|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AAReST  **Deformable Mirror Box ICD** | | | | |
| Caltech Author(s):  (if joint document SSC Author(s):  Approved on behalf of Caltech by  S. Pellegrino  Approved on behalf of Reviewers by  John Baker | | | T. Talon, S. Ferraro, A. Goel, Y. Wei  C. Bridges) |  |
| Subject | | | | |
| ICD for the ARReST deformable mirror box. | | | | |
|  | | | | |
| Version History | | | | |
| Version | Date | Changes | | |
| A | 1st December 2016 | First Issue | | |
| B | 21st December 2016 | First draft of sections 1, 2, 3, 4, 6 | | |
| C | 5th July 2017 | Update of Sections 4 and 6 | | |
| D | 2nd August 2017 | Changed picture of connector and updated grounding scheme in 3.7 | | |
| E | 8th November 2017 | Added bootloader section, updated register codes and function codes | | |
| F | 11th November 2017 | Updated communication protocol with camera | | |
| G | 20th December 2017 | Update Software section (Architecture, TCTM) | | |

**Intentionally Blank**

**Distribution List:**

|  |  |  |
| --- | --- | --- |
| **Participant No.** | **Participant organisation name** | **Country** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Intentionally Blank**

**Table of Contents**

[1 INTRODUCTION 1](#_Toc501199511)

[1.1 Scope 1](#_Toc501199512)

[1.2 Applicable Documents 2](#_Toc501199513)

[1.3 Reference Documents 2](#_Toc501199514)

[1.4 Acronyms and Abbreviations 2](#_Toc501199515)

[2 Deformable Mirror Box 3](#_Toc501199516)

[2.1 General Instrument Description 3](#_Toc501199517)

[2.2 Instrument Coordinate System 3](#_Toc501199518)

[2.3 Operational Modes 3](#_Toc501199519)

[3 Mechanical Interface and Design 4](#_Toc501199520)

[3.1 Structural Design 4](#_Toc501199521)

[3.2 Instrument to Platform Interfaces 5](#_Toc501199522)

[3.3 Payload Accommodation 5](#_Toc501199523)

[3.4 Alignment 6](#_Toc501199524)

[3.5 Mass Budget 6](#_Toc501199525)

[3.6 Payload Envelope 6](#_Toc501199526)

[3.7 Grounding Strategy 6](#_Toc501199527)

[3.8 Field of View 6](#_Toc501199528)

[3.9 Access 6](#_Toc501199529)

[4 Electrical Power Interface and Design 7](#_Toc501199530)

[4.1 DMB to MirrorSat Interface 7](#_Toc501199531)

[4.2 Instrument Power Supply 8](#_Toc501199532)

[4.3 Power Budget 8](#_Toc501199533)

[4.4 Overcurrent Requirements 9](#_Toc501199534)

[4.5 Payload Current Inrush 10](#_Toc501199535)

[4.6 Grounding Scheme 10](#_Toc501199536)

[5 Thermal 10](#_Toc501199537)

[6 Software Design and Interface 11](#_Toc501199538)

[6.1 Application Software Design 11](#_Toc501199539)

[6.1.1 Software Modes of Operation 11](#_Toc501199540)

[6.1.2 Software Data Rates 12](#_Toc501199541)

[6.1.3 Software Architecture 12](#_Toc501199542)

[6.1.4 Telecommand and Telemetry Points (TC/TM) 14](#_Toc501199543)

[6.1.5 Data Management 32](#_Toc501199544)

[6.1.6 Any other relevant requirements 32](#_Toc501199545)

[6.2 Bootloader Design 33](#_Toc501199546)

[6.2.1 Bootloader Modes of Operation 33](#_Toc501199547)

[6.2.2 Bootloader Data Rates 33](#_Toc501199548)

[6.2.3 Bootloader Architecture 34](#_Toc501199549)

[6.2.4 Telecommand and Telemetry Points (TC/TM) 34](#_Toc501199550)

[6.2.5 Loading New Application Code 36](#_Toc501199551)

[6.3 Instrument Autonomy and FDIR 37](#_Toc501199552)

[6.3.1 Watchdog timer prescaler setting 37](#_Toc501199553)

[6.3.2 Initialization routine 37](#_Toc501199554)

[6.3.3 System reset event description 38](#_Toc501199555)

[7 Harness Interface and Design 39](#_Toc501199556)

[7.1 Instrument to Platform Harness 39](#_Toc501199557)

[7.2 Connectors and Pin-Outs 39](#_Toc501199558)

[8 Instrument Handling & Storage 40](#_Toc501199559)

[8.1 Transport Container 40](#_Toc501199560)

[8.2 Physical Handling 40](#_Toc501199561)

[8.3 Mechanical Ground Support Equipment 40](#_Toc501199562)

[8.4 Electrical Ground Support Equipment 40](#_Toc501199563)

[8.5 Qualification Status 40](#_Toc501199564)

[9 Any other data 40](#_Toc501199565)

**List of Figures**

[Figure 1: DMB Diagram 1](#_Toc501199566)

[Figure 2: CAD model coordinate system with origin on one of the side structures of the box 3](#_Toc501199567)

[Figure 3: Deformable mirror box assembly. Two side walls were purposely removed from this picture to show internal components 4](#_Toc501199568)

[Figure 4: Deformable mirror box/Mirrorsat interfaces. From the left, the first two figures show the external side of the base plate, which directly touches the mirrorsat. The last figure on the right shows the inner side of the base plate with countersink holes to host the M2.5 screws. 5](#_Toc501199569)

[Figure 5: Payload Accommodation to be updated 5](#_Toc501199570)

[Figure 6: (left) Bottom view of the DMB showing the interface connector 7](#_Toc501199571)

[Figure 7: Pin description for the interface connector 8](#_Toc501199572)

[Figure 8: Software Architecture Diagram 12](#_Toc501199573)

[Figure 9: Watchdog Timer Routine Flow Chart 37](file:///C:\Users\Thibaud\Box%20Sync\AAReST\02_MIRRORS\00_ICD\20171113_yw_DeformableMirrorBoxICD.docx#_Toc501199574)

[Figure 10: Health Monitoring Diagram 38](file:///C:\Users\Thibaud\Box%20Sync\AAReST\02_MIRRORS\00_ICD\20171113_yw_DeformableMirrorBoxICD.docx#_Toc501199575)

**List of Tables**

[Table 1: Applicable Documents 2](#_Toc487061916)

[Table 2: Reference Documents 2](#_Toc487061917)

[Table 3: Deformable Mirror Box Power Budget 8](#_Toc487061918)

[Table 4: Software Modes of Operation (Software Dependencies Organized from High to Low Level) 10](#_Toc487061919)

[Table 5: System Timer Interrupts 12](#_Toc487061920)

[Table 6: Schedule Utility Functions 12](#_Toc487061921)

[Table 7: Task Structure 12](#_Toc487061922)

[Table 8: TC/TM Points Detail 13](#_Toc487061923)

[Table 9: List of Commands to the Mirror 13](#_Toc487061924)

[Table 10: Entries of the Register for Telemetry 20](#_Toc487061925)

[Table 11: Error Codes 22](#_Toc487061926)

[Table 13: Function IDs 24](#_Toc487061927)

**Intentionally Blank**

# INTRODUCTION

## Scope

This document is aimed at ensuring the all payloads are described sufficiently to a) highlight any assembly and integration issues before flight model parts are procured, and b) to enable deliverable “MirrorSat” to be completed at a sufficient standard. It is hoped that this document can be used to help assess your own progress too. Below you will find an updated system diagram which I would like to you to confirm is correct for your payload.

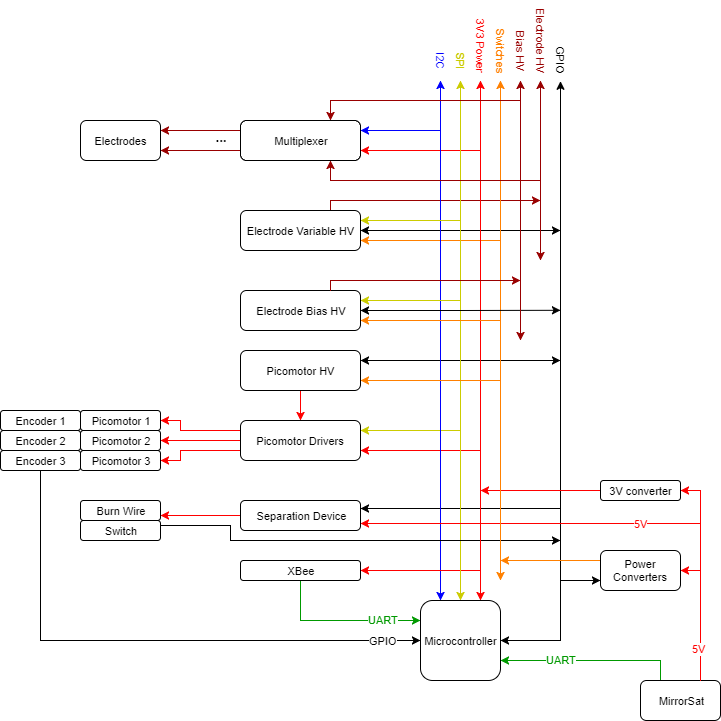


Figure : DMB Diagram

It is anticipated that there will be discrepancies and work is required towards resolving them with you – and to get us on track. This is the interface control document for the **Deformable Mirror Box** for the **AAReST** mission.

## Applicable Documents

Applicable Documents identified in the following text are identified by AD-n, where “n” indicates the actual document, from the following list:

Table : Applicable Documents

You may wish to relate your designs and work to standards. E.g. the PC/104 or CubeSat mech standards or ECSS software standards.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AD#** | **Title** | **Doc #** | **Revision** | **Date** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Reference Documents

Documents referenced in the following text, are identified by RD-n, where “n” indicates the actual document, from the following list:

Table : Reference Documents

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **RD#** | **Title** | **Customer Ref. #** | **Doc #** | **Revision** | **Date** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Acronyms and Abbreviations

The following abbreviations are used within this document:

|  |  |
| --- | --- |
| FDIR | Failure Detection, Isolation and Recovery |
| SSC | Surrey Space Centre |
| DM | Deformable mirror |
| DMB | Deformable Mirror Box |
| GND | Electrical ground |
| UART | Universal Asynchronous Receiver/Transmitter |
| SPI | Serial Peripheral Interface |
| I2C | Inter-Integrated Circuit |
| HV | High-voltage |
| ADC | Analog-to-Digital Converter |
|  |  |

# Deformable Mirror Box

## General Instrument Description

The Deformable Mirror Box (DMB) holds the deformable mirror for the AAReST mission. The primary mirror of AAReST is made out of 4 mirrors: 2 deformable and 2 rigid. The Deformable Mirror Boxes are placed on top of the MirrorSats. The tip/tilt and piston of the mirror can be adjusted through a set of 3 picomotors. Electrodes placed on the mirror can locally correct its shape. The electronics to perform these controls are hosted inside the box.

## Instrument Coordinate System

The current CAD model coordinate system is shown in the Figure 2 below.

