NanoCam
C1U

Manual

Camera Payload for nano-satellites

Release 5.0.8

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

Document Details

Title: NanoCam C1U
Reference: 1018167
Revision: 5.0.8
Date: 06 April 2022

Confidentiality Notice

Table of Contents

1. Introduction

The GomSpace NanoCam C1U is a flexible and modular system to rapidly implement tailored imaging systems based on customer requirements. The C1U is an off-the shelf configuration of the NanoCam system consisting of lens, lens table, image acquisition and processing board and software.

NanoCam C1U has been designed to be implementable in a standard 1U CubeSat structure together with GomSpace's on-board computers, attitude control system, radio transceiver and power products to allow low cost Earth observation using CubeSats.

This manual describes the NanoCam C1U firmware version 5.0.8:

- Command Interface
- CSP Client API

1.1 Unpacking and Handling Precautions

Warning: The NanoCam C1U is an ESD sensitive device. 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 parts, as these are 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.

2. General Operation

The NanoCam can be operated through the Cubesat Space Protocol (CSP) over CAN, I^2C or serial port. This integrates well with other GomSpace products, and GomSpace provides a client library for commanding the C1U from other systems running CSP. Refer to the *Client API* page for documentation of the client library API.

For integration and testing purposes, a command line interface is also available as explained in *Command Interface*. The command line interface uses the client API internally, so command names generally reflects the API function names.

The block diagram below shows the different stages of images through the camera software.

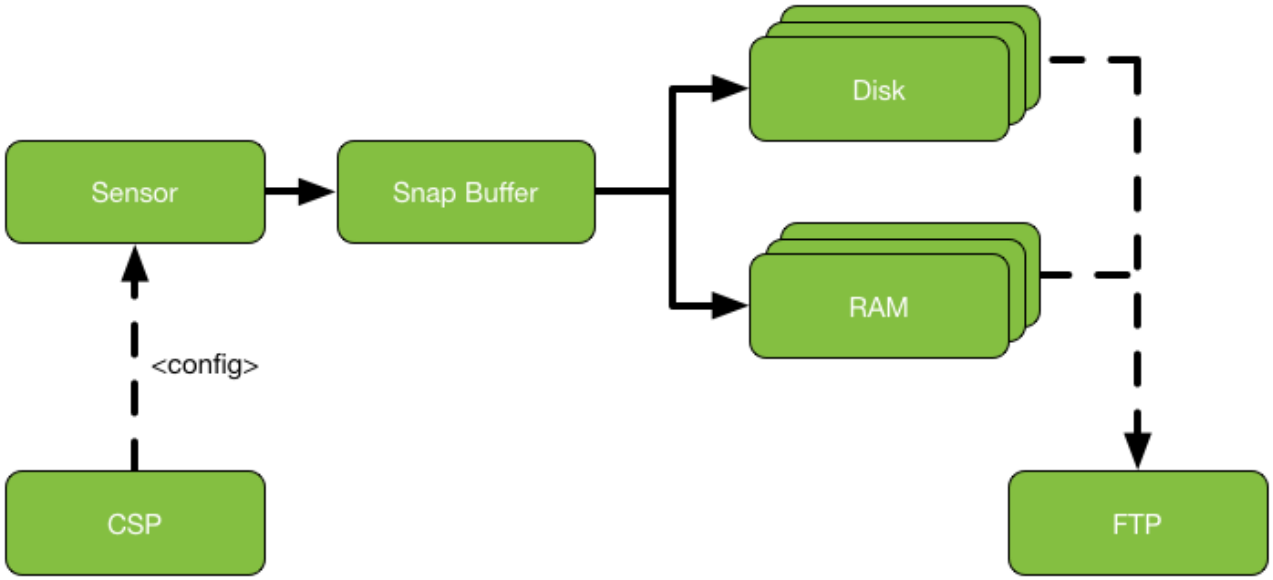


Fig. 2.1: NanoCam C1U Block Diagram

Images are captured using the *snap* command. By default, this captures a single full-size image to an in-memory snap buffer and returns image information such as histogram and average brightness for each color channel. The operator can then chose to store this image to flash memory and/or RAM using the *store* command, or adjust image parameters and capture a new image.

This mode of operation is most useful during integration and early operations, when a human operator is present to assess the quality of the captured images. It also requires the spacecraft to be in range of the ground station. For automatic mode, the snap command can be configured to capture a sequence of images with a fixed interval and automatically store the captured images to nonvolatile memory.

Stored images can be downloaded from RAM or flash using the GomSpace File Transfer Protocol. Use of the FTP through CSP-Term is explained in *Transferring a Picture using FTP*.

It is possible to configure the desired exposure time and target brightness of the captured images. The NanoCam will then attempt to automatically adjust the sensor shutter width and gain stages to match these values. Image parameters can also be adjusted by manually setting the sensor registers directly, but this should only be used if the built-in algorithms do not produce an acceptable result.

3. Command Interface

For testing and debugging purposes, the C1U can be operated using the GomSpace Shell (GOSH) interface. The console interface is available on the DRXD/DTXD pins in the J3 connector (refer to the datasheet for connector placement). The serial connection is 8 databits, 1 stopbit, no parity at 115200 baud. On Linux, the serial connection can be opened using the minicom or tio tools.

The C1U runs the GomSpace Linux distribution and will boot to a Linux command shell. To open the GOSH interface, simply run the `gosh` command:

```
root@nanocam:~# gosh
connected to 127.0.0.1:5001
1568287533.971040 I nanocam: working directory to /mnt/data
1568287533.974815 W param: Failed to load table [config] (id=0) - initialized to default␣
↪values
1568287533.975336 W param: Failed to load table [image] (id=1) - initialized to default␣
↪values
1568287533.978312 W param: Failed to load table [tag] (id=3) - initialized to default␣
↪values
1568287533.979306 I nanocam: csp address 6
1568287533.994808 I nanocam: rtable: 28/5 KISS, 5/5 I2C, 0/0 CAN
nanocam #
```

You can now execute commands in the GOSH command interpreter. An example command is `cmp ident` which lists identification info of the current subsystem:

```
nanocam # cmp ident
Hostname: Nanocam
Model:    C1U
Revision: 5.0.0
Date:     Sep 19 2019
Time:     11:03:13
```

A list of available commands can be shown by running the `help` command in GOSH. Pressing the *tab* key automatically completes commands and subcommands, and shows a usage string. Commands specific to the C1U are assembled under the `cam` command group. These commands are also available in CSP-Term for operating the camera via CSP:

```
nanocam # cam
  client: NanoCam C1U
    node                Set CAM address in host table
    timeout             Set timeout
    format              Set store format (0=raw, 1=bmp, 2=jpg, 3=dng, 4=raw10)
    snap                Snap picture
    store               Store picture
    read                Read sensor register
    write               Write sensor register
    list                List RAM pictures
    flush               Flush RAM pictures
    focus               Run focus assist algorithm
    recoverfs           Recreate data file system
    peek                Dump picture from RAM to ./cam_hexdump.bin via 'cmp peek'
```

Note: All of the `cam` commands run over loopback CSP using GOSH command wrappers around the *Client API*

3.1 Global Settings

The camera command line interface is written such that there are a few global variables that can be set first. The default values of the variables are:

- The camera CSP address is set to 6.
- Timeout is set to 5000 ms.
- Store format is set to JPG.

These values can be set using the `cam node <node>`, `cam timeout <timeout>` and `cam format <format>` commands. So to e.g. set the store format to BMP instead of JPG, run the `cam format bmp` command. To show the current setting, run each command without an argument.

3.2 Taking a Picture

In order to take a picture, the `cam snap` command should be executed. This sends a request immediately to the C1U and waits the timeout amount of milliseconds for a snap reply to come back. A successful snap reply will output the minimum/maximum brightness values and a histogram of the color distribution of the captured buffer. This can be used to assess the quality (e.g. exposure level and color balance) of the picture before storing it.

