibCSP (Opensource CSP implementation for Linux, Windows, and FreeRTOS, available at http://libcsp.org)

This code could be implemented as a part of your on-board or ground-station software to make commanding and retrieval of data more simple.

Using the command line utility

As a further extension of the NanoCom interface, there is also a command line utility available. This command system works by interpreting commands given on the serial port of the NanoCom and calling the NanoCom C-library functions. This means that most functions of the NanoCom are available directly on the serial port by typing commands. This is great for debugging and getting started with the NanoCom or TNC system.

The GomSpace Shell (gosh) currently runs on NanoMind, NanoCom, NanoCam, TNC1 and even on the ground-station PC in a small application called CSP-Term which is freely available. This means that the NanoCom commands can be executed from several different sources, depending on where you have access to a gosh shell. When issuing a command from the Ground-station, TNC or OBC, the underlying CSP-protocol will ensure to route the commands correctly to the NanoCom and back again.

By typing ‘help’, in any gosh shell, a list of commands will be printed. The NanoCom commands are all prefixed with the word ‘com’. In order to list the ‘com’ commands type ‘com <tab>‘.

```
com # com
  node                Set node
  gs                  Get status
  rssi                Get last RSSI readings
  hk                  Get last HK readings
  cal                 Get calibration data
  do                  set DAC offset
  cg                  Get configuration
  cs                  Set configuration
  cr                  Restore configuration
  preamble            Configure Preamble in [ms]
  baudrx              Configure baud RX
  baudtx              Configure baud TX
  menable             Configure Morse Enable
  minterval           Configure Morse interval
  mwpm                Configure Morse wpm
  mbatt               Configure Morse battery level
  mpone               Configure Morse postpone
  vit                 Configure do Viterbi
  rs                  Configure do Reed-Solomon
  random              Configure do randomization
```

The cmd client is configured to use the node 5 as a default, unless running on the TNC where it will be 9 as default. Otherwise the active node can be changed using the ‘com node x’ command.

Here are a few command examples:

```
com # com gs
Bits corrected total                          0
Bytes corrected total                         8
RX packets                                    21
RX checksum errors                            21
TX packets                                    0
Freq. Error                                   3529
Last RSSI                                     -125
Last A temp                                   28
Last B temp                                   28
Last TX current                               0
Last Battery Voltage                          0
Bootcount                                     69

com # com hk
Time 462000,    Temp A 28,      Temp B 28,      BattV 0
Time 461000,    Temp A 28,      Temp B 28,      BattV 0
```

```
Time 460000,    Temp A 28,      Temp B 28,      BattV 0
Time 459000,    Temp A 28,      Temp B 28,      BattV 0
Time 458000,    Temp A 28,      Temp B 28,      BattV 0
Time 457000,    Temp A 28,      Temp B 28,      BattV 0
Time 456000,    Temp A 28,      Temp B 28,      BattV 0
Time 455000,    Temp A 28,      Temp B 28,      BattV 0

com # com cg
FEC: rs 1, random 1, viterbi 0
BAUD: rx 24, tx 24, preamble 50
MORSE: enable: 1, delay 20, pospone 120, wpm 20, batt level 500, text NANOCOM
HK: interval 10
```