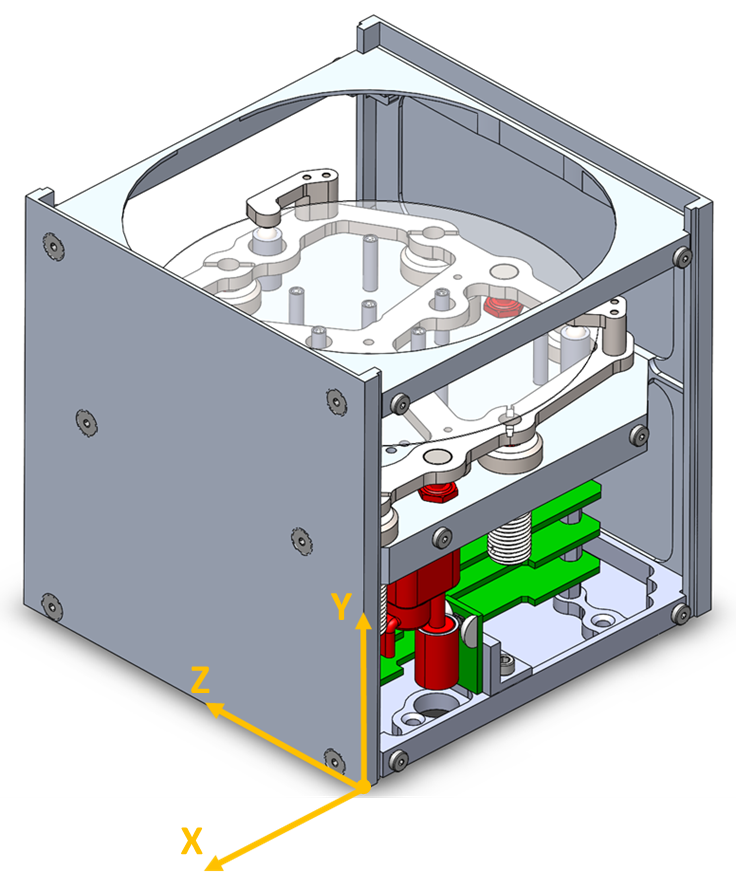


Figure : CAD model coordinate system with origin on one of the side structures of the box

## Operational Modes

Please provide description of operational modes that must be considered and how you expect your payload to operate.

# Mechanical Interface and Design

## Structural Design

* The key purposes of the deformable mirror box are to:
* House the deformable mirror and the electronics.
* Restrain the mirror during launch.
* Provide rigid body rotation and axial motion of the mirror during operations.

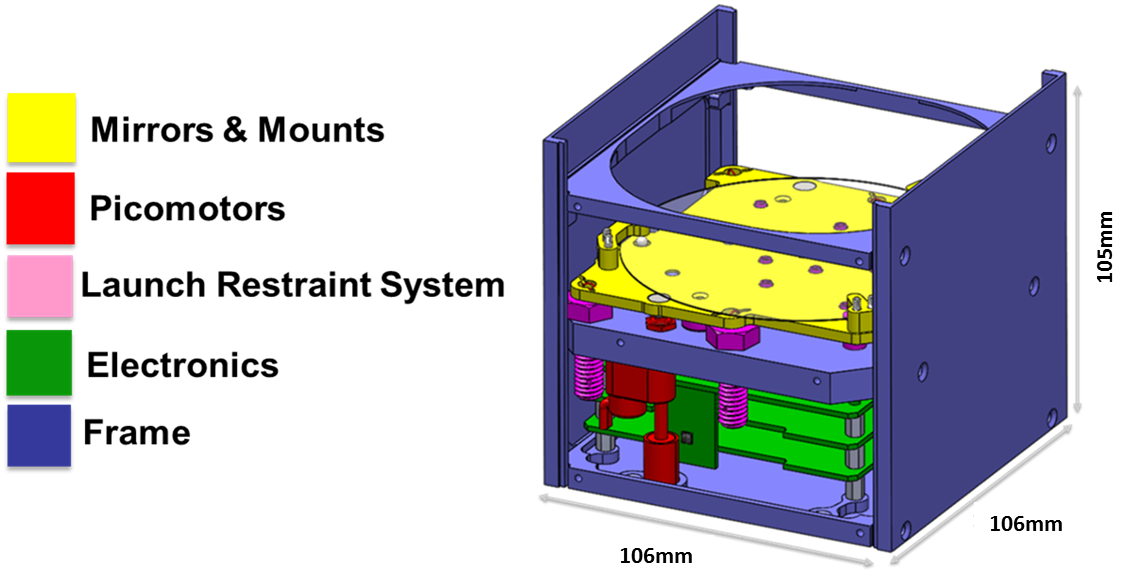


Figure : Deformable mirror box assembly. Two side walls were purposely removed from this picture to show internal components

The main components of the system are:

1. **Mirror mounting plate/mirror mounts:**

Provide support to the deformable mirror, which is held in place by mounts consisting of magnets and ball bearings.

1. **Picomotors:**

Linear actuators that provide piston, tip and tilt motion of the mirror. The picomotors are fully retracted during launch and lift up the mirror plate to begin normal operation.

1. **Electronics:**

Three printed circuit boards, whose architecture and functionality are described in detail in Section 4: Electrical Power Interface and Design.

1. **Frame:**

Provides protection from external environment and structural support to the entire assembly. Consists of two identical side structures (1mm thick), two identical side walls (1mm thick), a base that interfaces with the MirrorSat (1mm thick), a top baffle that bocks stray light, and a reference plate that is the only internal component of the frame (between 1mm and 9.5mm thick).

1. **Launch restraint system:**

Consists of a separation device and soft springs. The separation device, embedded in the reference plate, holds the mirror in place during launch and releases it afterwards. The soft springs restraint the mirror during operations.

## Instrument to Platform Interfaces

The boxes are mounted on the mirrorsats with four of the following screws:

* McMaster-Carr part number 91294A020, black-oxide alloy steel, hex drive, flat-head screw, M2.5 x 0.45 mm thread, 12 mm long.

Figure 4 shows the position of the holes on the base plate, which is in contact with the mirrorsat. The contact area is 106 X 106 mm. The screws are inserted from the mirror box and fastened into the mirrorsat. In fact, a countersink was manufactured on the inner side of the base plate. The assembly procedure requires to remove one side wall at a time, to access the countersink holes, and tighten the screws with a low profile allen wrench:

* McMaster-Carr part number 5503A36, Ball-End L-Key, 1.5 mm Size.

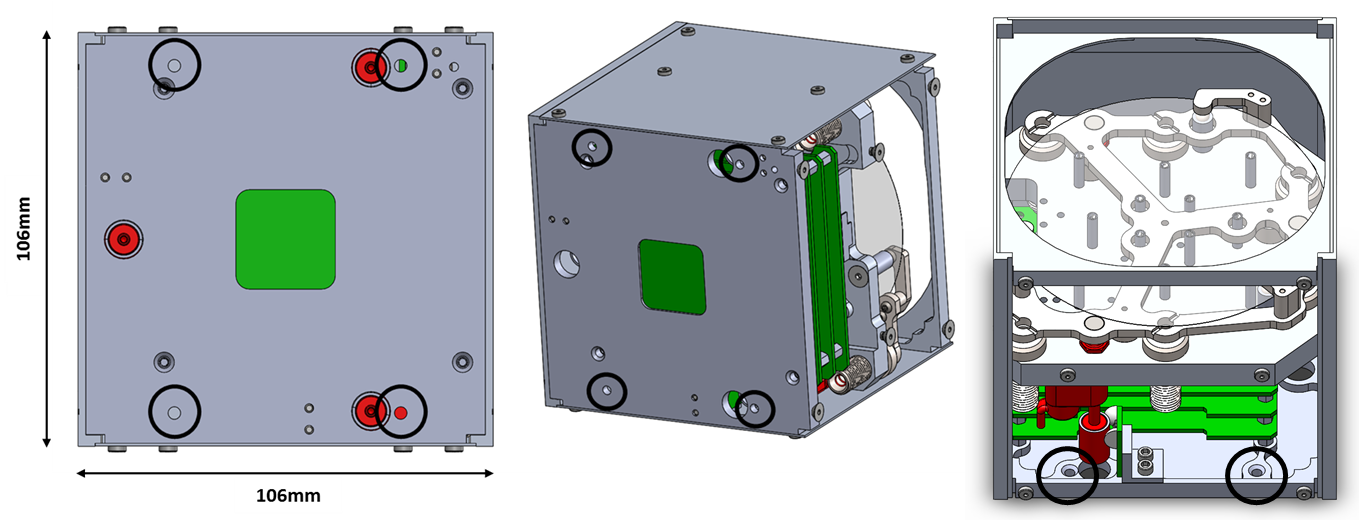


Figure : Deformable mirror box/Mirrorsat interfaces. From the left, the first two figures show the external side of the base plate, which directly touches the mirrorsat. The last figure on the right shows the inner side of the base plate with countersink holes to host the M2.5 screws.

## Payload Accommodation

SSC to provide updated images of where payload is accommodated – for present Figure 5 shows the current design.

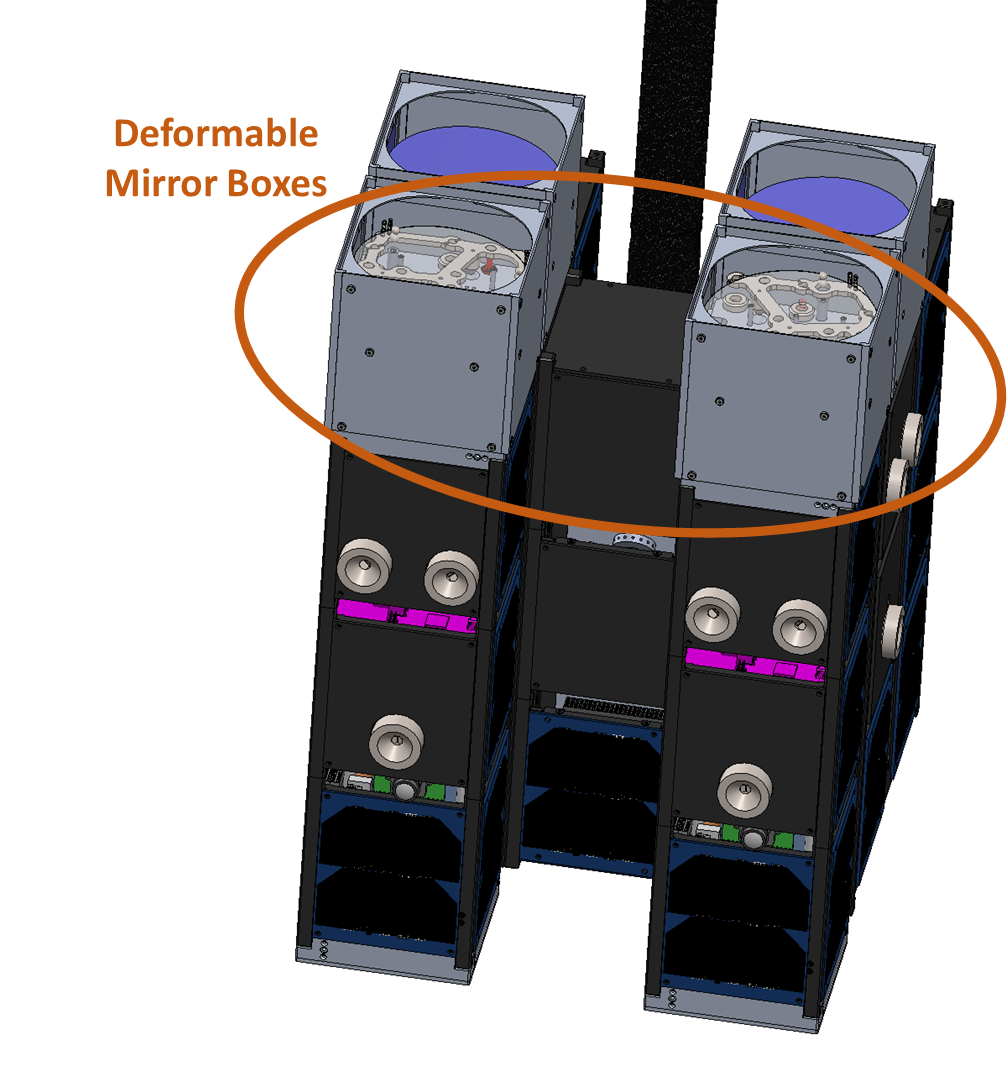


Figure : Payload Accommodation to be updated

## Alignment

The deformable mirrors will be optically aligned after integration on optical testbed at Caltech.

