Interface Control Document – Mechanical
10 mNm-sec Reaction Wheel
Sinclair Interplanetary
January 27, 2021, Rev 1.4

1. Revision Notes

This revision of the document contains the following changes relative to the previously released version:

- Connector specified in Section 3.2 changed from M80-8450445 to M80-8420442. The former part number applies to the mating connector and was specified in error. The latter part number is for the connector on the wheel.

2. Scope

This document describes the mechanical interface for the 10 mNm-sec reaction wheels built by Sinclair Interplanetary and SFL. For it to be relevant to your wheel, your part number must be of the form:

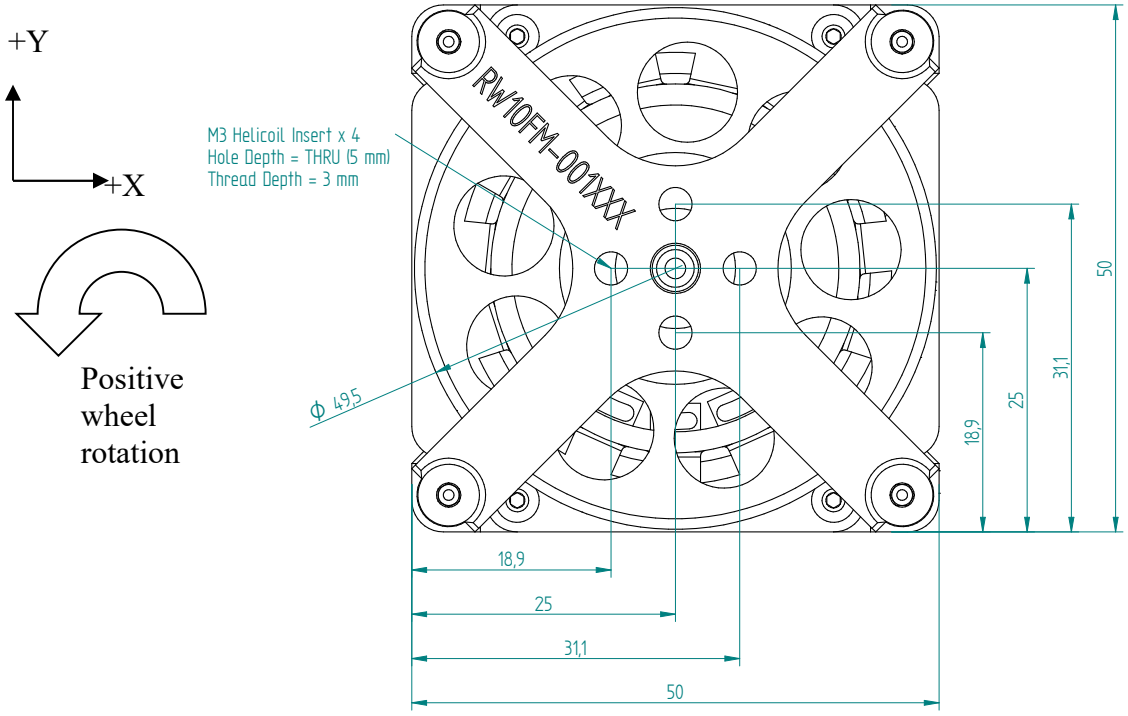
RW-0.01-xx-xxxxx-x-x-x

For additional interface definitions see:

- The electrical ICD for your particular electronics unit
- The software ICD for your particular software load

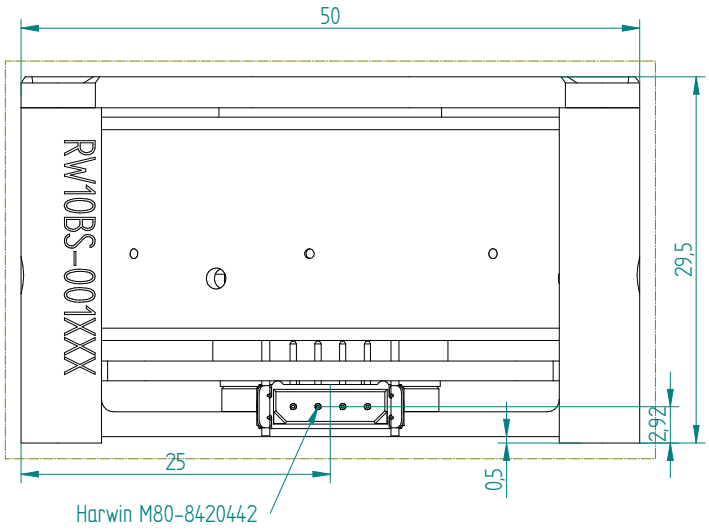
3. Mechanical Drawings

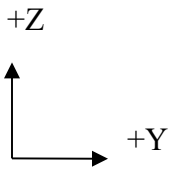
3.1. Top View



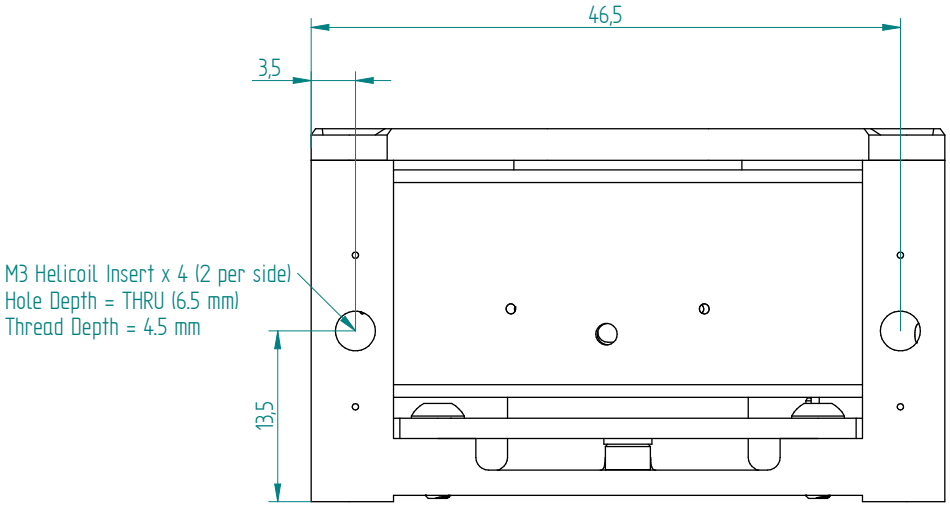
The X and Y axes are defined as shown in the figure. The Z-axis (illustrated in the following figures) completes the right-handed set. The rotation arrow shows the direction of wheel rotation that is considered positive speed. Rotation in the opposite direction is considered negative wheel speed.

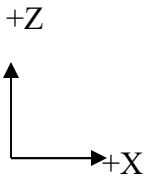
3.2. Side View



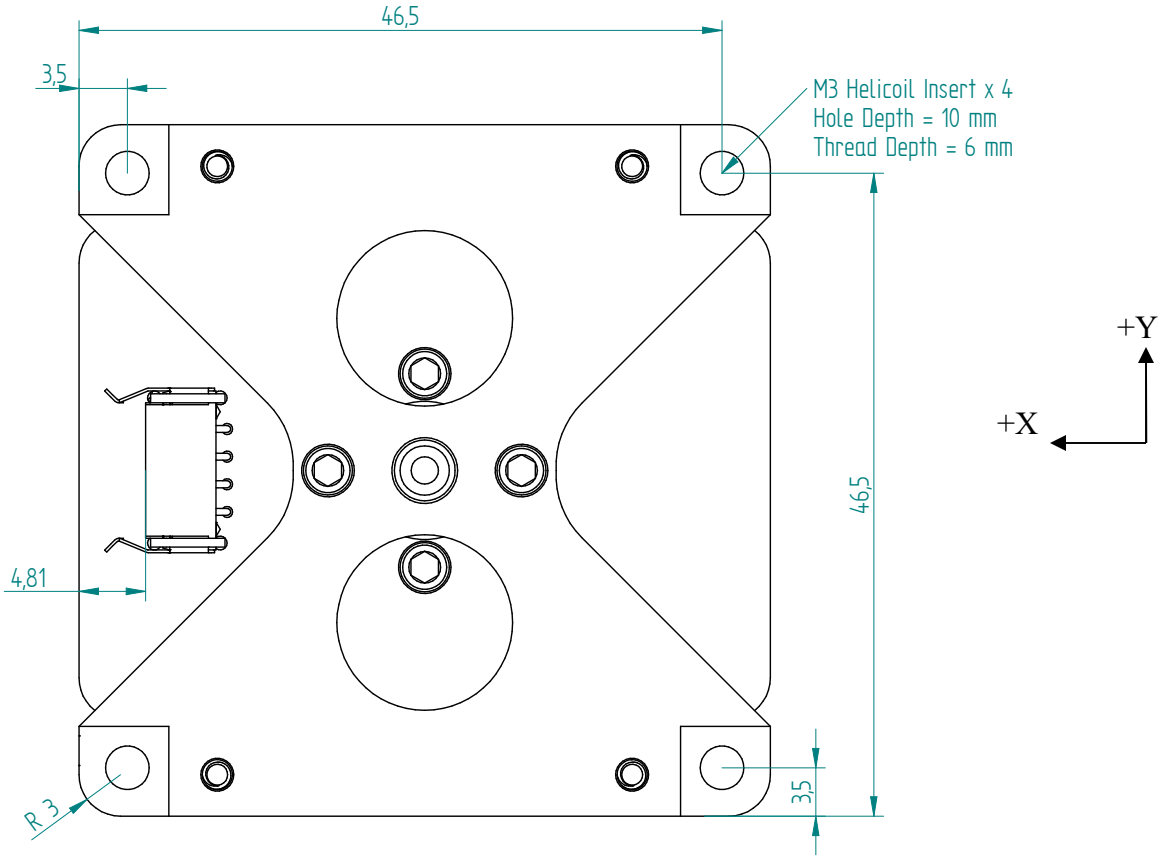


3.3. Front View





3.4. *Bottom View*



4. Materials

The following materials are used as structural elements in the reaction wheel.

- Aluminum 6061-T6, with yellow chemical film coating, for primary structural elements
- Nitronic 60™ (UNS S21800) stainless steel helical inserts for all internal threads
- Windform LX 2.0 composite polymer as an insulator in the motor
- Windform XT 2.0 composite polymer as a magnet positioner in the rotor
- Samarium Cobalt, with nickel overplate, in the rotor magnets
- Stainless Steel 416, for the bulk of the rotor

All of the materials and processes for the wheel have been selected for compatibility with high vacuum and low outgassing requirements.

5. Mounting

The wheel has mounting points on the bottom, front, back, and top surfaces. This flexibility allows the customer to mount a number of orthogonal wheels to a plate while reducing the need for additional brackets. All mounting holes are threaded for M3x0.5. The wheel mounting points provide mechanical, thermal and electrical bonding paths.

The bottom mounting points may be used alone, as the four widely-spaced holes provide a solid anchor for the wheel. Alternatively, a combination of the top and front or back holes can be used to securely mount the wheel.