The C1U implements an optional algorithm to automatically adjust the gain before taking the picture. It does so by adjusting the global-gain parameter to achieve an average brightness of eg. 30%. Passing the -a flag to the snap command will enable the auto-gain algorithm. Note the global-gain parameter will set/overwrite the 3 individual color gains. The command below shows an example of the snap output:

```
nanocam # cam snap -a
Snapping full-size image, timeout 5000
Snap result: 0
Brightness
  All   : 31% (min/max/peak/avg 014/255/087/081)
  Red   : 32% (min/max/peak/avg 015/255/087/082)
  Green : 31% (min/max/peak/avg 014/255/071/080)
  Blue  : 32% (min/max/peak/avg 015/255/087/082)
Histogram
    Min +-------------------------------------------------+
   0.0% |                                                 | 000-015
  10.2% |##################                               | 016-031
  11.8% |####################                             | 032-047
   9.8% |################                                 | 048-063
  16.9% |###################################              | 064-079
  23.5% |################################################ | 080-095
  10.2% |####################                             | 096-111
   5.9% |###############                                  | 112-127
   4.3% |#########                                        | 128-143
   1.6% |#####                                            | 144-159
   0.8% |##                                               | 160-175
   0.8% |##                                               | 176-191
   0.4% |#                                                | 192-207
   0.0% |#                                                | 208-223
   0.0% |                                                 | 224-239
   1.2% |##                                               | 240-255
    Max +-------------------------------------------------+
```

3.3 Saving a Picture

The `snap` command captures the image to the in-memory snap buffer. In order to store the picture to non-volatile storage, use the `cam store <filename>` command. The storage format should already be set by

the `cam format` function. A good practice is to name the filename such that the ending is .raw, .bmp or .jpg. Pictures are stored in the `/mnt/data/images` directory.

The converted picture that is written to file system will be kept in memory until either the next subsystem reboot or the `cam flush` command is executed. This enables direct RAM download, thereby bypassing the permanent storage. If you wish to only perform the picture conversion and store to RAM, use the `cam store` command without a filename.

The store command returns the address and length of the stored image in memory. The example below shows an image stored as test.jpg. After processing, the image is also located in RAM at address 0xAEEFC008 with a size of 601995 bytes:

```
nanocam # cam format jpg
Format set to JPG (2)
nanocam # cam store test.jpg
Result 0
Image 0xaeefc008 601995
```

3.4 Modify Sensor Registers

If you wish to modify the default picture settings, you can use the `cam write <reg> <value>` function to set a register, or `cam read <reg>` to read it back. For example to set register 0x2D to a value of 0x0123 the command would be `cam write 0x2d 0x0123`, or you could omit the 0x and write `cam write 2d 0123`.

Notice that the snap auto-gain algorithm may overwrite the register settings if enabled.

3.5 List Pictures in Memory

Every time the C1U stores a picture, it saves a copy of that picture in its RAM memory. This list of pictures can be acquired using the `cam list` command. It will tell you the memory address and the length in bytes of the picture data.

This information can be used to download the image using the direct memory backend of the GomSpace File Transfer Protocol. If you wish to delete the contents of the memory, send a `cam flush` command or reboot the camera by sending the `reboot` command to the C1U address.

3.6 Focus Assist Routine

The `cam focus` focus routine will take a picture and encode a part of this picture as a JPG and return the resulting compressed size. The theory is that a more detailed picture results in a bigger image size.

This can be used to help focusing the lens by running it continuously with the watch command: `watch 0 cam focus`. Adjust the lens focus until the maximum value is found.

3.7 File System Recovery

Images are stored on the data file system partition, which is available at `/mnt/data`. The `cam recoverfs` command can be used to recreate the file system. Note that this erases ALL images stored on the C1U. If you want to delete all the captured images, running the `ftp rm images/*` command is much faster and a safer option than rebuilding the entire file system.

3.8 Transferring a Picture using FTP

The GomSpace FTP Client, built into CSP-Term, can be used to list, retrieve, upload and delete files on the C1U. This means that after running `cam snap` and `cam store`, it is possible to download the picture directly from the CSP-Term client.

The following FTP commands are available directly from CSP-Term:

```
csp-term # ftp
File Transfer Protocol client
  ls                  list files
  rm                  rm files
  mkfs                make file system
  mkdir               make directory in file system
  rmdir               remove a directory from the file system
  mv                  move files
  cp                  copy files
  zip                 zip file
  unzip               unzip file
  local_zip           zip local file
  local_unzip         unzip local file
  server              set server, chunk size and mode
  upload              Upload url
  download            Download url
  timeout             Set general ftp timeout
```

Before using the FTP, you must configure the FTP client to use the correct server address and port number. The default address of the C1U is 6 and the port number it uses for FTP is 9. This is done using the 'ftp server 6 9' command.

You can then use the `ftp ls /mnt/data/images` command to get a list of images on the data partition. The images directory is created when the first image is stored. Notice that the `/mnt/data/images` prefix must be present when using the FTP client, whereas the `cam store` command should not include `/mnt/data/images` in the filename. When the `ftp ls` command works, you can download a file by using the `ftp download_file <filename>` command. Example:

```
csp-term # ftp server 6
csp-term # ftp download /mnt/data/images/test.jpg
```

The `ftp download` function can also used to download from ram. It takes three arguments, a hex address, a length in decimal bytes and a local filename to store the file to. The two first values are returned on each `cam store`, and a list of memory addresses and lengths can be called using the `cam list` command. Example:

```
csp-term # cam node 6
csp-term # cam snap -arx
Snapping full-size image, timeout 5000
Snap result: 0
csp-term # cam list
[00] F:0, T:1526645931 0xb1aff008 6291456
[01] F:2, T:1526645931 0xb05fc008 1162047
csp-term # ftp server 6
csp-term # ftp download mem://0xb05fc008++1162047 test.jpg
File size is 1162047
Checksum: 0x73cc520a
Transfer Status: 0 of 6282 (0.00%)
100.0% [########################################] (6282/6282)
CRC Remote: 0x73cc520a, Local: 0x73cc520a
```

4. Parameter System

A number of parameters on the NanoCam C1U can be adjusted through the GomSpace parameter system. The parameters are divided across five tables depending on the parameter types.

Table 0 is used for system level parameters that are expected to stay fixed for the entire mission. Table 1 is used for adjusting the image capture and processing parameters. Table 2 provides direct access to the image sensor registers and table 3 is used for configuring image tagging. Table 4 is used to store telemetry data.

Table	Description
0	System Configuration
1	Image Configuration
2	Sensor Registers
3	Image Tag Configuration
4	Telemetry Data

Through GOSH it is possible to list all parameters in a table by running the `param list <table>` command. Here it shows the default image parameters:

```
nanocam # param list 1
Parameter list 1:
  0x0000 exposure-us        U32 10000
  0x0004 hist-skip          U8  3
  0x0005 gain-target        U8  50
  0x0006 gain-slack         U8  5
  0x0007 gain-iters         U8  10
  0x0008 gain-global        U16 32767
  0x000A gain-red           U16 32767
  0x000C gain-green         U16 32767
  0x000E gain-blue          U16 32767
  0x0010 store-format       U8  0
  0x0011 jpeg-qual          U8  85
  ...
```

A parameter is modified by first switching to its table with `param select <table>` and then running `param set <parameter> <value>`. E.g., to adjust the exposure time from 10 to 20 ms we run the commands:

```
nanocam # param select 1
nanocam # param set exposure-us 500
nanocam # param get exposure-us
  exposure-us = 500
```

The updated parameter values are only valid until next reboot. To store a parameter permanently, the parameter table must be stored. To make our change to the exposure time permanent, we then need to run `param save <table>`:

```
nanocam # param save 1
```

The parameter system protects the stored table with two checksums: one to protect the data itself against corruption and one to ensure that the stored data matches the table structure.

System parameters in table 0 are stored in locked FRAM sectors and must be unlocked prior to saving the parameter table. This is done using the `param unlock framstore-protected` command. After the `param save 0` command has been executed, the FRAM sectors can be relocked by calling `param lock framstore-protected` or by rebooting the system.

Note: As a safety measure, the `lock/unlock` commands are only available through the GOSH interface and

can not be executed via CSP. It is thus not possible to unlock and modify the table 0 parameters remotely. This prevents accidental modification of e.g. the CSP address in orbit.

Telemetry data in table 4 is not saved to permanent storage.

Until table 0, 1 and 3 are saved first time, a warning is printed at boot telling default values have been loaded. Once the tables have been saved this will disappear.