## Mass Budget

From the CAD model we can get the information reported below. Some components are still missing in the CAD, hence the information is **inaccurate** at this stage.

* **Mass** = 472.54 grams
* **Volume** = 190932.19 cubic millimeters
* **Surface area** = 281813.61 square millimeters
* **Center of mass**: (millimeters)

X = -52.40

Y = 52.59

Z = 52.53

* **Principal axes of inertia and principal moments of inertia taken at the center of mass**: (grams \* square millimeters)

Ix = (-0.33, 0.00, 0.94) Px = 891029.29

Iy = (0.94, -0.03, 0.33) Py = 930355.86

Iz = (0.03, 1.00, 0.01) Pz = 1244928.11

## Payload Envelope

The payload unit is a box of 106 X 106 X 105 mm. Refer to figure in section 3.1: Structural Design.

## Grounding Strategy

All the boards in the DMB share the same ground that is communicated through the headers that are used for stacking the boards. The digital ground, which is same as the ground coming in from the MirrorSat power supply is not connected to the DMB chassis at any point.

## Field of View

Please provide any FoV clearances that may be needed for payload operations.

## Access

The box is assembled by removing one side wall at a time and tightening four screws from the inside of the box to the mirrorsat. This procedure requires at least 100mm of clearance in front of each side wall in order to mount the box to the mirrorsat. For this reason, each box should be first assembled on its mirrorsat and in a second step the mirrorsat/mirror box assembly should be integrated to the coresat.

# Electrical Power Interface and Design

The deformable mirror box (DMB) and the MirrorSat are electrically connected using a 4-wire interface. Two of these wires carry power (+5V and GND) while the other two wires are for UART communication. The UART interface is only meant to serve as a means for backup communication in case the wireless link between the camera and the DMB stops working. The DMB comprises a stack of 3 boards. From top to bottom, these are

1. Multiplexer board for distributing electrode voltages
2. HV board for generating the high voltages for the mirror and the picomotors
3. Microcontroller board for controlling the various processes, power conditioning, release mechanism etc.

The interface connector for the MirrorSat is located on the microcontroller board.

## DMB to MirrorSat Interface

The M80-5130405 connector from Harwin is provided on the microcontroller board in the DMB for interfacing with the MirrorSat. This is a 4-pin, 2-row, 2.0 mm pitch connector with jackscrews for mating with the connector on the harness. The connector can be accessed through the 1” X 1” hole in the bottom plate on the DMB. The exact location and dimensions can be found in the SolidWorks assembly file. Figure 6 shows the bottom and side view of the DMB, highlighting the location of the connector.

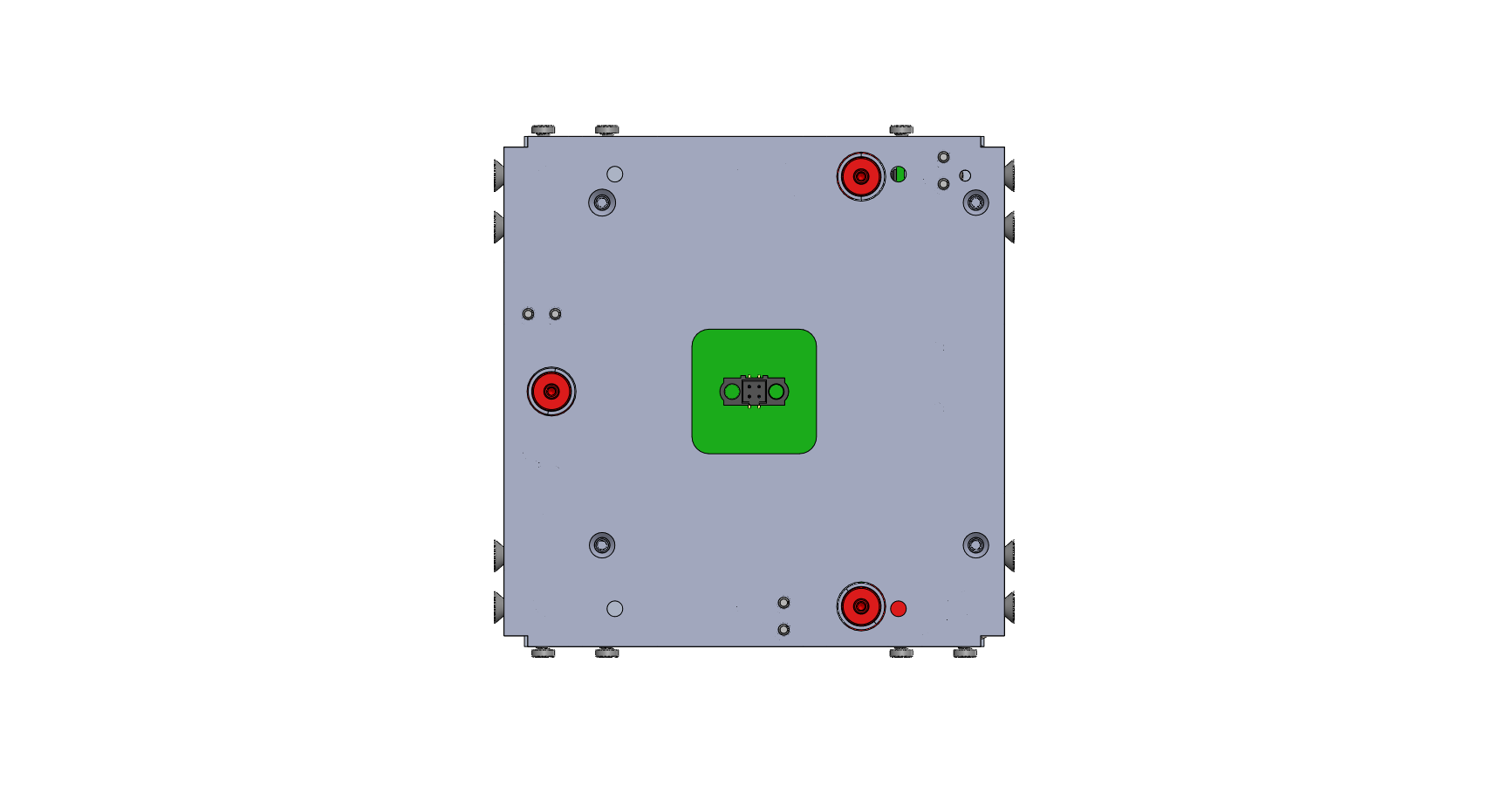
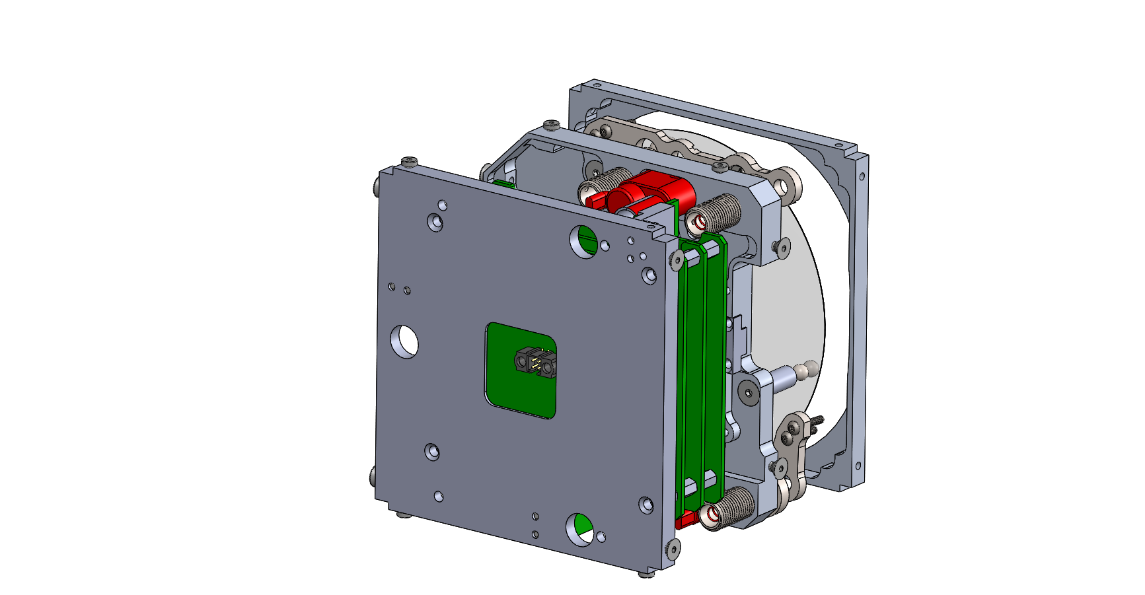
 

Figure : (left) Bottom view of the DMB showing the interface connector

Figure 7 contains the connectivity diagram for the connector. The DMB receives data from the MirrorSat on the RX pin and transmits data on the TX pin. Note that the microcontroller operates at 3.3 V and hence the UART must operate using 3.3 V logic. Therefore, the voltage on the RX and TX pins should nominally vary between 0 and 3.3 V only.