In order to change the actual configuration, the best way is to first run a 'com cg' to retrieve the current configuration from the device. Then make the change using one of the config commands shown in the help-output, and finally committing the change to the nanocom by issuing the 'com cs' command. Here is an example of how to change the baud-rate of the nanocom.

```
com # com cg
FEC: rs 1, random 1, viterbi 0
BAUD: rx 24, tx 24, preamble 50
MORSE: enable: 1, delay 20, pospone 120, wpm 20, batt level 500, text NANOCOM
HK: interval 10
com # com baudrx 48
Setting RX baud to 48
FEC: rs 1, random 1, viterbi 0
BAUD: rx 48, tx 24, preamble 50
MORSE: enable: 1, delay 20, pospone 120, wpm 20, batt level 500, text NANOCOM
HK: interval 10
com # com baudtx 48
Setting TX baud to 48
FEC: rs 1, random 1, viterbi 0
BAUD: rx 48, tx 48, preamble 50
MORSE: enable: 1, delay 20, pospone 120, wpm 20, batt level 500, text NANOCOM
HK: interval 10
com # com cs
com # COM_CONF_SAVE: Saving config from RAM to FLASH
COM_CONF_LOAD: Loading config from FLASH to RAM
        Watchdog reset: 8
        CPU clock :  64000000 Hz
        PBA clock :  16000000 Hz

Enter 'help' or press <tab> for a list of commands.
```

Notice here that after the NanoCom retrieves the new configuration it stores it to FLASH and then reboots in order to reload the configuration and start using the new baud-rate.

Automatic system restore

In case the NanoCom does not talk with anybody for a period of time, the system will reboot itself. If this has happened several times, the system assumes that it is malfunctioning and will reset to factory defaults. This function can be adjusted by setting the intervals or completely disabling it. (must be specified compile time). In order to prevent the system from rebooting and thereby restoring the default configuration, a ping or any command must be sent from any on-board-computer, or from the ground-station, directly to the NanoCom subsystem (CSP id 0x05). In order to reset the reboot counter, and prevent a config restore, the SET_CONF command must be run at least once per day in order to maintain the configuration.

Using the additional debug commands

The NanoCom has a set of additional commands, which are not available on the I2C or CSP interface but only on the serial debug interface, hence the name ‘comdebug’

```
com # comdebug
'comdebug' contains sub-commands:
  bootcause           Display Bootcause
  calibrate_rssi      Run RSSI calibration routine
  calibrate_rferr     Run RF-Error calibration routine
  qostest             Run QoS test
  up                  Key up transmitter
  dn                  Key down transmitter
  baud                Set baud rate [XX]
  restore             Configuration restore
  store               Store configuration to flash
  store_factory       Store configuration to as new factory settings
  modrun10            Modem send 0101010
  modrun1             Modem send 1111111
  modrun0             Modem send 0000000
```

These commands are for debugging only and should only be used if the consequences are understood.

Most of these commands are self explanatory, however a few comments should be made to the store_factory command. This command is used to write the current config from RAM to a reserved factory settings flash page. Before calling this command, all settings should be set to the desired factory setting and the RSSI and RF-Error calibration routines must have been successfully completed. This operation is done in the lab at gomspace during checkout of the NanoCom, and not necessary for customers to run.

A few other useful commands are the ‘up’, ‘dn’ and ‘modrun’ commands. These can be used to key up the transmitter and transmit a tone of either 1’s, 0’s or a sequence of 1’s and 0’s. After running the ‘modrun’ command, a reset is needed to stop the tone.

Note: Do not key up the transmitter without carefully monitoring the temperature of the power amplifier.

Using the GOSH commands

Apart from the NanoCom specific commands detailed above, the gosh shell also has a set of very useful debugging functions, which will be explained below:

```
com # help
  comdebug            COM debug commands
  ping                Ping subsystem
  rps                 Remote ps
  memfree             Memory free
  buffree             Buffers free
  reboot              Rebout subsystem
  uptime              Uptime of subsystem
  cmp                 CSP management protocol
  route               Show routing table
  ifc                 Show interfaces
  conn                Show connection table
  debug               Toggle CSP debug levels ON/OFF
  com                 COM subsystem
  i2cdump             Dump PCA9665 registers
  history             Show command history
  help                Show this help and command description
  clear               Clear screen
  echo                Display a line of text
  args                Print command line arguments
  sleep               Sleep for a specified period of time
  version             Show console version and compile time
  watch               Run command at defined intervals
  reset               Reset the CPU
  tdebug              Toggle driver debug
  ps                  List tasks
  stats               Get runtime stats
```

The ‘com’ and ‘comdebug’ commands has already been explained. The remaining commands can be used to debug the network and the microprocessor. The ping command is very useful to test network connectivity and the radio itself.

