Sirius TCM User Manual - with TCM Core Application

v1.22.0

TABLE OF CONTENTS

1. Introduction

The AAC Clyde space Sirius line of products, which will be referred to as "the Sirius products" in this document, consist of:

- Sirius OBC
- Sirius TCM
- Sirius TCM - with TCM Core Application

This manual describes the functionality and usage of the Sirius Leon3 TCM - with TCM Core Application.

The Sirius OBC or Sirius TCM differ in certain areas such as the SoC, interfaces etc. see the electrical and mechanical ICD documents, [RD1] and [RD2], for details on the interfaces.

1.1. Applicable releases

This version of the manual is applicable to the following software releases:

Sirius Leon3 TCM - with TCM Core Application: v1.22.0

1.2. Intended users

This manual is written for engineers that will use the Sirius Leon3 TCM installed with the Sirius Leon3 TCM - with TCM Core Application. It describes the capabilities and the interfaces of the Sirius Leon3 TCM - with TCM Core Application. The electrical and mechanical interface is described in more detail in the electrical and mechanical ICD document [RD2] .

1.3. Getting support

If you encounter any problem using the Sirius products or another AAC Clyde Space product, please use the following address to get help:

Email: support@aac-clydespace.com

1.4. Reference documents

[RD1] “Sirius OBC electrical and mechanical ICD,” 205-088. AAC Clyde Space.

[RD2] “Sirius TCM electrical and mechanical ICD,” 205-089. AAC Clyde Space.

[RD3] “GRLIB IP Core User’s Manual,” GRIP, May 2019, Version 2019.2.

[RD4] “Electrostatics - Part 5-1: Protection of electronic devices from electrostatic phenomena - General requirements,” SS-EN 61340-5-1.

[RD5] “ GRMON3 User’s Manual,” GRMON3-UM, June 2019, Version 3.1.0.

[RD6] “Sirius TCM User Manual,” 206-307. AAC Clyde Space.

[RD7] “Sirius SoC Configuration Document,” 206-222. AAC Clyde Space.

[RD8] “TM Space Data Link Protocol,” CCSDS 132.0-B-2.

[RD9] “Telecommand, part 1, Channel service,” CCSDS 201.0-B-3.

[RD10] “TC Space Data Link Protocol,” CCSDS 232.0-B-2.

[RD11] “Space engineering - Telemetry and telecommand packet utilization,” ECSS-E-ST-70-41C.

[RD12] “SpaceWire - Remote memory access protocol,” ECSS-E-ST-50-52C.

[RD13] “SpaceWire - Links, nodes, routers and networks,” ECSS-E-ST-50-12C.

[RD14] “The SPARC Architecture Manual,” sparcv8, SAV080SI9308, Version 8.

[RD15] “TM Synchronization and Channel Coding,” CCSDS 131.0-B-4.

[RD16] “Space Packet Protocol,” CCSDS 133.0-B-2.

[RD17] “Time Code formats,” CCSDS 301.0-B-4.

2. System overview

2.1. Description

The Sirius OBC and Sirius TCM products are depicted in Figure 3.1 and Figure 3.2.

In addition to the external interfaces, the Sirius products also include both a debugger interface for downloading and debugging software applications and a JTAG interface for programming the FPGA during manufacturing.

The FPGA firmware implements a SoC built around a LEON3FT processor [RD3] running at a system frequency of 50 MHz and with the following key peripherals:

- Error manager - error handling, tracking and log of e.g. memory error detection.
- SDRAM controller - 64 MB data + 64 MB EDAC running @100MHz.
- Spacecraft Elapsed Timer (SCET) - including a PPS (Pulse Per Second) time synchronization interface for accurate time measurement with a resolution of 15 µs.
- SpaceWire - including a three-port SpaceWire router, for communication with external peripheral units.
- UARTs - RS422 and RS485 line drivers on the board with line driver mode set by software.
- GPIOs
- Watchdog - a fail-safe mechanism to prevent a system lockup
- System flash - 2 GB of EDAC-protected flash for storing boot images in multiple copies.
- Pulse command inputs - for reset to a specific software image
- NVRAM - for storage of metadata and other data that requires a large number of writes that shall survive loss of power.

For the Sirius TCM the following additional peripherals are included in the SoC:

- CCSDS - communications IP with RS422/LVDS interfaces for radio communication and an UMBI interface for communication with EGSE.
- Mass memory - 32GB of EDAC-protected NAND flash based, for storage of mission critical data.

For the Sirius OBC:

- An analog interface is included for external analog measurements.

The input power supply provided to the Sirius products shall be between +4.5 and +16 VDC. Power consumption is highly dependent on activities and peripheral loads and ranges from 1.2 W to 2 W.

2.2. OBC/TCM peripherals

Figure 2.1 shows an overview of the System-on-Chip (SoC) together with the peripheral circuitry of the Sirius OBC and Sirius TCM products. The color coding in the figure shows what parts are included for which products. The CPU is a LEON3FT.

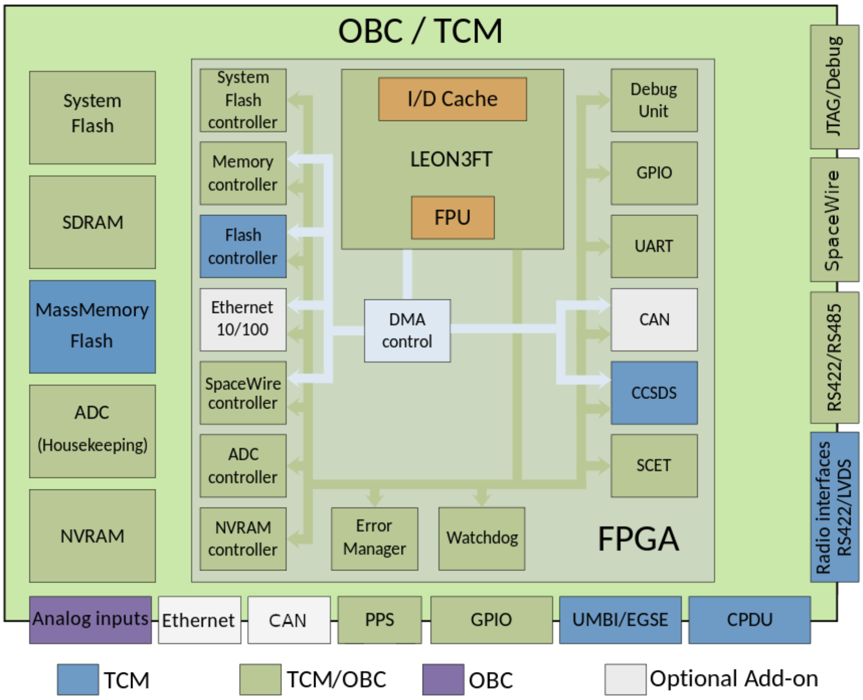

Figure 2.1 - The Sirius OBC / Sirius TCM SoC Overview

2.3. Fault tolerant design

The Sirius OBC and Sirius TCM are both fault tolerant by design to withstand the environmental loads that the modules are subjected to when used in space applications. The following error mitigation techniques are used.

- Continuous EDAC scrubbing of SDRAM data with at least 1 bit error correction and 2 bit error detection for each 16-bit word. Non-correctable errors cause a processor interrupt to allow the software to handle the error differently depending on in which section of the memory it appeared, unless the error appear in the execution path (see below).
- EDAC checking of instructions before execution and on data used in the

instruction (at least 1 bit error correction and 2 bit error detection as described in the previous point). Non-correctable errors cause automatic reboot.

- Parity checking of Instruction and Data caches when they are enabled. Errors cause a processor interrupt with a cache reload as the default error handling.
- Parity checking of peripheral FIFOs. Errors cause processor interrupt.
- EDAC checking on system flash with double bit error correction and extended bit error detection in combination with interleaving that corrects bursts with up to 16 bits in error.
- Triple Modular Redundancy (TMR) on all FPGA flip-flops
- All software stored in boot flash is, in addition to the EDAC protection of the flash data, encoded with a header for checksum and length. Each boot image is stored in three copies to allow for an automatic fallback option if the ECC and/or length check fails on one copy.
- Watchdog, tripping leads to automatic reboot of the device.
- Advanced Error Manager keeping the detected failures during reset/reboot for later analysis.

2.4. Usage and concept

This section describes the concept and normal intended use for the Sirius OBC and Sirius TCM in the default product configuration.

2.4.1. Combined setup

The OBC and TCM are intended to be used together to form the data processing and data handling portion of an on-board satellite system.

The OBC and TCM connect via spacewire, which provides the main interface for both commanding and data transfers.

Figure 2.2 shows an overview of an example setup with the OBC, TCM, a radio, and a pair of payloads in a suggested normal setup.

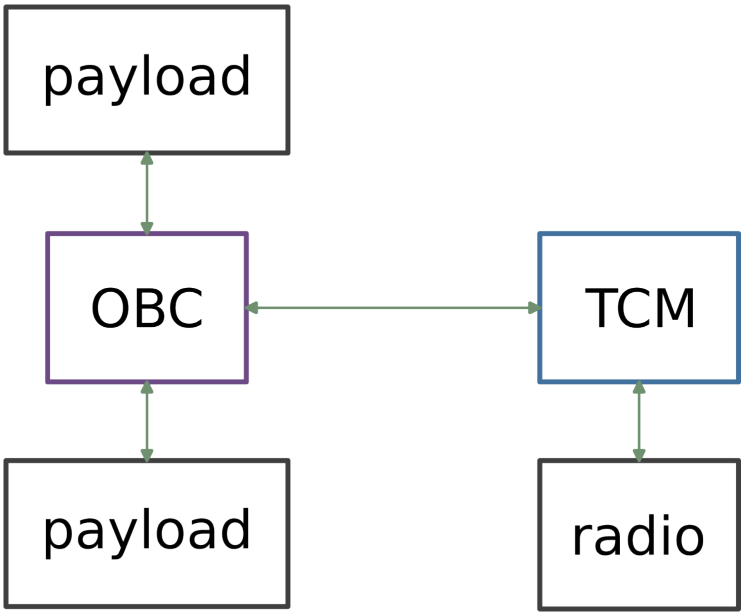

Figure 2.2 - Conceptual design of an on-board data handling system

2.4.2. OBC concept

The OBC provides a platform for hosting mission-specific flight software developed by the user, it is intended to handle the overall command and control handling of the on-board satellite system.

The OBC is also intended to handle the main data processing, and several interfaces for connecting to payloads and other on-board modules are provided.

The OBC Board Support Package (BSP) contains the RTEMS operation system along with drivers for use when developing its software.

2.4.3. TCM concept

2.4.3.1. Use with pre-programmed flight software

The TCM contains pre-programmed flight software (TCM Core Application). This software is conceptually passive and relies on external command and control, intended to be provided by the OBC.

The TCM is intended to be connected to a radio and provide a TM/TC communications interface for use by the OBC. The TCM also provides a data storage interface which can be used by the OBC for both custom data and pre-prepared telemetry for later downlinking.

The TCM is configured by the user to fit the specific mission parameters (see Section 6.1).

2.4.3.2. Use without pre-programmed flight software

The TCM may be used without the pre-programmed flight software and a TCM BSP is provided to allow the user to develop mission-specific software on the TCM, in a similar procedure as is normal for the OBC.

Using the TCM without the pre-programmed flight software is normally not the main intended use.

3. Setup and operation

3.1. User prerequisites

The following hardware and software are needed for the setup and operation of the Sirius products.

PC computer

- 1 GB free space for installation (minimum)
- Debian 10 or Debian 11 64-bit with super user rights
- USB 2.0

JTAG debugger

- AAC JTAG debugger hardware including harness (104452)

Recommended applications and software packages

- Installed serial communication terminal, e.g. *gtkterm* or *minicom*
- GPG for encryption/decryption of files containing sensitive data
- Host build system, e.g. the debian package build-essential
- AAC toolchain for LEON3 with RTEMS 4.11
- BCC2 bare metal toolchain from Frontgrade Gaisler

For FPGA update capabilities

- Microsemi FlashPro Express v11.9 (www.microsemi.com/products/fpga-soc/design-resources/programming/flashpro#software)
- FlashPro5 programmer

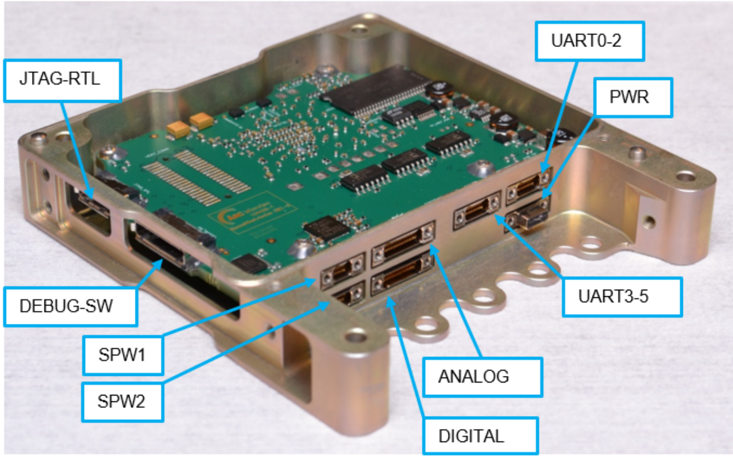

Figure 3.1 - Sirius OBC with connector naming

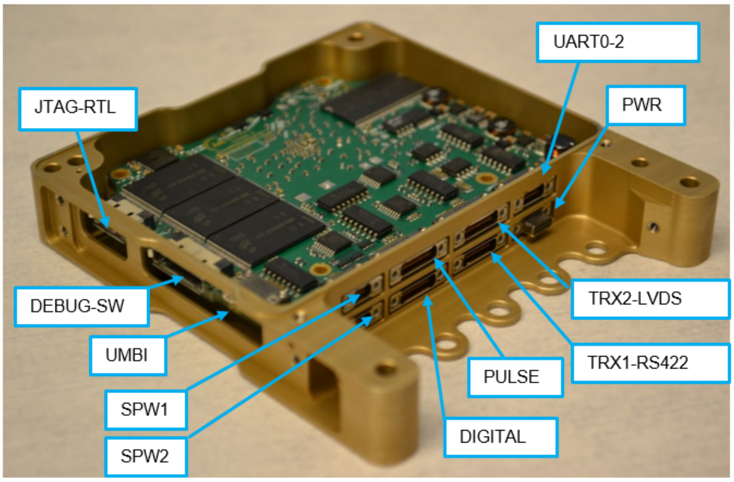

Figure 3.2 - Sirius TCM with connector naming

3.2. Connecting cables to the Sirius products

- All products and ingoing material shall be handled with care to prevent damage of any kind.
- ESD protection and other protective measures shall be considered. Handling should be performed according to applicable ESD requirement standards such as [RD4] or equivalent.
- Ensure that all mating connectors have the same zero reference (ground) before connecting.
- Connect the nano-D connector to the PWR connector with 4.5 - 16 V DC. The units will nominally draw about 260-300 mA @5V DC.
- The AAC debugger is mainly used for development of custom software for the Sirius OBC or Sirius TCM and has both a debug UART for monitoring and a JTAG

interface for debug capabilities. It is also used for programming an image to the system flash memory. For further information refer to Section 3.6. When it is to be used, connect the 104452 AAC Debugger to the DEBUG-SW connector. Connect the adapter USB-connector to the host PC.