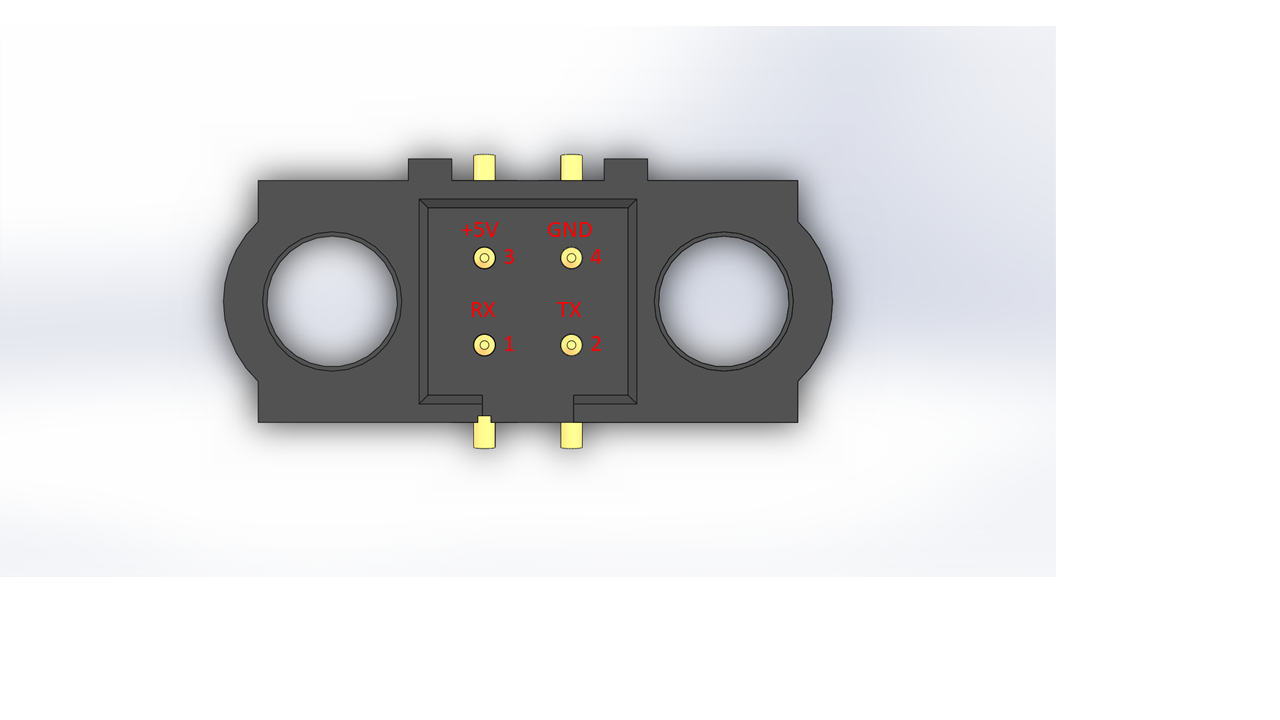


Figure : Pin description for the interface connector

## Instrument Power Supply

The DMB needs regulated 5 V power supply from the MirrorSat. Acceptable input voltage range is 4.5 V to 5.5 V. The current requirement varies depending on the mode of operation. Highest currents are needed when heating the nichrome wire to burn the Vectran cable for releasing the mirror after launch. The nominal power consumption is low (~ 1 W) but more currents are needed when driving the mirror at high voltages and while moving the picomotors. This information is captured in Table 4.

## Power Budget

The power budget of the DMB is as shown in Table 3.

Table : Deformable Mirror Box Power Budget

|  |  |  |
| --- | --- | --- |
| **Mode** | **Duration** | **Current / Power** |
| Activate release mechanism | ~ 10 s | 0.7 A / 3.5 W |
| Nominal mode (no actuation of mirror or picomotors) |  | 0.2 A / 1W |
| Mirror Actuation | ~ 10 min (duration of science mode of AAReST) | 0.60 A / 3 W |
| Picomotor actuation | <1 s to ~ 1 min | 0.60 A / 3 W |

## Overcurrent Requirements

There is overcurrent protection built into the high voltage electronics. The DMB does not impose any specific overcurrent protection requirements on the MirrorSat. The MirrorSat designers may choose to incorporate their own overcurrent protection circuits, provided they can meet the power requirements outlined in Table 4 with sufficient margin.

## Payload Current Inrush

Please provide figures, graphs and results of current inrush tests on the payload.

## Grounding Scheme

All the boards in the DMB share the same ground that is communicated through the headers that are used for stacking the boards. The digital ground, which is same as the ground coming in from the MirrorSat power supply is not connected to the DMB chassis at any point.

# Thermal

In this section, we would like to know what your expected limits are on your payload and what risk strategy you have in the event of failure during EVT at SSC, and on orbit.

# Software Design and Interface

## Application Software Design

### Software Modes of Operation

Table : Software Modes of Operation

|  |  |  |  |
| --- | --- | --- | --- |
| Mode | Function | Software Dependencies | Hardware Dependencies |
| Activate release mechanism | Release the Vectran cable restraining the mirror during launch | Timer  Watchdog  UART I2C  Internal EEPROM | Voltage supply  Cutting wire  Limit switch |
| Nominal/Safe mode | No actuation | Timer  Watchdog  UART I2C  Internal EEPROM | Temperature sensors |
| Electrode (Mirror) actuation | Actuate the electrodes of the mirror, one at a time | Timer  Watchdog  UART SPI  I2C  ADC  Internal EEPROM | 12V, 5V supplies  Variable HV  Bias HV  Multiplexer board  Electrodes (x41) |
| Picomotor actuation | Actuate the three picomotors | Timer  Watchdog  UART SPI  I2C  ADC  Internal EEPROM | 12V. 2.8V supplies  Picomotor HV  Picomotor actuation board  Encoders (x3)  Picomotors (x3) |

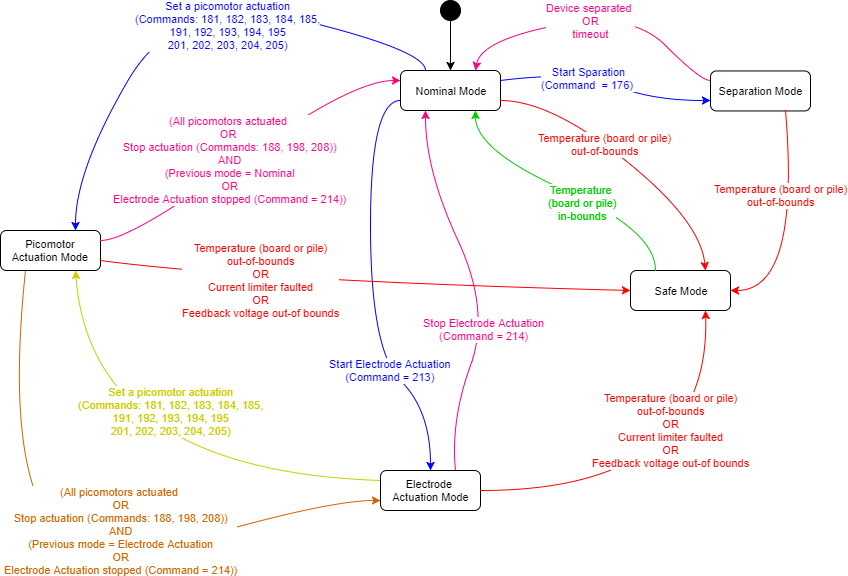


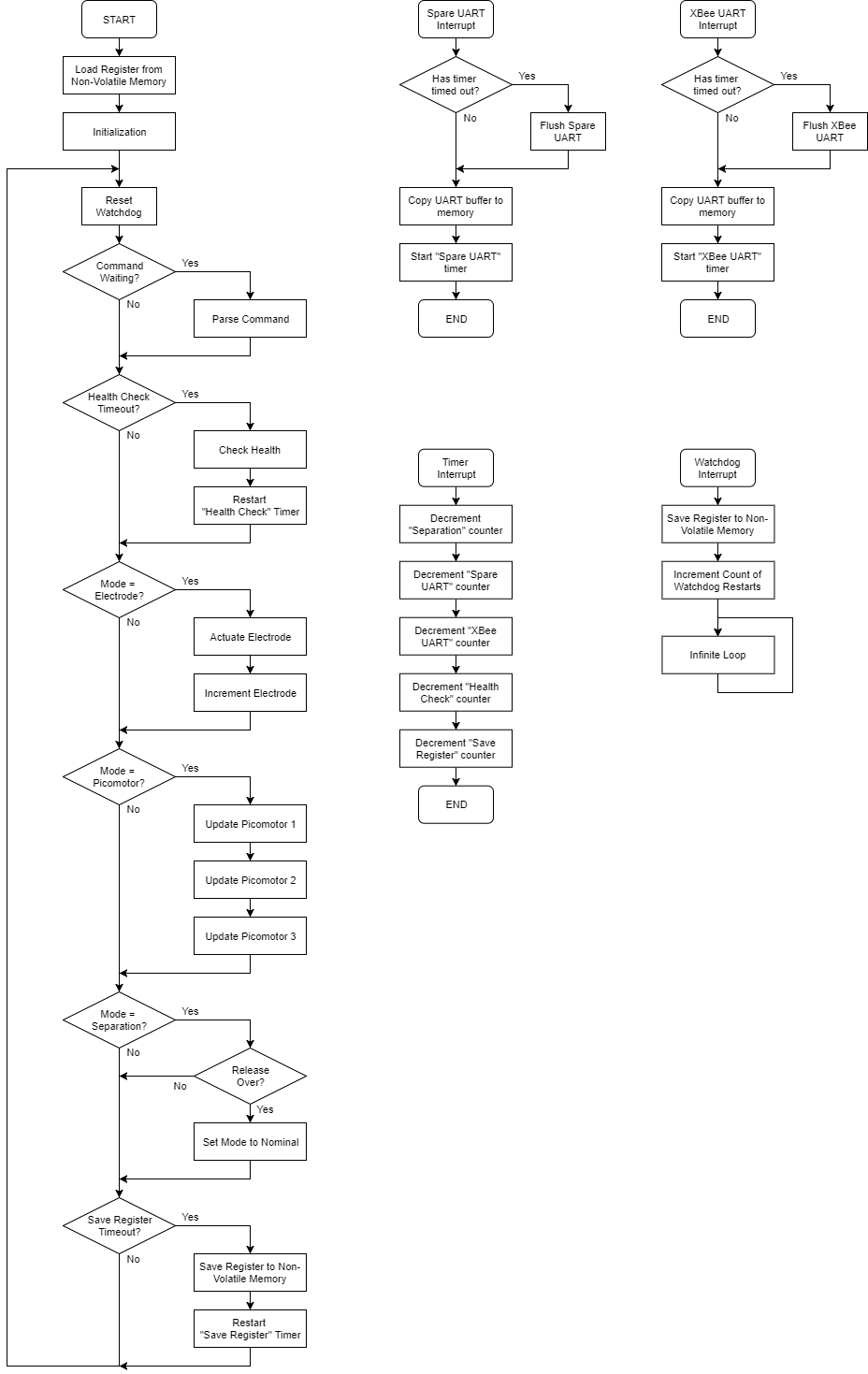
Figure - State Diagram

### Software Data Rates

Data rates for the UART interface:

* Baud rate: **9600**
* Parity: **None**
* Data Bits: **8 bits**
* Stop Bits: **1 bit**
* Hardware Flow control: **None**

### Software Architecture



### Telecommand and Telemetry Points (TC/TM)

#### TCTM Format

Command  
(1 byte)

Checksum  
(1 byte)

Data  
(4 byte)

Error  
(1 byte)

Table : TC/TM Points Detail

|  |  |  |
| --- | --- | --- |
| Name | Utility | Note |
| Command | See Table |  |
| Data | Data to be passed to the function. See Table | Type corresponding to the 4 bytes can be uint32\_t, int32\_t, or float |
| Error | Error returned from function. See Table | Only used in message from the DMB. |
| Checksum | Parity check | Sum of all (7) bytes of the message add up to 255 modulo 256 (0xFF) |

#### Telecommands

* Command and Data are listed in the table below
* Each command is acknowledged as described in the table
* To READ telemetry, the command is 150
* 1 – 130 corresponds to the entry of the register that saves parameters and telemetry data
* The error byte is discarded in the command message