In order to send a message from node 5 (COM) to 9 (TNC), a ping command can be executed on node 5 like this: ‘ping 9 <timeout> <packet-size>’, if no timeout or packet size is specified the default is 5 seconds and 1 byte.

Another helpful command is the ‘cmp ident’, which will give a result similar to this:

```
com # cmp ident 5 1000
Hostname: com
Model:    GomSpace U482
Revision: 38cf7bc
Date:     Jan 12 2012
Time:     15:52:37
```

This shows the identity of the system, and can be used on any CSP node. Finally the ‘uptime’, ‘buffree’, ‘reboot’ and ‘memfree’ are also part of the commands that can be used on any CSP node.

RF Transmission Format

NanoCom uses a variable frame format to transmit and receive packets over the spacelink.

ASM	Golay	(optionally) FEC Encoded Data
4 bytes	3 bytes	0 - 255 bytes

ASM

The first 4 bytes of each packet is an Attached Sync Marker containing 0xC3AA6655 used to synchronize the beginning of each packet. The ASM word is not FEC encoded, but the microcontroller accepts up to 3 bit errors in the correlator.

Golay

The next 3 bytes are the golay field which defines the FEC type and length of the frame. This field uses 12 bits to represent the length and data field, and 12 bit data-check parity. The golay decoder is able to correct up to 3-bit errors in the string of 24-bits, making it fairly robust. The three bytes are defined in a big-endian order like this:

12-bit parity
1 bit reserved
1 bit reed-solomon enabled flag
1 bit randomization enabled flag
1 bit viterbi enabled flag
8 bit payload length field

The purpose of the golay field is to: 1) specify the length of the remainder of the frame, which enables the use of very short frames for commands / acks, and long frames for data. 2) Specify which FEC code is used on a per-packet basis, which eliminates the need for the receiver to know this in advance and makes the receiver more versatile. 3) To provide additional packet synchronization confidence. Only if the sync-word is detected (with up to 3 bit errors) and the golay decoder returns a success (able to correct up to 3 bit-errors), the frame is actually received. This ensures a quick detection of false positives from the ASM correlator, which gives a lower packet loss probability.

Forward Error Correction

If the optional Viterbi, Reed solomon or Randomization is enabled, this only applies for the Data field.

The Viterbi decoder is excellent at correcting single bit-errors if they are spaced evenly throughout the frame. If 4 or more bit-errors occur in a row, the K=7 viterbi decoder does not have enough information in the preceding bits to correct it. This will then yield a block error, which may take some bits to recover from. However the Reed-Solomon code is excellent at correcting block errors, since it works on a byte-level and not bit-level. The (223,255) code is able to correct 16 bytes in a frame. If a shorter than 223 byte frame is used, the frame is padded with “virtual zeros” just before the decoder. Finally the CCSDS randomization can be enabled or disabled. It does not have a big effect on an FM channel, but ensures there are a close to 50% distribution of 0’s and 1’s in the raw data-stream. This is particularly helpful for bit-synchronization.

The Reed-Solomon code served a dual purpose since it both corrects errors but also detects if the frame was not correctable. This is therefore used as a final approval of the checksum of a frame. It is therefore recommended to always use the Reed-Solomon coder. However if the it is turned off, the 32 RS bytes are replaced with a simpler 4 byte CRC-32 (CCITT).

If you do not use the Gomspace TNC1 device at your ground-station, and wish to implement the FEC en/decoder yourself, please contact Gomspace for specific information about the Viterbi symbol table and encoder polynomial.

Preamble

When initiating transmission, a preamble of alternating 101010101... must be sent for at least 25ms. NanoCom uses 50 ms preamble as default but this can be altered in the factory configuration.