- For FPGA updating only: Connect a FlashPro5 programmer to the JTAG-RTL connector using the 104470 FPGA programming cable assembly. For further information how to update the SoC refer to Chapter 25.
- For connecting the SpaceWire interface, connect the nano-D connector to connector SPW1 or SPW2.

For more detailed information about the connectors, see [RD2] .

3.3. Installation of toolchain

This chapter describes instructions for installing the AAC toolchains.

3.3.1. Supported Operating Systems

- Debian 10 64-bit
- Debian 11 64-bit

When installing Debian, we recommend using the “netinst” (network install) method. Images for installing are available via www.debian.org/releases/bullseye/debian-installer/

To install the toolchain below, a Debian package server mirror must be added, either in the installation procedure (also required during network install) or after installation.

On Debian 11 some packages required to build the BSP have been noted to not be installed by default. These need to be installed in order to configure and build:

```
sudo apt-get update
sudo apt-get install m4 autoconf
```

3.3.2. Installation Steps

3.3.2.1. Store the key for the AAC package archive

In order to obtain the key to verify the packages, run the following commands

```
wget -O key.asc http://repo.aacmicrotec.com/archive/key.asc
gpg --dearmor --yes --output key.gpg key.asc
sudo mkdir -p /etc/apt/keyrings
```

```
sudo cp key.gpg /etc/apt/keyrings/aac-repo.gpg
```

3.3.2.1.1. Old AAC package archive key in global trusted apt keyring

Previous toolchain instructions described installing the AAC package archive key in the global trusted apt keyring, this is no longer recommended practise in Debian. If the AAC package key has previously been added to the global trusted apt keyring, it can be removed via

```
sudo apt-key del "39D5 F87E 457C 8EA5 0DEE B148 FA81 C4F9 0257 7CF0"
```

where the argument string is the fingerprint of the AAC package archive key.

NOTE If the key is deleted from the global trusted apt keyring it must instead be available in an individual keyring and the package archive source files must be rewritten to use it via the `signed-by` option, as described in Section 3.3.2.1 and Section 3.3.2.2.

3.3.2.2. Add the package archive as a source

In order to add the AAC package archive as a source; create a new repository source file and open it, for example via

```
sudoedit /etc/apt/sources.list.d/aac-repo.list
```

add the following lines to the file

```
deb [signed-by=/etc/apt/keyrings/aac-repo.gpg]
http://repo.aacmicrotec.com/archive/ aac/
deb-src [signed-by=/etc/apt/keyrings/aac-repo.gpg]
http://repo.aacmicrotec.com/archive/ aac/
```

then save and close the file.

3.3.2.3. Install the packages

In order to install the packages, run the following commands

```
sudo apt update
```

```
sudo apt install aac-sparc-toolchain
```

3.3.2.4. Setup PATH

The toolchain PATH setup file can be sourced manually to use the toolchain in the current instance of the shell via

```
. /opt/aac-sparc/aac-path.sh
```

In order to make the toolchain available automatically in all instances of the bash shell, which is recommended for convenience; open the bash run commands file for the current user in an editor, for example via

```
editor ~/.bashrc
```

add the following lines to the end of the file

```
# AAC LEON3 toolchain PATH setup
if [ -f /opt/aac-sparc/aac-path.sh ]; then
. /opt/aac-sparc/aac-path.sh >/dev/null
fi
```

then save and close the file.

New instances of the bash shell will now automatically have access to the toolchain.

3.3.3. AAC toolchain is RTEMS-only

NOTE	The AAC toolchain for LEON3 only supports RTEMS application development, for bare metal software the BCC2 toolchain from Cobham Gaisler is recommended (available at www.gaisler.com/index.php/downloads/compilers).

3.4. Installing the Board Support Package (BSP)

Board support packages can be found at repo.aacmicrotec.com/bsp. Download the file `aac-<cpu>-<board>-bsp-<version>.tar.bz2`, where `<cpu>` is the processor type (currently only leon3); `<board>` is obc-s or tcm-s; and <version> is the wanted version number of that BSP; and extract it to a directory of your choice.

The extracted directory `aac-<cpu>-<board>-bsp` now contains the drivers for both bare-metal applications and RTEMS. See the included README and Section 4.1 for build instructions.

3.5. Deploying a Sirius application

3.5.1. Establish a debugger connection to the Sirius products

The Sirius products are shipped with debuggers that connect to a PC via USB and have two interfaces towards the board:

- One JTAG interface to the SoC debug unit.
- One debug UART to exchange information with the running software.

3.5.2. JTAG connection

To communicate with the debug unit in LEON3 based SoC's the program GRMON from Frontgrade Gaisler is used. This is not included in the AAC toolchain package as it requires a special license and thus needs to be installed separately.

GRMON3 Pro version 3.0.10 or higher is required. This can be downloaded from Gaisler at www.gaisler.com/index.php/downloads/debug-tools. For further instructions please refer to the GRMON3 manual, which is available at www.gaisler.com/doc/grmon3.pdf.

GRMON3 can be used as a standalone debug monitor to load and run applications, set breakpoints and read/write system registers and memory, and it is scriptable using TCL. It can also run as a server for the GNU Debugger if that interface is preferred.

3.5.3. Setup a serial terminal to the device debug UART

The device debug UART may be used as a debug interface for printf output etc.

A serial communication terminal such as minicom or gtkterm is necessary to communicate with the Sirius product, using these settings:

Baud rate: 115200
Data bits: 8
Stop bits: 1
Parity: None
Hardware flow control: Off

On a clean system with no other USB-to-serial devices connected, the serial port will appear as `/dev/ttyUSB1`. However, the numbering may change when other USB devices are connected, and the user must make sure to use the correct device number to communicate to the board's debug UART.

On Debian, a more foolproof way of identifying the terminal to use is the by-id mechanism using the serial number of the debugger obtained in Section 3.5.4. When the AAC debugger is connected the system automatically creates named symbolic links to the device files under `/dev/serial/by-id`. The interface to use is `usb-AAC_Microtec_JTAG_Debugger_FTZ7QCMF-if01-port0`, where `FTZ7QCMF` is the serial number in this case. The debug UART is on `if01`, while `if00` is used for the JTAG interface (any serial device created for `if00` should disappear when a debug monitor is started).

3.5.4. Using multiple debuggers on the same PC

In order to use multiple debuggers connected to the same PC, each instance of run_aac_debugger.sh must be configured to connect to the specific debugger serial number and to use unique ports.

To determine the serial number for a specific device, run the following command before connecting the debugger:

```
sudo tail -f /var/log/kern.log
```

This initially prints the last 10 lines of the kernel log file, which can be ignored. When plugging in the debugger USB cable into the PC, this should produce new output similar to:

```
[363061.959120] usb 1-1.3.3.3: new full-speed USB device number 15 using
ehci_hcd
[363062.058152] usb 1-1.3.3.3: New USB device found, idVendor=0403,
idProduct=6010
[363062.058176] usb 1-1.3.3.3: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[363062.058194] usb 1-1.3.3.3: Product: JTAG Debugger
[363062.058207] usb 1-1.3.3.3: Manufacturer: AAC Microtec
[363062.058220] usb 1-1.3.3.3: SerialNumber: FTZ7QCMF
```

where `FTZ7QCMF` is the serial number for the debugger.

For GRMON3 the port to use for the GDB server needs to be unique. The default is `50001`.

For example, two debuggers with serial numbers `FTZ7QCMF` and `FTZ7IB10` can be setup via

```
run_aac_debugger.sh -s FTZ7QCMF -g 50001
run_aac_debugger.sh -s FTZ7IB10 -g 50002
```

Two instances of GDB can then be opened and connected to the different debuggers through the chosen ports.

3.5.5. Alternative USB library for GRMON

Some versions of GRMON have had issues communicating with the USB connected debugger hardware, particularly when dumping memory. This shows as error messages at the GRMON3 prompt noting “usb bulk write failed”, “usb bulk read failed” or similar. These come from the open source libftdi and libusb libraries included with GRMON. In case of such issues a workaround is to use the proprietary D2XX library from FTDI instead.

To install the library, download the D2XX driver package for linux from FTDI: ftdichip.com/drivers/d2xx-drivers/

The package contains a lot of examples and things needed to build applications that communicate with FTDI USB devices, but the only thing needed here is the file `libftd2xx.so.<version>`. This can be extracted and copied to a suitable directory on the computer running GRMON, for example `/usr/local/lib`. Then a symbolic link should be created in the same directory so that there appears to be a file without the version:

```
sudo ln -s libftd2xx.so.1.2.27 libftd2xx.so
```

GRMON can then be started with this library instead of the included open source libftdi:

```
LD_LIBRARY_PATH=/usr/local/lib /opt/grmon-pro-3.3.2/linux/bin64/grmon -v -abaud
115200 -ftdi d2xx -ftdigpio 0x08100000 -gdb 50001 -stack 0x04000000
```

To handle multiple debugger units connected to the same computer when using the D2XX library, the user can select the unit to use by serial number by adding the command line switch `-jtagserial FTZ7QCMF`, or alternatively listing the available debuggers using

```
LD_LIBRARY_PATH=/usr/local/lib /opt/grmon-pro-3.3.2/linux/bin64/grmon -ftdi d2xx
-jtaglist
```

and selecting the wanted unit using `-jtagcable <num>`.

3.5.6. Loading an application on LEON3

An application can either be loaded only to the board SDRAM, which is easier and typically used during the development stages, or to the system flash (see Section 3.6). In this manual it is done using GDB, but it could also be done using only GRMON (see sections 3.4.2 and 3.4.3 in the GRMON3 User’s Manual [RD5]). From GDB the user can also pass commands to GRMON by prefixing them with the GDB command `monitor`.

1. Start GDB with the following command from a shell to debug RTEMS executables:

   ```
   sparc-aac-rtems4.11-gdb
   ```

2. When GDB has opened successfully, connect to the hardware through the GRMON server using the GDB command target.

   ```
   target extended-remote localhost:50001
   ```

3. Specify the executable file for GDB to work with. Make sure the file is in ELF format.

   ```
   file <path/to/executable>
   ```

4. Transfer into the target RAM

   ```
   load
   ```

5. Start the application.

   ```
   run
   ```

3.5.7. Debugging software

Halting and reloading software via GRMON or GDB may leave peripheral units in an unknown state, and thus give unexpected behavior, especially if there is communication running on SpaceWire and UARTs. When working with software through the debugger it is good to start from a system reset, preferably with a very simple software in flash.

The Watchdog timer is enabled by default and can only be disabled when the debugger is connected. To avoid unexpected resets while debugging it is good to have a prepared command in GRMON or GDB to disable the Watchdog as soon as possible after software is halted. If the watchdog is not being kicked for a set time, it will trigger a reset of the board. Under normal operation, the TCM core application automatically kicks the watchdog.

In GRMON: `wmem 0xCB000000 0x0`

In GDB: `set *(unsigned int) 0xCB000000 = 0`

In GRMON: `wmem 0xC0000000 0xFFFFFFFF`

In GDB: `set *(unsigned int) 0xC0000000 = 0xFFFFFFFF`

If GRMON gives the error “CPU not in debug mode” when executing a command, that usually means that the board has reset, and the Debug Support Unit in the SoC is not in control of the CPU. To take back control the attach command is used.

In GRMON: `attach`

In GDB: `monitor attach`

This should be immediately followed by disabling the Watchdog to avoid losing the connection again.

3.6. Programming an application (boot image) to system flash

NOTE	The steps to build the nandflash_program can be ignored when using the TCM Core Application, as the TCM units are generally delivered with the application already loaded to the system flash.

To have an application start automatically when the board is powered the application image must be programmed to the system flash. This is done by taking the boot image binary and building it into the NAND flash programming application. The NAND flash programming application is then uploaded to the target and started using GDB, as described in the previous section. The maximum recommended size for the boot image is 16 MB. The `nandflash_program` application can be found in the BSP.

The below instructions assume that the toolchain is in the PATH, see Section 3.3 for how to accomplish this.

1. Compile the boot image binary according to the rules for that program.
2. Ensure that this image is in a binary-only format and not ELF. This can be accomplished with the help of the GCC objcopy tool included in the toolchain:

   ```
   sparc-aac-rtems4.11-objcopy -O binary boot_image.elf boot_image.bin
   ```

3. See Section 3.4 for installing the BSP and enter

   ```
   cd path/to/bsp/aac-<cpu>-<board>-bsp/src/nandflash_program/src
   ```

4. Now, compile the nandflash-program application, bundling it together with the boot image binary.

   ```
   make nandflash-program.elf PROGRAMMINGFILE=/path/to/boot_image.bin
   ```

5. Load the nandflash-program.elf onto the target RAM with the help of GDB and execute it, see Section 3.5.6. The programmer application will output progress

information on the debug UART.

3.7. Re-initialising the NVRAM

In some situations, it may be desirable to clear and re-initialise the NVRAM from scratch, for example if a test application has written data to the NVRAM which does not match the expected format for the system flash bad block table.

Clearing the NVRAM will cause loss of the following data, which should be read out, backed up, and written back after re-initialising if critical:

- Bad block markings for discovered bad blocks in the system flash (Both OBC and TCM), may degrade reliability if cleared.
- Bad block markings for discovered bad blocks in the mass memory (TCM with the TCM core application software), may degrade reliability if cleared.
- Ongoing operation markers for the mass memory handler (TCM with TCM core application), may cause partial loss of stored partition data if cleared.
- Internal write pointers for the mass memory handler (TCM with TCM core application), may case loss of start and end location in a completely full partition if cleared.

The following steps are required in order to clear and re-initialise the NVRAM:

1. Compile and run the nvram_clear application using the debugger. This application is located in the src/example/ directory in the OBC or TCM BSP; the steps for compiling it are described in Section 4.1. This will clear the NVRAM.
2. Program a boot image to the system flash as described in Section 3.6. This will initialize the system flash bad block table in the NVRAM.

The following additional steps are needed to re-initialize the TCM with the TCM core application:

3. Compile and run the board_initialiser application using the debugger. This application is located in the `src/nv_config/src/board_initialiser/` directory in the TCM BSP; it is compiled as an RTEMS application in a similar fashion as the example applications described in Section 4.1. This will initialize the mass memory bad block table in the NVRAM.

4. Compile and run the nv_config utility as described in Section 6.2 This will initialize the NVRAM configuration parameters.

4. Deploying the TCM core application

The TCM units are generally delivered with the TCM core application already loaded to the system flash as boot image, see Section 3.6. Hence, the TCM core application will automatically boot on power-on. However, the configuration (see Chapter 6) needs to be written to the NVRAM using the nv_config tool of the BSP, as described in Section 6.2.

To build the nv_config tool, the BSP needs to be compiled first. How to do this is explained in the following section.

Note that the RTEMS drivers are also contained within the BSP, but are not documented in this manual, as they are not directly needed when using the TCM core application. They are documented in [RD6] instead.

4.1. RTEMS step-by-step compilation

4.1.1. Compiling the BSP and compiling an example

The BSP is supplied with an example of how to write an application for RTEMS and engage all the available drivers.

Please note that the toolchain described in Section 3.3 needs to be installed and the BSP unpacked as described in Section 3.4.