4.1 Table 0: System Configuration

Table 0 holds system level configuration. Modification of these parameters require a reboot of the system, before they take effect. Note that the FRAM must be unlocked before this parameter table can be saved.

Table 4.1: Parameter Table 'config'

Name	Addr.	Type	
csp-addr	0x00	uint8	CSP address Default: `6`
csp-rtable	0x01	string	CSP routing table Default: `28/5 KISS, 5/5 I2C, 0/0 CAN`
can-bitrate	0x64	uint32	CAN bitrate Default: `1000000`
kiss-bitrate	0x68	uint32	KISS bitrate Default: `500000`
kiss-device	0x6c	string	KISS uart (ttyS1 or ttyS2) Default: `/dev/ttyS1`

The `kiss-device` parameter defines which uart is used for the KISS interface, only one can be active at a time:

HW interface	Description
/dev/ttyS1	TTL level serial port (**default**)
/dev/ttyS2	RS-422

4.2 Table 1: Image Configuration

Table 1 contains configuration of the image parameters. The most important parameters is `exposure-us` which is used to adjust the exposure time in microseconds and `gain-target` which is used to set the target average brightness for the auto-gain algorithm.

The `color-correct`, `gamma-correct` and `white-balance` parameters enables or disables a number of image enhancement algorithms that run before the image is stored. They can be disabled to increase the storage speed.

Table 4.2: Parameter Table 'image'

Name	Addr.	Type	
exposure-us	0x00	uint32	Exposure time in us Default: `10000`
hist-skip	0x04	uint8	Histogram calculation skip Default: `3`
gain-target	0x05	uint8	Auto-gain target Default: `30`
gain-slack	0x06	uint8	Auto-gain target tolerance Default: `3`

Continued on next page

Table 4.2 – continued from previous page

Name	Addr.	Type	
gain-iters	0x07	uint8	Max auto-gain iterations Default: `10`
gain-global	0x08	uint16	Global gain (0-65535), will overwrite gain-red, green and blue Default: `2048`
gain-red	0x0a	uint16	Red channel gain (0-65535), when using -a option gain-global is used Default: `2048`
gain-green	0x0c	uint16	Green channel gain (0-65535), when using -a option gain-global is used Default: `2048`
gain-blue	0x0e	uint16	Blue channel gain (0-65535), when using -a option gain-global is used Default: `2048`
gain-max	0x10	uint16	Maximum auto-gain (0-65535) Default: `65535`
jpeg-qual	0x12	uint8	JPEG quality setting (0-100) Default: `85`
jpeg-prog	0x13	bool	Store progressive JPEG Default: `False`
bayer-algo	0x14	uint8	Bayer demosaicing algorithm Default: `2`
color-correct	0x15	bool	Perform color correction Default: `True`
gamma-correct	0x16	bool	Perform gamma correction Default: `True`
white-balance	0x17	bool	Perform white balancing Default: `False`
gamma	0x18	float	Gamma-correction Default: `2.2`
gamma-break	0x1c	float	Linear gamma breakpoint Default: `0.1`
ccm-red	0x20	float[3]	Color-correction, red Default: `[1.6809, 0.1496, -0.6606]`
ccm-green	0x2c	float[3]	Color-correction, green Default: `[-0.7613, 2.4481, -0.5168]`
ccm-blue	0x38	float[3]	Color-correction, blue Default: `[-0.3891, -0.5243, 2.0834]`
thumb-scale	0x44	uint8	Thumbnail scaling factor Default: `8`
led-en	0x45	bool	LED enable Default: `True`

4.3 Table 2: Sensor Register

Parameter table 2 provides direct access to the image sensor registers. Please refer to the *Aptina MT9T031* datasheet for a description of the registers.

4.4 Table 3: Image Tagging

This table controls the creation of tag files. If enabled with `tag-adcs`, the NanoCam will attempt to read basic housekeeping information from a GomSpace ADCS system and store this in a .tag file next to the image.

Table 4.3: Parameter Table 'tag'

Name	Addr.	Type	
tag-adcs	0x00	bool	Add ADCS tags to tag file Default: `False`
tag-adcs-host	0x01	uint8	ADCS tagging host Default: `4`
tag-adcs-port	0x02	uint8	ADCS tagging port Default: `20`
tag-adcs-tout	0x04	uint16	ADCS tagging timeout in ms Default: `200`

4.5 Table 4: Telemetry

This table contains telemetry data. The values are automatically updated once per second by the housekeeping task. Only the `boot-count` and `image-count` parameters are stored to persistent memory. Images stored using the autostore flag of `cam snap` will use the image count number as file name.

Table 4.4: Parameter Table 'telem'

Name	Addr.	Type	
boot-count	0x00	uint32	Number of system boots Auto Persist: `Yes`
image-count	0x04	uint32	Number of captured images Auto Persist: `Yes`
unixtime	0x08	uint64	System time in ns
uptime	0x10	uint32	System uptime in seconds
loads	0x14	uint8[3]	System loads in 5,10,15 min
freeram	0x18	uint32	Free RAM in bytes
procs	0x1c	uint16	Number of running processes
temp1	0x1e	int16	Temperature 1 in 1/10 C
temp2	0x20	int16	Temperature 2 in 1/10 C
icore	0x22	uint16	Core current in mA
iddr	0x24	uint16	DDR current in mA
ivcc	0x26	uint16	VCC current in mA
vddcore	0x28	uint16	Core voltage in mV
vddioddr	0x2a	uint16	VDD voltage in mV
vcc	0x2c	uint16	VCC voltage in mV

5. Focus

To set focus on the C1U Nanocam, below produce can be used.

5.1 Aperture and exposure time

Before setting focus it is a good idea to set the aperture and exposure time. If the C1U Nanocam is used indoor, it is advised to set the aperture close to max, ie. F lower than 4. Pr default the exposure time it set to 10000 us, but if used indoor, it can be increased to eg. 100000 by setting the `exposure-us` in param table 0 for better results.

5.2 Brightness

Verify the brightness level by taking a picture running `cam snap -a` (enable auto-gain algorithm)

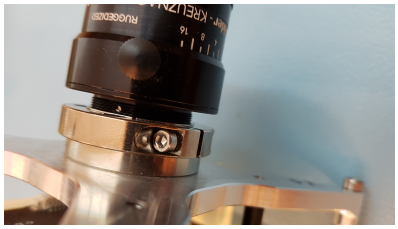
```
nanocam # cam snap -a
Snapping full-size image, timeout 5000
1500968455.242 sensor: failed to reach target brightness, found 3%
Snap result: 0
Brightness
  All   : 3% (min/max/peak/avg 010/013/008/010)
  Red   : 3% (min/max/peak/avg 010/013/008/010)
  Green : 3% (min/max/peak/avg 010/013/008/010)
  Blue  : 3% (min/max/peak/avg 010/013/008/010)
```

If the target brightness level is not reached, repeat `cam snap -a` until target is reached

```
nanocam # cam snap -a
Snapping full-size image, timeout 5000
Snap result: 0
Brightness
  All   : 29% (min/max/peak/avg 015/255/039/076)
  Red   : 29% (min/max/peak/avg 025/255/039/075)
  Green : 30% (min/max/peak/avg 015/255/039/078)
  Blue  : 29% (min/max/peak/avg 022/255/039/074)
```

5.3 Focus procedure

- Loosen the focus lock screw



- Run `watch 0 cam focus`

```
nanocam # watch 0 cam focus
Execution delay: 0
Command: cam focus
Result 42197
Result 42081
Result 42270
Result 42377
Result 42246
Result 42219
Result 42290
Result 42193
```

- Turn the lens 1/4 or 1/2
- Wait 1-2 seconds and observe the *Result* value from `cam focus`
- Keep turning the lens as long as the *Result* value increases
- When the *Result* value decrease, turn lens 1/8 back
- Keep adjusting until max value is reached
- ...
- Verify the focus by taking a picture running `cam snap -as` (enable auto-gain and autostore)
- Download picture:

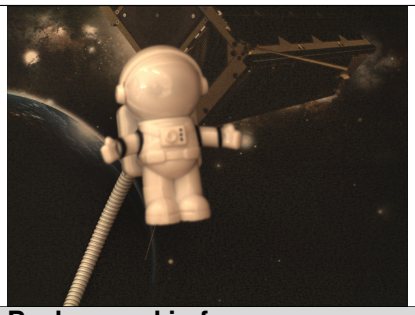
```
ftp server 6
ftp ls /mnt/data/images
ftp download /mnt/data/images/xxxxx.jpg
```

- When focus is verified tighten the focus screw again

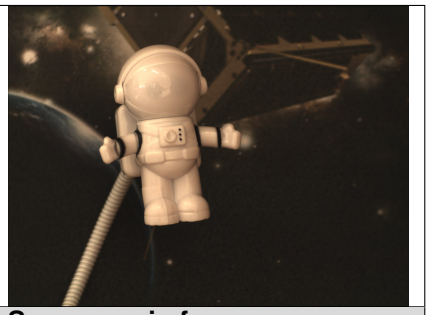
The cam focus focus routine will take a picture and encode a part of this picture as a JPG and return the resulting compressed size. The theory is that a more detailed picture results in a bigger image size.