Maximum transfer unit (MTU)

The MTU depends on the amount of FEC added to the frame

FEC	Viterbi = 0	Viterbi = 1
RS = 0	247	118
RS = 1	219	90

Medium Access Control

The NanoCom uses a Half-Duplex UHF link as its physical media and as with any half-duplex system, it has a risk of signal collisions. These collisions are avoided by a technique known as CSMA (Carrier Sense Multiple Access), which if often also referred to as “Listen Before Talk”. This greatly reduces the number of collisions but does not eliminate the risk completely. If both radios decides to transmit at the same time they will not have heard the other end transmitting and both messages will be lost. In order to mitigate this problem, a few Collision Avoidance techniques have been implemented.

Controlled Initiative

The first and most simple method to avoid collision is to define who has the transmission initiative. This technique works well if the satellite does not transmit unless requested to do so by the ground station operator. This method is very common for cubesats and also recommended by gomspace. However, no rule without an exception. It is much easier to locate the satellite if it has a tracking beacon, or some signal indicating its presence. The NanoCom features an automatic beacon that can start after power-up and help locate the satellite. Because this beacon could also cause a collision, it is only turned on at given time intervals, and will shut off completely (for a period) if a packet is received. Since the beacon is not a data-packet and thus not detected by the TNC, the ground station “Listen Before Talk” function does not work. As a consequence, some collisions with the morse beacon should be expected during the first contact. A recommended method to overcome this problem is to have automatic ping packets being sent out from the ground station at an interval about 4 times shorter than the morse interval. At the first ping reply the ground station

software will know that the satellite is now in range, and that the morse beacon is turned off. Hence the ground station now has the initiative to complete the planned tasks for the pass.

Inter Packet Delay

During two-way communication with the satellite, there is a possibility that either the ground station or satellite may need to transmit two or more frames in succession. In fact, this is the preferred method of retrieving data, since it reduces the overhead of the link. If some of the first incoming frames results in some outgoing frames being buffered at the radio, it should typically start transmission of these as soon as the link is available. However, after each received packet there is a small gap where the synchronizer has not seen the next packet yet, and the link may appear as free. This problem is solved by introducing a fixed waiting time after each RX frame before declaring the link as free. The current waiting time is 12 bytes, which gives enough time for the synchronizer to lock on the next frame, if they were transmitted in succession.

Minimum Listen Time

Another method used in order to avoid collisions takes the assumption that there are only two transceivers in the system. If both transmitters implement a minimum listen time before they are allowed to key-up, they should know that the other transmitter had it's chance to occupy the link if it so wanted. In other words, it is illegal for at transmitter to turn off after sending a packet, and then suddenly turning on again in order to send a new frame, if it has not waited the minimum listen time. If both transmitters employ this technique, the neighbor radio knows for sure, that when it has just received a message and there has not been detected a new frame 12 bytes later, the link will be available for a certain period. This period should be set to be long enough so that the receiver will have time to detect the transmitter. Typically this is related to the preamble period and the frame length. As default this is set to 150 ms. If you require a longer listen time (maybe due to a longer preamble period) please use the 'conf tx_guard' command to change this setting.

Maximum TX ON period

There is a safety feature built into the transmitter, which will ensure that the TX is only turned on for a maximum of 10 seconds (default). After this period has elapsed, the TX will automatically turn off, and remain turned off for the "minimum listen time" defined with the 'tx_guard' setting. This ensures that even if the satellite is constantly transmitting (the TX buffer is never empty), there is reserved a short timeslot for uplink commands. The default period of 10 seconds can be changed with the 'conf tx_max_time' command.

TDMA

Even with the "Controlled Initiative" approach and the extra Collision Avoidance techniques, problems can occur which may require additional collision mitigation. This technique is presented here as a possible scenario, not as a part of the NanoCom standard software. There are several possibilities of how to implement TDMA, which will depend on the mission and the satellite operating modes.