The following instructions detail how to build the RTEMS environment and a test application

1. Enter the BSP src directory
   ```
   cd path/to/bsp/aac-<cpu>-<board>-bsp/src/
   ```
2. Run make to build the RTEMS target
   make
3. Once the build is complete, the build target directory is librtems
4. Set the `RTEMS_MAKEFILE_PATH` environment variable to point to the librtems directory containing Makefile.inc:
   ```
   export RTEMS_MAKEFILE_PATH=path/to/librtems/sparc-aac-rtems4.11/leon3/
   ```
5. Enter the example directory and build the test application by issuing
   ```
   cd example
   make
   ```

Load the resulting application using the debugger according to the instructions in Section 3.5.

4.1.2. Compiling the BSP with debug output removed

During development, debug output from the RTEMS drivers can be very useful for detecting errors. During flight, debug output is unlikely to be useful (it is expected that the debug UART will be disconnected) and may decrease performance in case of large amounts of warnings/errors.

The RTEMS BSP can be compiled without debug output by replacing the make command in step 2. above with instead:

```
make clean
make BSP_AAC_DISABLE_DEBUG_OUTPUT=y
```

(The make clean command is only required if the BSP has previously been compiled with a different configuration.)

4.2. Software disclaimer of warranty

This source code is provided "as is" and without warranties as to performance or merchantability. The author and/or distributors of this source code may have made statements about this source code. Any such statements do not constitute warranties and shall not be relied on by the user in deciding whether to use this source code.

5. TCM core application overview

The Sirius TCM core application is a software product which provides an enhanced interface for accessing the functionality of the Sirius Leon3 TCM.

The Sirius TCM core application is intended to be used as a node in an on-board SpaceWire network with a separate controlling node. The Sirius Leon3 OBC running a mission-tailored customer software is commonly used as this controlling node.

An interface based on Remote Memory Access Protocol (RMAP) packets is used to control the Sirius TCM core application via SpaceWire.

The Sirius TCM core application uses the Sirius Leon3 TCM CCSDS functionality to decode and process telecommand packets uplinked from the ground segment. These telecommand packets may be forwarded over SpaceWire as part of RMAP write commands to be processed by other on-board nodes; they may also be sent directly to the Sirius TCM core application using a limited set of ECSS PUS packet services, these PUS services allow direct commanding of the application from the ground segment.

Access to storage based on mass memory flash is provided via RMAP packets, this can be used both for general data storage and for storage of pre-packaged telemetry for later downlinking.

The Sirius TCM core application allows downlinking live telemetry packets to the ground segment based on RMAP packets received directly from other nodes on the SpaceWire network. It also allows downlinking pre-packaged telemetry directly from the mass memory flash storage.

The Sirius Leon3 TCM UARTs are also made available by the Sirius TCM core application for access over SpaceWire using RMAP packets.

The Sirius TCM is highly configurable and can be tailored to different mission requirements. Pre-flight configuration is provided via a (non-volatile) magnetoresistive RAM which stores customer-specific mission parameters which are applied on start-up of the Sirius TCM core application, see Chapter 6. In addition, a lot of these parameters are also available to be dynamically set in-flight during the mission (but will return to the pre-flight configuration on reset).

Figure 5.1 shows an overview of the different components of the Sirius Leon3 TCM on which the Sirius TCM core application is hosted.

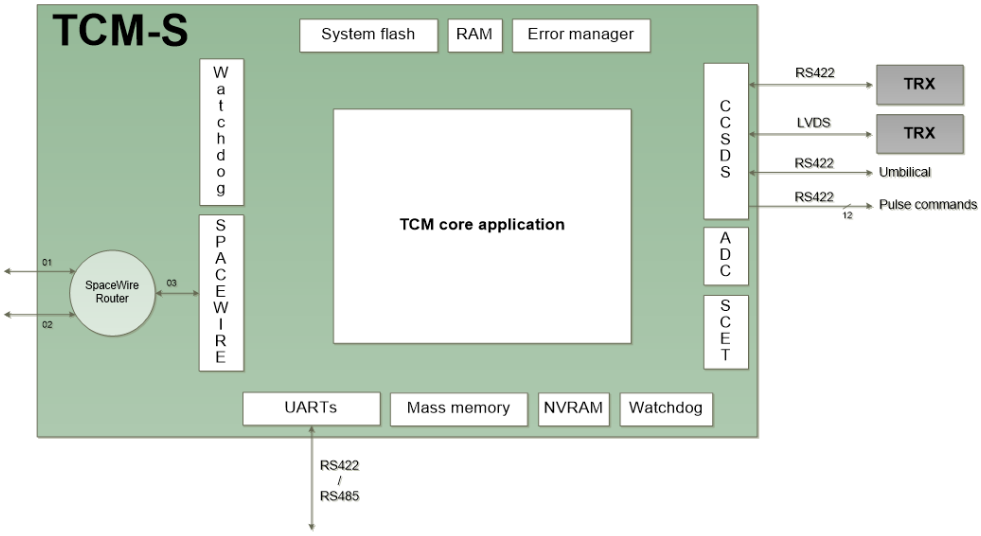

Figure 5.1 - Sirius TCM functionality layout with the external ports depicted

6. Configuration

The TCM can be configured for specific missions by parameters in NVRAM described in Section 6.1. The parameters from NVRAM are read during initialization of the TCM application. Section 6.2 describes how to write an example configuration to the NVRAM of an actual unit. If reading from NVRAM fails during initialization, a set of fallback parameters are used instead. The fallback parameters are described in Section 6.3.

6.1. Configuration parameters

The description and format of the different configuration parameters are detailed in the following tables.

6.1.1. Mass Memory partition configuration

Partition configuration of mass memory is specified in Table 6.1 below.

Table 6.1 - PARTITION_CFG

Data	Type	Description
0	UINT32	Starting block number of the partition.
4	UINT32	Ending block number of the partition (exclusive).
8	UINT8	Partition mode. 0 - Direct 1 - Continuous 2 - Circular 3 - Auto-padded Continuous 4 - Auto-padded Circular
9	UINT8	Specifies type of data stored on the partition. 0 - Packets 1 - Raw Data (not supported for download) 2 - TC Storage
10	UINT8	Specifies which virtual channel to be used for downloading of the data in the partition. See [RD7] for VC allocation.

Data	Type	Description
11	UINT8	Segment size for the partition. 1 - 16 kbyte 2 - 32 kbyte 3 - N/A 4 - 64 kbyte
12	UINT32	The data source identifier for the partition. Can be used to set a custom identifier of a data producer to a partition. Setting of this value is not required to successfully configure a partition.

6.1.2. UART routing to SpaceWire

Data from different sources can be routed to the SpW-network. Routing info is set by format specified in Table 6.2

Table 6.2 - UART_ROUTING

Data	Type	Description
uart	UINT8	Source of message: Which UART device n should route the data. See [RD7] for the available UARTs in the applicable SoC. 0 - UART0 1 - UART1 2 - UART2 3 - UART3 4 - UART4 5 - reserved 6 - UART6 (PSU Ctrl) 7 - UART7 (Safe Bus)
Path	UINT16	The index of the SpW-path for the routing. See Table 6.6.
Backup SpW path	UINT16	The index of the backup SpW-path for UART config. See Table 6.6.
Backup SpW reply path	UINT16	The index of the SpW write reply path for UART config. See Table 6.7.
Reserved	UINT8	Reserved padding.

6.1.3. UART configuration

Configuration of UART-devices is done via an array of fixed length representing all potential UART devices. Refer to [RD7] for the available UARTs in the applicable SoC. Additionally, each UART should be enabled/disabled via a configuration flag. If an unavailable UART is enabled and configured, opening it will fail and the TCM will use the fallback UART configuration (Table 6.20) instead. The configuration paremeters of one individual device are shown in Table 6.3 below.

Table 6.3 - UART_CONFIG

Data	Type	Description
reserved	UINT8	reserved
Bitrate	UINT8	UART bitrate: 12 = 223200 baud 11 = 375000 baud 10 = 347200 baud 9 = 153600 baud 8 = 115200 baud (default) 7 = 75600 baud 6 = 57600 baud 5 = 38400 baud 4 = 19200 baud 3 = 9600 baud 2 = 4800 baud 1 = 2400 baud 0 = 1200 baud
Mode	UINT8	UART mode: 0 = RS422 mode 1 = RS485 mode 2 = Loopback
UART configuration flags	UINT8	Configuration of UART via various flags, see Table 6.4 for details

Table 6.4 describes the detailed bit layout of the UART flags field. The UART device has to be enabled by the enable flag.

Table 6.4 - UART configuration flags

Data	Bit	Description
Parity	0-1	0 - no parity 1 - odd parity 2 - even parity

Data	Bit	Description
Reserved	2-5	reserved
PUS access blocked	6	0 - no (allowed) 1 - yes (blocked, PUS services cannot access UART)
UART enabled	7	0 - disabled - this uart will not be opened 1 - enabled

6.1.4. SpaceWire logical address of TCM core application

Table 6.5 describes the format of the SpW logical address configuration.

Table 6.5 - SPW lgl addr configuration

Data	Type	Description
SpW logical address	UINT32	TCM CA logical SpW address for receiving data. Only allowed values 0x20-0xFE, except for 0x43 which is reserved for future use.

6.1.5. SpaceWire paths

Paths on SpW-network are specified by table Table 6.6 below. This table can fit 20 different SpW paths, each path can fit 8 bytes.

NOTE All SpW paths must contain a terminating null character.

Table 6.6 - NVRAM SpW path storage

Data	Type	Description
Path 0	Array of UINT8	A path on SpW network including the logic address of the receiving node.
Path 1	Array of UINT8	A path on SpW network including the logic address of the receiving node.
Path N	Array of UINT8	A path on SpW network including the logic address of the receiving node.

6.1.6. SpaceWire backup routing

Backup reply paths on the SpW-network are specified by Table 6.7 below. When the TCM SW is requesting a write-reply from an external SpW node, the TCM SW must provide the path for the write reply. Since it is not possible to determine the reply path from the corresponding backup path, the user must also provide one write-reply path for every defined SpW path. This table can fit 20 different paths, each path can fit 8 bytes.

NOTE All SpW paths must contain a terminating null character.

Table 6.7 - NVRAM SpW Backup Reply Paths

Data	Type	Description
Reply path 0	Array of UINT8	A write-reply path from an external SpW node to the TCM SW.
Reply path 1	Array of UINT8	A write-reply path from an external SpW node to the TCM SW.
Reply path N	Array of UINT8	A write-reply path from an external SpW node to the TCM SW.

Enabling and timeout of the backup SpW paths are specified by Table 6.8 below.

NOTE Since the granularity of the system is 10 ms, values not divisible by 10 ms will be truncated to the nearest multiple if 10 ms. Setting a timeout less than 10 ms will result in a timeout of 0 ms.

Table 6.8 - NVRAM Backup SpW Configuration Storage

Data	Type	Description
SpW backup routing config	UINT32	Bit 0:15 - Sets the timeout in milliseconds. Bit 16 - Enable SpW backup routing. 1 = ENABLE 0 = DISABLE Bit 17:31 - Reserved

6.1.7. SpaceWire RIRP

RIRP can be enabled/disabled as specified in Table 6.9 below

Table 6.9 - RIRP Config

Data	Type	Description
RIRP Config	UINT32	Enabling/Disabling of RIRP 0 = DISABLE 1 = ENABLE

6.1.8. Telecommand APID routing

Telecommands can be routed to nodes on the SpW by APID as specified in Table 6.10

and Table 6.11 below.

Table 6.10 - NVRAM APID Routing

Byte	Type	Description
0-1	UINT16	APID or lower APID in APID range Bit15 0 = Single APID Routing, 1 = APID range Bit14:13 Routing destination type Bit12:11 Not used Bit10:0 APID
2-3	UINT16	Upper APID in APID range Bit15 0 = Single APID Routing, 1 = APID range Bit14:13 Routing destination type Bit12:11 Not used Bit10:0 APID
4-5	UINT16	The index of the primary SpW-path of the APID. See Table 6.6.
6-7	UINT16	The index of the backup SpW-path of the APID. See Table 6.7.
8-9	UINT16	The index of the SpW write reply path of the APID. See Table 6.8.
10 - 11	-	Reserved for future use

Table 6.11 - Routing destination type - mapping

Bit 14:13	Bit 14	Bit 13	Description
0	0	0	Routing via SPW
1	0	1	Reserved
2	1	0	TCM APID
3	1	1	Routing to TC queue

NOTE	If the *Routing destination path* is set to *TCM APID*, then bit 15 should be set to 0, *Single APID Routing*. This is because the TCM will only handle the APID that it is assigned to from NVRAM configuration and that is a single APID.

6.1.9. Telecommand configuration

Configuration of the TC path is described in Table 6.12 below:

Table 6.12 - TC_CONFIG

Data	Type	Description
TC Config	UINT32	Configuration of TC path. Bit0: 0 - Disable Derandomizer 1 - Enable Derandomizer

6.1.10. Telecommand APID of TCM Core Application

Configuration of the TC handler APID described in Table 6.13 below:

Table 6.13 - TC_HANDLER_APID

Data	Type	Description
TC Handler APID	UINT32	APID configuration of APID of TC Handler in TCM Core Application

6.1.11. Telecommand VC configuration

The virtual channel for the TCM to receive telecommands on is configured in NVRAM according to the format given in Table 6.14.

Table 6.14 - TC_VC_CONFIG

Data	Type	Description
Telecommand Virtual Channel	UINT32	VC number 0 - 63.

6.1.12. Telemetry configuration

Configuration of the TM path is described in Table 6.15 below.

Table 6.15 - TM_CONFIG

Data	Type	Description
TM Clk divisor	UINT16	The resulting TM bitrate is determined as described in Section 17.7.9.

Data	Type	Description
TM Config	UINT16	Configuration of TM path. Bit5: 0 - Disable Conv. Encoder 1 - Enable Conv. Encoder Bit4: 0 - Disable Randomizer 1 - Enable Randomizer Bit3: 0 - Disable Idle Frames 1 - Enable Idle Frames Bit2: 0 - Disable MCFC 1 - Enable MCFC Bit1: 0 - Disable FECF 1 - Enable FECF Bit0: 0 - Disable CLCW 1 - Enable CLCW

6.1.13. GPIO configuration

Base configuration of the GPIOs that can be controlled through the TCM RMAP interface is described in Table 6.16.

Table 6.16 - GPIO Configuration

Data	Type	Description
GPIO Configuration	UINT8	Bit 0 - GPIO Value, 0 = Low, 1 = High Bit 1 - GPIO Mode, 0 = Normal (Single Ended), 1 = Differential Bit 2 - GPIO Direction, 0 = Output, 1 = Input Bit 3:7 - Reserved

Refer to [RD7] for the number of available GPIOs in the applicable SoC.

NOTE The TCM SW only uses as many GPIOs as are configured in NVRAM.

> Due to requirements on memory alignment, the amount of configured GPIOs must be a multiple of 4.

6.1.14. Time synchronisation configuration

The time synchronisation NVRAM configuration parameters are described in Table 6.17.

For details regarding time synchronisation parameters, see Section 9.1.

Table 6.17 - Time synchronisation configuration

Data	Type	Description
Flags	UINT32	Bit 31:1 (MSB) - Reserved Bit 0 (LSB) - External synchronisation 0 = disabled 1 = enabled
Consecutive qualified PPS count	UINT32	Number of qualified PPS before synchronisation occurs
PPS qualification threshold	UINT16	Individual PPS qualification threshold window size in steps of ±10.24µs
Pad	UINT8	Reserved
Pad	UINT8	Reserved

6.2. Creating and writing a new configuration

A modified configuration should be created and written to the NVRAM using the nv_config utility from the TCM BSP.