Please note if you have a scene with a small object in front of a detailed background, the focus will be set to the background using above procedure (see picture in the middle below).

Blurred picture



Background in focus



Spaceman in focus

6. Client API

The client API consists of a set of wrapper functions that simplify the CSP interface to the NanoCam C1U. These functions are implemented in the `nanocam.c` file and can be integrated in custom code by including the `nanocam.h` header file. The `cmd_nanocam.c` implements the GOSH commands for the NanoCam and can be used as an additional reference for the use of the client API.

All the client functions specify a `timeout` argument that is used to specify the maximum number of milliseconds to wait for a reply. The client interface automatically performs endian conversion to network byte order on all arguments.

The functions return 0 on success and a non-zero error code on error. The error codes are listed below:

```
#define NANOCAM_ERROR_NONE              0 /** No error */
#define NANOCAM_ERROR_NOMEM             1 /** Not enough space */
#define NANOCAM_ERROR_INVAL             2 /** Invalid value */
#define NANOCAM_ERROR_IOERR             3 /** I/O error */
#define NANOCAM_ERROR_NOENT             4 /** No such file or directory */
```

Similar to the GOSH interface, the client API operates on a single C1U at a time. The CSP address of this C1U is set using the `nanocam_set_node` function. By default, the commands operate on CSP node `NANOCAM_DEFAULT_ADDRESS` which is currently set to 6. If the camera address has not been changed, it is not necessary to call `nanocam_set_node`.

void **`nanocam_set_node`**(uint8_t *node*)
: This function sets the CSP address of the NanoCam C1U. All other API functions use this CSP address.

6.1 Image Capture

Capture of images is provided by the `nanocam_snap` function.

int **`nanocam_snap`**(*cam_snap_t *snap*, *cam_snap_reply_t *reply*, unsigned int *timeout*)
: This function is used to capture an image. The capture parameters should be set in the `snap` structure argument prior to calling this function. The reply from the camera is returned in the `reply` struct.

`cam_snap_t`
: The `cam_snap_t` struct is used to specify the arguments for the snap command. Each field of the structure is documented below.

uint32_t **`cam_snap_t.flags`**
: This argument supplies optional flag bits that modifies the behavior of the snap request.

```
/* NANOCAM_PORT_SNAP */
#define NANOCAM_SNAP_FLAG_AUTO_GAIN     (1 <<  0) /** Automatically adjust gain */
#define NANOCAM_SNAP_FLAG_STORE_RAM     (1 <<  8) /** Store snapped image to RAM */
#define NANOCAM_SNAP_FLAG_STORE_FLASH   (1 <<  9) /** Store snapped image to flash */
#define NANOCAM_SNAP_FLAG_STORE_THUMB   (1 << 10) /** Store thumbnail to flash */
#define NANOCAM_SNAP_FLAG_STORE_TAGS    (1 << 11) /** Store image tag file */
#define NANOCAM_SNAP_FLAG_NOHIST        (1 << 16) /** Do not calculate histogram */
#define NANOCAM_SNAP_FLAG_NOEXPOSURE    (1 << 17) /** Do not adjust exposure */
```

uint8_t **`cam_snap_t.count`**
: Number of images to capture. If set to zero a single image capture will be performed. When capturing multiple images, the `nanocam_snap` function will only return the `cam_snap_reply_t` for the first image.

uint8_t **cam_snap_t.format**
Output format to use when NANOCAM_SNAP_FLAG_STORE_FLASH or NANOCAM_SNAP_FLAG_STORE_RAM is enabled in the flags field. Valid output formats are:

```
/* NANOCAM_PORT_STORE */
#define NANOCAM_STORE_RAW       0 /* Store RAW sensor output (1 pixel into 2 bytes) */
#define NANOCAM_STORE_BMP       1 /* Store bitmap output */
#define NANOCAM_STORE_JPG       2 /* Store JPEG compressed output */
#define NANOCAM_STORE_DNG       3 /* Store DNG output (Raw, digital negative) */
#define NANOCAM_STORE_RAW10     4 /* Store packed RAW output (4 pixels into 5 bytes) */
#define NANOCAM_STORE_UNKNOWN   UINT8_MAX
```

RAW pictures from the camera are 10 bit Bayer pattern (GRGB) stored in 16 bits. The 10 data bits are stored in bit [13-4] (mask 0x3FF0):

- [B3:B0] is expected to be low
- [B13:B4] is the image data
- [B15:B14] is expected to be high.

```
|--------------
| G R G R G R
| B G B G B G
|
```

uint16_t **cam_snap_t.delay**
Optional delay between captures in milliseconds. Only applicable when count > 1.

uint16_t **cam_snap_t.width**
Image width in pixels. Set to 0 to use default (maximum = 2048) size.

uint16_t **cam_snap_t.height**
Image height in pixels. Set to 0 to use default (maximum = 1536) size.

uint16_t **cam_snap_t.top**
Image crop rectangle top coordinate. Must be set to 0.

uint16_t **cam_snap_t.left**
Image crop rectangle left coordinate. Must be set to 0.

---

cam_snap_reply_t
This struct contains the reply of a image capture. The reply contains arrays with information of average brightness and distribution. A couple of defines are used for the length of these arrays:

```
#define NANOCAM_SNAP_COLORS           4
#define NANOCAM_SNAP_HIST_BINS        16
```

uint8_t **cam_snap_reply_t.result**
Result of the capture. One of the error codes listed in the introduction.

uint8_t **cam_snap_reply_t.seq**
Zero-index sequence number when capturing multiple images, i.e. when count > 1 in the cam_snap_t argument.

uint8_t[NANOCAM_SNAP_COLORS] **cam_snap_reply_t.light_avg**
Array of NANOCAM_SNAP_COLORS elements corresponding to the average brightness of all pixels plus the red, green and blue channel pixels. The numbers are scaled from 0-255, so e.g. 128 corresponds to an average brightness of 50%.

uint8_t[NANOCAM_SNAP_COLORS] **cam_snap_reply_t.light_peak**
Array of NANOCAM_SNAP_COLORS elements corresponding to the estimated peak brightness of all pixels plus the red, green and blue channel pixels.

uint8_t[NANOCAM_SNAP_COLORS] **cam_snap_reply_t.light_min**
Array of NANOCAM_SNAP_COLORS elements corresponding to the minimum brightness of all pixels plus the red, green and blue channel pixels.

uint8_t[NANOCAM_SNAP_COLORS] **cam_snap_reply_t.light_max**
Array of NANOCAM_SNAP_COLORS elements corresponding to the maximum brightness of all pixels plus the red, green and blue channel pixels.

uint8_t[NANOCAM_SNAP_COLORS][NANOCAM_SNAP_HIST_BINS] **cam_snap_reply_t.hist**
Array of NANOCAM_SNAP_COLORS elements each consisting on an array of NANOCAM_SNAP_HIST_BINS bins. Each bin contains a number from 0 to 255 matching the distribution of brightness, with the sum of all bins being 255. Thus, if a bin is 128, 50% of all pixels falls within the brightness range covered by that particular bin.

6.2 Image Storage

Storage of images captured to the snap buffer is provided by the nanocam_store functions. Images are stored in the /mnt/data/images directory on the camera file system.

int **nanocam_store**(*cam_store_t *store*, *cam_store_reply_t *reply*, unsigned int *timeout*)
This function is used to store a captured image from the snap buffer to persistant storage and/or RAM.