In the event that the radio link should not close with a sufficient link margin, and one of the receivers does not have a 100% detection rate. The "Listen Before Talk" mitigation technique will not work 100%. This means that the collision avoidance will only relies on the "Controlled Initiative", which may be fine for most missions, but another very simple alternative is also available.

The technique is called TDMA (Time Division Multiple Access) which essentially splits a half-duplex link into two separate channels (up/down) which take turns at being active. This is very effective at avoiding collisions since the channels is reserved for the transmitter, even if does not have any pending traffic. The downside is that it may be a waste of bandwidth if the link is idle.

The TDMA relies on an agreement between the two transmitters and some synchronization signal. Let's consider the scenario where the satellite is allowed to initiate transmissions on even- and the ground on odd seconds. This then requires an exact time (around 1/10 of a second) synchronization between the ground and the satellite. Something that can be done, but could be rather difficult. However if time synchronization is possible to only within seconds, the agreement could be changed to even ten's of seconds instead.

Another option if we assume that the uplink has a higher link-margin than the downlink, is that the ground station sends a CTS (Clear To Send) frame to the satellite with a specified time period where the ground station will remain listening.

Even a third method if the satellite is operating in telemetry mode utilizing a large percentage of the link, the satellite could occasionally send an End of Messages - Clear To Send packet, with a time period where the ground station would be allowed to uplink.

Operation and Handling

Warnings:

- The NanoPower system employs components based on FETs and therefore requires anti-static handling precautions to be observed. Do not touch or handle the product without proper grounding!

Customization Options

As GomSpace realizes that different applications place different requirements to a communications system, options to be agreed upon time of order placement include:

- All pin-connections indicated with red dots
- Transmitter supply voltage
- Conformal coating using NASA approved CV-1152 silicone coating (at an extra cost)
- More options may be available at the customers request
- Specific software changes (at an extra cost)

Quality Assembly

GomSpace space hardware is hand-assembled in a procedure where all parts are cleaned with IPA and then soldered in an anti-static environment to “IPC-A-610 Class 3” specifications. All solder-work is done under a microscope with tin-lead 63/37 using rosin flux. All solder joints are re-checked for class 3 compliance and the PCB is finally cleaned with IPA and ready for testing.

Materials

The circuit board of a NanoCom is the single most critical part of the system and is not allowed to suffer a mechanical failure. Therefore, we use polyamide P97 as the base material in the PCB because of its excellent temperature tolerance resulting in very little thermal expansion and hence very little stress on the copper and the via platings when the board is subjected to even broad temperature cycles. This also means, that the soldering process does not effect the peel-strength of the tracks and pads resulting in the best possible mechanical base for the electronic components.

The surface of the conducting areas is covered in tin/lead for optimum solderability with allowance for small differences thermal expansion without the risk of exceeding the maximum tensile strength of joins. Lead-free solder is never allowed in a GomSpace flight-grade product.

The EMI shield on the transceiver is made from 0.2mm tin-plated steel in order to ensure proper RF shielding and providing a level of radiation shielding as well. The shield also helps to serve as heat sink and a thermal radiator transport heat away from the transmitter power electronics.