The recommended way to create a new configuration is:

- Create a copy of the example configuration at src/nv_config/src/configs/example.h with a different name located it in the same directory.
- Modify the new file to match the desired configuration. The original example file and the definitions file at src/nv_config/src/nvram_common.h are useful references for the format and available parameters.
- Build the nv_config utility by executing the shell command `make` in the src/nv_config/src/ directory. This will compile the nv_config utility for each configuration file, with each resulting RTEMS executable located at src/nv_config/src/nv_config_<config name>.exe, where <config name> is the name

of the source configuration file, for example src/nv_config/src/nv_config_example.exe.

- Load and run the resulting binary RTEMS application using the debugger unit and GDB. Success is indicated via the output:

```
***************** NVRAM programming finished ****************

***************** System can be power cycled ****************
```

6.3. Fallback NVRAM parameters

If reading from NVRAM fails during initialisation of TCM Core Application, a set of fallback-parameters described in the tables below will be used.

Table 6.18 - Fallback Partition Configuration

Partition #	Start Block	End Block	Partition Mode	Data Type	Virtual Channel	Segment Size	Data Source
0	0	99	0 (Direct)	0 (Packet)	1	32 kbyte	0
1	100	5000	1 (Cont.)	0 (Packet)	1	32 kbyte	0
2	5001	7500	2 (Circ.)	0 (Packet)	1	32 kbyte	0

Table 6.19 - Fallback UART Routing

Uart #	SpW Path Index	SpW Backup Path Index	SpW Backup Reply Path Index	Reserved
0 (UART0)	0	0	0	0
1 (UART1)	0	0	0	0
2 (UART2)	0	0	0	0
3 (UART3)	0	0	0	0
4 (UART4)	0	0	0	0
6 (UART6)	0	0	0	0
7 (UART7)	0	0	0	0

Table 6.20 - Fallback UART Config

UART	Reserved	Bitrate	Mode	Extended configuration
UART0	0	8 (115200 baud)	0 (RS422)	0x80 (no parity, pus access unblocked, **enabled**)
UART1	0	8 (115200 baud)	0 (RS422)	0x80 (no parity, pus access unblocked, **enabled**)
UART2	0	8 (115200 baud)	0 (RS422)	0x80 (no parity, pus access unblocked, **enabled**)
UART3	0	8 (115200 baud)	0 (RS422)	0x80 (no parity, pus access unblocked, **enabled**)
UART4	0	8 (115200 baud)	0 (RS422)	0x80 (no parity, pus access unblocked, **enabled**)
UART5	0	8 (115200 baud)	0 (RS422)	0x00 (no parity, pus access unblocked, **disabled**)
UART6	0	8 (115200 baud)	1 (RS485)	0x80 (no parity, pus access unblocked, **enabled**)
UART7	0	8 (115200 baud)	1 (RS485)	0x80 (no parity, pus access unblocked, **enabled**)

Table 6.21 - Fallback SpW Logical Address

Parameter	Value	Description
SpW logical address	0x00000042	Fallback SpW logical address of the TCM-S SW

Table 6.22 - Fallback SpW Paths

Path #	Path data
0	\{0x01, 0x03, 0xFE, '\0'}
1	\{0x01, 0x01, 0x03, 0xFE, '\0'}
2	\{0x01, 0x02, 0x03, 0xFE, '\0'}
3	\{0x02, 0x03, 0xFE, '\0'}
4	\{0x02, 0x02, 0x03, 0xFE, '\0'}
5	\{0x02, 0x01, 0x03, 0xFE, '\0'}

Table 6.23 - Fallback SpW backup reply paths

Reply Path #	Reply Path data
0	\{0x01, 0x03, 0x42, '\0'}

Reply Path #	Reply Path data
1	\{0x01, 0x01, 0x03, 0x42, '\0'}
2	\{0x02, 0x03, 0x42, '\0'}

Table 6.24 - Fallback SpW backup routing configuration

Parameter	Description
SpW backup routing config	Fallback configuration of SpW backup routing: • SpW backup routing is disabled • The RMAP write-reply timeout is 100 ms

Table 6.25 - Fallback RIRP Config

Parameter	Description
RIRP Config	Fallback configuration of RIRP • RIRP is disabled

Table 6.26 - Fallback APID Routing

APID Routing #	Lower APID	Upper APID	SpW Path Index	SpW backupPath Index	SpW write-reply path Index
1	0xE000	0xE0AE	TC queue	TC queue	TC queue
2	0x40AF	0x40AF	Internal APID for TCM	Internal APID for TCM	Internal APID for TCM
3	0x8151	0x8300	0	3	0
4	0x8301	0x8450	1	4	1
5	0x8451	0x8600	2	5	2

Table 6.27 - Fallback TC Configuration

Parameter	Value	Description
TC Config	0x02	Configuration of TC path. • Derandomizer Disabled

Table 6.28 - Fallback TC Handler APID

Parameter	Value	Description
TC Handler APID	0xAF	Fallback APID configuration of APID of TC Handler in TCM Core Application

Table 6.29 - Fallback TC VC Configuration

Parameter	Value	Description
Telecommand Virtual Channel	0x00000000	Fallback Telecommand Virtual Channel

Table 6.30 - Fallback TM Configuration

Parameter	Value	Description
TM Clk divisor	250	The resulting TM bitrate will be 100 kbit/s (since convolutional encoding is disabled)
TM Config	0x4F	Configuration of TM path: • Conv. Encoder Disabled • Randomizer Disabled • Idle Frames Enabled • MCFC Enabled • FECF Enabled • CLCW Enabled

Table 6.31 - Fallback GPIO configuration

Parameter	Value	Description
GPIO 0 Configuration	0x04	Direction - Input Mode - Normal, single ended Value - 0
GPIO 1 Configuration	0x04	Direction - Input Mode - Normal, single ended Value - 0
GPIO 2 Configuration	0x04	Direction - Input Mode - Normal, single ended Value - 0
GPIO 3 Configuration	0x04	Direction - Input Mode - Normal, single ended Value - 0

Parameter	Value	Description
GPIO 4 Configuration	0x04	Direction - Input Mode - Normal, single ended Value - 0
GPIO 5 Configuration	0x04	Direction - Input Mode - Normal, single ended Value - 0

Table 6.32 - Fallback time synchronisation configuration

Parameter	Value	Description
External synchronisation	Enabled	Synchronisation to external PPS source enabled
Consecutive qualified PPS count	5	Synchronisation occurs after 5 qualified PPS
PPS qualification threshold	65535	Individual PPS qualification threshold window size is ~ ±0.67s

7. Telemetry

Telemetry is simultaneously sent on all the transceiver interfaces, i.e. the RS422 (TRX1), the LVDS (TRX2) and umbilical (UMBI) interfaces. See [RD7] for the VC allocation. The CCSDS IP generates complete TM Transfer Frames from CCSDS space packets. If a packet does not fit in one TM Transfer Frame, the CCSDS module splits the packet into several TM Transfer Frames. If a packet does not fill the whole TM Transfer Frame, an idle packet is added as padding to fill the frame. The following telemetry settings are configurable via the NVRAM configuration (see Section 6.1.12) or by RMAP-commands (see Chapter 17):

- Divisor of TM Clock
- Inclusion of CLCW of TM Transfer Frames
- Inclusion of Frame Error Control Field of TM Transfer Frames
- Updating of Master Channel Frame Counter
- Idle frame generation
- Convolutional encoding
- Pseudo randomization

The TCM supports the format of TM Transfer Frames described in [RD8].

8. Telecommands

8.1. Description

Telecommands can be received on the RS422 (TRX1), the LVDS (TRX2) or the umbilical (UMBI) interface.

The TCM actively searches for Command Link Transmission Units (CLTU), i.e. telecommands, on all three inputs simultaneously (as long as they are enabled). When a telecommand start sequence is detected, the other inputs are ignored during telecommand reception. The search will restart once the entire telecommand is either received or a reception error is detected. In short, the telecommand reception uses the following reception logic, also illustrated in Figure 8.1:

- All incoming signals on the inputs are synchronized to the system clock domain.
- The BCH decoder is configured in error-correcting mode.
- When the CLTU receptor has detected and decoded a start pattern, it sets an enable signal for the active path, indicating that this CLTU receptor is now active. If one or more bit-errors occur in the start pattern, the CLTU will be rejected, which is not compliant to handling of one-bit errors described in [RD9] since one-bit errors shall be corrected when BCH decoder is configured in error-correcting mode.
- The telecommand path activated is set until the reception status changes, i.e. the current telecommand is finished and a new start pattern is detected correctly on a different CLTU path.
- The selected telecommand clock, data and enable signals are now forwarded through the mux to the BCH decoder, rejecting data and clock on inactive data paths.
- When BCH has decoded the tail in the CLTU, all CLTU receptors are set in search mode again, scanning for the start pattern ready to receive a new telecommand. If one or more bit-errors occur in the tail sequence of the CLTU, the CLTU will be rejected.
- The BCH interface does not “see” the data/clock until the start pattern is decoded correctly and the enable signal is set.

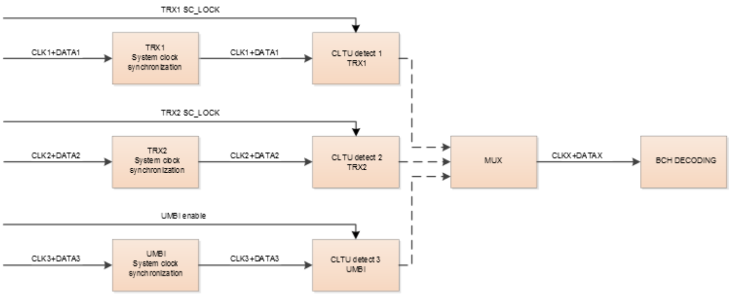

Figure 8.1 - Telecommand Input Multiplexer

Derandomization of TC can be enabled/disabled via the NVRAM configuration (see Section 6.1.9) or via RMAP command (see Chapter 17).

The TCM supports the format of TC Transfer Frames described in [RD10]. TC Transfer Frames sent to the TCM must include a segment header, see 4.1.3.2.2 in [RD10].

8.2. Pulse commands

The CCSDS IP in the TCM has a built-in Command Pulse Distribution Unit (CPDU) execution functionality with the possibility to execute up to twelve CPDUs without interaction from software. A pulse command is decoded directly in hardware, and it sets an output pin according to the pulse command parameters. The CPDU_DURATION_UNIT is defined to 12.5 ms and the output is hence a multiple of this signal length.

The CPDU function can for example be used to reset modules in a spacecraft and choose which software image to boot, an updated version or the safe image. The last executed pulse command can be read from the telecommand status data field.

For details about the format of pulse commands, see Section 14.3.

8.3. COP-1

The CCSDS COP-1 functionality on the spacecraft is implemented mainly in software where the command link control word (CLCW) is generated based on telecommand status. The CLCW is inserted when the OCF_CLCW flag is set in the control register, otherwise user data will be inserted instead. It will insert four bytes, and the CLCW is also included in the CRC calculation for the master frame on both idle and data frames. The NO RF AVAILABLE flag and NO BIT LOCK flag are set from external pins and will overwrite the respective bits in the CLCW word which hence cannot be controlled by software. The flag NO RF AVAILABLE is set by signal Carrier lock in and the flag NO BIT LOCK is set by signal Sub-carrier lock in.

9. Time Management

9.1. Time synchronisation

The TCM has an internal SCET timer that can be synchronised to an external time source. The synchronisation behaviour depends on if active synchronisation to an external PPS source is enabled or disabled.

9.1.1. External synchronisation enabled

When external synchronisation is enabled, the incoming PPS will be monitored for stability. After a given number of consecutive individually qualified PPS have been received, the incoming PPS source will be considered "fully qualified". Once "fully qualified" the SCET subseconds will be synchronised to the incoming PPS.

After synchronisation of subseconds has occurred and if the incoming PPS remains qualified, synchronisation of SCET seconds can occur via the SCETTime write command (see Section 17.7.20). A SCETTime write command will set the target seconds value to be written after the next qualified PPS arrives, so should normally use the current reference seconds count plus one.

9.1.1.1. Individual PPS qualification threshold

The individual PPS qualification criteria is determined using a threshold parameter which defines a qualification window size. The qualification window midpoint is at 1 second after the previous received PPS (or previous window midpoint) and its size is defined in steps of ±10.24µs based on the threshold parameter:

Table 9.1 - Individual PPS qualification threshold values

Value	Window size
0	±10.24µs
1	±20.48µs
…	…
65535	~ ±0.67s

9.1.1.2. Consecutive qualified PPS count

The number of qualified PPS after which the incoming PPS source will be considered "fully qualified" is defined using a consecutive qualified PPS count parameter. Synchronisation of SCET subseconds will occur on a qualified PPS following the qualified PPS which satisfied the consecutive qualified PPS count.

After the consecutive qualified PPS count has been reached, SCET subseconds synchronisation will be maintained and seconds synchronisation (via SCETTime write commands) can occur.

For example if the consecutive qualified PPS count parameter is 3:

- The incoming PPS source will be considered "fully qualified" after 3 qualified PPS.
- Subseconds synchronisation will occur on the 4th qualified PPS.
- Seconds synchronisation can occur on the 5th qualified PPS or above.

Any unqualified PPS or expired PPS events will:

- Cause the qualification procedure to restart from a count of 0.
- Cause subseconds synchronisation to the incoming PPS to stop being maintained.

If the consecutive qualified PPS count parameter is 0:

- The incoming PPS source will always be considered "fully qualified".
- Subseconds synchronisation will occur on the 1st qualified PPS.
- Seconds synchronisation can occur on any qualified PPS.

9.1.1.3. Seconds synchronisation

SCET seconds synchronisation is possible via a SCETTime write command (see Section 17.7.20). This command will set a pending seconds value which will be used to set the SCET seconds on the next PPS, provided:

- The PPS is qualified.
- Subseconds synchronisation has already occurred, i.e. the number of consecutive qualified PPS, prior to the current PPS, was greater than the consecutive qualified PPS count parameter (see Section 9.1.1.2).

The pending seconds value will be silently discarded after a successful update of the SCET seconds or at the next PPS event if the above criteria was not met.

Since the SCET seconds are updated on the next PPS, the seconds value in the SCETTime write command should normally be the reference seconds count plus one.

9.1.1.4. PPS source

The TCM uses the PPS1 port (DIGITAL connector) as the incoming PPS source.

9.1.1.5. Further reading

For more details regarding the SCET PPS synchronisation behaviour please refer to the SCET section in [RD6].

9.1.2. External synchronisation disabled

If external synchronisation is disabled, the incoming PPS will be ignored and no qualification nor synchronisation of SCET subseconds will occur.

When external synchronisation is disabled, SCET seconds can always be written and will be set immediately on write.

9.1.3. Configuration

Time synchronisation parameters can be configured both via NVRAM as described in Table 6.17 and via Spacewire RMAP commands as described in Section 17.7.27. The NVRAM configuration defines the persistent mission base parameters which are applied on power on or reset. Spacewire RMAP commands can be used to change the parameters non-persistently during operation.

9.2. TM time stamps

A timestamp can be generated when a TM Transfer Frame is sent on VC0. The rate of timestamp generation is configurable through an RMAP command, and the latest timestamp is readable on the same interface. See Section 17.7.11 and Section 17.7.12 for further info.