Be careful when using the front or back mounting points that the spacecraft structure does not interfere with the rotor. There is only 0.5 mm of clearance between the rim of the rotor and the edge of the wheel footprint. If mounting to a plate it must be flat, or some sort of standoff will be required.

6. Electrical Connector

The electrical connector is mounted to the underside of the circuit board. Specific details on the connector may be found in the electrical ICD. The type and polarity of connector shown in this document is for illustration only.

7. Mass Properties

The mass of the complete wheel assembly is 122 grams. The mass center is on the spin axis, 16.0 mm in the +Z direction above the base.

8. Magnetic Properties

The rotor is contains a 10-pole magnet array on its inner surface. The stainless steel of the rotor prevents the vast majority of the field lines from leaving the wheel. The large number of poles means that the field decays very quickly with increasing distance. There is no dipole moment, and so no unwanted attitude torques upon the spacecraft.

9. Pressure Environment

The reaction wheel is designed to operate in the vacuum of space, and in the atmosphere of a terrestrial laboratory environment. The bearings are lubricated with space-grade grease and will give many years of service on-orbit. This lubricant is not moisture sensitive, and is not degraded by atmospheric operation.

The wheel has not been designed to operate through critical pressure, and has not been tested for coronal discharge. For safety it should be unpowered while the host spacecraft is attached to the launch vehicle.

10. Thermal

10.1. Thermal Environment

Operating Temperature	-40 ºC to +75 ºC
Survival Temperature	-40 ºC to +85 ºC

The temperature is defined as the temperature of the base plate in the nominal mounting configuration.

Note that the motor magnets are high-temperature samarium-cobalt and are at no risk of demagnetization from reasonable spacecraft temperatures.

10.2. Thermal Interface

The thermal interface to the spacecraft is through conduction. There are 4 mounting feet, with a total surface area of 130 mm^2. The contact material is aluminum 6061-T6, with yellow chemical film coating.

There is no convective thermal interface. Radiation is not considered a significant thermal interface. The reaction wheel rotor is thermally isolated, as conduction through the bearings is minimal.

In normal operation, the maximum heat load from the reaction wheel is 1.0 W.

11. Vibration and Shock Environment

The reaction wheel has been designed to be compatible with the vibration and shock environment of most launch vehicles. Please contact the factory for further details.

The total vibration lifetime of the wheel bearings is limited. The user must be careful not to over-test flight wheels during unit-level or spacecraft-level acceptance testing. Test plans should be designed so that the sum of the acceptance testing plus the actual launch vibration does not exceed the duration of the qualification tests.

12. Contamination Environment

This wheel is not hermetically sealed, and so great care must be taken to avoid contamination of the electronics and the bearings by dust and debris. It should be bagged when not used, and handled only in a clean-room environment.

Interface Control Document – Electrical
Reaction Wheel with 4 V Power
Sinclair Interplanetary
February 11, 2017, Rev 1.2

1. Scope

This document describes the electrical interface for the Sinclair Interplanetary reaction wheels with 4 V nominal power bus. For it to be relevant to your reaction wheels, your part number must be:

RW-x.xx-4-ASYNC-x-1-x

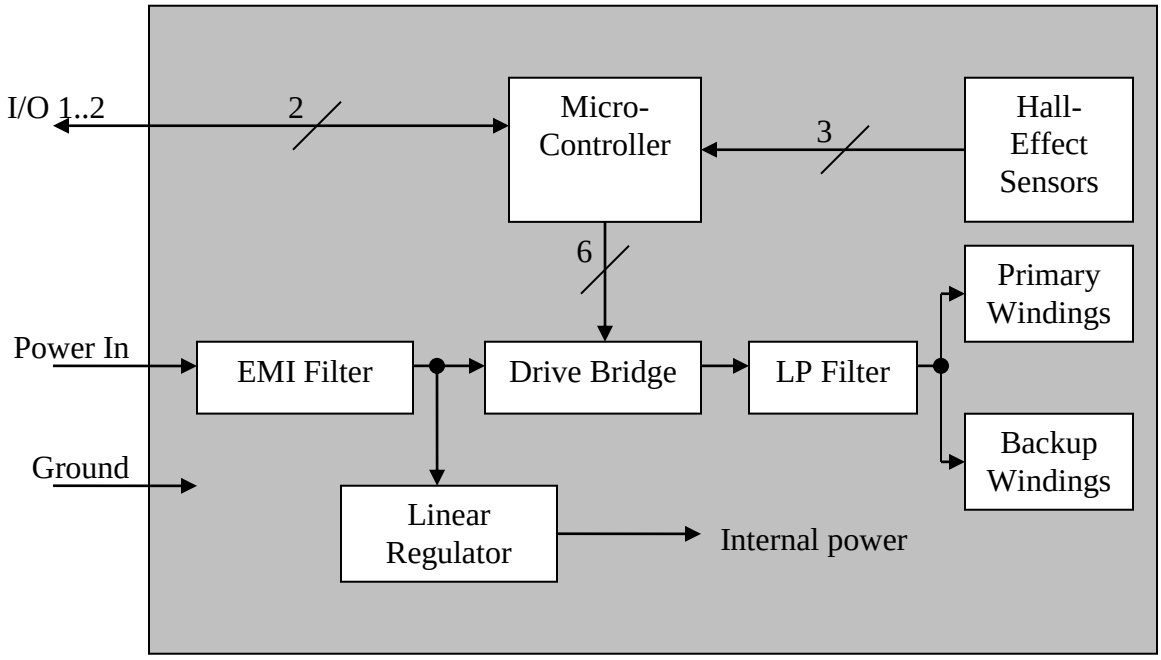
Or

RW-x.xx-4-I2C-x-1-x

For additional interface definitions see:

- The mechanical ICD for your particular size of wheel
- The NSP packet protocol
- The software ICD for your particular software load

2. Block Diagram



3. Principle of Operation

The motor is a three-phase brushless design with redundant windings. Unregulated spacecraft power passes through the front-end EMI filter and into the drive bridge. Here it is chopped by a 350 kHz PWM waveform. The LP (low pass) filter smoothes the PWM, leaving only the DC drive components that pass on to the motor. Backup windings are wired in parallel with the primary.

Three Hall-effect sensors detect the position of the rotor magnets. This data is processed by a microcontroller where it is used for commutation and for speed estimation. The microcontroller controls the three current-mode half-bridges allowing it to set the motor torque.

Two I/O signals from the microcontroller are carried on the electrical connector. The command and telemetry interface is implemented using these signals. The details of this interface depend on which onboard software load the customer has requested.

4. Connector

The connector is a Harwin Datamate M80-8420442. This is a 4-pin single row plug with retention latches.

Mating Connector Part Number	Comments
M80-8450445 (Digikey 952-1184-ND)	Receptacle connector takes wires 24-28 AWG. Special crimp tooling required.
M80-8400401	Through-hole connector with SnPb termination, suitable for flex circuit.

Care must be taken when demating the connector as the retention latches are delicate. If they are treated well they will positively retain a mating connector without any need for staking.

If customer access to crimp tooling is problematic, Sinclair Interplanetary can pre-terminate wires or prepare harnesses on a special order basis.

5. Connector Saver

Reaction wheels are provided with connector savers to protect the flight connectors from unnecessary mate/demate cycles during testing. These savers are fabricated from short lengths of Teflon wire and Harwin Datamate crimp plug and socket connectors. The connector saver should be removed prior to final integration with the spacecraft.

6. Pinout

The connector has the following pin assignments:

Pin	Type	Signal
1	Digital I/O	IO/2
2	Power	Power In
3	Digital I/O	I/O1

4	Power	Ground

7. Grounding

The default factory option is RF multipoint grounding, whereby the chassis is connected to the internal electronics ground via 4 x 10 nF capacitors (50 V rated), and one 1 MΩ resistor. Upon request, wheels can be furnished with complete isolation between chassis and electronics. Alternatively, wheels can be supplied with low-impedance DC connection between chassis and electronics.

8. Signals

8.1. Ground

Ground is the reference used for all other signals.

8.2. Power In

Absolute Maximum Voltage	0 to +7.0 V (see note)
Operating Voltage	+3.5 V to +6.0 V
Fault Protection	6 V TVS

The Power In signal provides power to the all of the wheel circuits. It is intended to be compliant with busses that are regulated from either a single cell Li-ion battery or two triple-junction GaAs solar cells.

A transient voltage suppressor (TVS) is included for fault protection. This part is rated for a standoff voltage of 6 V, and at this voltage it does not impact the circuit. At 6.67 V it may begin to conduct up to 10 mA. At a maximum voltage of 10.3 V it conducts a clamping current of 58.3 A. It will also conduct if a negative voltage is applied to the Power In signal.

High (or negative) voltages must not be applied for long durations from low impedance sources or the TVS will overheat and become damaged. It is intended to absorb capacitive discharges, load dumps, and other short-lived phenomena. A power of 600 W may be applied for no longer than 1 msec.

8.3. I/O[1..2]

Absolute Maximum Voltage	-0.5 to 5.5 V
Input Low Voltage	0.9 V max
Input High Voltage	2.1 V min
Output Low Voltage	0.7 V max, 3 mA external load
Output High Voltage	2.3 V min, 3 mA external load
Pull-up	120 kΩ nominal to +3.0V
ESD Protection	5.6 V Zener
Power-off impedance	High

The use of the I/O signals is dependent on the software running in the wheel. They may be configured as digital inputs or open-drain or push-pull outputs.

I/O1 and I/O2 may be driven above the internal +3.0 V rail, though they must always respect the absolute maximum limits.

9. Connections

The connections made to the I/O pins depend on the communication option specified in the part number:

Part Number	Communications
RW-x.xx-4-ASYNC-x-x-x	Asynch Serial with NSP
RW-x.xx-4-I2C-x-x-x	I2C with NSP

Signals are assigned to I/O pins as follows:

Signal	ASYNC	I2C
I/O1	Transmit	SDA
I/O2	Receive	SCL

Using the Asynch Serial and I2C options a number of devices may share a common data bus, and each requires a unique address to identify it. However, the small 4-pin connector does not allow any additional signals for addressing. Instead, each wheel has an NSP address programmed into it in the factory. The user must take care not to put two wheels with the same NSP address on the same bus.

The asynchronous option configures the UART in the following way. Note that the wheel's master oscillator is not crystal driven and so there can be some variation in actual baud rates.

Data Bits per Byte	8
Parity	None
Stop Bits	1
Nominal Baud Rate	57600 bps
Actual Output Rate	56338 bps to 58685 bps
Permissible Input Rate	56000 bps to 59000 bps

Using the I2C option, connections SDA and SCL are open-drain digital lines. The internal pull-up device will be turned on for each line, but this pull-up is weak. It is recommended that the user install an additional pull-up resistor to ensure fast transitions.

NSP Bootloader for Sinclair Interplanetary Hardware
Rev 1.2
September 5 2008

1 Scope

This document describes how to use the NSP bootloader software that is found on a number of Sinclair Interplanetary spacecraft components. These include reaction wheels and sun sensors that are not configured for CAN communications. Sinclair Interplanetary devices that do not permit on-orbit code modification, such as the battery charge/discharge regulator, do not use this bootloader.