Table : List of Commands to the Mirror

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Data *(data type)* range *(default)* | Description | Acknowledgment |
| 1 | *(uint32\_t)*  0 – 1 *(0)* | Define the boot mode (“BOOT\_SAFE“) 0 = Normal (Default) 1 = User driven It is set to 1 by the bootloader and to 0 by the application code once a successful ping command has been received and answered. | **Command:** 1  **Data:** Set value  **Error:** |
| 2 | *(uint32\_t)*  0 – 3 *(0)* | Set address of the code for boot up | **Command:** 2  **Data:** Set value  **Error:** 0 |
| 3 | *(uint32\_t)*  1,200 - 460,800 *(9,600)* | Set Baud rate of Spare UART set during start-up (data in bits per seconds) | **Command:** 3  **Data:** Set value  **Error:** |
| 4 - 6 | READ-ONLY | | |
| 7 | *(uint32\_t)*  1,200 - 460,800 *(9,600)* | Set Baud rate of XBee UART set during start-up (data in bits per seconds) | **Command:** 7  **Data:** Set value  **Error:** |
| 8 - 10 | READ-ONLY | | |
| 11 | *(uint32\_t)*  40,000 – 4,000,000 *(4,000,000)* | Set Frequency of SPI set during start-up (data in bits per seconds) | **Command:** 11  **Data:** Set value  **Error:** |
| 12 | READ-ONLY | | |
| 13 | *(uint32\_t)*  100,000 -400,000 *(400,000)* | Set Frequency of I2C set during start-up (data in bits per seconds) | **Command:** 13  **Data:** Set value  **Error:** |
| 14 | *(uint32\_t)*  0 – 100 *(3)* | Set Maximum number of restarts when device is not acknowledging | **Command:** 14  **Data:** Set value  **Error:** |
| 15 - 18 | READ-ONLY | | |
| 19 | *(uint32\_t)*  5,000 – 200,000 *(200,000)* | Set Frequency of ADC set during start-up (data in bits per seconds) | **Command:** 19  **Data:** Set value  **Error:** |
| 20 - 24 | READ-ONLY | | |
| 25 | *(uint32\_t)*  0 – 60,000 *(100)* | Set Communication timeout (time to wait for a full message on either UART in milliseconds) | **Command:** 25  **Data:** Set value  **Error:** |
| 26 -27 | READ-ONLY | | |
| 28 | *(uint32\_t)*  0 – 512 *(50)* | Set tolerance to measure a feedback voltage | **Command:** 28  **Data:** Set value  **Error:** |
| 29 - 33 | READ-ONLY | | |
| 34 | *(int16\_t)*  -7,040 – 12,800 *(-7,040)* | Minimum allowable temperature on the microcontroller board | **Command:** 34  **Data:** Set value  **Error:** |
| 35 | *(int16\_t)*  -7,040 – 12,800 *(12,800)* | Maximum allowable temperature on the microcontroller board | **Command:** 35  **Data:** Set value  **Error:** |
| 36 | *(int16\_t)*  -7,040 – 12,800 *(-7,040)* | Minimum allowable temperature on the HV board | **Command:** 36  **Data:** Set value  **Error:** |
| 37 | *(int16\_t)*  -7,040 – 12,800 *(12,800)* | Maximum allowable temperature on the HV board | **Command:** 37  **Data:** Set value  **Error:** |
| 38 | *(int16\_t)*  -5,120 – 12,800 *(-5,120)* | Minimum allowable temperature on the thermopile 1 | **Command:** 38  **Data:** Set value  **Error:** |
| 39 | *(int16\_t)*  -5,120 – 12,800 *(12,800)* | Maximum allowable temperature on the thermopile 1 | **Command:** 39  **Data:** Set value  **Error:** |
| 40 | *(int16\_t)*  -5,120 – 12,800 *(-5,120)* | Minimum allowable temperature on the thermopile 2 | **Command:** 40  **Data:** Set value  **Error:** |
| 41 | *(int16\_t)*  -5,120 – 12,800 *(12,800)* | Maximum allowable temperature on the thermopile 2 | **Command:** 41  **Data:** Set value  **Error:** |
| 42 | *(int16\_t)*  -5,120 – 12,800 *(-5,120)* | Minimum allowable temperature on the thermopile 3 | **Command:** 42  **Data:** Set value  **Error:** |
| 43 | *(int16\_t)*  -5,120 – 12,800 *(12,800)* | Maximum allowable temperature on the thermopile 3 | **Command:** 43  **Data:** Set value  **Error:** |
| 44 | *(uint32\_t)*  0 – 1,000,000 *(1,000)* | Set maximum number of ticks when moving a picomotor | **Command:** 44  **Data:** Set value  **Error:** |
| 45 – 53 | READ-ONLY | | |
| 54 | *(float)* 0.0 - 1.0 | Set Mean value of a positive picomotor 1 step (relative to the size of an encoder 1 interval) | **Command:** 54  **Data:** Set value  **Error:** |
| 55 | *(float)* 0.0 - 1.0 | Set Mean value of a positive picomotor 2 step (relative to the size of an encoder 2 interval) | **Command:** 55  **Data:** Set value  **Error:** |
| 56 | *(float)* 0.0 - 1.0 | Set Mean value of a positive picomotor 3 step (relative to the size of an encoder 3 interval) | **Command:** 56  **Data:** Set value  **Error:** |
| 57 | *(float)* 0.0 - 1.0 | Set Standard deviation of a positive picomotor 1 step (relative to the size of an encoder 1 interval) | **Command:** 57  **Data:** Set value  **Error:** |
| 58 | *(float)* 0.0 - 1.0 | Set Standard deviation of a positive picomotor 2 step (relative to the size of an encoder 2 interval) | **Command:** 58  **Data:** Set value  **Error:** |
| 59 | *(float)* 0.0 - 1.0 | Set Standard deviation of a positive picomotor 3 step (relative to the size of an encoder 3 interval) | **Command:** 59  **Data:** Set value  **Error:** |
| 60 | *(float)* -0.0 to -1.0 | Set Mean value of a negative picomotor 1 step (relative to the size of an encoder 1 interval) | **Command:** 60  **Data:** Set value  **Error:** |
| 61 | *(float)* -0.0 to -1.0 | Set Mean value of a negative picomotor 2 step (relative to the size of an encoder 2 interval) | **Command:** 61  **Data:** Set value  **Error:** |
| 62 | *(float)* -0.0 to -1.0 | Set Mean value of a negative picomotor 3 step (relative to the size of an encoder 3 interval) | **Command:** 62  **Data:** Set value  **Error:** |
| 63 | *(float)* 0.0 - 1.0 | Set Standard deviation of a negative picomotor 1 step (relative to the size of an encoder 1 interval) | **Command:** 63  **Data:** Set value  **Error:** |
| 64 | *(float)* 0.0 - 1.0 | Set Standard deviation of a negative picomotor 2 step (relative to the size of an encoder 2 interval) | **Command:** 64  **Data:** Set value  **Error:** |
| 65 | *(float)* 0.0 - 1.0 | Set Standard deviation of a negative picomotor 3 step (relative to the size of an encoder 3 interval) | **Command:** 65  **Data:** Set value  **Error:** |
| 66 | *(uint32\_t)*  0 | Reset errors related with the actuation of picomotor 1 | **Command:** 66  **Data:** 0  **Error:** |
| 67 | *(uint32\_t)*  0 | Reset errors related with the actuation of picomotor 2 | **Command:** 67  **Data:** 0  **Error:** |
| 68 | *(uint32\_t)*  0 | Reset errors related with the actuation of picomotor 3 | **Command:** 68  **Data:** 0  **Error:** |
| 69 | *(uint32\_t)*  0 – 1,000 *(15)* | Set duration for variable electrode HV to stabilize (data milliseconds) | **Command:** 69  **Data:** Set value  **Error:** |
| 70 | *(uint32\_t)*  0 – 8114 *(8088)* | Set voltage limit of the electrodes from ground | **Command:** 70  **Data:** Set value  **Error:** |
| 71 | *(uint32\_t)*  0 – 1,000 *(337)* | Set voltage step when initializing the mirror | **Command:** 71  **Data:** Set value  **Error:** |
| 72 | *(uint32\_t)*  0 – 16,000  *(8191)* | Set bias voltage | **Command:** 72  **Data:** Set value  **Error:** |
| 73 | READ-ONLY | | |
| 74 - 114 | **First 2 bytes:**  Time duration to activate electrode [ms]  *(uint16\_t)*  0 -255  **Last 2 bytes:** Voltage of the electrode  *(uint16\_t)* 0 – 16 229 | Set Data for electrode 1 – 41 (respectively)  Example:  data = 0x000A1FFF  sets the electrode to 0V for 10 ms | **Command:** 74 – 114 (respectively)  **Data:** Set value  **Error:** |
| 115 | *(uint32\_t)*  0 | Reset the errors associated with the electrodes actuation | **Command:** 115  **Data:** 0  **Error:** |
| 116 | READ-ONLY | | |
| 117 | *(uint32\_t)*  0 - 100 | Number of errors that set the Safe Mode | **Command:** 117  **Data:** Set value  **Error:** |
| 118 | READ-ONLY | | |
| 119 | *(uint32\_t)*  0 | Reset the errors that are triggering Safe Mode | **Command:** 119  **Data:** 0  **Error:** |
| 120 | *(uint32\_t)*  0 – 600,000 *(30,000)* | Duration between each automatic health check of the electronics (in milliseconds) | **Command:** 120  **Data:** Set value  **Error:** |
| 121 | *(uint32\_t)*  0 – 3,600,000 *(60,000)* | Duration between each automatic save operation of the register to non-volatile memory (in milliseconds) | **Command:** 121  **Data:** Set value  **Error:** |
| 122 | *(uint32\_t)*  0 | Reset the errors related with the automatic health check of the electronics | **Command:** 122  **Data:** 0  **Error:** |
| 123 – 127 | READ-ONLY | | |
| 128 | *(uint32\_t)*  0 | Reset the errors related with the parsing of telecommands | **Command:** 128  **Data:** 0  **Error:** |
| 129 – 130 | READ-ONLY | | |
| 150 | **READ REGISTER. SEE Table 10** | | |
| 151 | *(uint32\_t)*  1,200 - 460,800 *(9,600)* | Re-initialize UART with MirrorSat with baud rate equal to *data* [bps] | **Command:** 151  **Data:** Baud rate  **Error:** |
| 152 | *(uint32\_t)*  1,200 - 460,800 *(9,600)* | Re-initialize UART with XBee with baud rate equal to *data* [bps] | **Command:** 152  **Data:** Baud rate  **Error:** |
| 153 | *(uint32\_t)*  40,000 – 4,000,000 *(4,000,000)* | Re-initialize SPI with frequency equal to *data* [bps] | **Command:** 153  **Data:** Frequency  **Error:** |
| 154 | *(uint32\_t)*  100,000 -400,000 *(400,000)* | Re-initialize I2C with frequency equal to *data* [bps] | **Command:** 154  **Data:** Frequency  **Error:** |
| 155 | *(uint8\_t)* 0 – 254 | Test I2C device ACK with address equal to *data* (LSB defines read / write operation) | **Command:** 155  **Data:** Address  **Error:** |
| 156 | *(uint32\_t)*  5,000 – 200,000 *(200,000)* | Re-initialize ADC with frequency equal to *data* [Hz] | **Command:** 156  **Data:** Frequency  **Error:** |
| 157 | *(uint32\_t)*  0 – 60,000 *(1,000)* | Re-initialize the communication with MirrorCraft and XBee with timeout equal to *data* [ms] | **Command:** 157  **Data:** Timeout [ms]  **Error:** |
| 160 | *discarded* | Re-initialize the power lines | **Command:** 160  **Data:** 0  **Error:** |
| 161 | *(uint8\_t)* 0, 1, or 2 | Enable supply voltage:  0 = 5V  1 = 12V  2 = 2.8 V | **Command:** 161  **Data:** 0  **Error:** |
| 162 | *(uint8\_t)* 0, 1, or 2 | Disable supply voltage:  0 = 5V  1 = 12V  2 = 2.8 V | **Command:** 162  **Data:** 0  **Error:** |
| 163 | *(uint8\_t)* 0, 1, 2, 3, or 4 | Measure feedback voltage: 0 = Variable HV  1 = Bias HV  2 = Picomotor HV  3 = Bus current  4 = Bus voltage | **Command:** 163  **Data:** *(uint16\_t)* Image of the measure voltage  **Error:** |
| 164 | **First byte:** Index of the line  *(uint8\_t)* 0, 1, 2, 3, or 4  **Middle 2 bytes:**  Target voltage:  *(uint16\_t)* 0 – 1024  *(776)*  **Last byte:** Maximum number of tries  *(uint8\_t)* 0 – 255  *(50)* | Tries (last byte) number of times to check if the feedback voltage #(first byte) is out-of-bounds (compared to (middle 2 bytes) voltage):  First byte: 0 = Variable HV  1 = Bias HV  2 = Picomotor HV  3 = Bus current  4 = Bus voltage  Middle 2 bytes:  Note: This function return false if the measured voltage within the tolerance (Register entry 28) from the target voltage | **Command:** 164  **Data:** *(bool)* true is voltage is out-of- bounds,  false if the voltage is in- bound  **Error:** |
| 165 | *(uint8\_t)* 0, 1, or 2 | Enable current limiter  0 = CL1  1 = CL2  2 = CL3 | **Command:** 165  **Data:** 0  **Error:** |
| 166 | *(uint8\_t)* 0, 1, or 2 | Disable current limiter  0 = CL1  1 = CL2  2 = CL3 | **Command:** 166  **Data:** 0  **Error:** |