10. Error Management and System Supervision

The Error Manager in the TCM provides information about different errors and operational status of the system such as:

- Counters for correctable and uncorrectable errors
- Watchdog trips

Error Manager related information and housekeeping data is available by RMAP. See Section 17.7.19

The status of the TM Downlink and TC Uplink are available through RMAP. See Section 17.7.14 and Section 17.7.1.

A watchdog is enabled in the TCM that must be kicked by the TCM Application or a reset will occur. The watchdog is kicked only when all active tasks in the TCM report that they are alive.

11. Mass Memory Handling

11.1. Description

The mass memory in the TCM is primarily intended for storage of telemetry data while awaiting transfer to ground but can also be used for internal data storage. The mass memory is configurable as described in Section 11.2.

Table 11.1 - Mass memory page and block size

Mass Memory size	Page size [byte]	Block size [byte]	Pages per block
16 GB	16 * 1024	2 * 1024 *1024	128
32 GB	32 * 1024	4 * 1024 * 1024	128

The mass memory is accessed through the MM* RMAP commands described in Section 17.7. The mass memory is nandflash-based and that also slightly colours its user interface, even though the detailed handling has been abstracted away. The total amount of mass memory available is 16 or 32 GB, depending on hardware and SW configuration. As shown in Table 11.1 the page size is 16kB and the block size is 2MB for 16 GB of Mass Memory. For a mass memory of 32 GB, the page size is 32kB and the block size is 4MB. The number of pages per block is independent of mass memory size.

Due to the flash nature of the mass memory, each new block will require erasing before accepting writes, but the TCM software will handle this automatically. For each 32-bit word stored in mass memory, there are 8 bits stored as EDAC to be able to detect double errors and correct single errors. During erases or writes the operation may fail, and the software will then mark this block as bad and skip this in all future transactions. The bad block list is stored in NVRAM and will thus survive a reboot and/or power cycling. This graceful degradation behaviour of the mass memory implies that partitions may shrink in size and this phenomenon needs to be considered when planning partition sizes. Another effect of the bad blocks is that available space on a partition may decrease by more than the actual data written and this might need tracking by the user.

To simplify divisions between different types of data with different configurations, the mass memory is divided into logical partitions where each partition is configured by its mode, type, segment size and TM virtual channel for downloading. All partitions have an address space of 4 GiB regardless of their physical size and this is also the maximum size of a partition.

The number of blocks allocated to a partition determines the maximum amount of data that can be stored on the partition, up to the 4 GiB maximum. The number of blocks allocated to a partition does not change the address space of the partition.

There is no limit to the number of blocks allocated to a partition. If more blocks than than would be needed for the maximum 4GiB physical size is allocated, the extra blocks will be used for wear leveling and as reserve blocks if bad blocks occur. Such over-allocation can be used to ensure that a partition remains at a 4GiB physical capacity despite a number of discovered bad blocks.

Reading and writing to partitions behaves slightly different between different types of partitions, but when a partition is full, it requires a *free* operation to allow for further writes. New space for writing will only become available once a block is completely freed (that is, when a free operation passes over a block boundary).

Figure 11.1 illustrates this with an example two-block partition, showing in the last picture that new data cannot be written until free has reached the block boundary. To simplify operations for the user, free operations can be requested on more data than is available in the mass memory, see Section 17.7.34 for details.

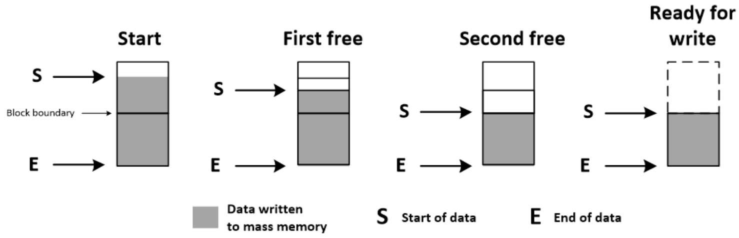

Figure 11.1 - Illustration of free behaviour and block boundaries.

11.2. Partition configuration

Partitions are configured via the NVRAM configuration tool, according to the format in Section 6.1.1, below follows some detailed information regarding certain configuration items.

11.2.1. Partition mode

Each partition can be configured as Continuous, Circular or Direct mode.

In **Continuous mode**, all write accesses are sequential and can be of any size but will return with an error when the partition is full. The MM handler internally implements free and write pointers to keep track of the data in the partition. The write pointer is used as the address for storing the data and is updated after each successful write. The free pointer is used as the address when freeing data and is updated after each successful free. Read access and download of data is available on any arbitrary address within the partition (between the free and write pointer addresses). Obsoleted data need to be freed to enable further writes when the partition is full.

Continuous Auto-padded mode operates in the same way as Continuous mode, with additional automatic segment padding, see Section 11.2.4.

Circular mode operates much in the same way as Continuous mode except that writes will never fail when the partition is full. Instead, it will automatically free one or more blocks used for the oldest written data and update the free pointer accordingly. Thus, data never needs to be freed manually, but the operation is available.

Circular Auto-padded mode operates in the same way as Circular mode, with additional automatic segment padding, see Section 11.2.4.

For both Continuous and Circular mode (with or without automatic padding), an internal cache of one page is used to hold any data that does not fit a page. As soon as the cache is filled, the data is written to physical memory. Any restarts or power cycling will result in loss of any data only written into this cache. If loss of cache data is an issue, ensure that all writes end on a page boundary as this will make sure all data is always written to flash.

In **Direct mode**, a write access can be to any arbitrary address in the address space provided that writing starts at a block boundary and is continuously written within this block. Each access must also be a multiple of the page size and thus keeps no cache of data not stored in physical memory. To determine the actual page size in use, the current page size can be read out using the RMAP command MMGetPageSize described in Section 17.7.37. Read access and download of data is available from any arbitrary address within a partition, given that it has valid data (previously written). Obsoleted data or data to overwrite need to be freed here as well but can be freed on any valid address in the address space.

IMPORTANT	Due to considerably increased initialisation times when using direct partitions, it is recommended to only allocate a maximum of 200 blocks (400 Mbytes for 16 GB mass memory or 800 Mbytes for 32 GB mass memory) in total to direct partitions. Increasing the amount of direct partition blocks significantly above this limit will cause initialisation failure due to the watchdog timeout being triggered.

The direct partition mode does not utilise free and write pointers.

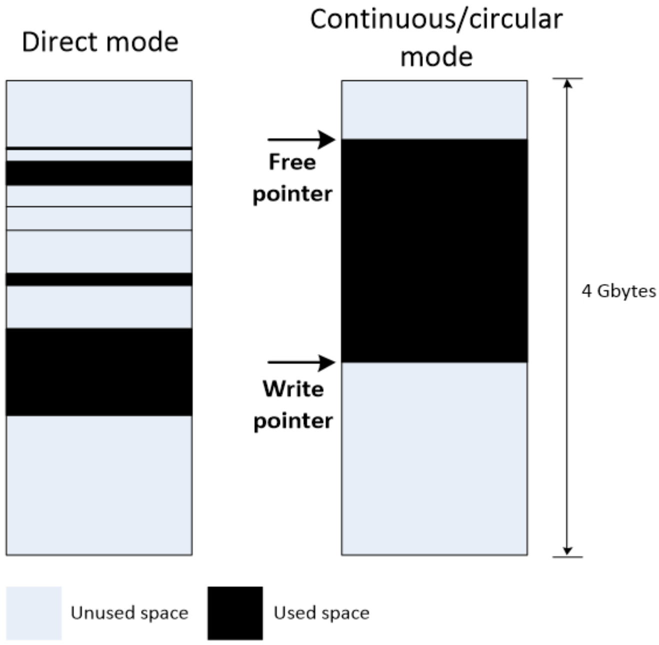

Figure 11.2 - Illustration of partition modes and the free/write pointers

11.2.2. Partition segment size

The segment size is only applicable for downloading and for partitions of type PUS (see below). The mass memory supports segment sizes of 16, 32, and 64 kbyte.

11.2.3. Partition type

Partitions can be of three types, PUS, raw and TC storage.

Partitions of **type PUS** require that each segment will begin with a CCSDS space packet and unless auto-padding is used, it is up to the software writing into the mass memory to maintain this segmentation. There are no limitations on the number of CCSDS space packets that can be contained in one segment, but if the written data doesn't fit exactly into the segment size it must be padded up to the segment boundary. Padding can be achieved either with a CCSDS idle space packet (which also will be transferred to ground) or with a bit pattern of 0xF5, allowing padding of as little as one byte. During a download operation when the padding bit pattern is discovered, download will skip to the next segment (if available).

NOTE	Despite the "PUS" naming, partitions of this type only requires CCSDS space packets (which is a superset of PUS packets), the name is used for historical reasons.

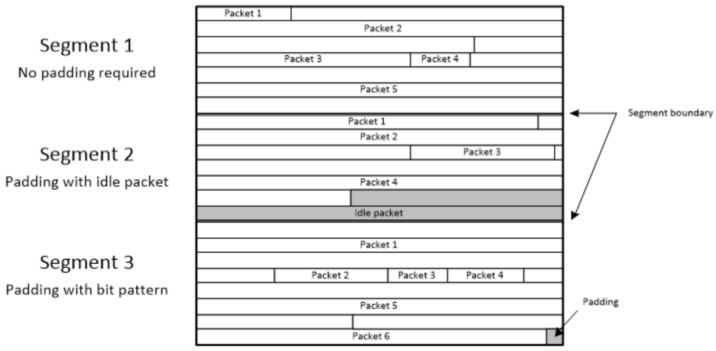

Figure 11.3 - Illustration of packet placement inside segments with different padding (marked in grey)

Partitions of **type raw** can be used to store data on-board if that is needed for the mission (to be written/read by other units in the system), but only PUS type partitions can be downlinked to ground through the CCSDS block.

Configuring a partition with **type TC storage** dedicates this partition for use by the TC storage, see Chapter 12 for more info. A partition with this type must use the continuous partition mode (without automatic padding) and no more than one partition may be configured with this type the same time.

11.2.4. Automatic padding

Continuous and circular partitions can be configured with automatic padding of segments, which automatically pads data written to the partition with a 0xF5 bit pattern, such that written data never overlaps a segment boundary, and is fit for download.

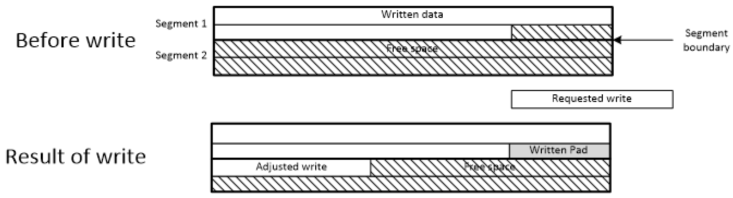

Figure 11.4 - Illustration of auto-padding of a requested write.

No examination or validation of data contents are done in the padding process, and if a write command with data containing multiple packets is received, it will be padded as if it was a single large packet.

Auto-padding will never split the data in a received write command, and thus writing with data that is larger than a segment is not supported.

If writing packets to an auto-padded partition, each write should contain data that starts at the beginning of a packet and ends at the end of a packet, in order to ensure that it is possible to download the data correctly.

Reads from an auto-padded partition will return padding and data as it was written to the partition in the auto-padding procedure.

Downloads from an auto-padded partition should consider the additional padding size for written data when calculating the download size. The free and write pointers can be used to determine the total current size of all written data including padding.

Automatic padding limits single writes to the segment size, therefore using segments of 16kB together with pages of 32kB makes it impossible to write a full page immediately to the physical mass memory. In this case some of the data to be written will always be kept in the SDRAM cache. To be able to write a full page of 32kB immediately to the physical mass memory when using automatic padding, the segment size must be equal to or larger than the page size (>=32kB).

11.2.5. Partition virtual channel

This specifies which CCSDS virtual channel to be used for downloading of the data in the partition. See [RD7] for the supported channels.

11.3. Recovery

The mass memory handler utilises the NVRAM to store on-going operation data, which is used in the initialisation step in order to recover consistency after aborted write or free operations, caused for example by a power failure reset.

If errors or inconsistencies are detected when the stored on-going parameters are read from NVRAM at initialisation, the recovery associated with the unavailable item will be skipped and the initialisation will continue.

The initialisation recovery is aggressive and will prioritise a usable system over data retention; any single block which exhibits metadata inconsistencies that make it impossible to safely add it to the translation table will be erased and considered free.

For continuous and circular partitions, further recovery is performed to ensure that the partition data range is continuous (which is required for the partition to be usable). If a discontinuity is discovered, the recovery process will erase data blocks from the highest logical partition address and downwards, until a continuous range of data is left on the partition. Such discontinuities can for example occur due to corrupt blocks, or if a partition is configured to include blocks with unknown contents (e.g. changing a direct partition into a continuous partition).

Recovery does not take into account the format of the stored data and may for example leave a partition with data that no longer fulfils segmentation requirements for download.

Recovery may cause the free and write pointers of continuous/circular partitions to move.

An empty continuous/circular partition, where the write pointer is located exactly at the start of a block, will have the free and write pointers reset to address 0 if a reset and subsequent re-initialisation occurs.

Recovery will cause rediscovery of previously freed data in a block in the following scenarios:

- If the block was not completely freed.
- If data was freed from the block in a continuous/circular partition and the free did not move past the block boundary.
- If data was freed from the block in a continuous/circular partition and the write pointer was located inside the block.

For continuous/circular partitions, this data rediscovery will only occur in the block where the free pointer was last located. For direct partitions, it will occur in every block which provides one of the scenarios listed above.

12. TC Storage

The TCM provides a ”TC storage“ which consists of a non-persistent storage that can be written to directly from the ground segment using PUS service request telecommands and can be read and cleared by the space segment via RMAP.

The TC storage functionality allows burst uploading of data while avoiding directly routing this data as telecommands over the SpaceWire network, which could be used to defer certain processing.

The format of the data stored in the TC storage is not defined by the TCM and instead needs to be coordinated between the ground segment writing into the storage and the space segment reading from the storage. This is especially important if the space segment needs to distinguish the boundary between individual data chunks in the TC storage.

If the existing error correction protections of the mass memory are insufficient for the current mission specification to guarantee that the space segment can always parse the data read from the TC storage, additional synchronization features could be added to the data format to allow discarding invalid data, for example a fixed data chunk size, a synchronization pattern, etc. Such features are the responsibility of the mission-specific ground and space segment.

A suggested use case for the TC storage is to provide deferred telecommand processing, where the data chunks written into the TC storage are in the form of telecommands, which can be read by the space segment when it is ready to process them. In this case the telecommands to be stored need to be embedded in the service data unit field of PUS service request telecommands - one (or more) telecommand(s) wrapped inside another telecommand.

The TC storage is non-persistent, and the stored data along with the status information will be cleared if the TCM is reset, despite using a mass memory partition for its storage. This clear is done in order to eliminate any potential data inconsistency which could occur due to write cache loss on reset or data recovery during initialization.

To enable the TC storage, a single mass memory partition must be configured with the TC storage partition type as described in Section 11.2.3.

The PUS service interface for writing into the TC storage from the ground segment is described in Section 15.2.

The space segment can use the MMData (read only), MMDataRange, MMPartitionConfig and MMPartitionSpace commands to read data from and information about the TC storage partition in the same way as from a standard continuous partition.

In addition, the space segment can use the MMTCStorageStatus command to read specific status information from the TC storage and can use the MMTCStorageClear command to clear the TC storage.