2 Overview

The NSP bootloader is a small, reliable piece of software that is loaded into the device in the factory. It is started whenever the device processor is reset.

The bootloader allows NSP communications over an appropriate link. It allows the user to load new application software into the device, or to validate existing application software. Finally, it allows the user to execute application software. While the application software executes the bootloader remains running in the background providing continued NSP communications.

Very careful effort has gone into protecting the bootloader from damage. While it can be used to load and modify application software, the bootloader cannot change itself. Nor can application software modify the bootloader. There should be no way to "brick" the device with a badly chosen command.

3 NSP Module Commands

The NSP bootloader interprets incoming telecommands. The command code section of the message control field is compared to the following table of known commands:

0x00	PING
0x01	INIT
0x02	PEEK
0x03	POKE
0x04	TELEMETRY
0x06	CRC
0x07	APPLICATION-TELEMETRY
0x08	APPLICATION-COMMAND

If the command code is unknown (and if the poll bit is set) then a reply is generated with a NACK and no further processing is performed. Otherwise the command is executed as shown in the following sections.

3.1 PING Command (0x00)

The PING command is typically used during testing to verify communications. Incoming data is ignored. The reply packet contains a human-readable text string containing:

- The type of device and the manufacturer
- The compile date and time of the NSP bootloader
- The name, and compile date and time of the currently running application module, if applicable.

The string is not NULL terminated.

This command is unusual in that data sent in the telecommand is lost. The reply message contains only human-readable string.

3.2 INIT Command (0x01)

The INIT command is used to change the operating mode of a device. An INIT with no data causes a complete reset of the device. It will reboot to the NSP bootloader, with no application module running. Note that if a reply has been requested ("Poll" bit set to '1') then the reset will occur after the reply transmission is complete.

An INIT command with 4 bytes of data causes the NSP bootloader to run an application module at the corresponding 32-bit start address. If an application module is already running, the request will be NAKed. A NAK will also be generated if the start address lies within the NSP module or is outside the valid memory range.

By convention, devices will ship from the factory with their primary application program stored at address 0x00001000. Thus, a command of INIT 0x00001000 will start the default behaviour.

3.3 PEEK Command (0x02)

The PEEK command is used to read the device memory. Five bytes of data are required. The first four are interpreted as a 32-bit pointer to the start of the memory to be read. The fifth byte is the number of bytes to read. A value of zero in this field will cause 256 bytes to be read. The reply message contains the start address, followed by bytes read from memory. The number of bytes to read is not echoed back.

The memory map is as follows:

Address Range	Function
0x00000000 – 0x00007DFF	32 kB Program memory (flash)
0x00010000 – 0x000107FF	2 kB XRAM (RAM)
0x00020000 – 0x000200FF	256 B IRAM (RAM)
0x00030080 – 0x000300FF	128 B SFR (RAM)

Address ranges not mentioned above are unsupported, and attempted reads will generate NAKs. The SFR space refers to the Special Function Registers of the processor. Reads to some of these registers will have consequences, and there is no guarantee that careless PEEKs will not cause unexpected software operation.

3.4 POKE Command (0x03)

The POKE command is used to write the device memory. Five or more bytes of data are required. The first four are interpreted as a 32-bit pointer to the start of the memory to be written. Each subsequent byte is then a byte to be written. A maximum of 256 bytes

may be written in a single POKE. The reply message contains all of the data from the telecommand message. The message control field will contain ACK if the write has been successful, or NAK otherwise.

The memory map is as follows:

Address Range	Function
0x00001000 – 0x00007DFF	32 kB Program memory (flash)
0x00010000 – 0x000107FF	2 kB XRAM (RAM)
0x00020000 – 0x000200FF	256 B IRAM (RAM)
0x00030080 – 0x000300FF	128 B SFR (RAM)

POKEs to flash memory are not permitted when any application module is running. Flash memory 0x00000000 to 0x00000FFF and 0x000007C00 to 0x00007DFF is reserved for the NSP bootloader and may not be written to.

Each 512 byte block of program memory has a lifetime of only 20,000 write cycles. One cycle is consumed for each POKE telecommand that accesses a particular block. This lifetime is more than sufficient for occasional software patches, but the user is cautioned that a looping sequence of POKE telecommands could easily wear out a block.

3.5 TELEMETRY Command (0x04)

The TELEMETRY command gathers telemetry from the NSP bootloader only. A TELEMETRY request has a single data byte to indicate the telemetry address. The response adds a 32-bit telemetry quantity.

Telemetry Address	Function
0	Last reset reason
1	Reset count
2	Framing error count
3	Runt packet count
4	Oversize packet count
5	Bad CRC count

All of the "count" channels are stored internally as 16-bit unsigned integers. They will wrap around to 0 after reaching 65535.

3.5.1 Last Reset Reason

The last reset reason telemetry channel indicates the reason for the most reset processor reset.

0	Power cycle
1	Realtime clock reset
2	Flash memory reset
3	Comparator0 reset
4	Watchdog timer reset
5	Missing clock reset
6	External /reset pin reset
7	INIT reset
8+	Application triggered reset

3.5.2 Reset Count

This telemetry channel counts the number of times the processor has been reset since the last power cycle. When first turned on, reset count will be zero.

3.5.3 Framing Error Count

After each reset this count is zeroed. It is incremented each time a character sequence is received that violates the SLIP framing rules (FESC must be followed by TFESC or TFEND).

3.5.4 Runt Packet Count

After each reset this count is zeroed. It is incremented each time two FEND characters are received, separated by between one to four bytes. The minimum length for a telecommand is five bytes, so these cannot be valid. Note that consecutive FEND characters do not count as runts. For a runt to be counted its first byte (the nominal destination address) must be equal to the device's NSP address or the device must use an SPI link.

3.5.5 Oversize Packet Count

After each reset this count is zeroed. It is incremented each time an incoming telecommand is determined to be too large. The NSP bootloader tolerates a maximum of 260 data bytes on incoming telecommands. Oversize packets are only counted if they are addressed to this device.

3.5.6 Bad CRC Count

After each reset this count is zeroed. It is incremented each time an incoming telecommand, addressed to this device, has a bad CRC.

3.6 CRC Command (0x06)

The CRC command is used to calculate a checksum on an area of program memory. It takes a total of eight data bytes: a 32-bit start address, and a 32-bit end address. Both addresses must be within the flash memory limits (0x00000000 – 0x00007DFF) and the start must be before the end. Failure of any of these criteria will result in a NAK reply with no data.

If successful, the return message appends a 32-bit CRC of the flash memory between the start and the end. This feature can be used to verify that a large block of program memory (bootloader or application) has not suffered any corruption.

Consult Sinclair Interplanetary for details on the polynomial and setup conditions for the 32-bit CRC.

3.7 APPLICATION-TELEMETRY Command (0x07)

Application telemetry requests are NAKed if no application program is running. Otherwise the command packet is passed to the application program which interprets it and returns an appropriate response.

3.8 APPLICATION-COMMAND Command (0x08)

Application commands are NAKed if no application program is running. Otherwise the command packet is passed to the application program which interprets it and returns an appropriate response.

4 Revision History

4.1 Rev 1.0 to Rev 1.2

Corrected erroneous correction to APPLICATION_TELEMETRY and APPLICATION_COMMAND codes.

NSP Packet Protocol for Sinclair Interplanetary Hardware
Rev 1.0
February 11 2008

1. Scope

This document is intended to be read by users of Sinclair Interplanetary satellite components that use the common NSP communications protocol. It describes how NSP is used inside a spacecraft to move data to and from sensors and actuators.

The full and definitive NSP document is available from Daniel Kekez at UTIAS/SFL. It contains additional features (such as space-Earth links) not used by Sinclair Interplanetary equipment and thus not of interest to the target reader.

2. Concept

NSP is the Nanosatellite Protocol, originally developed at UTIAS/SFL for use on the CanX nanosatellites. This in turn is descended from the Simple Serial Protocol (SSP) used by UTIAS/SFL and Dynacon on the MOST and CHIPSAT spacecraft as well as the Dynacon reaction wheels in the wider market.

These protocols were developed as more and more microprocessors found their way into small spacecraft. Each satellite component will typically contain a processor, and it is convenient to manage all of its command and telemetry requirements using digital communications. In contrast to a more traditional set of analog telemetry and high-level command lines, digital links require fewer wires in the harness and are less susceptible to noise. NSP specifies a digital link standard that is used by many pieces of Sinclair Interplanetary equipment. Use of the same standard on multiple devices allows reuse of code on the part of the customer and significant savings in GSE.

The basic unit of communication in NSP is the message. A message from the satellite on-board computer (OBC) to a Sinclair Interplanetary device carries a single telecommand. The device may then reply to the OBC with a single response message. This may be a simple acknowledgement, or it may contain additional telemetry.

NSP messages can be carried on a number of different types of link: asynchronous, synchronous and I2C. Large NSP networks may contain different types of links with routers and bridges to connect them.

3. Message Format

3.1. Byte Stream

NSP messages are composed of an ordered sequence of 8-bit bytes. Wherever multiple adjacent bytes are grouped together to form larger storage units (16-bit integers, 32-bit floats, etc) the least significant bytes are transmitted and received first.

3.2. Message Fields

Length	Field
1 byte	Destination Address
1 byte	Source Address
1 byte	Message Control Field
0 or more bytes	Data
2 bytes	Message CRC

The table above shows the fields that make up an NSP message. The data field has variable length. If its length is zero (the message carries no data) then the total message contains five bytes. The maximum number of data bytes that can be placed in a message depends on the NSP implementation in each particular device. 260 bytes is a common limit.

3.3. Message Control Field

Bit 7 (MSB)	"Poll" Bit
Bit 6	"B" Bit
Bit 5	"A" (ACK) BIT
Bits 4 – 0	Command code

Within each NSP message is a message control field byte. The table above shows how the individual bits in this byte are decoded. There are three Boolean quantities, labeled "A", "B" and "Poll". There is also a five bit command code, interpreted as an unsigned integer between 0 and 31.

3.4. Message CRC

Each NSP message contains a 2 byte (16-bit) CRC to guard against errors in transmission. The 16-bit CCITT polynomial is used: x^16 + x^12 + x^5 + 1. The initial shift register value is 0xFFFF. Bytes are fed into the CRC computation starting with the destination address, and concluding with the last byte of the data field. Within a byte, bits are fed in LSB first.

The following fragment of C code, courtesy of Henry Spencer, illustrates how the CRC can be computed. It is worth noting that most Sinclair Interplanetary devices actually perform an identical calculation but using a hardware CRC engine.

```
#define POLY 0x8408 /* bits reversed for LSB-first */
unsigned short crc = 0xffff;
while (len-- > 0) {
        unsigned char ch = *bufp++;
        for (i = 0; i < 8; i++) {
                crc = (crc >> 1) ^ ( ((ch ^ crc) & 0x01) ? POLY : 0 );
                ch >>= 1;
        }
}
```

4. Telecommand Format

The OBC will send telecommands to Sinclair Interplanetary devices. These messages have the following attributes:

Destination Address	The address of the SI device in question
Source Address	The unique address of the OBC
"Poll" bit	'1' If a reply is requested '0' If no reply is desired
"B" bit	Ignored by SI device (But see Reply Format)
"A" bit	Ignored by SI device
Command Code	Desired command
Data	Data bytes, as appropriate for command
CRC	CRC, computed from the rest of the message

The list of command codes, and the formatting of the data to accompany them, is specific to the individual Sinclair Interplanetary device.