| 167 | *(uint8\_t)* 0, 1, or 2 | Check if current limiter has faulted  0 = CL1  1 = CL2  2 = CL3 | **Command:** 167  **Data:** *(bool)* true = yes  false = no  **Error:** |
| 168 | *discarded* | Activate the electrode HV lines | **Command:** 168  **Data:** 0  **Error:** |
| 169 | *discarded* | Deactivate the electrode HV lines | **Command:** 169  **Data:** 0  **Error:** |
| 170 | *(uint16\_t)* 0 – 16 229 | Set voltage of variable HV manually (actual voltage seen by electrode is the difference with the bias voltage): | **Command:** 170  **Data:** 0  **Error:** |
| 171 | *(uint16\_t)* 0 – 16 229 | Set voltage of bias HV manually | **Command:** 171  **Data:** 0  **Error:** |
| 172 | *discarded* | Activate the picomotor HV line | **Command:** 172  **Data:** 0  **Error:** |
| 173 | *discarded* | Deactivate the picomotor HV line | **Command:** 173  **Data:** 0  **Error:** |
| 175 | *discarded* | Re-initialize the separation device | **Command:** 175  **Data:** 0  **Error:** |
| 176 | *(uint32\_t)* 0 – 4,294,967,295 | Start cutting the vectran cable for a maximum of *data* milliseconds  *Note: The user needs to probe the end of deployment with command 177)* | **Command:** 176  **Data:** 0  **Error:** |
| 177 | *discarded* | Check if mirror is still constrained | **Command:** 177  **Data:** (bool) true = constrained  false = released  **Error:** |
| 180 | *discarded* | Re-initialize the picomotor actuation | **Command:** 180  **Data:** 0  **Error:** |
| 181 | *(int32\_t)* -150,000 to 150,000 | Move picomotor 1 by *data* ticks  *Note: Need to check for end of actuation (Command = 187)* | **Command:** 181  **Data:** 0  **Error:** |
| 182 | *(int32\_t)* -75,000 to 75,000 | Move picomotor 1 by *data* intervals of its encoder.  *Note: Need to check for end of actuation (Command = 187)* | **Command:** 182  **Data:** 0  **Error:** |
| 183 | *(float)* -75,000 to 75,000 | Set position of picomotor 1 to the absolute position *data* [in intervals]  *Note: Need to check for end of actuation (Command = 187)* | **Command:** 183  **Data:** 0  **Error:** |
| 184 | *(uint32\_t)* 0 to 150,000 | Initialize position of picomotor 1 (with a maximum of *data* ticks of actuation)  *Note: Need to check for end of actuation (Command = 187)* | **Command:** 184  **Data:** 0  **Error:** |
| 185 | *(int16\_t)* -1,000 to 1,000 | Re-calibrate picomotor 1 using *data* intervals of its encoder (each direction of actuation has a different calibration)  *Note: Need to check for end of actuation (Command = 187)* | **Command:** 185  **Data:** 0  **Error:** |
| 186 | *discarded* | Measure the state of the encoder 1 of picomotor 1 | **Command:** 186  **Data:** state (0, 1, 2, 3)  **Error:** |
| 187 | *discarded* | Check if actuation of picomotor 1 is done | **Command:** 187  **Data:** (bool) true = actuation done, false = actuation running  **Error:** |
| 188 | *discarded* | Stop actuation of picomotor 1 | **Command:** 188  **Data:** 0  **Error:** |
| 191 | *(int32\_t)* -150,000 to 150,000 | Move picomotor 2 by *data* ticks  *Note: Need to check for end of actuation (Command = 197)* | **Command:** 191  **Data:** 0  **Error:** |
| 192 | *(int32\_t)* -75,000 to 75,000 | Move picomotor 2 by *data* intervals of its encoder.  *Note: Need to check for end of actuation (Command = 197)* | **Command:** 192  **Data:** 0  **Error:** |
| 193 | *(float)* -75,000 to 75,000 | Set position of picomotor 2 to the absolute position *data* [in intervals]  *Note: Need to check for end of actuation (Command = 197)* | **Command:** 193  **Data:** 0  **Error:** |
| 194 | *(uint32\_t)* 0 to 150,000 | Initialize position of picomotor 2 (with a maximum of *data* ticks of actuation)  *Note: Need to check for end of actuation (Command = 197)* | **Command:** 194  **Data:** 0  **Error:** |
| 195 | *(int16\_t)* -1,000 to 1,000 | Re-calibrate picomotor 2 using *data* intervals of its encoder (each direction of actuation has a different calibration)  *Note: Need to check for end of actuation (Command = 197)* | **Command:** 195  **Data:** 0  **Error:** |
| 196 | *discarded* | Measure the state of the encoder 2 of picomotor 2 | **Command:** 196  **Data:** state (0, 1, 2, 3)  **Error:** |
| 197 | *discarded* | Check if actuation of picomotor 2 is done | **Command:** 197  **Data:** (bool) true = actuation done, false = actuation running  **Error:** |
| 198 | *discarded* | Stop actuation of picomotor 2 | **Command:** 198  **Data:** 0  **Error:** |
| 201 | *(int32\_t)* -150,000 to 150,000 | Move picomotor 3 by *data* ticks  *Note: Need to check for end of actuation (Command = 207)* | **Command:** 201  **Data:** 0  **Error:** |
| 202 | *(int32\_t)* -75,000 to 75,000 | Move picomotor 3 by *data* intervals of its encoder.  *Note: Need to check for end of actuation (Command = 207)* | **Command:** 202  **Data:** 0  **Error:** |
| 203 | *(float)* -75,000 to 75,000 | Set position of picomotor 3 to the absolute position *data* [in intervals]  *Note: Need to check for end of actuation (Command = 207)* | **Command:** 203  **Data:** 0  **Error:** |
| 204 | *(uint32\_t)* 0 to 150,000 | Initialize position of picomotor 3 (with a maximum of *data* ticks of actuation)  *Note: Need to check for end of actuation (Command = 207)* | **Command:** 204  **Data:** 0  **Error:** |
| 205 | *(int16\_t)* -1,000 to 1,000 | Re-calibrate picomotor 3 using *data* intervals of its encoder (each direction of actuation has a different calibration)  *Note: Need to check for end of actuation (Command = 207)* | **Command:** 205  **Data:** 0  **Error:** |
| 206 | *discarded* | Measure the state of the encoder 3 of picomotor 3 | **Command:** 206  **Data:** state (0, 1, 2, 3)  **Error:** |
| 207 | *discarded* | Check if actuation of picomotor 3 is done | **Command:** 207  **Data:** (bool) true = actuation done, false = actuation running  **Error:** |
| 208 | *discarded* | Stop actuation of picomotor 3 | **Command:** 208  **Data:** 0  **Error:** |
| 209 | *discarded* | Check if actuation of all picomotors is done | **Command:** 209  **Data:** (bool) true = actuation done, false = actuation running  **Error:** |
| 210 | *discarded* | Reset the electrode actuation functions | **Command:** 210  **Data:** 0  **Error:** |
| 211 | *(uint8\_t)* 0-40 | Connect electrode # *data* to HV line | **Command:** 211  **Data:** 0  **Error:** |
| 212 | *(uint8\_t)* 0-40 | Disconnect electrode # *data* to HV line | **Command:** 212  **Data:** 0  **Error:** |
| 213 | *discarded* | Start the electrode actuation | **Command:** 213  **Data:** 0  **Error:** |
| 214 | *discarded* | Stop the electrode actuation | **Command:** 214  **Data:** 0  **Error:** |
| 215 | *(uint8\_t)*  0, 1, 2, 3, 4, or 5 | Reset temperature sensors  0 = all sensors  1 = board sensor 1  2 = board sensor 2  3 = thermopile 1  4 = thermopile 2  5 = thermopile 3 | **Command:** 215  **Data:** 0  **Error:** |
| 216 | *(uint8\_t)*  0, or 1 | Check if board temperature sensor # *data* is active | **Command:** 216  **Data:** (bool) true = active, false = inactive  **Error:** |
| 217 | *(uint8\_t)*  0, or 1 | Enable board temperature sensor # *data* (for health check function) | **Command:** 217  **Data:** 0  **Error:** |
| 218 | *(uint8\_t)*  0, or 1 | Disable board temperature sensor # *data* (for health check function) | **Command:** 218  **Data:** 0  **Error:** |
| 219 | *(uint8\_t)*  0, or 1 | Measure temperature from board sensor # *data* | **Command:** 219  **Data:** (int16\_t) temperature \* 128 (in Celsius)  **Error:** |
| 220 | *(uint8\_t)*  0, 1 or 2 | Check if thermopile # *data* is active | **Command:** 220  **Data:** (bool) true = active, false = inactive  **Error:** |
| 221 | *(uint8\_t)*  0, 1 or 2 | Enable thermopile # *data* (for health check function) | **Command:** 221  **Data:** 0  **Error:** |
| 222 | *(uint8\_t)*  0, 1 or 2 | Disable thermopile # *data* (for health check function) | **Command:** 222  **Data:** 0  **Error:** |
| 223 | *(uint8\_t)*  0, 1 or 2 | Measure temperature from board sensor # *data* | **Command:** 219  **Data:** (int16\_t) temperature \* 128 (in Celsius)  **Error:** |
| 225 | *(uint8\_t)* 0 – 3 | Save register to internal EEPROM memory, slot #*data* | **Command:** 225  **Data:** 0  **Error:** |
| 226 | **First 2 bytes:**  Memory slot of the internal EEPROM  *(uint8\_t)* 0 – 1  **Last 2 bytes:**  Register parameter to save:  *(uint16\_t)* 0 – 130 | Save value of the register to internal EEPROM memory | **Command:** 226  **Data:** 0  **Error:** |
| 227 | *(uint8\_t)* 0 – 3 | Load register from internal EEPROM memory, slot #*data* | **Command:** 227  **Data:** 0  **Error:** |
| 228 | **First 2 bytes:**  Memory slot of the internal EEPROM  *(uint8\_t)* 0 – 1  **Last 2 bytes:**  Register parameter to save:  *(uint16\_t)* 0 – 130 | Load value of the register from internal EEPROM memory | **Command:** 228  **Data:** Loaded value  **Error:** |
| 229 | **First 2 bytes:**  Memory slot of the internal EEPROM  *(uint8\_t)* 0 – 1  **Last 2 bytes:**  Register parameter to save:  *(uint16\_t)* 0 – 130 | Read value of the register from internal EEPROM memory (without loading it to the RAM) | **Command:** 229  **Data:** Read value  **Error:** |
| 230 | *(uint8\_t)* 0 – 4 | Set state (not recommended):  0 = Nominal  1 = Electrode actuation  2 = Picomotor actuation  3 = Separation  4 = Safe mode | **Command:** 230  **Data:** 0  **Error:** |
| 231 | *discarded* | Read current state:  0 = Nominal  1 = Electrode actuation  2 = Picomotor actuation  3 = Separation  4 = Safe mode | **Command:** 231  **Data:** Current state  **Error:** |
| 232 | *discarded* | Read previous state:  0 = Nominal  1 = Electrode actuation  2 = Picomotor actuation  3 = Separation  4 = Safe mode | **Command:** 232  **Data:** Previous state  **Error:** |
| 233 | *discarded* | Trigger Health Check manually | **Command:** 233  **Data:** 0  **Error:** |
| 245 | **First 2 bytes:** location of the code:  *(uint16\_t)*  0 – 3  **Last 2 bytes:** size of the code in bytes:  (uint8\_t)  0 - 255 | Command to load a code of size *data* [in number of pages] in external EEPROM. | **Command:** 245  **Data:** 0  **Error:** |
| After the first acknowledgment, the code is waiting for the first page [256 bytes] to be transferred (no command or checksum, just the 256 bytes). The MCU will then acknowledge the transfer (as described on the right) and wait for the next page. This is repeated until the last page is received | | **Command:** 245  **Data:** 0  **Error:** |
| 246 | *(uint16\_t)*  0 – 3 | Get the size of the code [in pages] in external EEPROM, location #*data* | **Command:** 246  **Data:** Size of code  **Error:** |
| 247 | **First 2 bytes:** location of the code:  *(uint16\_t)*  0 – 3  **Last 2 bytes:** address to read:  (uint16\_t)  0 - 65535 | Read a DWord (4 bytes) in the external EEPROM | **Command:** 247  **Data:** Read DWord  **Error:** |
| 250 | *discarded* | Reboot - Trigger the watchdog to reset the MCU | No acknowledgment (MCU restarts) |
| 255 | *(uint32\_t)* | Ping | **Command:** 255  **Data:** Sent data  **Error:** |
| else | | Wrong received command | **Command:** 254  **Data:** Erroneous sent command  **Error:** |
| Error parsing incoming message (checksum error) | **Command:** 253  **Data:** First 2 bytes: buffer index of UART  3rd byte: parsed command  Last byte: parsed checksum  **Error:** |