Writing to, freeing from, or downloading from a TC storage partition by the space segment is not supported.

See Chapter 17 for detailed descriptions of the space segment commands.

13. TC Queue

The TCM supports a functionality to route received TC packets to a queue, rather than consuming the TC packets directly, depending on the APID of the TC packet. It also defines an interface to read a packet from this queue, and to remove a previously read packet from the queue.

The TC queue stores individual TC packets and the queue size is 50 packets. The maximum size of TC packets to be stored is 1016 bytes, which represents the maximum size of a transfer frame data field without the segment header (see Section 24.6) . The queue is implemented as a circular queue, meaning that the FIFO (first in, first out) principle applies, and upon a queue overflow, the oldest packet is overwritten.

When a TC packet is added to the TC queue, it is assigned a queue item ID. This is an incremental counter, which is returned as part of the metadata when reading the packet. It can be used to check e.g., whether a queue overflow has occurred.

TC packets routed via the TC queue will be treated as CCSDS space packets and therefore only the CCSDS space packet primary header will be validated. If validation fails, the packet will be dropped (no PUS verification reports will be generated).

14. ECSS standard services

The TCM supports a subset of the services described in [RD11].

14.1. PUS-1 Telecommand verification service

Incoming telecommands directed to the TCM core application (based on their APID) will be treated as PUS requests and will undergo acceptance verification accordingly. If the verification fails, the telecommand is rejected and a report (as described in [RD11]) will be sent back to the ground segment as live telemetry. On successful acceptance verification, the request will be processed and executed. If the execution of the request fails, a failed completion of execution report will be sent back to the ground segment as live telemetry.

The TCM core application provides the following set of verification reports as part of its acceptance and execution verification of incoming PUS requests directed to the TCM core application.

- TM[1,1] successful acceptance verification report
- TM[1,2] failed acceptance verification report
- TM[1,7] successful completion of execution verification report
- TM[1,8] failed completion of execution verification report

Successful verification reports TM[1,1] and TM[1,7] will be sent back to the ground segment as live telemetry as part of the acceptance and execution verification only if the corresponding acknowledgement flag is set in the request.

Telecommands routed to other on-board applications (based on their APID) will only be assumed to be CCSDS space packets and not a necessarily PUS requests; no PUS request verification of these telecommand will occur and no PUS verification reports will be generated. It is the responsibility of the destination on-board application to handle the verification of these telecommands and send any potential verification reports if relevant.

The formats of the PUS acceptance verification reports TM[1,1] and TM[1,2] generated by the TCM core application are shown in Table 14.1 and Table 14.2.

Table 14.1 - Telecommand Acceptance Report - Success [1,1] data

Packet ID	Packet Sequence Control
UINT16	UINT16

Table 14.2 - Telecommand Acceptance Report - Failure [1,2] data

Packet ID	Packet Sequence Control	Code
UINT16	UINT16	UINT8. 0 - Illegal APID 1 - Invalid packet length 2 - Incorrect CRC 3 - Illegal packet type 4 - Illegal packet subtype 5 - Illegal application data 6 - Illegal PUS version

14.2. PUS-2 Distributing Register Load Command

By PUS service [2,2] it is possible to write data to devices on the TCM by a telecommand. One register load command per telecommand is supported.

Using this service if the PUS access to the UART is blocked (see Table 6.4) will result in a Telecommand execution completed report - failure.

Table 14.3 - **Distributing Register Load Command**

Register Address	Register Data
0xFF04000100 - UART0 0xFF04000101 - UART1 0xFF04000102 - UART2 (5 octets)	Array of UINT8

14.3. PUS-2 Device Command Distribution Service

The TCM supports the command pulse distribution unit (CPDU) pulse commands in hardware as defined in 6.2.6 in [RD11]. The CPDU listens for a specific virtual channel - APID pair, see the configuration document [RD7].
The TCM has 12 controllable (0-11) output lines and can be toggled to supply different pulse lengths according to the following scheme:

Table 14.4 - **CPDU Command [2,3]**

Output Line ID	Duration
0-11 (1 octet)	0 - 7 (1 octet)

The duration is a multiple of the CPDU_DURATION_UNIT (D), defined to 12.5 ms, as detailed below.

Table 14.5 - CPDU Duration

Duration in bits	Duration in time (ms)
000	1 x D = 12.5
001	2 x D = 25
010	4 x D = 50
011	8 x D = 100
100	16 x D = 200
101	32 x D = 400
110	64 x D = 800
111	128 x D = 1600

NOTE The APIDs reserved for the CPDU are 1 - 9 for future use.

14.4. PUS-2 Distributing Device Command

14.4.1. Description

By PUS service [2,128] it is possible to write a command to devices on the TCM by a telecommand. One device command per telecommand is supported. UART-devices have a fixed configuration of 8 data bits and 1 stop bit.

Using this service if the PUS access to the UART is blocked (see Table 6.4) will result in a Telecommand execution completed report - failure.

Table 14.6 - Distributing Device Command

Device Address	Bitrate	Mode	Parity
0xFF04000100 - UART0 0xFF04000101 - UART1 0xFF04000102 - UART2 (5 octets)	12 = 223200 baud 11 = 375000 baud 10 = 347200 baud 9 = 153600 baud 8 = 115200 baud (default) 7 = 75600 baud 6 = 57600 baud 5 = 38400 baud 4 = 19200 baud 3 = 9600 baud 2 = 4800 baud 1 = 2400 baud 0 = 1200 baud (1 octet)	0=RS422 mode (default) 1=RS485 mode 2=Loopback (1 octet)	0=No parity (default) 1=Odd parity 2=Even parity (1 octet)

14.4.2. Execution Verification Profile

Table 14.7 - Distributing Device Command - execution verification completion failed report - failure notice codes

Failure Code	Description
EIO (5)	I/O error. The UART device cannot be accessed
EINVAL (22)	Invalid argument values.

The execution verification completion failure notice is made of only of a failure code value of 8 bits (see table above), with no associated parameters.

15. Custom services

15.1. PUS-130 Software upload

During the lifetime of a satellite, the on-board software might need adjustments as bugs are detected or the mission parameters adjusted. This service solves that by providing a means for updating the on-board software in orbit. See Chapter 22 for further info.

15.2. PUS-131 TC Storage

15.2.1. Description

The TC storage service provides the capability to store data into the TC storage (see Chapter 12) for later retrieval by the space segment.

The TC storage service does not provide any capability to read or clear the data stored into the TC storage, this responsibility is delegated completely to the space segment.

The TC storage service provides a storage area into which data chunks can be appended. The storage area is configured via the mass memory partition configuration in the NVRAM, using the special TC storage partition type.

When the TC storage partition becomes full, no more data can be appended into the storage area and attempted stores will be discarded and an execution failure report sent. The space segment is responsible for clearing the storage area for re-use.

The amount of data that can be stored in the TC storage before it becomes full depends on the number of blocks configured for the TC storage partition. The maximum number of data chunks that can be appended into the TC storage before clearing is $2^{32}-1$, exceeding this limit is not supported (although it is highly unlikely based on the telecommand uplink speed).

The TC storage service maintains status information about the number of bytes used in the storage area, the amount of data chunks currently stored in the storage area and the amount of data chunks which has failed to be stored due to the storage area being full. This status information is only accessible by the space segment.

Requests to the TC storage must use the TCM core application APID, configured in Table 6.13.

The TC storage service defines a single service request named “TC storage store data” with service type 131 and service subtype 0.

The service data unit associated with the TC storage store data service request is a single “deduced parameter” in the form of a “fixed-length octet string” which is deduced from the request telecommand packet length, see [RD11] for details. In other words, the payload is treated as raw data bytes and will not be parsed in any way.

When the TC storage service receives a TC storage store data service request it will attempt to append the data in the service data unit of this request into the storage area. The result of the append action will be provided via an execution failure or execution success report (see Section 14.1).

15.2.2. Execution Verification Profile

For a general overview of PUS telecommand verification, see Section 14.1.

The format of the failure notice field in the failed completion of execution verification report for the TC storage service is shown below.

Table 15.1 - TC storage store data - failed completion of execution verification report failure notice format

Failure Code
1 - Unable to append due to data store being full. 2 - No TC storage partition is configured. (1 octet)

16. RIRP RMAP Interface

RIRP is an alternative interface for RMAP command access to the TCM.

Specific RMAP addresses for devices and sub-systems are allocated for RIRP-interface accesses to the TCM, see Table 17.3 for info about addresses.

With RIRP, the reply uses standard RMAP status codes as described in [RD12] and the specific execution status is not generally returned in the reply, but instead stored in a transaction status buffer to be read out separately.

The transaction status buffer is not used in the case of acceptance errors, successful reads, or reads from the transaction status buffer itself.

See Section 17.2 for more information.

17. Spacewire RMAP

A general description of how the Spacewire RMAP is used by the TCM is given in Section 17.1. Section 17.2 describes the alternative RIRP interface. Section 17.3 deals with the incoming RMAP commands that the TCM application supports. Any RMAP commands issued by the TCM are described in Section 17.4. Section 17.5 deals with the status codes returned with the replies to incoming commands, and Section 17.6 explains the use of transaction ID's to keep track of where replies shall be sent. Finally, Section 17.7 and Section 17.8 provide further details about the incoming and outgoing RMAP commands.

17.1. Description

According to [RD12], a 40-bits address consisting of an 8-bit Extended Address field and a 32-bit Address field is used in RMAP. This has been utilized in the TCM according to Table 17.3 to separate between configuration commands and mass memory storage of data (partition handling).

The initiator logic address of output messages from the TCM, and the RMAP key that needs to be used for input messages and should be expected from output messages, are shown in Table 17.1.

Table 17.1 - RMAP predefined fields

Field	Value
Initiator Logical Address	configurable, see Section 6.1.4 (default 0x42).
Key	0x30

In addition, target address and reply address must be added to the RMAP header in commands targeting the Sirius TCM to compensate for topology external to the Sirius TCM and the embedded SpaceWire router. As can be seen in Figure 5.1, if the Sirius TCM were to be addressed from SpaceWire port 1, the example addresses below must be added to the routing addresses in the RMAP header.

Table 17.2 - RMAP predefined fields for routing

Field	Value
Target Spw Address	0x01, 0x03
Reply Address	0x01, 0x03

IMPORTANT

The size requested in RMAP read commands will be ignored and the returned data by the reply will be of a fixed size determined by the TCM. Except for the commands MMData,

> and RIRPTransactionStatus, where the size requested will be used. Refer to the respective subsection of Section 17.7 for details about the size returned by the individual commands.

In the RMAP header Instruction field there is a Verify-Data-Before-Write bit. In the TCM that is used as follows:

- If an RMAP command is received by the TCM SW with Verify-Data-Before-Write set, the data integrity is verified before the command is processed, according to the RMAP standard.
- If an RMAP command is received by the TCM SW with the Verify-Data-Before-Write bit not set, the data payload integrity is not verified before nor after the command is processed. This is a deliberate deviation from the RMAP standard to allow high throughput writing of data where immediate indications of any data corruption is not critical, and/or will be provided via other means.

17.2. RIRP Interface

Specific addresses have been allocated to be used for the RIRP interface as described in Table 17.3.

The command for reading from the transaction status buffer is described in Section 17.7.53.

Limitation:

Using both the standard RMAP interface and the RIRP interface in parallel is not supported. The desired interface must be configured in NVRAM (see Table 6.9) and only the addresses corresponding to the configured interface may be used.

17.2.1. Command Acceptance

If an invalid command is received by the TCM it is discarded without sending a reply.

If a command is received by the TCM which contains an invalid extended address, a reply is sent with a status set to 1 (General error code). In this case the command is not stored in the transaction status buffer.

When using RIRP and the RIRP transaction status buffer is full, all incoming RMAP commands will be rejected. When a RIRP command is rejected for this reason, a reply with status 1 (General Error) will be sent to the initiator. Reading from the RIRP transaction buffer must be performed before the TCM-SW can handle new RMAP commands.

Up to 200 transactions can be stored in the transaction status buffer.

17.2.2. Write Commands

If a RIRP write command is received and accepted by the TCM, a reply will be sent directly with a status indicating success, the command is then added to the transaction status buffer with an “ongoing” status.

When an accepted write command completes execution either successfully or with an error, the entry in the transaction status buffer is updated with a “finished” status and the specific execution status. The initiator of the write command is expected to read from the transaction status buffer to determine the execution status.

17.2.3. Read commands

For RIRP read commands, a reply is sent within 100 ms (excluding any delays due to spacewire network congestion).

Read commands which have been received and accepted will be added to the transaction status buffer with an “ongoing” operation state.

If a read command has been received and accepted and the read execution finishes within 100ms, a reply will be sent with the read data and a status indicating success. The read command is no longer stored in the transaction status buffer after success and the initiator is not expected to read from the transaction status buffer.

If a read command has been received and accepted and the read execution does not finish within 100ms, a reply will be sent with no read data and a status set to 1 (General error code). The entry in the transaction status buffer is updated with a “timed out” operation state (execution status is unspecified in this case). The initiator of the read command is expected to read from the transaction status buffer to determine the timed out status of the read command.

If a read command has been received and accepted and the read execution completes with an error, a reply will be sent with a status set to 1 (General error code) which may include read data, depending on the error. The entry in the transaction status buffer is updated with a “finished” operation state and the specific execution status. The initiator of the read command is expected to read from the transaction status buffer to get the specific execution error.

17.2.4. Reading from the Transaction Status Buffer

RIRP read commands which reads from the transaction status buffer is an exception to the general read command handling:

- Transaction status buffers read commands are never added as transaction status buffer entries.
- Transaction status buffer reads cannot time out.

- If a transaction status buffer read fails due to the read length being longer than the transaction status buffer size, a reply will be sent with a status set to 11 (RMW Data Length error), this is a non-standard use of this RMAP status code.
- No other observable execution failures exist for transaction status buffer reads.

Commands with has completed execution or timed out will be cleared from the transaction status buffer once the transaction status entry has been fully read.

17.3. Input

The RMAP commands supported by the TCM are specified in table below. See Section 17.7 for details on each specific command.