5. Telecommand Validation

When a sequence of bytes that may be a telecommand are received, a Sinclair Interplanetary device will take the following steps to validate it:

- The destination address will be compared to the set of addresses that the device will accept. Details of addressing are covered in a later section.
- The message must meet the minimum message length (5 bytes)
- The message must not exceed the maximum message length, determined by the device implementation.
- There must not have been an error in SLIP framing (see SLIP section for details).
- The message CRC must be good.

Messages that fail any of these criteria are rejected and forgotten. Valid telecommands are passed on to the next software layer for execution.

6. Reply Format

After a telecommand has been executed, a Sinclair Interplanetary device will generate a reply message if (and only if) the telecommand "Poll" bit is '1'. The reply will have the following attributes:

Destination Address	The source address from the telecommand
Source Address	The destination address from the telecommand
"Poll" bit	'1' in all cases
"B" bit	The "B" bit from the telecommand
"A" bit	'1' if the device reports an ACK condition '0' if the device reports a NACK condition
Command Code	The command code from the telecommand
Data	The data bytes from the telecommand, followed by zero or

	more bytes of telemetry
CRC	CRC, computed from the rest of the message

It can be seen that much of the reply message is a direct echo of the telecommand message. This makes inefficient use of the link bandwidth, but makes the context of each reply completely unambiguous.

7. SLIP Framing

NSP messages are encapsulated into packets using SLIP framing before being transmitted to another device. Upon reception the framing is removed. SLIP framing is described in RFC 1055.

A special character FEND (0xc0) marks both the beginning and end of each NSP message. Wherever FEND would occur within the message, it is replaced by two bytes: FESC TFEND (0xdb 0xdc). Wherever FESC would occur within the message, it is replaced by FESC TFESC (0xdb 0xdd).

When processing a SLIP framed message, it is an error to see FESC followed by anything other than TFESC or TFEND.

SLIP framing is required because not all of the link types supported by Sinclair Interplanetary feature out-of-band message completion signals.

8. NSP Addresses

NSP addresses are eight bits long. Recommended address ranges are between 1 and 127. Any user of Sinclair Interplanetary devices should pick an NSP address for the spacecraft OBC (or GSE test computer). By convention, 0x11 is used as the NSP address for the primary computer.

Some Sinclair Interplanetary devices have unique NSP addresses programmed into them in the factory. Others have pins on their connectors that can be shorted to ground to select between a number of different addresses. Finally, devices operating on synchronous (SPI) busses use out-of-band addressing and do not have a fixed NSP address. Consult the particular documentation for your device to determine its NSP address capabilities.

9. Asynchronous Encapsulation

Certain Sinclair Interplanetary devices are configured for asynchronous communications. They have one transmit and one receive channel. This may operate at +3.3 V or +5 V logic levels, or it may use an RS-485 driver.

Both channels operate at the same baud rate. Consult your device documentation for its baud rate and tolerance. There are 8 bits per byte, no parity, and 1 stop bit (8N1).

The OBC sends telecommands into the receive channels of devices. Since each device has a unique NSP address, all of the receive channels can be wired together if desired. Telecommand bytes can be sent as rapidly as desired (limited only by the baud rate) or as

slowly as desired. There is no requirement that a message be completed in a particular period of time.

A reply message (if "Poll" bit is '1') will occur quickly following the reception of the telecommand. The device will send out the reply on its transmit channel. At the conclusion of the reply (after the stop bit has been transmitted on the final FEND character) the device will turn off its output channel driver leaving only a weak pull-up. This permits multiple devices to have their transmit channels directly wired together.

A device can process only one telecommand at a time. Once a telecommand has been received, the link's receive channel is disabled and incoming bytes are ignored. The receiver is re-enabled only when the telecommand execution is complete, and the reply (if any) has been transmitted.

Both the telecommand message and the reply message (if any) are framed using SLIP.

10. Synchronous Encapsulation (SPI slave)

[Basic familiarity with the SPI standard is required to understand this section. See http://en.wikipedia.org/wiki/Spi for background.]

Some Sinclair Interplanetary devices are configured for synchronous communications (SPI slave). Each device has a Master-In-Slave-Out (MISO) pin, a Master-Out-Slave-In (MOSI) pin, a Slave Select (/SS) pin and a Serial Clock (SCK) pin. The MISO, MOSI and SCK signals are common for all of the devices on the SPI bus. However, each device has its own /SS signal, connected directly to the OBC.

MOSI, SCK and /SS are always inputs on the Sinclair Interplanetary device. When /SS is at a logic low, MISO is an output. Otherwise it is high impedance with a weak pull-up. The device outputs new data on MISO on the falling edge of SCK, and samples data from MOSI on the rising edge of SCK.

A device receives and transmits data when /SS is low. /SS must be low for the entire duration of an 8-bit byte transfer. The OBC may drive /SS high between bytes, or /SS may remain low for the entire duration of a telecommand/response transaction. /SS should not be wired low, even in applications where there is only one device on the SPI bus, as noise on SCK can then cause the device to become irrecoverably confused about where bytes begin and end.

Messages are SLIP framed as usual. They begin and end with FEND, and instances of FEND and FESC inside them are escaped. The SLIP framing protocol is also expanded to compensate for the full-duplex nature of the bus.

On an SPI connection one bit is transferred from device to OBC, and one bit from OBC to device, each SCK cycle. Thus, while receiving a telecommand the device is required to transmit something. The SLIP framing character FEND (0xc0) is sent as each telecommand byte is received.

The SCK signal is controlled by the OBC, so the Sinclair Interplanetary has no control of when it transmits and receives bytes. Consequently the OBC must poll the device when it expects a reply packet. The OBC does this by sending repeated FEND characters following the end of its telecommand message. The device will respond with FEND as it

executes the telecommand, and will then begin to respond with sequential bytes from the reply packet. Once the reply packet is finished the device will once again respond with FEND characters until the next telecommand is received.

If the device receives bytes while busy with telecommand execution it will ignore them – it can only process one telecommand at a time. However, if a byte that is not FEND is received while the device is transmitting its reply message the reply is immediately aborted and the device begins to receive the new telecommand.

Devices on an SPI bus are hardware addressed by their /SS pins. This takes priority over the destination address field in telecommands. Bytes are not received when /SS is high, and all bytes received when /SS is low are considered part of a telecommand addressed to the receiving device. For ease of software compatibility, NSP messages still contain source and destination address fields. The Sinclair Interplanetary device will simply swap the two addresses in its reply message but will otherwise ignore them.

11. I2C Encapsulation

[Basic familiarity with the I2C standard is required to understand this section. See http://en.wikipedia.org/wiki/I2c for basic background.]

The final link option available for Sinclair Interplanetary NSP devices is I2C. The devices are never bus-masters, and all transactions are initiated by the OBC. The NSP address of a device is also used as its I2C address – note that I2C addresses are only 7 bits long, while NSP implemented on other links will accept up to 8 bits.

NSP telecommand over I2C with no reply		
Transmitter	Data	Notes
OBC	START	
OBC	Slave address, Write	Slave ACK indicates slave receiving
OBC	Source address	
OBC	Message Control Field	Poll bit = 0
OBC	Outgoing data (0 – N bytes)	
OBC	16-bit CRC	
OBC	FEND	Slave ACK only if CRC valid
OBC	STOP	

The table above shows the OBC sending a telecommand to a device where no reply is required. The message is SLIP framed, except that the leading FEND is omitted as it is redundant – I2C provides an out-of-band frame start signal.

The destination address is not transmitted. Instead, the transaction begins with the slave address and the write bit, in accordance with the I2C specification. Note that the CRC is computed based on the NSP message, not on the bytes transmitted over the I2C link! When the final FEND is received the device calculates the CRC for the packet and sets the I2C ACK bit only if the CRC is valid.

NSP telecommand over I2C with reply		
Transmitter	Data	Notes
OBC	START	
OBC	Slave address, Write	Slave ACK indicates slave receiving

OBC	Source address	
OBC	Message Control Field	Poll bit = 1
OBC	Outgoing data (0 – N bytes)	
OBC	16-bit CRC	
OBC	FEND	Slave ACK only if CRC valid
OBC	START	
OBC	Slave address, Read	Slave ACK if reply packet available
Device	Message Control Field	Poll bit = 1
Device	Reply data (0 – N bytes)	
Device	16-bit CRC	
Device	FEND	
OBC	STOP	

The table above shows the case of the OBC sending a telecommand to a device where a reply is required. Instead of sending a STOP at the end of the telecommand, the OBC sends a repeated START. To comply with the I2C specification it must then re-send the slave address in read mode.

The slave then sends the reply message. This is SLIP framed, but as before the leading FEND is omitted. The destination address and source address are also both omitted. An I2C transaction is atomic, and so there can be no question of which telecommand produced which reply. Even though these fields are omitted, the CRC is still computed based on the entire NSP message in the normal fashion.

Sinclair Interplanetary devices use SCL clock stretching for flow control. While SCL is normally driven by the bus master, the device may hold it low while its internal software decides what to do next. The longest such bus stall will be just after the second slave address is received when a reply is requested. The internal processor must execute the incoming telecommand and format the reply before it releases SCL.

Sinclair Interplanetary devices implement the SMBus SCL timeout. If SCL is held low for longer than 25 msec the device will reset its I2C state.

NSP Application Software for Reaction Wheel
Sinclair Interplanetary
April 29, 2008
Revision 1.0

1. Scope

This document describes how to use the application software that is typically loaded into Sinclair Interplanetary SS-411 reaction wheels that have been configured for NSP communications. It does not pertain to units configured for CAN operation.

The application software may be patched or replaced through the NSP bootloader. Changes that affect the user interface will be accompanied by revisions of this document.

2. Overview

The reaction wheel computer maintains an internal parameter file, consisting of 256 floating-point values. Some parameters may be written to control the behaviour of the unit. Other parameters may be read as telemetry to discover the state of the unit.

Parameter zero is a special case, as it contains the mode register. In addition to a 32-bit floating-point value (the mode value), the mode register contains an 8-bit integer (the mode type). Both of these values are always read or written together. The mode is the primary command interface to the wheel. For example, a mode type might indicate "closed-loop speed mode" while the accompanying mode value might read "200 rad/sec". In this case, the wheel would operate with a constant speed of 200 rad/sec.

3. Low-Voltage and High-Voltage Wheels

The low voltage (4 V) and high voltage (28 V) versions of the wheel electronics have quite different motor drive schemes. The low voltage electronics uses a 12-bit digital-to-analog converter (DAC) to feed an analog current-mode pulse-width modulator. The pulse-width modulator's transfer function is not precisely known, but this is acceptable since the current and power control loops are robust against parameter variation. The processor does not know the exact duty cycle or frequency of the motor drive, and so all other operations are asynchronous.