#### Register

The Register is a vector of 4 byte values used by the code to operate. It is saved in a non-volatile memory so any change can be remembered upon restart of the DMB.

The register is saved periodically to the internal EEPROM of the MCU (non-volatile memory). It is loaded from the EEPROM to the RAM at the beginning of the application code. There are 2 slots to save the register in this EEPROM. The slot 0 is the usual slot where the current register is saved/loaded. The slot 1 is used by the watchdog timer only to save a “screenshot” of the register before resetting the system. This helps understand why the watchdog had to reset the system.

To read the register values, the message to send is the following:

* Command: 150
* Data: Entry of register to read (1 – 130)

The mirror answers with the following message:

* Command: Entry of the register specified in the incoming data field
* Data: Value at the entry
* Error = 0

To read slot 1 of the internal EEPROM without loading its values to the copy of the register in the RAM, use command 229.

Table : Entries of the Register for Telemetry

|  |  |  |
| --- | --- | --- |
| Entry of the Register | Type | Description |
| 1 | uint32\_t | Define the boot mode (“BOOT\_SAFE“) 0 = Normal (Default) 1 = User driven It is set to 1 by the bootloader and to 0 by the application code once a successful ping command has been received and answered. |
| 2 | uint32\_t | Address of the code for boot up |
| 3 | uint32\_t | Baud rate of Spare UART set during start-up (data in bits per seconds) |
| 4 | uint32\_t | Last 4 bytes transmitted on Spare UART   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last transmitted bytes | Third to last transmitted bytes | Second to last transmitted bytes | Last transmitted bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 5 | uint32\_t | Last 4 bytes received on Spare UART   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last received bytes | Third to last received bytes | Second to last received bytes | Last received bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 6 | uint32\_t | Current number of bytes in Spare UART buffer |
| 7 | uint32\_t | Baud rate of XBee UART |
| 8 | uint32\_t | Last 4 bytes transmitted on XBee UART   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last transmitted bytes | Third to last transmitted bytes | Second to last transmitted bytes | Last transmitted bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 9 | uint32\_t | Last 4 bytes received on XBee UART   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last received bytes | Third to last received bytes | Second to last received bytes | Last received bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 10 | uint32\_t | Current number of bytes in XBee UART buffer |
| 11 | uint32\_t | Frequency of SPI |
| 12 | uint32\_t | Last 4 bytes transmitted on SPI   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last transmitted bytes | Third to last transmitted bytes | Second to last transmitted bytes | Last transmitted bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 13 | uint32\_t | Frequency of I2C |
| 14 | uint32\_t | Number of maximum restarts when contacting a device through I2C |
| 15 | uint32\_t | Number of restarts used in the last transmissions through I2C |
| 16 | uint32\_t | Slave address used in the last 4 transmissions through I2C   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last used address | Third to last used address | Second to last used address | Last used address | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 17 | uint32\_t | Last 4 bytes transmitted on I2C   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last transmitted bytes | Third to last transmitted bytes | Second to last transmitted bytes | Last transmitted bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 18 | uint32\_t | Last 4 bytes received on I2C   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last received bytes | Third to last received bytes | Second to last received bytes | Last received bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 19 | uint32\_t | Frequency of ADC |
| 20 | uint32\_t | Last 4 bytes received on ADC   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last received bytes | Third to last received bytes | Second to last received bytes | Last received bytes | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 21 | uint32\_t | Number of commands received from Spare UART |
| 22 | uint32\_t | Number of commands received from XBee UART |
| 23 | uint32\_t | Number of messages sent to Spare UART |
| 24 | uint32\_t | Number of messages sent to XBee UART |
| 25 | uint16\_t | Timeout while receiving a byte on UART [in ms] |
| 26 | uint16\_t | Current high voltage on the Variable HV line |
| 27 | uint16\_t | Current high voltage on the Bias HV line |
| 28 | uint16\_t | Tolerance to measure a voltage [in bits] |
| 29 | uint16\_t | Feedback voltage from the picomotor voltage supply |
| 30 | uint16\_t | Feedback voltage from the bias HV |
| 31 | uint16\_t | Feedback voltage from the variable electrode HV |
| 32 | uint16\_t | Feedback voltage from the 5V Bus current |
| 33 | uint16\_t | Feedback voltage from the 5V Bus voltage |
| 34 | int16\_t | Minimum allowable temperature on the microcontroller board |
| 35 | int16\_t | Maximum allowable temperature on the microcontroller board |
| 36 | int16\_t | Minimum allowable temperature on the HV board |
| 37 | int16\_t | Maximum allowable temperature on the HV board |
| 38 | int16\_t | Minimum allowable temperature on the thermopile 1 |
| 39 | int16\_t | Maximum allowable temperature on the thermopile 1 |
| 40 | int16\_t | Minimum allowable temperature on the thermopile 2 |
| 41 | int16\_t | Maximum allowable temperature on the thermopile 2 |
| 42 | int16\_t | Minimum allowable temperature on the thermopile 3 |
| 43 | int16\_t | Maximum allowable temperature on the thermopile 3 |
| 44 | uint32\_t | Set maximum number of ticks when moving a picomotor |
| 45 | float | Location of picomotor 1 (in intervals) |
| 46 | float | Location of picomotor 2 (in intervals) |
| 47 | float | Location of picomotor 3 (in intervals) |
| 48 | int32\_t | Number of intervals left actuating picomotor 1 |
| 49 | int32\_t | Number of intervals left actuating picomotor 2 |
| 50 | int32\_t | Number of intervals left actuating picomotor 3 |
| 51 | int32\_t | Number of ticks left actuating picomotor 1 |
| 52 | int32\_t | Number of ticks left actuating picomotor 2 |
| 53 | int32\_t | Number of ticks left actuating picomotor 3 |
| 54 | float | Mean value of a positive picomotor 1 step (relative to the size of an encoder 1 interval) |
| 55 | float | Mean value of a positive picomotor 2 step (relative to the size of an encoder 2 interval) |
| 56 | float | Mean value of a positive picomotor 3 step (relative to the size of an encoder 3 interval) |
| 57 | float | Standard deviation of a positive picomotor 1 step (relative to the size of an encoder 1 interval) |
| 58 | float | Standard deviation of a positive picomotor 2 step (relative to the size of an encoder 2 interval) |
| 59 | float | Standard deviation of a positive picomotor 3 step (relative to the size of an encoder 3 interval) |
| 60 | float | Mean value of a negative picomotor 1 step (relative to the size of an encoder 1 interval) |
| 61 | float | Mean value of a negative picomotor 2 step (relative to the size of an encoder 2 interval) |
| 62 | float | Mean value of a negative picomotor 3 step (relative to the size of an encoder 3 interval) |
| 63 | float | Standard deviation of a negative picomotor 1 step (relative to the size of an encoder 1 interval) |
| 64 | float | Standard deviation of a negative picomotor 2 step (relative to the size of an encoder 2 interval) |
| 65 | float | Standard deviation of a negative picomotor 3 step (relative to the size of an encoder 3 interval) |
| 66 | uint32\_t | Errors related with the actuation of picomotor 1 |
| 67 | uint32\_t | Errors related with the actuation of picomotor 2 |
| 68 | uint32\_t | Errors related with the actuation of picomotor 3 |
| 69 | uint32\_t | Duration for variable electrode HV to stabilize (data milliseconds) |
| 70 | uint32\_t | Voltage limit of the electrodes from ground |
| 71 | uint32\_t | Voltage step when initializing the mirror (the variable and bias voltage are successively ramped up by this amount): |
| 72 | uint32\_t | Bias voltage: |
| 73 | uint32\_t | Active channel on the multiplexer (0xFFFFFFFF means no channel is active) |
| 74 - 114 | uint32\_t | Data for electrode 1 – 41 (respectively) **First 2 bytes:** time of actuation in milliseconds **Last 2 bytes**  Example:  data = 0x000A1FFF means the electrode is set to 0V for 10 ms |
| 115 | uint32\_t | Errors associated with the electrodes actuation. Each time an error occurs, this value if incremented:   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last error | Third to last error | Second to last error | Last error | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 116 | uint32\_t | Number of safe mode errors (each time a measured parameter such as the temperature of a board is out-of-bounds, the value in incremented by 1) |
| 117 | uint32\_t | Number of errors that set the Safe Mode (when the number of safe mode error reaches this value, safe mode is triggered) |
| 118 | uint32\_t | Number of safe modes (reset to 0 when the MCU restarts) |
| 119 | uint32\_t | Last errors that are triggering Safe Mode   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last error | Third to last error | Second to last error | Last error | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 120 | uint32\_t | Duration between each automatic health check of the electronics (in milliseconds) |
| 121 | uint32\_t | Duration between each automatic save of the register to non-volatile memory (in milliseconds) |
| 122 | uint32\_t | Errors related with the automatic health check of the electronics   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last error | Third to last error | Second to last error | Last error | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 123 | uint32\_t | Timer associated with the separation device (in milliseconds) |
| 124 | uint32\_t | Timer associated with the Spare UART (in milliseconds) |
| 125 | uint32\_t | Timer associated with the XBee UART (in milliseconds) |
| 126 | uint32\_t | Timer associated with the Health Checking function (in milliseconds) |
| 127 | uint32\_t | Timer associated with the Register saving (in milliseconds) |
| 128 | uint32\_t | Errors related with the parsing of telecommands   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last error | Third to last error | Second to last error | Last error | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |
| 129 | uint32\_t | Number of watchdog resets of the MCU |
| 130 | uint32\_t | Last 4 executed functions (useful to understand when a watchdog timer reset has occurred)   |  |  |  |  | | --- | --- | --- | --- | | Fourth to last executed function | Third to last executed function | Second to last executed function | Last executed function | | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | 0 – 0xFF | | MSB |  |  | LSB | |

#### Error Codes

The following codes are returned in the error byte of the acknowledge message.