---

cam_store_t
This struct is used to supply store arguments to the nanocam_store function.

uint8_t **cam_store_t.format**
Output format of the stored image. See the argument list to cam_snap_t.format for a list of valid options.

uint8_t **cam_store_t.scale**
This argument is currently unused and should be set to 0.

uint32_t **cam_store_t.flags**
This argument supplies optional flag bits that modifies the behavior of the store request. If the NANOCAM_STORE_FLAG_FREEBUF flag is cleared, a copy of the stored image will be kept in the RAM list.

```
#define NANOCAM_STORE_FLAG_FREEBUF      (1 << 0) /* Free buffer after store */
#define NANOCAM_STORE_FLAG_THUMB        (1 << 1) /* Create thumbnail */
#define NANOCAM_STORE_FLAG_TAG          (1 << 2) /* Create tag file */
```

char[40] **cam_store_t.filename**
Filename of the stored image. The file type is not required to match the file format, but it is recommend to e.g. store JPEG images with a *.jpg* ending. Setting this field to an empty string, i.e. set filename[0] to \0, will only store the image in the RAM list.

---

cam_store_reply_t
This struct contains the reply of a image store command.

uint8_t **cam_store_reply_t.result**
Result of the store command. One of the error codes listed in the introduction.

uint32_t **cam_store_reply_t.image_ptr**
Address of the RAM copy of the stored image.

uint32_t **cam_store_reply_t.image_size**
Size in bytes of the RAM copy of the stored image.

6.3 Modifying Sensor Registers

The image sensor registers can be adjusted using the `nanocam_reg_read` and `nanocam_reg_write` functions. Any modifications of the registers are volatile, and may be overridden by the auto-gain and exposure setting algorithms.

Note: For normal operation, it is not necessary to adjust sensor registers directly. Instead the image configuration parameters from the image table should be used.

Please refer to the *Aptina MT9T031* datasheet for a description of individual sensor registers.

int **`nanocam_reg_read`**(uint8_t *reg*, uint16_t **value*, unsigned int *timeout*)

This function reads a sensor register and returns the current value. The `reg` argument contains the address of the register to read and the current value is returned in the `value` pointer.

int **`nanocam_reg_write`**(uint8_t *reg*, uint16_t *value*, unsigned int *timeout*)

Use this function to update the value of a register. The `reg` contains the register address and `value` contains the new value to write to the register.

6.4 In-memory Images

int **`nanocam_img_list`**(*nanocam_img_list_cb cb*, unsigned int *timeout*)

Call this function to list all images in the RAM list. The `nanocam_img_list_cb` callback will be called once for each image element in the list.

typedef void **`(*nanocam_img_list_cb)`**(int *seq*, cam_list_element_t **elm*)

Implement an image listing callback function matching this prototype, and supply it to the `nanocam_img_list` list function. If no images are available in memory, the callback is called with `elm` set to NULL.

int **`nanocam_img_list_flush`**(unsigned int *timeout*)

This function flushes all images stored in the RAM image list. Note that the current image in the snap buffer can not be flushed, so a single image will always be returned by `nanocam_img_list`.

6.5 Focus Assist Routine

int **`nanocam_focus`**(uint8_t *algorithm*, uint32_t **af*, unsigned int *timeout*)

This function runs a single iteration of the focus assist algorithm. The `algorithm` argument is used to select between different algorithms. Currently, `NANOCAM_AF_JPEG` is the only supported option.

The focus assist algorithm captures an image, JPEG compresses the center of the image and returns the size of the compressed data in the `af` pointer. The premise is that a more focused image will be more difficult to compress, giving a larger size of the compressed data. Continuously running this algorithm can thus be used to adjust the focus until a maximum size is found.

6.6 Data Partition Recovery

int **`nanocam_recoverfs`**(unsigned int *timeout*)

This function can be used to recreate the data file system. Note that this erases ALL images stored on the camera. If you just want to delete all captured images, using the FTP `rm` command is much faster and a safer option than rebuilding the entire file system.

7. CSP Header

The NanoCam C1U uses the CubeSat Space Protocol (CSP) to transfer data to and from other CSP nodes on the main system bus. All reference code provided is based on this. If it is chosen not to use *libcsp*, the following chapter contains information regarding the CSP HEADER and examples how to fill it out for the 3 commands: SNAP, LIST and PEEK.

It is advised to read this chapter together with the provided src code (based on *libcsp*), mainly nanocam.c and nanocam_types.h are of interest to get the full understanding.

Please note if CAN is used as HW interface, CAN Fragmentation Protocol (CFP), a data-link layer protocol specially developed for CSP to handle CSP packages larger than 8 bytes must also be added. CFP is not described further in this chapter.

```
CAM            OBC
CSP            CSP
(CFP)          (CFP)
Interface      Interface
hw bus<------>hw bus
```

C1U Nanocam support 3 different interfaces, I^2C, CAN and KISS, these are not described further in this chapter either.

7.1 CSP Header format

All commands for the C1U Nanocam are based on *libcsp*, meaning it contains a CSP HEADER + some parameters. The CSP HEADER consists of the following fields:

Priority|**s**ource|**D**estination|Destination-**P**ort|source-**p**ort|**u**nused|S**F**P|H**M**AC|**X**TEA|**R**DP|**C**RC32

```
Bit31                              Bit0
|                                    |
PPss|sssDDDDD|PPPP|PPpp|pppp|uuuF|MXRC
```

Below table contains a short description and a recommended value to be used for each field:

[Bit] Name	Description	Recommended Value
[31:30] **P**riority (2 bit)	CSP Priority of the message	2
[29:25] **s**ource (5 bit)	CSP address of the source. Use recommended value if csp routing table is not set (`csp-rtable`)	CAN: **s**ource = 1 I^2C: **s**ource = 2 KISS: **s**ource = 8
[24:20] **D**estination (5 bit)	C1U Nanocam CSP address	6
[19:14] destination **P**ort (6 bit)	Depends on the CMD, defines can be found in nanocam_types.h (see also examples below)	-
[13:8] source **p**ort (6 bit)	Defined by source/client	0 (can be freely chosen)
[7:3] **u**nused (3bit)	Unused	0
[4:4] S**F**P (1bit)	Simple Fragmented Protocol	0
[3:0] H**M**AC, **X**TEA, **R**DP, **C**RC32 (4 bit)	Optional, set to 0 for disabling	0 (**1 when using KISS**)

Note: When **CAN** is used, the first parameter must always be a uint16_t with the size of the following parameters (excluding the size parameter itself).

Note: When **KISS** is used Gomspace products enforces CRC32 two times, if not applied, C1U Nanocam will discard the commands.

To enable 2 times CRC32:

1. Set the CRC32 bit in the CSP header
2. Append the first CRC32 checksum (4 bytes) to the command
3. Append the second CRC32 checksum, calculated including the first checksum

The CRC32 value is calculated without the CSP HEADER, the implementation can be found in *libcsp* under "src/csp_crc32.c" (*libcsp* is available at http://www.libcsp.org).