The high voltage electronics use direct digital pulse-width modulation. The PWM waveform is generated from the processor clock with 8-bits of resolution. Analog telemetry measurements are made synchronously to the motor drive waveform to control noise.

The software for the two types of wheel is implemented as a single collection of C files, using #if preprocessor directives to determine a low-voltage or high-voltage target. From a user perspective the interfaces are essentially identical. The only differences lie in the motor drives, as detailed above. Low voltage wheels implement commands to directly set the DAC code, while high-voltage wheels have commands to set the PWM duty cycle. Similarly one has DAC telemetry while the other has duty cycle telemetry.

The response to a PING telemcommand while running application software will show whether that software has been built for low or high voltage electronics.

4. Startup

The application program begins at memory location 0x00001000. When the wheel first boots up it will be running the NSP bootloader. Use an INIT telecommand with a data value of 0x00001000 to start the application. A PING telecommand will verify that the program is running.

The application program runs forever and has no exit condition. To stop the program the user must send an INIT with no data to force a complete wheel reset.

5. APPLICATION_COMMAND Telecommands

APPLICATION_COMMAND telecommands are used by the host spacecraft to write into the parameter file. The reaction wheel will reply to APPLICATION_COMMAND telecomands provided the "Poll" bit is set. The reply will contain an echo of the data from the telecommand. The "A" bit will be set if the telecommand contained the correct number of bytes, and cleared otherwise. There is no other checking to make sure that commanded values are within acceptable ranges.

Two types of commands are accepted: mode commands and parameter commands.

5.1. Mode Commands

Byte 1	0
Byte 2	Mode type (unsigned 8-bits)
Bytes 3-6	Mode value (32-bit floating point)

Mode commands are APPLICATION_COMMAND telecommands with six data bytes where the first byte is zero. The second byte encodes the mode type, while the remaining four encode the mode value. Upon receipt the reaction wheel will transition to the new mode type and value.

5.2. Parameter Commands

Byte 1	Parameter number (unsigned 8-bits, 1-255)
Bytes 2-5	Parameter value (32-bit floating point)

Parameter commands are APPLICATION_COMMAND telecommands with five data bytes where the first byte is non-zero. The first byte encodes the parameter number, while the remaining four encode the parameter value. Upon receipt the reaction wheel will store the parameter value in the appropriate spot of the parameter file.

6. APPLICATION_TELEMETRY Telecommands

APPLICATION_TELEMETRY telecommands are used by the host spacecraft to read from the parameter file. The reaction wheel will reply to APPLICATION_TELEMETRY telecommands provided the "Poll" bit is set. Valid APPLICATION_TELEMETRY telecommands have one byte of data. In replying to these, the "A" bit is set and additional bytes of telemetry data are appended to the original message. Telecommands with more or less than one byte of data are echoed back with the "A" bit cleared.

Two types of telemetry requests are accepted: mode telemetry and parameter telemetry.

6.1. Mode Telemetry

Telecommand	
Byte 1	0

A mode telemetry telecommand has a single byte of data with value zero, as shown above.

Reply	
Byte 1	0
Byte 2	Mode type (unsigned 8-bits)
Bytes 3-6	Mode value (32-bit floating point)

The reply message is shown above. The telemetry message contains the wheel's current mode type and mode value.

6.2. Parameter Telemetry

Telecommand	
Byte 1	Parameter number (unsigned 8-bits, 1-255)

A parameter telemetry telecommand has a single byte of data with non-zero value, as shown above.

Reply	
Byte 1	Parameter number (unsigned 8-bits, 1-255)
Bytes 2-5	Parameter value (32-bit floating point)

The reply message is shown above. The telemetry message contains the current value of the requested parameter.

7. Modes

The mode type is an 8-bit enumerated type. The mode value is a 32-bit floating point quantity, encoded per IEEE-754. The encodings are:

Mode type	Mode name	Mode value units
0x00	MODE_IDLE	N/A (value ignored)
0x01	MODE_DAC	16-bit DAC counts + sign (low-voltage wheels) PWM duty cycle + sign (high-voltage wheels)
0x02	MODE_CURRENT	Amperes + sign
0x03	MODE_POWER	Watts + sign
0x04	MODE_BRAKE	8-bit brake count
0x05	MODE_SPEED	rad/sec
0x06	MODE_DAC_H1	16-bit DAC counts (low-voltage wheels) PWM duty cycle (high-voltage wheels)
0x07	MODE_DAC_H2	
0x08	MODE_DAC_H3	
0x09	MODE_DAC_H4	
0x0A	MODE_DAC_H5	
0x0B	MODE_DAC_H6	
0x0C	MODE_DAC_BIT	
0x0D	MODE_CURRENT_H1	Amperes
0x0E	MODE_CURRENT_H2	Amperes
0x0F	MODE_CURRENT_H3	Amperes
0x10	MODE_CURRENT_H4	Amperes
0x11	MODE_CURRENT_H5	Amperes
0x12	MODE_CURRENT_H6	Amperes
0x13	MODE_CURRENT_BIT	Amperes
0x14	MODE_ACCEL	rad/sec^2
0x15	MODE_MOMENTUM	N-m-sec
0x16	MODE_TORQUE	N-m
0x17	MODE_BURNIN	rad/sec
0x18	MODE_SFFT	N/A (value ignored)
0x19	MODE_LIFE	N/A (value ignored)
0x1A	MODE_POWER_H1	Watts
0x1B	MODE_POWER_H2	Watts
0x1C	MODE_POWER_H3	Watts
0x1D	MODE_POWER_H4	Watts
0x1E	MODE_POWER_H5	Watts
0x1F	MODE_POWER_H6	Watts

In general the mode type encodes “what to do” and the mode value “how much to do it”. Thus, a mode type of “MODE_SPEED” tells the wheel that it should operate as a closed-loop speed controller, and the target speed is encoded in the mode value.

7.1. MODE_IDLE

In this mode type the motor drive stages are turned off. If the wheel is stationary it will remain so. Otherwise it will spin-down under the influence of friction alone. The mode value is ignored.

When it is first started the application program defaults to a mode type of MODE_IDLE, and a mode value of 0.0.

7.2. MODE_DAC

The behaviour of this mode depends on whether the wheel has low-voltage or high-voltage electronics.

7.2.1. Low-Voltage Electronics

When using low-voltage electronics, valid mode values are between -65535.0 and +65535.0. In this mode type the motor drive stages are turned on and the drive digital-to-analog converter (DAC) is fed a 16-bit code equal to the absolute value of the mode value. The sign of the mode value determines the direction of the drive stages, and commutation is dictated by the Hall-effect sensors.

The relationship between DAC code and motor drive current is non-linear and varies significantly from phase to phase. This mode is largely used for testing. A mode value of 65535.0 will cause the wheel to generate its maximum torque, subject to safety limits.

7.2.2. High-Voltage Electronics

When using high-voltage electronics, valid mode values are between -1.0 and +1.0. The absolute value of the mode value is used as the PWM duty-cycle, while the sign determines the direction of the drive stages. Commutation is dictated by the Hall-effect sensors.

The relationship between PWM duty-cycle and motor drive current is linear and stable. This mode is largely used for testing. A mode value of 1.0 will cause the wheel to generate its maximum torque, subject to safety limits.

7.3. MODE_CURRENT

In this mode the motor drive stages are turned on, and the drive is servoed to keep the current consumption equal to the absolute value of the mode value. The sign of the mode value determines the direction of the drive stages, and commutation is dictated by the Hall-effect sensors.

Note that the current consumption of the drive stages is not equal to the current in the motor windings. At low speeds the motor winding current will be much larger than the stage current consumption due to the step-down nature of the DC/DC conversion.

If the specified current cannot be reached, the drive will saturate at is maximum output.

7.4. MODE_POWER

In this mode the motor drive stages are turned on, and the drive digital-to-analog converter (DAC) is servoed to keep the power consumption equal to the absolute value of the mode value. The sign of the mode value determines the direction of the drive stages, and commutation is dictated by the Hall-effect sensors. Power is estimated by multiplying the current consumption by the bus voltage.

If the specified power cannot be reached, the drive will saturate.

7.5. MODE_BRAKE

MODE_BRAKE is not intended for operational use. It is a hold-over from previous code generations, and may be removed in the future. Valid mode values are between 0.0 and +255.0. In addition, correct behaviour may not occur at values close to these extremes.

In this mode the drive stages are configured for dissipative braking. The larger the mode value, the greater the brake. There is no closed-loop control, and so the braking torque will also be roughly proportional to the wheel speed. At speeds close to zero there will be no useful braking.

This mode produces an acoustic tone around 8 kHz when the wheel turns at high speed. This is normal.

7.6. MODE_SPEED

This mode provides closed-loop control over the wheel speed. It overlies MODE_POWER, and an additional low-rate controller servoes the power setpoint and drive direction to achieve the target speed. The wheel speed and direction is estimated from the Hall-effect sensors, and used as the control feedback.

The gains and limits of the speed controller are reconfigurable – see the parameter file documentation for more information.

7.7. MODE_DAC_Hx

There are six mode types, for H = 1 to 6. Valid mode ranges are between 0.0 and +65535.0 (low-voltage electronics) or 0.0 and +1.0 (high-voltage electronics).

This mode is equivalent to MODE_DAC. However, the Hall-effect sensors are not used for commutation. Instead, the motor drive stages are configured as if the Hall-effect sensors are currently reading a pattern given by the binary value of x (1 to 6).

The wheel will typically not spin freely in this mode. Large continuous stall currents are possible. Be careful not to overheat the motor. This mode can be used to apply controlled preheat to a wheel that has become very cold on-orbit. The mode can also be used to implement a crude stepper motor drive which may be useful to explore mechanical or electrical failures.

7.8. MODE_DAC_BIT

This mode steps through a pre-programmed built-in-test sequence lasting approximately 30 seconds to explore and record the health of the drive stages. Valid mode ranges are between 0.0 and +65535.0 (low-voltage electronics) or 0.0 and +1.0 (high-voltage electronics)..

The sequence steps through functionality equivalent to each of MODE_DAC_H1 to MODE_DAC_H6, spending approximately 6 seconds in each. At the completion of the sequence the mode register is re-written to put the wheel into MODE_IDLE.

At the end of each of the steps the current telemetry is recorded into parameter file BIT_Hx_FILE. These parameters can be later read back and analyzed to determine the functionality of the drive stages.

7.9. MODE_CURRENT_Hx

There are six mode types, for H = 1 to 6. For proper operation the mode range should be greater or equal to zero.

This mode is equivalent to MODE_CURRENT. However, the Hall-effect sensors are not used for commutation. Instead, the motor drive stages are configured as if the Hall-effect sensors are currently reading a pattern given by the binary value of x (1 to 6).

The wheel will typically not spin freely in this mode. Large continuous stall currents are possible. Be careful not to overheat the motor. This mode can be used to apply controlled preheat to a wheel that has become very cold on-orbit. The mode can also be used to implement a crude stepper motor drive which may be useful to explore mechanical or electrical failures.