Table : Error Codes

|  |  |
| --- | --- |
| Error ID | Description |
| 0 | OK (no error) |
| 1 | Internal EEPROM overloaded |
| 21 | Incorrect stop when receiving command from the Spare UART |
| 22 | Frame lost when receiving command from the Spare UART |
| 23 | Wrong parity check when receiving command from the Spare UART |
| 31 | Incorrect stop when receiving command from the XBee UART |
| 32 | Frame lost when receiving command from the XBee UART |
| 33 | Wrong parity check when receiving command from the XBee UART |
| 41 | Frequency of SPI out of bounds |
| 42 | SPI: Timeout |
| 51 | Frequency of I2C out of bounds |
| 52 | I2C: Timeout |
| 53 | I2C: Arbitration lost when sending start |
| 54 | I2C: Critical error when sending start |
| 55 | I2C: Arbitration lost when sending restart |
| 56 | I2C: Critical error when sending restart |
| 57 | I2C: Address non acknowledged |
| 58 | I2C: Arbitration lost when sending address |
| 59 | I2C: Critical error when sending address |
| 60 | I2C: Data non acknowledged |
| 61 | I2C: Arbitration lost when sending data |
| 62 | I2C: Critical error when sending data |
| 63 | I2C: Critical error when reading data |
| 71 | Frequency of ADC too low for proper functioning |
| 72 | Frequency of ADC too high for proper functioning |
| 73 | Frequency of ADC out of bounds |
| 74 | ADC: Timeout |
| 81 | Wrong UART port when reading a message |
| 82 | Wrong UART port when sending a message |
| 83 | Wrong checksum of received message |
| 91 | Variable electrode HV feedback out of bounds |
| 92 | Bias HV feedback out of bounds |
| 93 | Picomotor HV feedback out of bounds |
| 94 | Fault of Current Limiter 1 |
| 95 | Fault of Current Limiter 2 |
| 96 | Fault of Current Limiter 3 |
| 101 | Index of thermal sensor out of bounds |
| 111 | Critical state of encoder |
| 112 | Current state of encoder out of bounds |
| 113 | Previous state of encoder out of bounds |
| 114 | Incorrect state change of encoder |
| 115 | Critical error monitoring the state of the encoder |
| 116 | Number of ticks actuated within an encoder interval is larger than the maximum |
| 117 | Encoder moving in wrong direction |
| 121 | ID of the electrode to switch ON/OFF out-off-bounds |
| 131 | Change mode while in safe mode |
| 132 | Change mode while picomotors are running |
| 133 | Set separation mode while not in nominal mode |
| 134 | Set mode while separation is running |
| 135 | Set picomotor or electrode actuation from separation mode |

### 

### Data Management

For a 10 min long science mode, considering a command every 1 s, and both messages to and from the mirror saved, this leads to 2 x 7 x 600 = 8,400 bytes of data.

### Any other relevant requirements

File handling or interpretation

## Bootloader Design

While booting up the DMB, the microcontroller starts the bootloader before the application code. The bootloader is located at the bottom of the internal flash memory of the MCU.

During a normal boot, the bootloader loads the application code from an external EEPROM and starts it.   
During a safe boot (either triggered by failing to communicate with the previous application code, or forced by the user by programming the first entry of the register to 1) the user can control the bootloader to perform basic debugging operations.

The bootloader cannot be reprogrammed in flight.

### Bootloader Modes of Operation

|  |  |  |  |
| --- | --- | --- | --- |
| Mode | Function | Software Dependencies | Hardware Dependencies |
| Normal Boot | Automatically load the application code from the external EEPROM and starts it | * I2C initialization and reading * Flash writing function | Microcontroller  External EEPROM |
| Safe Boot | Gives the freedom to the user to debug the loading of the application code | * All UART functions * I2C initialization and reading * All flash management functions * All external EEPROM management functions | Microcontroller External EEPROM  (XBee) |

### Bootloader Data Rates

Data rates for the UART interface:

* Baud rate: **9600**
* Parity: **None**
* Data Bits: **8 bits**
* Stop Bits: **1 bit**
* Hardware Flow control: **None**

### C:\Users\Thibaud\Box Sync\AAReST\02_MIRRORS\01_CODE\FLIGHT\FLIGHT_BOOTLOADER_V2\Bootloader_Architecture.pngBootloader Architecture

### Telecommand and Telemetry Points (TC/TM)

During a safe boot, the user can interact with the bootloader to debug the loading an application code to the flash memory of the MCU.

**TC/TM format**

Data  
(4 byte)

Command  
(1 byte)

Checksum  
(1 byte)

**Explanation of TC/TM points**

* Command and data are listed in the table below
* Checksum is a parity checksum such that adding each byte of the total message (checksum included) equals 0xFF
* The same format applies for sent and received messages
* Each command is acknowledged as described in the table

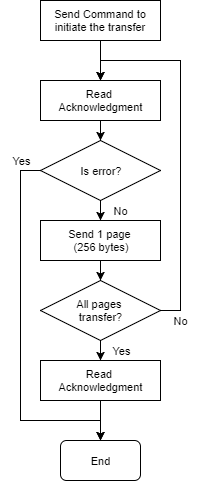
|  |  |  |  |
| --- | --- | --- | --- |
| Command | Data | Description | Acknowledgment |
| 1 | 122 – 500 000 (uint32\_t) | Re-initialize UART with MirrorSat with baud rate equal to *data* [Hz] | Command: 1  Data: error (see Table 7) |
| 2 | 122 – 500 000 (uint32\_t) | Re-initialize UART with XBee with baud rate equal to *data* [Hz] | Command: 2  Data: error (see Table 7) |
| 3 | 15 209 - 500 000 (uint32\_t) | Re-initialize I2C with frequency equal to *data* [Hz] | Command: 3  Data: error (see Table 7) |
| 4 | 0 – 254 (uint8\_t) | Test I2C device ACK with address equal to *data* (LSB discarded) | Command: 4  Data: error (see Table 7) |
| 10 | 0 – 110 | Read value from the Register (internal EEPROM) | Command: 10  Data: Read value |
| 11 | 0 – 131072 (uint32\_t) | Read DWord (uint32\_t) from the Flash | Command: 11  Data: Read DWord (flipped) |
| 12 | **First 2 bytes:**  0 – 3 (Code ID) **Last 2 bytes:** 1 – 65535 | Read DWord (uint32\_t) from the code #(Code ID) in the external EEPROM | Command: 12 Data: Read DWord (flipped) |
| 20 | *discarded* | Starts the application code | *None* |
| 21 | 0 – 3 | Loads the code #(Code ID) from the external EEPROM to the Flash memory | Command: 21  Data: error (see Table 7) |
| 22 | 1 – 255 | Loads the code sent by the UART to the Flash memory directly. This code has a number of pages = Data. (1 page = 256 bytes) | Command: 22  Data: error (see Table 7) |
| 255 | *discarded* | Ping | Command: 255  Data: 1 |
| else | | **Erroneous command** | **Command: 254**  **Data: received command** |
| **Erroneous checksum** | **Command: 253**  **Data:**  **Second byte = received command**  **Last 2 bytes = received checksum** |

### Loading New Application Code

New code can be loaded either directly to the Flash memory (from bootloader) or to one of the 4 partitions of the external EEPROM (from application code).

The method is the following:

1. Send command to initiate the transfer  
   **Bootloader**: Command = 22, Data = Number of pages  
   **Application Code**: Command = 245, Data = Partition of the EERPOM and Number of pages
2. Read Acknowledgment   
   **Bootloader**: Command = 22, Data = 0 (OK) or error  
   **Application Code**: Command = 245, Data = 0 (OK) or error
3. Loop between sending a page (256 bytes) and reading the acknowledgment of that page being loaded (same as above)  
   **If the last page has less than 256 bytes, fill the rest of the page with 0xFF bytes**



## Instrument Autonomy and FDIR

Instrument safety and FDIR for mirror box is provided through watchdog timer overflow triggered system reset. Additionally, a health monitoring function checks the proper functioning of the electronics. In case of an improper measured value (temperature, feedback voltage) a safe mode is triggered turning off all the electronics (except communication and MCU) and preventing any mode switching until values are back to nominal.

### Health Check

Depending on the mode of operation, the MCU checks the different values. Each anomaly increments a counter. When it overflows, safe mode is triggered.

The health check function is called periodically and after each mode change.

**All modes:**Temperature of MCU and HV boards + thermopiles.

**Electrode actuation mode:**Feedback voltages and current limiter faults of both HV supplies (variable and bias). Note that a current limiter fault forces safe mode (no increment of the counter).

**Picomotor actuation mode:**Feedback voltage and current limiter fault of the HV supply. Note that a current limiter fault forces safe mode (no increment of the counter).

### Safe Mode

Safe mode deactivates supply voltages and high voltage lines. The user can still communicate with the MCU but cannot change mode. the software changes the mode to nominal if the temperatures are all within range.

### System reset event description

Watchdog timer is scale to overflow every 8 seconds. It is reset in the main loop and specific long-lasting functions.

Watchdog timer (WDT) is used as both the system-resetting source, also as the interrupt source. Before system reset, the WDT interrupt routine will save the current Register in non-volatile memory (slot 1). Since the loaded Register when the Application Code starts is from slot 0 in the non-volatile memory, this saving operation puts a snapshot of the Register just before reset which can help understand the cause for reset.

To debug the reset, the code for the last 4 functions used before reset are saved in the register.

# Harness Interface and Design

## Instrument to Platform Harness

## Connectors and Pin-Outs

# Instrument Handling & Storage

## Transport Container

## Physical Handling

## Mechanical Ground Support Equipment

## Electrical Ground Support Equipment

## Qualification Status

# Any other data

Please provide here any other data you think requires to successfully integrate your payload into the satellite.

Delete if not used

1. APPENDIX TITLE [PLEASE DELETE IF NOT USED]