7.2 SNAP

Reference implementation can be found in:

- nanocam_cmd.c: **cmd_cam_snap()**
- nanocam.c: **nanocam_snap()**

Parameter definition can be found in:

- nanocam_types.h: **cam_snap_t**
- nanocam_types.h: **cam_snap_reply_t**

Snap command defines:

```
#define NANOCAM_DEFAULT_ADDRESS      6
#define NANOCAM_PORT_SNAP            20
```

Snap command **I²C**:

```
CSP HEADER | uint32 flags | uint8 count | uint8 format | uint16 delay | uint16 width |
uint16 height | uint16 top | uint16 left
```

Snap command **KISS**:

```
CSP HEADER | uint32 flags | uint8 count | uint8 format | uint16 delay | uint16 width |
uint16 height | uint16 top | uint16 left | uint32 crc32 | uint32 crc32
```

Snap command **CAN**:

```
CSP HEADER | uint16 size | uint32 flags | uint8 count | uint8 format | uint16 delay |
uint16 width | uint16 height | uint16 top | uint16 left
```

7.2.1 SNAP HEADER Example

Name	Value
Priority	2
source	1 (CAN)
Destination	6 (NANOCAM_DEFAULT_ADDRESS)
destination **P**ort	20 (NANOCAM_PORT_SNAP)
source **p**ort	62
unused	0
S**F**P	0
H**M**AC, **X**TEA,**R**DP, **C**RC32	0

```
PP ss|sss D|DDDD |PPPP|PP pp|pppp| uuuF | MXRC
10 00|001 0|0110 |0101|00 11|1110| 0000 | 0000
  8     2     6     5    3     E    0    | 0    = 0x82653E00
```

7.2.2 SNAP Reply

Snap reply defines:

```
#define NANOCAM_SNAP_COLORS          4
#define NANOCAM_SNAP_HIST_BINS       16
```

Snap reply **I²C**:

```
CSP HEADER | uint8_t result | uint8_t seq | uint8_t reserved[2] |
uint8_t light_avg[NANOCAM_SNAP_COLORS] | uint8_t light_peak[NANOCAM_SNAP_COLORS] |
uint8_t light_min[NANOCAM_SNAP_COLORS] | uint8_t light_max[NANOCAM_SNAP_COLORS] |
uint8_t hist[NANOCAM_SNAP_COLORS][NANOCAM_SNAP_HIST_BINS]
```

Snap reply **KISS**:

```
CSP HEADER | uint8_t result | uint_t 8 seq | uint8_t reserved[2] |
uint8_t light_avg[NANOCAM_SNAP_COLORS] | uint8_t light_peak[NANOCAM_SNAP_COLORS] |
uint8_t light_min[NANOCAM_SNAP_COLORS] | uint8_T light_max[NANOCAM_SNAP_COLORS] |
uint8_t hist[NANOCAM_SNAP_COLORS][NANOCAM_SNAP_HIST_BINS] | uint32_t crc32 | uint32 crc32
```

Snap reply **CAN**:

```
CSP HEADER | uint16_t size | uint8_t result | uint_t 8 seq | uint8_t reserved[2] |
uint8_t light_avg[NANOCAM_SNAP_COLORS] | uint8_t light_peak[NANOCAM_SNAP_COLORS] |
uint8_t light_min[NANOCAM_SNAP_COLORS] | uint8_t light_max[NANOCAM_SNAP_COLORS] |
uint8_t hist[NANOCAM_SNAP_COLORS][NANOCAM_SNAP_HIST_BINS]
```

7.3 Images LIST

Reference implementation can be found in:

- nanocam_cmd.c: **cmd_cam_list()**
- nanocam.c: **nanocam_img_list()**

Parameter definition can be found in:

- nanocam_types.h: **cam_list_t**
- nanocam_types.h: **cam_list_reply_t**

- nanocam_types.h: **cam_list_element_t**

List command defines:

```
#define NANOCAM_DEFAULT_ADDRESS      6
#define NANOCAM_PORT_IMG_LIST        22
```

List command **I²C**:

```
CSP HEADER | uint8_t elements | uint8_t reserved[3] | uint32_t flags
```

List command **KISS**:

```
CSP HEADER | uint8_t elements | uint8_t reserved[3] | uint32_t flags |   | uint32 crc32 |␣
↪uint32 crc32
```

List command **CAN**:

```
CSP HEADER | uint16_t size | uint8_t elements | uint8_t reserved[3] | uint32_t flags
```

7.3.1 LIST HEADER Example

Name	**Value**
Priority	2
source	1 (CAN)
Destination	6 (NANOCAM_DEFAULT_ADDRESS)
destination **P**ort	22 (NANOCAM_PORT_IMG_LIST)
source **p**ort	61
unused	0
S**F**P	0
H**M**AC, **X**TEA,**R**DP, **C**RC32	0

```
PP ss|sss D|DDDD |PPPP|PP pp|pppp| uuuF | MXRC
10 00|001 0|0110 |0101|10 11|1101| 0000 | 0000
  8     2     6     5     B     D    0    | 0    = 0x8265BD00
```

7.3.2 LIST Reply

cam_list_element_t:

```
uint8_t id | uint8_t format | uint16_t flags | uint16_t width | uint16_t height |
uint32_t ptr | uint32_t size | uint64_t time
```

List reply **I²C**:

```
CSP HEADER | uint16_t total | uint16_t seq | uint8_t count | uint8_t reserved[3] |
cam_list_element_t images[]
```

List reply **KISS**:

```
CSP HEADER | uint16_t total | uint16_t seq | uint8_t count | uint8_t reserved[3] |
cam_list_element_t images[] | uint32 crc32 | uint32 crc32
```

List reply **CAN**:

```
CSP HEADER | uint16_t size | uint16_t total | uint16_t seq | uint8_t count |
uint8_t reserved[3] | cam_list_element_t images[]
```

7.4 Transfer Picture via PEEK

Normally Gomspace FTP client is used, but in case *libcsp* is not used, this might bee too much to implement, and it could be considered to use “cmp peek” instead. Please note this require images are stored in RAM on the C1U Nanocam (“cam list” will provide the memory address):

Reference implementation can be found in:

- nanocam_cmd.c: **cmd_cam_peek()**

Parameter definition can be found in:

- csp_cmp.h: **csp_cmp_message/peek**

Peek command defines:

```
#define NANOCAM_DEFAULT_ADDRESS      6
#define CSP_CMP                      0 (csp_types.h)

#define CSP_CMP_REQUEST              0x00
#define CSP_CMP_PEEK                 4
#define CSP_CMP_PEEK_MAX_LEN         200
```

Peek command parameters:

```
CSP HEADER | uint8_t type | uint8_t code | uint32_t addr | uint8_t len |
char data[CSP_CMP_PEEK_MAX_LEN]

type = 0x00 (CSP_CMP_REQUEST)
code  = 4 (CSP_CMP_PEEK)
addr = memory address to read (obtained from cam LIST)
len  = length to read in bytes (max CSP_CMP_PEEK_MAX_LEN)
data = data buffer (only reply command will have valid data)
```

Peek command **I²C**:

```
CSP HEADER | uint8_t type | uint8_t code | uint32_t addr | uint8_t len |
char data[CSP_CMP_PEEK_MAX_LEN]
```

Peek command **KISS**:

```
CSP HEADER | uint8_T type | uint8_t code | uint32_t addr | uint8_t len |
char data[CSP_CMP_PEEK_MAX_LEN] | uint32 crc32 | uint32 crc32
```

Peek command **CAN**:

```
CSP HEADER | uint16_t size | uint8_t type | uint8_t code | uint32_t addr | uint8_t len |
char data[CSP_CMP_PEEK_MAX_LEN]
```

7.4.1 PEEK HEADER Example

Name	**Value**
Priority	2
source	1 (CAN)
Destination	6 (NANOCAM_DEFAULT_ADDRESS)
destination **P**ort	0 (CSP_CMP)
source **p**ort	60
unused	0
S**F**P	0
H**M**AC, **X**TEA,**R**DP, **C**RC32	0

```
PP ss|sss D|DDDD |PPPP|PP pp|pppp| uuuF | MXRC
10 00|001 0|0110 |0000|00 11|1100| 0000 | 0000
  8     2     6     0    3     C    0    | 0    = 0x82603C00
```

7.4.2 PEEK reply

Peek reply defines:

```
#define CSP_CMP_REPLY                0xFF
#define CSP_CMP_PEEK                 4
#define CSP_CMP_PEEK_MAX_LEN         200
```

Peek reply **I²C**:

```
CSP HEADER | uint8_t type | uint8_t code | uint32_t addr | uint8_t len |
char data[CSP_CMP_PEEK_MAX_LEN]
```

Peek reply **KISS**:

```
CSP HEADER | uint8_t type | uint8_t code | uint32_t addr | uint8_T len |
char data[CSP_CMP_PEEK_MAX_LEN] | uint32 crc32 | uint32 crc32
```

Peek reply **CAN**:

```
CSP HEADER | uint16_t size | uint8_t type | uint8_t code | uint32_t addr | uint8_t len |
char data[CSP_CMP_PEEK_MAX_LEN]
```

8. Firmware Update

The firmware on the NanoCam C1U can be updated via USB using the Device Firmware Update (DFU) protocol. To update the firmware, you need the dfu-util tool that can be installed from most Linux distribution package managers or from http://dfu-util.sourceforge.net/.