7.10. MODE_CURRENT_BIT

This mode steps through a pre-programmed built-in-test sequence lasting approximately 30 seconds to explore and record the health of the drive stages. For proper operation the mode range should be greater or equal to zero.

The sequence steps through functionality equivalent to each of MODE_CURRENT_H1 to MODE_CURRENT_H6, spending approximately 6 seconds in each. At the completion of the sequence the mode register is re-written to put the wheel into MODE_IDLE.

At the end of each of the steps the DAC code telemetry (low-voltage electronics) or PWM duty-cycle (high-voltage electronics) is recorded into parameter file BIT_Hx_FILE. These parameters can be later read back and analyzed to determine the functionality of the drive stages.

7.11. MODE_ACCEL

This mode provides closed-loop control over the wheel acceleration. It overlies MODE_POWER, and an additional low-rate controller servoes the power setpoint and drive direction to achieve the target acceleration. The wheel acceleration and direction is estimated from the Hall-effect sensors, and used as the control feedback.

The gains and limits of the acceleration controller are reconfigurable – see the parameter file documentation for more information.

7.12. MODE_MOMENTUM

This mode overlies MODE_SPEED. The target momentum setpoint is divided by the INERTIA_FILE parameter to determine the target speed.

7.13. MODE_TORQUE

This mode overlies MODE_ACCEL. The target torque setpoint is divided by the INERTIA_FILE parameter to determine the target acceleration.

7.14. MODE_BURNIN

This mode overlies MODE_SPEED. For the first hour of operation the mode value is passed directly to the speed controller. For the second hour, the sign of the target speed is reversed. The wheel alternates between forward and reverse operation once an hour for as long as this mode runs.

The mode is intended to be run overnight on brand-new wheels to properly distribute the grease within the bearings.

7.15. MODE_SFFT

This mode executes a Short-Form Functional Test (SFFT) profile. The wheel runs through a number of simulated commands and stores key telemetry to the parameter file for later analysis. More information is given in the SFFT section.

7.16. MODE_LIFE

This mode executes an endless sequence of simulated commands intended to stress the wheel mechanical and electrical components. It is used to validate the wheel lifetime estimates by running on qualification articles for months on end.

The life test sequences vary slightly for the low-voltage and high-voltage electronics.

7.16.1. Low-Voltage Electronics

The life test sequence is:

Mode type	Mode value
MODE_DAC	65535.0
MODE_IDLE	0.0
MODE_DAC	-65535.0
MODE_IDLE	0.0
MODE_POWER	0.5
MODE_BRAKE	128.0
MODE_POWER	-0.5
MODE_BRAKE	128.0

Each step lasts for 4 minutes. If polled the mode register will read a mode type of MODE_LIFE.

7.16.2. High-Voltage Electronics

The life test sequence is:

Mode type	Mode value
MODE_DAC	0.9
MODE_IDLE	0.0
MODE_DAC	-0.9
MODE_IDLE	0.0
MODE_POWER	0.5
MODE_DAC	0.0
MODE_POWER	-0.5
MODE_DAC	0.0

Each step lasts for 4 minutes. If polled the mode register will read a mode type of MODE_LIFE.

7.17. MODE_POWER_Hx

There are six mode types, for H = 1 to 6. For proper operation the mode range should be greater or equal to zero.

This mode is equivalent to MODE_POWER. However, the Hall-effect sensors are not used for commutation. Instead, the motor drive stages are configured as if the Hall-effect sensors are currently reading a pattern given by the binary value of x (1 to 6).

The wheel will typically not spin freely in this mode. Large continuous stall currents are possible. Be careful not to overheat the motor. This mode can be used to apply controlled preheat to a wheel that has become very cold on-orbit. The mode can also be used to implement a crude stepper motor drive which may be useful to explore mechanical or electrical failures.

This mode is intended to be used with the TEST_TONE feature to tune the power control loop.

8. Parameter File

8.1. Parameter List

Each entry in the parameter file consists of a floating-point number in 32-bit IEEE-754 format. Floating point values are used exclusively, even though some of the parameters represent integer quantities. All 255 parameters are implemented, though many are unused by the internal software.

Parameter number	Parameter name	Type	Default	Units
0x01	VOLTAGE_FILE	Read-only	0.0	Volts
0x02	CURRENT_FILE	Read-only	0.0	Amperes
0x03	TEMPERATURE_FILE	Read-only	0.0	Celsius

0x04	DAC_FILE	Read-only	0.0	DAC Counts (LV) or duty cycle (HV)
0x05	SPEED_FILE	Read-only	0.0	rad/sec
0x06	SPEED_P_FILE	Read/write	6.0×10^{-4}	
0x07	SPEED_I_FILE	Read/write	6.0×10^{-6}	
0x08	SPEED_D_FILE	Read/write	0.0	
0x09	SPEED_INTEGRATOR_FILE	Read/write	0.0	
0x0A	POWER_LIMIT_FILE	Read/write	0.7	Watts
0x0B	PREVIOUS_SPEED_FILE	Read-only	0.0	rad/sec
0x0C	BIT_H1_FILE	Read/write	0.0	rad/sec OR Watts
0x0D	BIT_H2_FILE	Read/write	0.0	rad/sec OR Watts
0x0E	BIT_H3_FILE	Read/write	0.0	rad/sec OR Watts
0x0F	BIT_H4_FILE	Read/write	0.0	rad/sec OR Watts
0x10	BIT_H5_FILE	Read/write	0.0	rad/sec OR Watts
0x11	BIT_H6_FILE	Read/write	0.0	rad/sec OR Watts
0x12	ACCEL_FILE	Read-only	0.0	rad/sec^2
0x13	ACCEL_P_FILE	Read/write	0.0	
0x14	ACCEL_I_FILE	Read/write	1.4×10^{-7}	
0x15	ACCEL_I2_FILE	Read/write	0.014	
0x16	ACCEL_D_FILE	Read/write	0.0	
0x17	ACCEL_INTEGRATOR_FILE	Read/write	0.0	
0x18	ACCEL_INTEGRATOR2_FILE	Read/write	0.0	
0x19	PREVIOUS_ACCEL_FILE	Read-only	0.0	rad/sec^2
0x1A	INERTIA_FILE	Read/write	5.12×10^{-5} (LV) 8.78×10^{-5} (HV)	$kg\text{-}m^2$
0x1B	MOMENTUM_FILE	Read-only	0.0	N-m-sec
0x1C	TORQUE_FILE	Read-only	0.0	N-m
0x1D	ACCEL_OFFSET_SPEED_FILE	Read/write	10.0	rad/sec
0x1E	ACCEL_FILTER_FILE	Read/write	0.0	rad/sec^2
0x1F	TORQUE_FILTER_FILE	Read-only	0.0	N-m-sec
0x20	FILTER_TAU_FILE	Read/write	1.0	sec
0x21	HALL_OVERFLOW_FILE	Read-only	0.0	counts
0x22	HALL_IMPOSSIBLE_FILE	Read-only	0.0	counts
0x23	HALL_SKIP_FILE	Read-only	0.0 or 1.0	counts

0x24	IDLE_CADENCE_FILE	Read-only	0.0	counts
0x25	IDLE_INHIBIT_FILE	Read/write	0.0	Boolean
0x26	OVERSPEED_LIMIT1_FILE	Read/write	680.0	rad/sec
0x27	OVERSPEED_LIMIT2_FILE	Read/write	700.0	rad/sec
0x28	ADC_CADENCE_FILE	Read-only	0.0	counts
0x29	SEU_COUNT_FILE	Read/write	0.0	counts
0x2A	MAX_PWM_FILE	Read/write (LV) Read-only (HV)	65535.0 (LV) 0.9 (HV)	DAC Counts (LV) or duty cycle (HV)
0x2B	MIN_PWM_FILE	Read-only		Duty cycle
0x2C	LOOP_I_GAIN_FILE	Read/write	16.0 (LV) 250.0 (HV)	
0x2D	MAX_CORRECTION_FILE	Read/write	1024.0 (LV) 0.015625 (HV)	
0x2E	TEST_TONE_FILE	Read/write	0.0	Boolean
0x2F	CURRENT_OFFSET_FILE	Read/write	0.0	
0x30 – 0x7F	Unused	Read/write	0.0	
0x80 – 0xE3	Short-form Functional Test Results	Read/write	0.0	
0xE4 – 0xFF	Unused	Read/write	0.0 (except 0xFF which is not initialized)	

(LV) refers to low-voltage electronics, and (HV) to high-voltage electronics.

8.2. Analog Telemetry Group

The analog telemetry group consists of:

- VOLTAGE_FILE
- CURRENT_FILE
- TEMPERATURE_FILE
- DAC_FILE
- CURRENT_OFFSET_FILE

These report the instantaneous state of the ADC or DAC channels. With the exception of TEMPERATURE_FILE, these channels are all undersampled. The parameters are only updated at the control frame rate (93 Hz), while the underlying VOLTAGE, CURRENT and DAC channels are updated at many tens of kilohertz.

VOLTAGE_FILE reports the bus voltage inside the wheel. It is scaled to engineering units, but reads 6% low. This is a known deficiency in the present hardware design, and will be corrected in future generations.

CURRENT_FILE reports the current entering the motor drive stages. It is scaled to Amperes, and should be accurate to within 2%. The sensing is unipolar, and if current is instead leaving the motor drive stages the telemetry will report 0.0.

TEMPERATURE_FILE reports the temperature of the onboard processor die. The absolute accuracy of this channel is poor, and scatter of ±5ºC or even ±10ºC between wheels is not unexpected. Fortunately the wheel operates over a wide range of temperatures so the precise temperature is seldom of interest.

DAC_FILE is interpreted in two ways, depending on whether the wheel uses low-voltage or high-voltage electronics. For low-voltage, DAC_FILE reports the instantaneous code at the motor drive digital-to-analog converter. For high-voltage, DAC_FILE represents the instantaneous PWM duty cycle of the motor drive (expressed as a number between -1.0 and +1.0).

If the wheel mode type is MODE_DAC or one of the variants then DAC_FILE should equal the mode value. Otherwise DAC_FILE can be used to see the output of the high-speed current or power controllers. Expect this channel to be very noisy as the controller reacts to high-frequency disturbances.

The CURRENT_OFFSET_FILE is implemented only on high-voltage units. These units have bi-directional motor drives, capable of acting as generators under certain circumstances. The current sensors on these units are also bi-directional. To compensate for telemetry offset, the current sensor is measured whenever the wheel is in IDLE_MODE. The ADC code is stored in CURRENT_OFFSET_FILE. This number is subtracted from the ADC result whenever normal current measurements are being made.

8.3. Motion Telemetry Group

The motion telemetry group consists of:

- SPEED_FILE
- ACCEL_FILE
- MOMENTUM_FILE
- TORQUE_FILE
- ACCEL_FILTER_FILE
- TORQUE_FILTER_FILE

These files are also closely associated:

- INERTIA_FILE
- FILTER_TAU_FILE

These report telemetry concerning the motion of the rotor as measured by the Hall-effect sensors.