Table 17.3 - RMAP commands TCM

Name	Ext. Addr	Address	Cmd	Description
TMStatus	0x90 - RIRP 0xFF - No RIRP	0x00000000	R	Reads latest telemetry status.
TMConfig	0x90 - RIRP 0xFF- No RIRP	0x00000200	R	Reads telemetry configuration.
TMControl	0x90 - RIRP 0xFF- No RIRP	0x00000300	W	Enable/Disable telemetry.
TMFEControl	0x90 - RIRP 0xFF - No RIRP	0x00000400	W	Enable/Disable Frame Error Control Field for TM Transfer Frames.
TMMCFCControl	0x90 - RIRP 0xFF - No RIRP	0x00000500	W	Enable/Disable Master Channel Frame Counter Control for TM Transfer Frames.
TMIFControl	0x90 - RIRP 0xFF - No RIRP	0x00000600	W	Enable/Disable Idle Frames.
TMPRControl	0x90 - RIRP 0xFF - No RIRP	0x00000700	W	Enable/Disable Pseudo Randomization for telemetry.
TMCEControl	0x90 - RIRP 0xFF - No RIRP	0x00000800	W	Enable/Disable Convolutional Encoding for telemetry.
TMBRControl	0x90 - RIRP 0xFF - No RIRP	0x00000900	W	Sets telemetry clock frequency divisor (bitrate)

Name	Ext. Addr	Address	Cmd	Description
TMOCFControl	0x90 - RIRP 0xFF - No RIRP	0x00000A00	W	Enable/Disable inclusion of Operational Control field in TM Transfer Frames.
TMTSControl	0x90 - RIRP 0xFF - No RIRP	0x00000B00	R/W	Configures Timestamp of telemetry.
TMTSStatus	0x90 - RIRP 0xFF - No RIRP	0x00000C00	R	Latest timestamp of telemetry on virtual channel 0.
TMSend	0x90 - RIRP 0xFF - No RIRP	0x0000100N	W	Sends telemetry on virtual channel N.See [RD7] for allowed VCs.
TCStatus	0x90 - RIRP 0xFF - No RIRP	0x01000000	R	Reads latest telecommand status.
TCDRControl	0x90 - RIRP 0xFF - No RIRP	0x01000100	W	Enables/Disables Derandomizer of telecommands.
TCQueueQuery	0x90 - RIRP 0xFF - No RIRP	0x01001000	R	Query the oldest packet from TC queue.
TCQueueRemoveAndQuery	0x90 - RIRP 0xFF - No RIRP	0x01001100	R	Remove packet from TC queue and query next.
TCQueueClear	0x90 - RIRP 0xFF - No RIRP	0x01001200	W	Clear the queue.
HKData	0x90 - RIRP 0xFF - No RIRP	0x02000000	R	Reads housekeeping data.
SCETTime	0x90 - RIRP 0xFF - No RIRP	0x02000100	R/W	Reads/Sets SCET time.
HKResetCause	0x90 - RIRP 0xFF - No RIRP	0x02000500	R	Retrieves the cause of the last TCM reset
HKLastBootStatus	0x90 - RIRP 0xFF - No RIRP	0x02000600	R	Reads out the status of the last failed boot.
HKDeathReports	0x90 - RIRP 0xFF - No RIRP	0x02000700	R	Reads out death reports to allow analysis of resets.
HKClearDeathReports	0x90 - RIRP 0xFF - No RIRP	0x02000800	W	Clears the death report area on NVRAM.

Name	Ext. Addr	Address	Cmd	Description
Reserved	0x90 - RIRP 0xFF - No RIRP	0x02000A00	R	Reserved for internal usage.
HKCpuUsage	0x90 - RIRP 0xFF - No RIRP	0x02000B00	R	Reads the CPU usage from TCM.
HKCpuUsageReset	0x90 - RIRP 0xFF - No RIRP	0x02000C00	W	Reset the CPU usage on TCM.
TimesyncConfig	0x90 - RIRP 0xFF - No RIRP	0x02000D00	R/W	Gets or sets the timesync configuration.
UARTCommand	0x90 - RIRP 0xFF - No RIRP	0x0400010n	W	Sends a command to UART device n. 0 - UART0 1 - UART1 2 - UART2 3 - UART3 4 - UART4 5 - reserved 6 - UART6 (PSU Ctrl) 7 - UART7 (Safe Bus)
MMData	0x80-0x8F-RIRP 0x00-0x0F - No RIRP	0xnnnnnnnn	R/W	Reads/writes data from/to a partition. The extended address field determine the partition number. The address field is used differently on different types of partitions, see command details.
MMDataRange	0x90 - RIRP 0xFF - No RIRP	0x0500010n	R	Address ranges of all stored data in partition n.
MMPartitionConfig	0x90 - RIRP 0xFF - No RIRP	0x0500030n	R	Configuration of partition n.
MMPartitionSpace	0x90 - RIRP 0xFF - No RIRP	0x0500040n	R	Space available in partition n.
MMDownloadPartitionData	0x90 - RIRP 0xFF - No RIRP	0x0500050n	W	Downloads partition n data via telemetry.

Name	Ext. Addr	Address	Cmd	Description
MMFree	0x90 - RIRP 0xFF - No RIRP	0x0500060n	W	Frees memory from partition n.
MMDownloadStatus	0x90 - RIRP 0xFF - No RIRP	0x0500070n	R	Amount of data downloaded in partition n.
MMStopDownloadData	0x90 - RIRP 0xFF - No RIRP	0x05000A0n	W	Stops download of data from partition n.
MMGetPageSize	0x90 - RIRP 0xFF - No RIRP	0x05000B00	R	Reads out size of page, block and spare area
MMTCStorageStatus	0x90 - RIRP 0xFF - No RIRP	0x05000C00	R	TC storage status information.
MMTCStorageClear	0x90 - RIRP 0xFF - No RIRP	0x05000D00	W	Clear the TC storage.
MMBadBlockCount	0x90 - RIRP 0xFF - No RIRP	0x05000E00	R	Read out number of bad blocks.
MMCombinedDataRange	0x90 - RIRP 0xFF - No RIRP	0x05000F00	R	Combined address ranges for each partition.
MMCombinedConfig	0x90 - RIRP 0xFF - No RIRP	0x05001000	R	Combined configuration for each partition.
MMCombinedSpace	0x90 - RIRP 0xFF - No RIRP	0x05001100	R	Combined available space for each partition.
MMCombinedDownloadStatus	0x90 - RIRP 0xFF - No RIRP	0x05001200	R	Combined amount of data downloaded for each partition.
SpwBackupRoutingEnableDisableSet	0x90 - RIRP 0xFF - No RIRP	0x07000200	W	Enables/disables backup SpW routing
SpwBackupRoutingEnableDisableGet	0x90 - RIRP 0xFF - No RIRP	0x07000300	R	Reads out the current SBR configuration
SpwRoutingPathSet	0x90 - RIRP 0xFF - No RIRP	0x07000400	W	Sets the SpW paths
SpwRoutingPathGet	0x90 - RIRP 0xFF - No RIRP	0x07000500	R	Reads out the SpW paths

Name	Ext. Addr	Address	Cmd	Description
SpwReplyPathSet	0x90 - RIRP 0xFF - No RIRP	0x07000600	W	Sets the SpW write-reply paths.
SpwReplyPathGet	0x90 - RIRP 0xFF - No RIRP	0x07000700	R	Gets the SpW write-reply paths.
SpwBackupRoutingTimeoutSet	0x90 - RIRP 0xFF - No RIRP	0x07000800	W	Sets the write reply timeout of the TCM in milliseconds
SpwBackupRoutingTimeoutGet	0x90 - RIRP 0xFF - No RIRP	0x07000900	R	Reads out the write reply timeout of the TCM in milliseconds
RIRPTransaction Status	0x90 - RIRP 0xFF - No RIRP	0x07000A00	R	Reads out the RIRP transaction status
GPIOGetConfig	0x90 - RIRP 0xFF - No RIRP	0x090001nn	R	Get configuration of GPIO nn
GPIOSetConfig	0x90 - RIRP 0xFF - No RIRP	0x090002nn	W	Set configuration for GPIO nn
GPIOGetValue	0x90 - RIRP 0xFF - No RIRP	0x090003nn	R	Get value of GPIO nn
GPIOSetValue	0x90 - RIRP 0xFF - No RIRP	0x090004nn	W	Set output value for GPIO nn

17.4. Output

The TCM supports routing of data received by some of its interfaces to other Spacewire nodes, according to the address map below:

NOTE All outgoing communication will use the Transaction ID range of 0x0000-0x0FFF.

Table 17.4 - Published data from TCM

Name	Ext. Addr.	Address	Cmd	Description
TCCommand	0xFF	0x00000000	W	Routed Telecommands

Name	Ext. Addr.	Address	Cmd	Description
UARTData	0xFF	0x0400000x	W	Data received on specified UART x. 0 - UART0 1 - UART1 2 - UART2 3 - UART3 4 - UART4 5 - reserved 6 - UART6 (PSU Ctrl) 7 - UART7 (Safe Bus)

17.5. Status code in reply messages

17.5.1. Status field, RIRP Disabled

In the status field of write/read, the values in table below can be returned, this replaces the standard RMAP status codes described in [RD12]. See individual commands for specific status code interpretations.

Table 17.5 - Status codes, RIRP disabled

Code	Numeric value
-	0
EIO	5
EAGAIN	11
ENOMEM	12
EEXIST	17
ENODEV	19
EINVAL	22
ENOSPC	28
ENODATA	61
EBADMSG	77
EALREADY	120
ESTALE	133
ENOTSUP	134
ECANCELED	140

17.5.2. Status field, RIRP Enabled

In RMAP Write/Read reply messages when using RIRP, the status field of the reply contains values according to [RD12].

The possible status codes are described in Table 17.6.

Table 17.6 - RIRP Status Codes

Numeric value	Command type	Meaning
0	read	Success.
0	write	Success.
1	read	One of: • Invalid extended address. • Read execution timed out. • Read execution failed. RIRP transaction status buffer is full
1	write	One of: • Invalid extended address. • Write execution failed
11	read	Length is greater than the maximum when reading from the transaction status buffer (non-standard use of RMAP status code).

If a write command is sent using RIRP without requesting a reply, no reply is returned.

Error code 1 (General error code) can be returned without the RIRP transaction status buffer being updated. This indicates that the RIRP transaction status buffer was full or that the extended address was wrong.

When error code 1 (General error code) is returned in a read reply, more details about the actual error can be obtained by reading from the transaction status buffer, Table 17.7 provides further details of how to distinguish the actual error in this case.

Table 17.7 - Determining actual error for read general errors

Actual error	Determined by
Invalid extended address	No command entry with corresponding transaction ID is present in transaction status buffer.
Read execution timed out	Command entry with corresponding transaction ID is present in transaction status buffer and contains a “timed out” operation state.
Read execution failed	Command entry with corresponding transaction ID is present in transaction status buffer and contains a “finished” operation state.
RIRP transaction status buffer is full	No command entry with corresponding transaction ID is present in transaction status buffer.

17.6. Transaction ID

The TCM uses the RMAP Transaction ID to separate between outstanding replies to different units. When several nodes are addressing the TCM, they need to be assigned a unique transaction id range to ensure correct system behaviour. To allow for similar transaction identification throughout the system, the TCM uses the Transaction ID range 0x0000-0x0FFF in all outgoing communication. The transaction id range 0x0000-0x0FDF is used for normal commands, and the range 0x0FDE-0x0FFF is used for resends of commands.

A single node addressing the TCM also must make sure that transaction IDs used for commands that can overlap in time are unique, in order to ensure that on-going transactions cannot be mixed up with new transactions. This also applies for commands without a requested reply.

17.7. RMAP input address details

The chapters below contain detailed information on the data accesses to the given RMAP addresses.

17.7.1. TMStatus

Reads the latest telemetry status.

Table 17.8 - TMStatus data

Byte	Type	Description
0	UINT8	0x00 - No Error 0x01 - FIFO error.
1	UINT8	Reserved

RMAP reply status:

Table 17.9 - TMStatus reply status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized.
EIO	I/O error. The TM device cannot be accessed

17.7.2. TMConfig

Reads the telemetry configuration.

Table 17.10 - TMConfig data

Byte	Type	Description
0-1	UINT16	Telemetry clock bitrate divisor value, default 250.
2	UINT8	Telemetry Control 0x00 - Disabled 0x01 - Enabled (default)
3	UINT8	OCF Control 0x00 - Disabled 0x01 - Enabled (default)
4	UINT8	Frame Error Counter Field Control 0x00 - Disabled 0x01 - Enabled (default)
5	UINT8	Master Channel Frame Count Control 0x00 - Disabled 0x01 - Enabled (default)

Byte	Type	Description
6	UINT8	Idle Frame Control 0x00 - Disabled 0x01 - Enabled (default)
7	UINT8	Convolutional Encoding Control 0x00 - Disabled (default) 0x01 - Enabled
8	UINT8	Pseudo Randomization Control 0x00 - Disabled (default) 0x01 - Enabled

RMAP reply status:

Table 17.11 - TMConfig reply status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized.
EIO	I/O error. The TM device cannot be accessed

17.7.3. TMControl

Controls generation of telemetry.

Table 17.12 - TMControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled 0x01 - Enabled (default)

RMAP reply status (if a reply is requested):

Table 17.13 - TMControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed

17.7.4. TMFEControl

Controls Frame Error Control Field inclusion for transfer frames.

Table 17.14 - TMFEControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled 0x01 - Enabled (default)

RMAP reply status (if a reply is requested):

Table 17.15 - TMFEControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed

17.7.5. TMMCFCControl

Controls the Master Channel Frame Counter generation for transfer frames.

Table 17.16 - TMMCFCControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled 0x01 - Enabled (default)

RMAP reply status (if a reply is requested):

Table 17.17 - TMMCFCControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed

17.7.6. TMIFControl

Controls the Idle Frame generation for transfer frames.

Table 17.18 - TMIFControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled 0x01 - Enabled (default)

RMAP reply status (if a reply is a requested):

Table 17.19 - TMIFControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed

17.7.7. TMPRControl

Controls the Pseudo Randomization for transfer frames.

Table 17.20 - TMPRControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled (default) 0x01 - Enabled

RMAP reply status (if a reply is requested):

Table 17.21 - TMPRControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed

17.7.8. TMCEControl

Controls the Convolutional Encoding for transfer frames.

NOTE Convolutional encoding **doubles** both the amount of telemetry data sent and also the telemetry clock frequency, keeping the same net datarate as without.

Table 17.22 - TMCEControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled (default) 0x01 - Enabled

RMAP reply status (if a reply is requested):

Table 17.23 - TMCEControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed

17.7.9. TMBRControl

Sets the telemetry clock frequency divisor.

The telemetry clock is fed to the radio and determines the TM output rate. The divisor is defined such that the useful payload bitrate (before possible convolutional encoding) is the same irrespective of whether convolutional encoding is performed or not. The frequency of the telemetry clock thus depends on the divisor and whether convolutional encoding is enabled or disabled according to:

Interface bitrate with convolutional encoding: $TM\_clk = \frac{50 \cdot 10^6}{divisor}$

Interface bitrate without convolutional encoding: $TM\_clk = \frac{50 \cdot 10^6}{2 \cdot divisor}$

Payload bitrate **irrespective** of convolutional encoding: $TM\_clk = \frac{50 \cdot 10^6}{2 \cdot divisor}$

Note that a 50% duty cycle will not be achieved with an odd divisor and convolutional encoding enabled.

The TM stream will be interrupted while the bitrate change takes place, as TM is disabled before updating the divisor and then reenabled if it was enabled before.

Table 17.24 - TMBRControl data

Byte	Type	Description
0-1	UINT16	Bitrate divisor value (default 25). Convolutional encoding: $6 \leq divisor \leq 12500$ No convolutional encoding: $3 \leq divisor \leq 6250$

RMAP reply status (if a reply is requested):

Table 17.25 - TMBRControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid.
EIO	I/O error. The TM device cannot be accessed

17.7.10. TMOCFControl

Controls Operational Control Field inclusion in TM Transfer frames.

Table 17.26 - TMOCFControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled 0x01 - Enabled (default)

RMAP reply status (if a reply is requested):

Table 17.27 - TMOCFControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed

17.7.11. TMTSControl

Configures the timestamping for transfer frames.

Table 17.28 - TMTSControl data

Byte	Type	Description
0	UINT8	The period of the generation is the power of two with the input as exponent. 0x00 - Take a timestamp every time frame sent (default) 0x01 - Take a timestamp every 2nd time frame sent 0x02 - Take a timestamp every 4th time frame sent ... 0x08 - Take a timestamp every 256th time frame sent

RMAP reply status (if a reply is requested):

Table 17.29 - TMTSControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is invalid
EIO	I/O error. The TM device cannot be accessed.