A full NanoCam firmware consists of 5 parts: at91bootstrap, u-boot bootloader, Linux kernel, Linux device tree (DTB) and the root file system. It is usually only needed to update the root filesystem which contains the nanocam application.

To prepare the update, attach a Micro-USB cable between the C1U and your development machine. Please ensure that the development machine is properly ground before connecting the C1U. Press and hold the “space” key while the C1U is booting to stop the boot sequence in the bootloader. You should see a boot sequence similar to the one below:

```
U-Boot 2015.01 (Aug 07 2015 - 09:30:37)

CPU: SAMA5D35
Crystal frequency:       12 MHz
CPU clock        :      528 MHz
Master clock     :      132 MHz
DRAM:  512 MiB
Flash: 64 MiB
MMC:   mci: 0
nanocam>
```

U-boot commands can now be entered in the “nanocam>” console.

In this example we flash the Linux root filesystem, but the approach is the same for all images. First run the `run flashroot` command in the U-boot console. The command does not produce any output:

```
nanocam> run flashroot
```

Next, on your development machine run dfu-util to transfer the image. The file that should be flashed should be passed with the `--download` argument and the altsetting should be passed with `-a` argument. The altsetting must be `emmc` when flashing the root filesystem and `ram` for all other images.

For the root filesystem, we run the following command:

```
dev % sudo dfu-util --download gomspace-nanocam-nanocam.emmc  -a emmc
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

dfu-util: Invalid DFU suffix signature
dfu-util: A valid DFU suffix will be required in a future dfu-util release!!!
Opening DFU capable USB device...
ID 03eb:3333
Run-time device DFU version 0110
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0110
Device returned transfer size 4096
Copying data from PC to DFU device
```

(continues on next page)

(continued from previous page)

```
Download        [=========================] 100%      3576232 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

If the download was successful, you should see the following output on the NanoCam. Press Ctrl+C to exit DFU mode and write the image file to the flash. Do not remove the power from the system while the image is being written to flash:

```
nanocam> run flashroot
#DOWNLOAD ... OK
Ctrl+C to exit ...
................................ done
Un-Protected 32 sectors
................................ done
Erased 32 sectors
Copy to Flash... done
nanocam>
```

To reset and boot the system normally run `reset`

8.1 U-boot commands

The complete list below contains the image file names and corresponding u-boot command and DFU altsettings, please do **NOT** flash at91bootstrap or u-boot without consulting GomSpace first.

Firmware Part	Filename	U-boot Command	dfu-util altsetting
at91bootstrap	at91bootstrap-nanocam.bin	run flashat91	ram
u-boot	u-boot-nanocam.bin	run flashuboot	ram
Linux kernel	zImage-nanocam.bin	run flashlinux	ram
Linux DTB	zImage-nanocam-v8.dtb	run flashdtb	ram
Root Filesystem	gomspace-nanocam-nanocam.emmc	run flashroot	emmc

8.2 ROM Bootloader Recovery

If the software update procedure fails, it is possible to recover the system using the processor's built-in ROM bootloader.

Warning: Do **NOT** attempt to run the ROM bootloader recovery without consulting GomSpace first. If the system boots to u-boot you do not need to run the recovery procedure.

In addition to dfu-util, you will need the SAM-BA In-system Programmer from Atmel (version 2.15 or newer 2.xx), which can be downloaded from https://www.microchip.com/developmenttools/ProductDetails/PartNO/SAM-BA%20In-system%20Programmer#:~:text=Summary%3A,the%20Secure%20SAM%2DBA%20edition. Unpack the tool and add the `sam-ba_cdc_linux` directory to your `$PATH` to install SAM-BA.

With the camera powered off, connect a Micro-B USB cable to the USB connector (J1) on the C1U.

To force the system to boot the ROM bootloader, you need to short two pads on the top side of the NanoCam PCB. The pad is located next to the serial debug connector, and is marked in red on the placement diagram below. Use a pair of tweezers to short the two pads, apply power to the system, and then remove the tweezers. A `/dev/ttyACM0` USB device should appear on your developement computer.

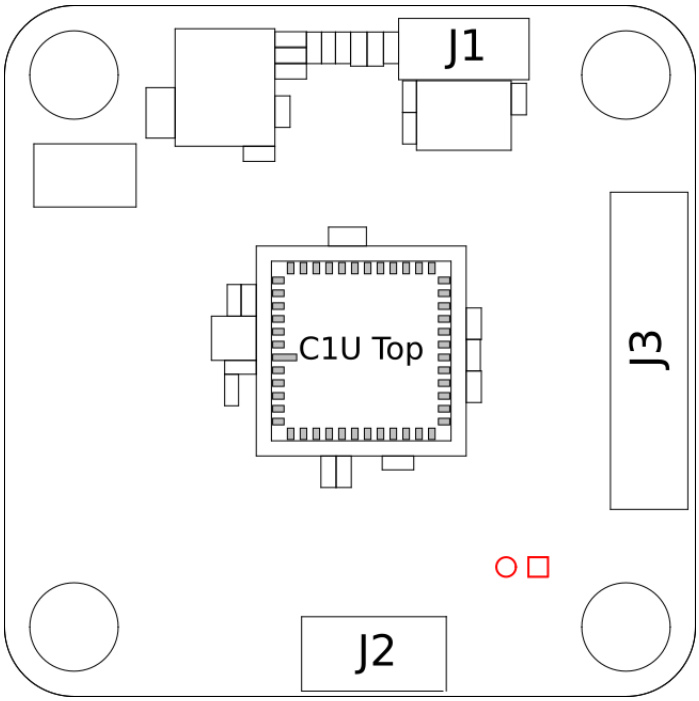


Fig. 8.1: Placement of NanoCam C1U Boot Mode Setting Pad

To recover the system, run the `nanocam-bootstrap.py` Python script. This will flash *all* the firmware parts to the NanoCam. When the script is finished, power cycle the device to boot into u-boot. The output from the recovery script should look like this:

```
dev % ./nanocam-bootstrap.py
GomSpace NanoCam C1Uv2 Bootstrap Utility

WARNING: this will erase ALL firmware components on the camera!
Please ensure that the system is powered off before continuing

To flash via SAM-BA and the ROM bootloader, short the bootloader
pads on the PCB and power on the board. To flash using an existing
U-Boot, just power on the board

Waiting for board to power on ...
Detected ROM bootloader, waiting for ACM device
Flashing bootloaders via SAM-BA
sam-ba_64 /dev/ttyACM0 at91sama5d3x-ek nanocam-bootstrap.tcl
-I- Waiting ...
-I- TCL platform : Linux
-I- SAM-BA CDC 2.16  on : linux
-I- Retrieved arguments from command line :
-I- argv 0 : /dev/ttyACM0
-I- argv 1 : at91sama5d3x-ek
-I- argv 2 : nanocam-bootstrap.tcl
-I- Connection /dev/ttyACM0
-I- Connection : /dev/ttyACM0 (target(comType) = 0)
-I- Board : at91sama5d3x-ek
```

(continues on next page)

(continued from previous page)