SPEED_FILE reports the speed of the rotor, averaged over the most recent complete revolution. Thus, at low speeds, the latency may become non-trivial. Speeds below approximately 0.5 rad/sec are reported as 0.0 in order to keep the latency bounded.

ACCEL_FILE reports the acceleration of the rotor, based on the difference in speed between the most recent measurement and a measurement in the last control frame 11 msec ago. As a derivative of a real quantity, ACCEL_FILE carries significant noise.

MOMENTUM_FILE and TORQUE_FILE are simply SPEED_FILE and ACCEL_FILE multiplied by INERTIA_FILE.

To mitigate the noise in ACCEL_FILE, ACCEL_FILTER_FILE is available. This provides acceleration telemetry that has been passed through a one-pole IIR low-pass filter. The time constant for this filter is stored in FILTER_TAU_FILE. ACCEL_FILTER_FILE can be written, and may be used to test the filter response to an upset.

TORQUE_FILTER_FILE is ACCEL_FILTER_FILE multiplied by INERTIA_FILE. This channel is not writeable.

8.4. Anomaly Count Group

The anomaly count group consists of:

- HALL_IMPOSSIBLE_FILE
- HALL_OVERFLOW_FILE
- HALL_SKIP_FILE
- SEU_COUNT_FILE

These count anomalous conditions that may indicate problems with the onboard electronics or software.

HALL_IMPOSSIBLE_FILE counts the number of times that the Hall-effect sensors transition to an “impossible” pattern. This would be when all three sensors read 0, or all three read 1. Counts here indicate a serious problem: damaged sensors, or a disintegrated rotor.

HALL_OVERFLOW_FILE counts the number of times that two or more Hall-effect sensors appear to change simultaneously. Counts here probably indicate a problem with realtime performance. The processor has been too busy to record the last sensor transition, and so when a second transition occurs it appears that both have happened at once.

HALL_SKIP_FILE counts the number of times that the Hall-effect sensors transition from one pattern to another, where the two patterns should be separated by more than one angular step. Note that due to the way the software is written a single count may accumulate on turn-on.

All three counters described above are based on internal 8-bit integer quantities. If more than 255 counts accumulate the counters will wrap around to 0.

SEU_COUNT_FILE counts the number of radiation-induced upsets in a target region of RAM. This region is approximately 600 bytes long, varying depending on the exact software version run. The RAM is initially loaded with a test pattern, and is then checked during the processor idle time. When an error is seen the counter is incremented and the byte is rewritten to its correct value. This telemetry allows a measure of the SEU rate, and hence the proton fluence, in the spacecraft orbit.

8.5. Realtime Cadence Group

The realtime cadence group consists of:

- ADC_CADENCE_FILE
- IDLE_CADENCE_FILE
- IDLE_INHIBIT_FILE

This group in intended to verify that the processor has adequate realtime margin in its target configuration.

The IDLE_INHIBIT_FILE, while formatted as a float, is treated as a Boolean quantity. If it is 0.0, it is considered FALSE. Any other value is considered TRUE. When FALSE, the processor is allowed to stop the clock to the ALU between interrupts. This saves a measureable amount of power (~1mA). When TRUE, the processor is not allowed to stop the clock. Between interrupts it will spend all of its time polling the NSP Module to see if a new packet has come in, running the SEU counter experiment, and incrementing the cadence counter.

IDLE_CADENCE_FILE counts the number of times the idle loop is executed in a given control frame. When IDLE_INHIBIT_FILE is FALSE, the loop should be run at a controlled 3 kHz. The control frame runs at 93 Hz, so the parameter will normally have a value around 32. When IDLE_INHIBIT_FILE is TRUE, the loop will run much faster and the count will be several thousand. The difference between the two values may be interpreted as a measure of the processor's realtime margin.

IDLE_INHIBIT_FILE is FALSE by default in order to save power. It may be set to TRUE to measure the realtime margin, as above. It may also be used to intentionally increase power consumption as a heater, or to decrease the packet communications latency if this becomes critical.

ADC_CADENCE_FILE counts the number of times the ADC runs through a combined measurement of current and voltage in a given control frame. The control frame runs at 93 Hz, and the ADC loop rate is on the order of 30 kHz, so counts should be expected around 300.

8.6. Power Controller Parameters

The power controller parameters group consists of:

- LOOP_I_GAIN_FILE
- MAX_CORRECTION_FILE
- TEST_TONE_FILE

These govern the behaviour of the high-frequency current and power control loops. Typically this loop can only be observed in the lab with an oscilloscope, so modifications on-orbit must be done very carefully.

The LOOP_I_GAIN_FILE controls the gain of the high-frequency closed-loop controllers. As presently implemented, the controllers have only an integral term and thus there is only an "I" gain. The gain must be high enough to track the motor back-EMF pulsations as it rotates, but not so high as to cause instability.

The MAX_CORRECTION_FILE limits the maximum change in DAC_FILE that can be caused in each frame of the high-frequency controller. It is intended to prevent a single bad ADC sample from causing the controller to make a large excursion. The values chosen are large enough that the controller will have no trouble tracking the motor back-EMF at even the fastest rotation speed.

The TEST_TONE_FILE allows laboratory testing of the high-frequency loops. When set to 1.0 a square-wave at ~300 Hz is injected into the controller. By observing the motor drive with an oscilloscope the step response of the high-frequency controller can be determined. The test tone should be left off when not needed.

8.7. Motion Controller Gains

The motion controller gains group consists of:

- SPEED_P_FILE
- SPEED_I_FILE
- SPEED_D_FILE
- SPEED_INTEGRATOR_FILE
- PREVIOUS_SPEED_FILE
- ACCEL_P_FILE
- ACCEL_I_FILE
- ACCEL_I2_FILE
- ACCEL_D_FILE
- ACCEL_INTEGRATOR_FILE
- ACCEL_INTEGRATOR2_FILE
- PREVIOUS_ACCEL_FILE
- ACCEL_OFFSET_SPEED_FILE
- POWER_LIMIT_FILE

These govern the behaviour of the speed and acceleration closed-loop controllers.

X_P_FILE, X_I_FILE and X_D_FILE are the P, I and D gains from the standard PID controller. These gains are based on a 93 Hz control frame. Divide the D gain by 93 and multiply the I gain by 93 to get the actual gains for a continuous time simulation.

The acceleration controller has an additional second-order integrator. Due to the way this is implemented, the gain is the product of ACCEL_I_FILE and ACCEL_I2_FILE.

The accumulator for each of the integrators is stored in SPEED_INTEGRATOR_FILE, ACCEL_INTEGRATOR_FILE and ACCEL_INTEGRATOR2_FILE. The integrators can be directly written and read if needed for testing, but generally they should be left alone.

The differential terms in the controller use PREVIOUS_SPEED_FILE and PREVIOUS_ACCEL_FILE to store the value from the previous frame. These will be of little interest to the user.

ACCEL_OFFSET_SPEED_FILE helps to determine the acceleration controller behaviour at low speeds. See the controller design section for more details.

The POWER_LIMIT_FILE caps the maximum actuation power that the speed and acceleration controllers can request. It prevents the wheel from unnecessarily blowing breakers in the power system. It also helps to determine the speed controller's response to large step commands. Note that it does not restrict the power in non-closed-loop modes.

8.8. Hardware Protection Group

The hardware protection group consists of:

- MAX_PWM_FILE
- MIN_PWM_FILE
- OVERSPEED_LIMIT1_FILE
- OVERSPEED_LIMIT2_FILE

These help to ensure that the reaction wheel hardware cannot be damaged by bad commands.

For low-voltage wheels, MAX_PWM_FILE is read/write. It limits the maximum DAC value that can be set in both open-loop and closed-loop control modes. The default value of 65535.0 essentially removes this limit. The design of the low-voltage electronics is such that high DAC values are not immediately dangerous though they may eventually lead to stator overheat. MIN_PWM_FILE is not used for low-voltage wheels.

For high-voltage wheels, MAX_PWM_FILE and MIN_PWM_FILE are read-only. The software continually computes the largest and smallest safe PWM duty cycles, based on the bus voltage and the wheel speed. The motor drive PWM is bounded by these two values in both open-loop and closed-loop control modes. This protection is necessary to prevent burn-out of the drive stages at high bus voltages. For this reason, the behaviour cannot be overridden without uploading new application code.

The two overspeed limits are intended to prevent excessive rotor speed due to bad commands. This is of limited importance at low voltages where the motor back-EMF will naturally limit the maximum speed. However, at high bus voltages the motor back-EMF is no longer an effective limit and the software limits must be used.

Closed-loop control modes (speed, torque, acceleration, momentum) will attempt to not exceed OVERSPEED_LIMIT1_FILE. Once the limit is reached they will hold at this speed.

If the rotor speed exceeds OVERSPEED_LIMIT2_FILE, either through an open-loop mode or by failure of a closed-loop mode, the motor drive will be turned off. As soon as the rotor speed drops below the limit the motor drive is re-enabled. This provides a highly reliable absolute limit to the speed. However, due to the bang-bang nature of the limit some oscillation should be expected in saturation.

To achieve the proper overspeed limit behaviour, OVERSPEED_LIMIT1_FILE should be less than OVERSPEED_LIMIT2_FILE.

8.9. Built-In Test Results Group

There are a total of 105 files devoted to storing the results of built-in tests. These are intended to simplify on-orbit diagnostics where the telemetry bandwidth for realtime logging may not exist. There are two sub-groups here:

- BIT_Hx_FILE
- SFFT results

The six BIT_Hx_FILEs store the results from MODE_DAC_BIT or MODE_CURRENT_BIT tests. In MODE_DAC_BIT the controller holds the DAC code constant and logs the current in these parameters. In MODE_CURRENT_BIT the controller holds the current constant and logs the DAC code in these parameters. Thus, the parameters may have units of either Amperes or DAC counts depending on which BIT test has been most recently run. The large difference in magnitude between the two measurement types eliminates any ambiguity.

9. Short-Form Functional Test

9.1. Concept

A short-form functional test (SFFT) is a sequence of commands and expected responses that can be used to verify the correct functioning of a device. An SFFT is typically run after any significant handling or integration activity to ensure that the device has not been damaged. A long-form functional test (LFFT) is comprised of a number of SFFTs run at different voltages, temperatures, pressures, etc.

Traditionally, an SFFT was performed by an operator moving through a checklist and writing down results. This is tedious, and carries the risk of mistake. Another approach is to program the command sequence into a ground-support computer to automatically test and log the data. This is attractive, but may be difficult to support at all levels of spacecraft integration.

Sinclair Interplanetary reaction wheels execute the SFFT and log the results internally as part of a built-in test. This takes the burden off both of the operator and the ground-support equipment. An SFFT can even be executed on-orbit to determine if wheel parameters have changed after launch.

To start an SFFT, send a MODE telecommand with mode type MODE_SFFT. The mode value is ignored. The test takes approximately 25 minutes to complete. Once it is done the wheel will go into MODE_IDLE. The test may be aborted at any time by sending another MODE telecommand to place the wheel into a different mode.

The wheel stores a complete set of health telemetry into parameter files 0x80 – 0xE3. These parameters can then be downloaded and examined. The total number of parameters is naturally limited by the modest amount of RAM available in the wheel. Users may choose to poll additional telemetry channels (speed, torque, etc) in real-time while the test is running. The wheel remains fully responsive during the SFFT.

