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.

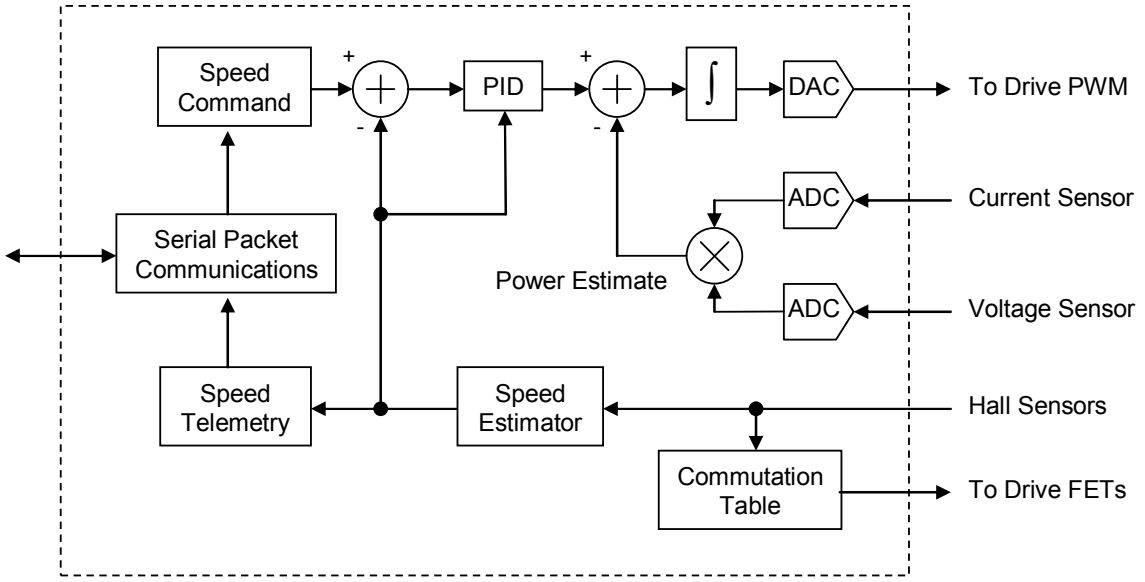
9.3. Telemetry Files

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.