```
-I- Traces Level : 4
-I- target(handle) : file10
-I- sourcing board description file /opt/utils/sam-ba_cdc_linux/tcl_lib/at91sama5d3x-ek/
↪at91sama5d3x-ek.tcl
Read device Chip ID at 0xFFFFEE40 --- get 0xffffffff8a5c07c2
-I- Found chip : at91sama5d3x (Chip ID : 0xffffffff8a5c07c2)
-I- Loading applet applet-lowlevelinit-sama5d3x.bin at address 0x308000
-I- Memory Size : 0x0 bytes
-I- Buffer address : 0x4
-I- Buffer size: 0x0 bytes
-I- Applet initialization done
-I- Low level initialized
-I- Loading applet applet-extram-sama5d3x.bin at address 0x308000
-I- Memory Size : 0x20000000 bytes
-I- Buffer address : 0x3093E0
-I- Buffer size: 0x0 bytes
-I- Applet initialization done
-I- External RAM initialized
-I- Command line mode : Execute script file : nanocam-bootstrap.tcl
-I- === Initialize the NOR access ===
-I- NORFLASH::Init (trace level : 4)
-I- Loading applet applet-norflash-sama5d3x.bin at address 0x20000000
-I- Memory Size : 0x4000000 bytes
-I- Buffer address : 0x20009884
-I- Buffer size: 0x20000 bytes
-I- Applet initialization done
-I- === Erase all the NOR flash blocs and test the erasing ===
-I- Complete 0%
-I-   Erasing  blocks at address 0x0
[Lines omitted]
-I- Complete 99%
-I-   Erasing  blocks at address 0x3FC0000
-I- === Load the bootstrap in the first sector ===
-I- Send File at91bootstrap-nanocam.bin at address 0
GENERIC::SendFile at91bootstrap-nanocam.bin at address 0x0
-I- File size : 0x158C byte(s)
-I- Complete 0%
-I-   Writing: 0x158C bytes at 0x0 (buffer addr : 0x20009884)
-I-   0x158C bytes written by applet
-I- === Load the u-boot in the next sectors ===
-I- Send File u-boot-nanocam.bin at address 0x00020000
GENERIC::SendFile u-boot-nanocam.bin at address 0x20000
-I- File size : 0x3D3A0 byte(s)
-I- Complete 0%
-I-   Writing: 0x20000 bytes at 0x20000 (buffer addr : 0x20009884)
-I-   0x20000 bytes written by applet
-I- Complete 52%
-I-   Writing: 0x1D3A0 bytes at 0x40000 (buffer addr : 0x20009884)
-I-   0x1D3A0 bytes written by applet
-I- === DONE. ===
Power-cycle the board to continue flashing via DFU
Waiting for U-Boot to start ...
Ready to accept commands
Flashing zImage-nanocam.bin via DFU to alt 'ram' (3579216 bytes)
dfu-util -D zImage-nanocam.bin -a ram
[Lines omitted]
Download      [=========================] 100%      3579216 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

(continues on next page)

(continued from previous page)

```
Flashing zImage-nanocam-v8.dtb via DFU to alt 'ram' (32424 bytes)
dfu-util -D zImage-nanocam-v8.dtb -a ram
[Lines omitted]
Download      [=========================] 100%        32424 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
Flashing gomspace-nanocam-nanocam.emmc via DFU to alt 'emmc' (335544320 bytes)
dfu-util -D gomspace-nanocam-nanocam.emmc -a emmc
[Lines omitted]
Download      [=========================] 100%    335544320 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
All flash operations completed
```

9. Software Changelogs

9.1 NanoCam C1U (server)

5.0.8 (2022-04-06)

- Bug: Missing remote shell init (missing log group)
- Bug: Manual missing info regarding 2 times CRC when using KISS

5.0.7 (2020-08-31)

- Bug: Wrong gain value saved in tag file

5.0.6 (2020-08-27)

- Improvement: Updated manual “Firmware Update” and RAW description in API section
- Improvement: updated nanocam2_client to 5.0.5

5.0.5 (2020-02-21)

- Improvement: to reflect sensor behavior gain-red, green and blue is set when gain-global is set
- Improvement: updated nanocam2_client to 5.0.4
- Improvement: added nanocam2_client changelog in manual

5.0.4 (2020-02-19)

- Bug: Fixed gain-red, gain-green and gain-blue in param table 1 not having any effect
- Bug: Fixed empty tag files (if powered off too quickly without shutdown command)
- Improvement: updated nanocam2_client to 5.0.3

5.0.3 (2019-11-28)

- Bug: Fixed RTC clock not saved on reboot
- Improvement: updated nanocam2_client to 5.0.2

5.0.2 (2019-10-25)

- Bug: autoexposure flag do not work
- Improvement: updated nanocam2_client to 5.0.1
- Improvement: updated libgscsp to 2.7.2

5.0.1 (2019-09-20)

- Bug: wrong libgscsp version in dependencies
- Bug: wscript update to fix build instability

5.0.0 (2019-09-19)

- Breaking: param tables changed
- Breaking: update all libs
- Breaking: switch to nanocam client 5.x.x (new libparam)
- Improvement: added rgosh support
- Improvement: more info in tag file

4.15.2 (2019-01-17)

- Improvement: added check for valid kiss-interface in table 0

4.15.1 (2018-05-18)

- Feature: distro: meta-atmel updated due to removed wl driver in github
- Improvement: use gsbuildtools for manual (new GS template)
- Bug: client fix when using new libs

4.14 (2017-10-18)

- Improvement: doc: csp section updated
- Improvement: doc: wrong paramters in cam peek example

4.13 (2017-08-04)

- Feature: added RS422 as optional KISS interface
- Improvement: added focus guide in manual
- Improvement: added CSP Header description in manual
- Feature: updated nanocam client
- Feature: added RS422 in table 0
- Feature: use cam format when snap options are set
- Bug: wrong arg check in cam peek
- Bug: wrong return parameter for csp transaction
- Bug: cpp warnings

4.12 (2017-06-19)

- Improvement: updated adcs client (libadcs)
- Bug: client printing wrong size when snapping
- Bug: snap should scale and not crop when smaller images size is used (fix for debug if)
- Feature: cam peek add to client as reference how to read image wo FTP
- Improvement: cam node return current setting if no parameters are provided
- Feature: rawtodng tool added
- Feature: internal design document added

4.8 (2016-11-16)

- Improvement: document use of nanocam-bootstrap.py

4.7 (2016-09-23)

- Feature: add support for packed RAW output format

4.6 (2016-08-15)

- Bug: update libcsp with RDP fixes for FTP

4.5 (2016-05-09)

- Feature: implement scaling support for store command

4.4 (2016-05-09)

- Feature: platform v1.2 support

4.3 (2016-02-24)

- Feature: platform v1.1 support
- Feature: add parameter to enable/disable LED
- Feature: update documentation

4.2 (2015-09-01)

- Improvement: rename ‘cam af’ command to ‘cam focus’
- Feature: add color correction
- Feature: add gamma correction
- Feature: add DNG output format
- Feature: add thumbnail support
- Feature: add ADCS basic housekeeping tagging
- Bug: fix zippering pattern on edges

4.1 (2015-08-06)

- Feature: added CSP via I2C support
- Feature: added CSP via KISS support

4.0 (2015-07-08)

- Feature: switched to revision 7 PCB (not compatible to previous revisions)
- Improvement: revised CSP protocol
- Feature: added FRAM support
- Feature: added RTC support
- Feature: added per-color histogram support
- Feature: added parameter system support
- Feature: added automatic gain adjustment support
- Bug: allow snapped images to be stored multiple times

9.2 NanoCam C1U (nanocam2_client)

5.0.5 (2020-08-26)

- Improvement: added RAW description in API section of documentation

5.0.4 (2020-02-21)

- Improvement: updated documentation for gain-global usage

5.0.3 (2020-02-19)

- Bug: cam snap missing width and height in help text

5.0.2 (2019-10-28)

- Bug: cam peek, use smallest value of CSP_CMP_PEEK_MAX_LEN and GS_CSP_DEFAULT_MAX_PAYLOAD_SIZE

5.0.1 (2019-10-25)

- Improvement: added retry count in cam peek command
- Improvement: updated build scripts and dependencies
- Improvement: remove wscript option, not needed anymore since commands must be registered.
- Improvement: added command register function to public header file.
- Improvement: replaced sscanf / command_args(ctx), with gs_string_to_xxx()
- Improvement: use mandatory_args and/or optional_args, instead of checking for minimum args in functions

- Improvement: use new GS_CSP_DEFAULT_MAX_PAYLOAD_SIZE (libgscsp), instead of CSP_CMP_PEEK_MAX_LEN
- Bug: cam peek now only enabled for linux as it does not work for freertos
- Improvement: register write function, now reads back the value and print it
- Improvement: libgscsp updated to 2.7.2

5.0.0 (2019-09-19)

- Breaking: switched to libparam 4.7 (new param layout)
- Breaking: renamed kiss-interface to kiss-device in table 0

4.16.6 (2019-07-03)

- Improvement: Changed command/gosh definitions to const.

4.16.5 (2019-05-20)

- Improvement: Updated command help text.

4.16.4 (2019-02-21)

- Improvement: Updated documentation.

4.16.3 (2018-12-20)

- Improvement: Updated documentation.

4.16.2 (2018-11-22)

- Improvement: Updated dependencies.

10. 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.