9.2. Test Sequence

During the SFFT, the wheel will execute a number of internally generated commands. These are equivalent to MODE telecommands, and they exercise the open-loop and closed-loop portions of the wheel software. Note however that MODE_TELEMETRY requests during the test will return MODE_SFFT, and not the control mode that is being currently used.

There are two slightly different test sequences. One is used for low-voltage wheels, and one for high-voltage wheels.

9.2.1. Low-Voltage Test Sequence

Time (sec)	Mode type	Mode value
0	MODE_IDLE	0.0
10	MODE_DAC	65535.0
70	MODE_IDLE	0.0
190	MODE_DAC	-65535.0
250	MODE_IDLE	0.0
370	MODE_POWER	0.5
430	MODE_BRAKE	128.0
550	MODE_POWER	-0.5
610	MODE_BRAKE	128.0
730	MODE_SPEED	100.0
740	MODE_SPEED	110.0
750	MODE_SPEED	100.0
760	MODE_SPEED	200.0
770	MODE_SPEED	210.0
780	MODE_SPEED	200.0
790	MODE_SPEED	300.0
800	MODE_SPEED	310.0
810	MODE_SPEED	300.0
820	MODE_SPEED	400.0
830	MODE_SPEED	410.0
840	MODE_SPEED	400.0
850	MODE_SPEED	500.0
860	MODE_SPEED	510.0
870	MODE_SPEED	500.0
880	MODE_ACCEL	-10.0
980	MODE_SPEED	0.0
1040	MODE_SPEED	10.0
1060	MODE_SPEED	-10.0
1080	MODE_SPEED	1.0
1100	MODE_SPEED	2.0
1120	MODE_SPEED	5.0
1140	MODE_SPEED	2.0
1160	MODE_BRAKE	128.0

1280	MODE_DAC_BIT	30000.0

9.2.2. High-Voltage Test Sequence

Time (sec)	Mode type	Mode value
0	MODE_IDLE	0.0
10	MODE_DAC	0.9
70	MODE_IDLE	0.0
190	MODE_DAC	-0.9
250	MODE_IDLE	0.0
370	MODE_POWER	0.5
430	MODE_DAC	0.0
550	MODE_POWER	-0.5
610	MODE_DAC	0.0
730	MODE_SPEED	100.0
740	MODE_SPEED	110.0
750	MODE_SPEED	100.0
760	MODE_SPEED	200.0
770	MODE_SPEED	210.0
780	MODE_SPEED	200.0
790	MODE_SPEED	300.0
800	MODE_SPEED	310.0
810	MODE_SPEED	300.0
820	MODE_SPEED	400.0
830	MODE_SPEED	410.0
840	MODE_SPEED	400.0
850	MODE_SPEED	500.0
860	MODE_SPEED	510.0
870	MODE_SPEED	500.0
880	MODE_ACCEL	-10.0
980	MODE_SPEED	0.0
1040	MODE_SPEED	10.0
1060	MODE_SPEED	-10.0
1080	MODE_SPEED	1.0
1100	MODE_SPEED	2.0
1120	MODE_SPEED	5.0
1140	MODE_SPEED	2.0
1160	MODE_BRAKE	128.0
1280	MODE_DAC_BIT	30000.0

Note that the final command really does use a value of 30000.0. Since the duty-cycle range is 0.0 to 1.0, this effectively supplies a duty-cycle equal to MAX_PWM_FILE.

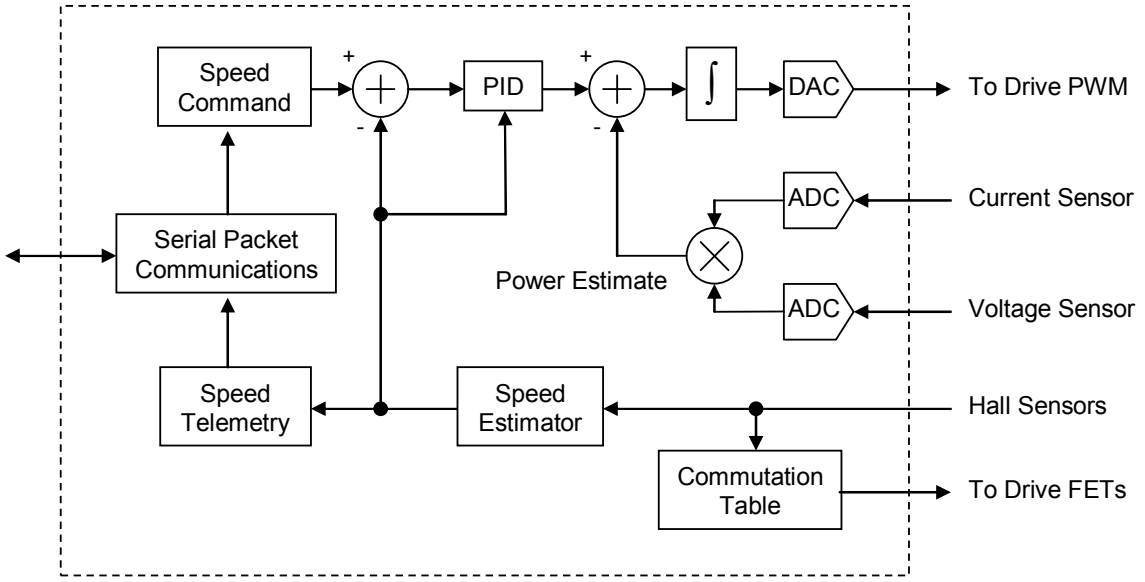
9.3. Telemetry Files

The SFFT logs telemetry to the parameter file as it runs. Telemetry is logged in batches of three samples at one time. These are detailed in the following table:

Time (sec)	Param 1	Telem 1	Param 2	Telem 2	Param 3	Telem 3
5	0x80	Voltage	0x81	Current	0x82	Temperature
65	0x83	Speed	0x84	Torque_filt	0x85	Current
130	0x86	Speed	0x87	Torque_filt	0x88	Current
245	0x89	Speed	0x8A	Torque_filt	0x8B	Current
310	0x8C	Speed	0x8D	Torque_filt	0x8E	Current
425	0x8F	Speed	0x90	Torque_filt	0x91	Current
490	0x92	Speed	0x93	Torque_filt	0x94	Current
605	0x95	Speed	0x96	Torque_filt	0x97	Current
670-	0x98	Speed	0x99	Torque_filt	0x9A	Current
735	0x9B	Speed	0x9C	Torque_filt	0x9D	Current
745	0x9E	Speed	0x9F	Torque_filt	0xA0	Current
755	0xA1	Speed	0xA2	Torque_filt	0xA3	Current
765	0xA4	Speed	0xA5	Torque_filt	0xA6	Current
775	0xA7	Speed	0xA8	Torque_filt	0xA9	Current
785	0xAA	Speed	0xAB	Torque_filt	0xAC	Current
795	0xAD	Speed	0xAE	Torque_filt	0xAF	Current
805	0xB0	Speed	0xB1	Torque_filt	0xB2	Current
815	0xB3	Speed	0xB4	Torque_filt	0xB5	Current
825	0xB6	Speed	0xB7	Torque_filt	0xB8	Current
835	0xB9	Speed	0xBA	Torque_filt	0xBB	Current
845	0xBC	Speed	0xBD	Torque_filt	0xBE	Current
855	0xBF	Speed	0xC0	Torque_filt	0xC1	Current
865	0xC2	Speed	0xC3	Torque_filt	0xC4	Current
875	0xC5	Speed	0xC6	Torque_filt	0xC7	Current
930	0xC8	Speed	0xC9	Torque_filt	0xCA	Current
1030	0xCB	Speed	0cCC	Torque_filt	0xCD	Current
1050	0xCE	Speed	0xCF	Torque_filt	0xD0	Current
1070	0xD1	Speed	0xD2	Torque_filt	0xD3	Current
1090	0xD4	Speed	0xD5	Torque_filt	0xD6	Current
1110	0xD7	Speed	0xD8	Torque_filt	0xD9	Current
1130	0xDA	Speed	0xDB	Torque_filt	0xDC	Current
1150	0xDD	Speed	0xDE	Torque_filt	0xDF	Current
1260	0xE0	Voltage	0xE1	Current	0xE2	Temperature

For example, at a point 930 seconds into the SFFT the speed is logged into parameter 0xC8, the filtered torque is logged into 0xC9 and the current is logged into 0xCA. In addition to those the standard BIT_Hx_FILE telemetry is logged during the MODE_DAC_BIT portion.

10. Controller Design



The figure above shows a block diagram of the software in speed control mode. The top-right quadrant is the high-rate power control loop. The ADC samples the bus current and voltage at high rate (~30 kHz each). These two measurements are multiplied in software and compared to a current setpoint target. The controller has only an integrator term, and the gain is given by LOOP_I_GAIN_FILE. The integrator is clamped by MAX_PWM_FILE and MIN_PWM_FILE.

Power control is used to eliminate the motor torque ripple. Ignoring winding resistance loss, the motor power is equivalent to the torque multiplied by the speed. By holding the power constant over short time periods (when the speed is also essentially constant) the torque is constant even if the motor torque constant is highly variable with angle.

Running outside the power controller is a speed controller. Speed is estimated from the Hall sensor pulse train, and compared to the speed command. The power setpoint is then computed from a PID controller. A complication is that the torque at constant power is a function of speed. Thus, a set of PID gains appropriate for one wheel speed do not give good results at a different wheel speed. Consequently, the PID controller is implemented with variable, speed-dependent gains.

$$G_D = \frac{|SPEED\_FILE| + |target\_speed|}{2} \times SPEED\_D\_FILE$$

$$G_P = \frac{|SPEED\_FILE| + |target\_speed|}{2} \times SPEED\_P\_FILE$$

$$G_I = \frac{|SPEED\_FILE| + |target\_speed|}{2} \times SPEED\_I\_FILE$$

This works well, but with the exciting side effect that at zero speed, when commanded to zero speed, the gain is zero! Special case code turns off the drive stages when this occurs.

The acceleration controller is slightly different, in that there is no target speed. Instead, the gains are computed as:

$$G_D = \frac{|SPEED\_FILE| + ACCEL\_OFFSET\_SPEED\_FILE}{2} \times ACCEL\_D\_FILE$$

$$G_P = \frac{|SPEED\_FILE| + ACCEL\_OFFSET\_SPEED\_FILE}{2} \times ACCEL\_P\_FILE$$

$$G_I = \frac{|SPEED\_FILE| + ACCEL\_OFFSET\_SPEED\_FILE}{2} \times ACCEL\_I\_FILE$$

$$G_{I2} = \frac{|SPEED\_FILE| + ACCEL\_OFFSET\_SPEED\_FILE}{2} \times ACCEL\_I\_FILE \times ACCEL\_I2\_FILE$$

Thus the acceleration controller suffers from no dead-spot where gain is zero.

Each of the integrators is bounded by POWER_LIMIT. This is used to prevent integrator windup during large slews.