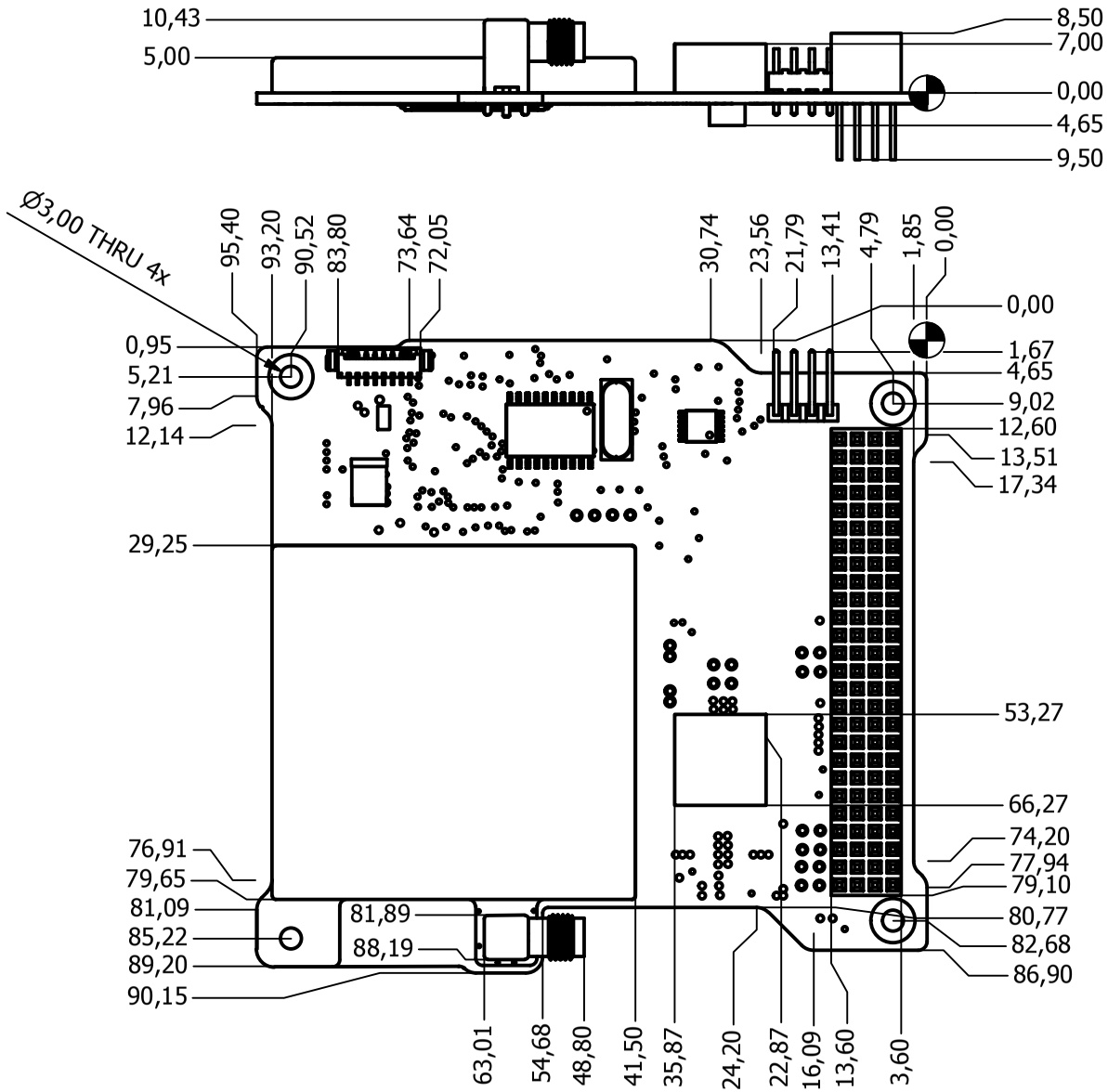
Revision history

Revision	Date	Changes performed
5.0	August 12, 2013	Main block diagram, page 6, pin numbers corrected from H1-27 and H1-28 to H2-27 and H2-28.

Physical Dimensions

See last page. Dimensions are given in mm.

PCB-type is glass/polyamide, 6 layer, tin-lead fused surface, 1.60 mm thick. Mass: 75g.



Designed by	Checked by	Approved by	Date		Date	
KKL					12-10-2011	

GomSpace ApS Niels Jernes Vej 10 DK-9220 Aalborg East Denmark	NanoCom U482C		
	PID: 0037 CID: 0977	Edition	Sheet 1 / 1