17.7.12. TMTSStatus

The latest timestamp of telemetry sent on virtual channel 0.

Table 17.30 - TMTSStatus data

Byte	Type	Description
0	UINT32	Seconds counter sampled when the frame event triggered
4	UINT16	Subseconds counter sampled when the frame event triggered

RMAP reply status:

Table 17.31 - TMTSStatus status codes

Status code	Description
0	Success.
EIO	I/O error. The TM device cannot be accessed

17.7.13. TMSend

Sends telemetry to the TM path on virtual channel N. See [RD7] for VC allocation. If RIRP is enabled and Live TM is sent to the TCM at a higher rate than the TCM can push it to the radio, it is indicated by the RMAP reply with status code 1 (General error). The payload of the TMSend command will be rejected. Reading RIRPTransactionStatus gets detailed information about the error that occurred in the TMSend command.

If TMSend commands provide data at a rate higher than the TM downlink can handle (depending on current bitrate configuration and any other ongoing downlink), it will result in the TM live buffer becoming full. If additional TMSend commands are received when the TM live buffer is full, these commands will be rejected with execution status 12 (“ENOMEM”).

The manner in which this execution status is presented depends on which API is used:

- When not using the RIRP API, this execution status will be provided in the error code field of the RMAP reply.
- When using the RIRP API, the RMAP reply error code will be 1 (“general error”) and the execution status will be provided in the RIRP transaction status buffer accessible via the RIRPTransactionStatus command.

NOTE The data must contain **at least one** telemetry CCSDS space packet.

Table 17.32 - TMSend data

Byte	Type	Description
0 - nn	Array of UINT8	Data containing CCSDS space packet(s).

RMAP reply status (if a reply is requested):

Table 17.33 - TMSend status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized.

Status code	Description
EIO	I/O error. The TM device cannot be accessed
ENOMEM	TM live buffer is full

17.7.14. TCStatus

Reads current telecommand status.

Table 17.34 - TCStatus data

Byte	Type	Description
0	UINT32	CLCW word of the last received telecommand.
4	UINT8	Number of missed TC frames due to overflow. Wraps after 0xFF.
5	UINT8	Number of rejected CPDU commands. Wraps after 0xFF.
6	UINT8	Number of rejected telecommands. Wraps after 0xFF.
7	UINT8	Number of parity errors generated by checksums in the telecommand path. Wraps after 0xFF.
8	UINT8	Number of received telecommands. Both TC and CPDU are counted. Wraps after 0xFF.
9	UINT16	Last CPDU pulse command. Logic 1 indicates the last activated line. Bit 15:12 - Unused Bit 11:0 - Line 11:0
11	UINT8	Number of accepted CPDU commands. Wraps after 0x0F.
12	UINT8	Derandomizer setting 0x00 - Disabled. 0x01 - Enabled.
13	UINT16	Length of the last received TC frame

RMAP reply status:

Table 17.35 - TCStatus status codes

Status code	Description
0	Success.
EINVAL	The driver for the TC device has not been initialized.
EIO	I/O error. The TC device cannot be accessed

17.7.15. TCDRControl

Configures derandomization for telecommand frames.

Table 17.36 - TCDRControl data

Byte	Type	Description
0	UINT8	0x00 - Disabled (default) 0x01 - Enabled

RMAP reply status (if a reply is requested):

Table 17.37 - TCDRControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TC device has not been initialized or invalid argument length.
EIO	I/O error. The TC device cannot be accessed

17.7.16. TCQueueQuery

Reads the oldest packet from the TC queue and some metadata.

Table 17.38 - TCQueueQuery data

Byte	Type	Description
0	UINT8	Number of packets in queue
1	UINT8	Queue item ID of current TC packet, [1,255]
2	UINT16	Size of TC packet (maximum 1016)
4 to ...	Array	TC packet

RMAP reply status:

Table 17.39 - TCQueueQuery status code

Status code	Description
0	Success
EAGAIN	TC queue is empty (no data returned)

17.7.17. TCQueueRemoveAndQuery

Remove the oldest packet from the TC queue (supposedly one that was read before) and read the next packet in the queue. Data returned by this command is the same as in Table 17.38.

RMAP reply status:

Table 17.40 - TCQueueRemoveAndQuery status code

Status code	Description
0	Success
EAGAIN	Removal succeeded, but there is no available TC packet in the queue to query (no data returned)
ENODATA	No packet to remove in the queue, queue is empty

17.7.18. TCQueueClear

This command clears the entire TC queue. This command does not require or provide any data.

RMAP reply status:

Table 17.41 - TCQueueClear status code

Status code	Description
0	Success

17.7.19. HKData

Reads the housekeeping data.

Table 17.42 - HKData data

Byte	Type	Description
0	UINT32	SCET Seconds
4	UINT16	SCET Subseconds
6	UINT16	Input voltage [mV]

Byte	Type	Description
8	UINT16	Regulated 3V3 voltage [mV]
10	UINT16	Regulated 2V5 voltage [mV]
12	UINT16	Regulated 1V2 voltage [mV]
14	UINT16	Input current [mA]
16	INT32	Temperature [m°C]
20	UINT8	S/W version 0-padding
21	UINT8	S/W major version
22	UINT8	S/W minor version
23	UINT8	S/W patch version
24	UINT8	Reserved
25	UINT8	Watchdog trips
26	UINT8	Correctable SDRAM errors found during CPU access to working memory
27	UINT8	Correctable SDRAM errors found by the scrubber or though a DMA access, in CPU working memory
28	UINT8	Uncorrectable SDRAM errors found during CPU access to working memory
29	UINT8	Uncorrectable SDRAM errors found by the scrubber or though a DMA access, in CPU working memory

RMAP reply status:

Table 17.43 - HKData status codes

Status code	Description
0	Success.
EINVAL	The driver for the HK device has not been initialized.
EIO	I/O error. The HK device cannot be accessed

17.7.20. SCETTime

Reads/sets the SCET time.

If external synchronisation is enabled, a write will set a pending seconds value which may be used to set the SCET seconds on the next PPS, hence the seconds value should normally be the reference seconds count plus one.

The command execution will only reflect if setting the pending seconds value was successful, it will not reflect whether or not it will actually be used to update the SCET seconds. The pending seconds will be silently discarded if the criteria for allowing seconds synchronisation is not met.

See Section 9.1.1.3 for further details.

If external synchronisation is disabled, setting the SCET seconds value is always possible and will occur immediately.

The subseconds value is ignored for write commands.

Table 17.44 - SCETTime data

Byte	Type	Description
0	UINT32	SCETSeconds
4	UINT16	SCETSubSeconds

RMAP reply status (if a reply is requested):

Table 17.45 - SCETTime status codes

Status code	Description
0	Success.
EINVAL	Insufficient command length.
EIO	I/O error. Reading from the SCET device failed.

17.7.21. HKResetCause

Gets the last cause of system reset.

Table 17.46 - HKResetCause data

Byte	Type	Description
0	UINT32	SCET seconds when latest reset was triggered. Zero following a hard reset or power-up.
4	UINT16	SCET subseconds when latest reset was triggered. Zero following a hard reset or power-up.

Byte	Type	Description
6	UINT8	Last cause of reset encoded as: 0x0 - Power-Up 0x1 - Watchdog 0x2 - Manual (SW initiated) 0x3 - CPDU (safe image) 0x4 - CPDU (default image) 0x5 - Uncorrectable SDRAM error during CPU access to working memory
7	UINT8	RESERVED

RMAP reply status:

Table 17.47 - HKResetCause status codes

Status code	Description
0	Success.

17.7.22. HKLastBootStatus

Gets status of last failed boot, if any. Otherwise get status of latest successful boot.

Table 17.48 - HKLastBootStatus data

Byte	Type	Description
0	UINT8	Steps defined: 1 - Init 2 - Init timer 3 - Init UART 4 - Read SoC info 5 - Wait for scrubber 6 - Read bad-block table 7 - Set image 8 - Check bad-block table 9 - Get SCET before load 10 - Init sysflash 11 - Load image 12 - Compute load time 13 - Verify checksum 14 - Handover to boot image 0x0E thus indicates boot successful 0x06 indicates an error occurred during read of the bad block table
1	UINT8	The SW image in error (0 to 5)

RMAP reply status (if a reply is requested):

Table 17.49 - HKLastBootStatus status codes

Status code	Description
0	Success.

17.7.23. HKDeathReports

Gets context of up to 5 anomalous events (A to E) that have led to an unhandled exception. Refer to Chapter 23 for more information on death reports.

Table 17.50 - HKDeathReports data

Byte	Type	Description	Trap category
0	UINT32	Number of death reports currently in table	-
4	UINT32	A: SCET Seconds	All
8	UINT32	A: SCET Subseconds	All
12	UINT32	A: Processor Status Register (PSR)	All

Byte	Type	Description	Trap category
16	UINT32	A: Trap Type	All
20	UINT32	A: Program Counter (PC)	Direct
24	UINT32	A: next Program Counter (nPC)	Direct
28	UINT32	A: Stack Pointer	Direct
32	UINT32	A: FPU Control/Status Register (FSR)	Floating point
36	UINT32	A: Instruction address (Deferred traps)	Floating point
40	UINT32	A: Instruction code (Deferred traps)	Floating point
44	UINT32	B: SCET Seconds	All
48	UINT32	B: SCET Subseconds	All
52	UINT32	B: Processor Status Register (PSR)	All
56	UINT32	B: Trap Type	All
60	UINT32	B: Program Counter (PC)	Direct
64	UINT32	B: next Program Counter (nPC)	Direct
68	UINT32	B: Stack Pointer	Direct
72	UINT32	B: FPU Control/Status Register (FSR)	Floating point
76	UINT32	B: Instruction address	Floating point
80	UINT32	B: Instruction code	Floating point
84	UINT32	C: SCET Seconds	All
88	UINT32	C: SCET Subseconds	All
92	UINT32	C: Processor Status Register (PSR)	All
96	UINT32	C: Trap Type	All
100	UINT32	C: Program Counter (PC)	Direct
104	UINT32	C: next Program Counter (nPC)	Direct
108	UINT32	C: Stack Pointer	Direct
112	UINT32	C: FPU Control/Status Register (FSR)	Floating point
116	UINT32	C: Instruction address	Floating point
120	UINT32	C: Instruction code	Floating point
124	UINT32	D: SCET Seconds	All
128	UINT32	D: SCET Subseconds	All
132	UINT32	D: Processor Status Register (PSR)	All

Byte	Type	Description	Trap category
136	UINT32	D: Trap Type	All
140	UINT32	D: Program Counter (PC)	Direct
144	UINT32	D: next Program Counter (nPC)	Direct
148	UINT32	D: Stack Pointer	Direct
152	UINT32	D: FPU Control/Status Register (FSR)	Floating point
156	UINT32	D: Instruction address	Floating point
160	UINT32	D: Instruction code	Floating point
164	UINT32	E: SCET Seconds	All
158	UINT32	E: SCET Subseconds	All
162	UINT32	E: Processor Status Register (PSR)	All
166	UINT32	E: Trap Type	All
170	UINT32	E: Program Counter (PC)	Direct
174	UINT32	E: next Program Counter (nPC)	Direct
178	UINT32	E: Stack Pointer	Direct
182	UINT32	E: FPU Control/Status Register (FSR)	Floating point
186	UINT32	E: Instruction address	Floating point
200	UINT32	E: Instruction code	Floating point

RMAP reply status (if a reply is requested):

Table 17.51 - HKDeathReports status codes

Status code	Description
0	Success.
EINVAL	The driver for the HK device has not been initialized
EIO	I/O error. The HK device cannot be accessed

17.7.24. HKClearDeathReports

Clears the stored death reports. Refer to Chapter 23 for more information on death reports.

Table 17.52 - HKClearDeathReports data

Byte	Type	Description
0	UINT8	0x01 - Clear death reports

RMAP reply status (if a reply is requested):

Table 17.53 - HKClearDeathReports status codes

Status code	Description
0	Success.
EINVAL	The driver for the HK device has not been initialized or invalid argument length or the argument is out of range
EIO	I/O error. The HK device cannot be accessed

17.7.25. HKCpuUsage

Gets the CPU usage for all tasks on the TCM. The returned data consists of a header and dynamic size table for all the tasks. The information about the table size is in the header. The header size is always 12 bytes long, meaning that the data table starts at the 13th byte.

Table 17.54 - HKCpuUsage header

Byte	Type	Description
0	UINT8	Number of tasks
1:3	UINT8	Padding
4	UINT32	Total number of seconds
8	UINT32	Total number of nanoseconds

Table 17.55 - HKCpuUsage data per task

Byte	Type	Description
0	UINT32	Task id
4:7	CHAR	Task name in Ascii format
8	UINT32	Seconds
12	UINT32	Nanoseconds
16	UINT8	Percent
17:19	UINT8	Padding
20	UINT32	Percent fraction, one thousandth of a percent.

RMAP reply status (if a reply is requested):

Table 17.56 - HKCpuUsage status codes

Status code	Description
0	Success.
ENFILE	Too many tasks, not all tasks are reported

17.7.26. HKCpuUsageReset

Resets the measured CPU time on the TCM. This command does not require or provide any data.

RMAP reply status (if a reply is requested):

Table 17.57 - HKCpuUsageReset status codes

Status code	Description
0	Success.

17.7.27. TimesyncConfig

Read or write the time synchronisation configuration.

If external synchronisation was previously enabled, writing a new configuration will result in a reset of the qualification and synchronisation procedure. If external synchronisation is enabled in the new configuration, re-qualification and re-synchronisation will occur using the new configuration.

If writing a new configuration where external synchronisation is disabled, the new configuration will take effect immediately. If writing a new configuration where external synchronisation is enabled, the new configuration will take effect after the next PPS event (expired, unqualified, or qualified).

For details regarding time synchronisation parameters, see Section 9.1.

Table 17.58 - TimesyncConfig data

Byte	Type	Description
0	UINT32	Option bit flags where:+ Bit 31:1 (MSB) - Reserved Bit 0 (LSB) - External synchronisation 0 - disabled 1 - enabled
4	UINT32	Consecutive qualified PPS count
8	UINT16	PPS qualification threshold

Byte	Type	Description
10	UINT8	Reserved
11	UINT8	Reserved

RMAP reply status (if a reply is requested):

Table 17.59 - TimesyncConfig status codes

Status code	Description
0	Success.
EINVAL	Invalid command length or invalid flags parameter value.

17.7.28. UARTCommand

Send a command on the specified UART interface. See also Section 6.1.3 for the UART configuration.

Table 17.60 - UARTCommand data

Byte	Type	Description
0 - nn	Array of UINT8	UART command data

RMAP reply status (if a reply is requested):

Table 17.61 - UARTCommand status codes

Status code	Description
0	Success.
ENODEV	This UART device has not been configured/initialized.
EINVAL	The value for the UART device is invalid.
EIO	I/O error. The UART device cannot be accessed

17.7.29. MMData

Reads or writes data from/to a partition.

17.7.29.1. Read

The address given in the RMAP command defines the starting byte address of the read and the RMAP data size determines the length of the read in bytes.

If no data is available at the starting address an error will be reported. If less than the requested data is available, a short read will be returned with an RMAP error status indication. If read errors occur based on uncorrectable read errors, the data will be returned along with an RMAP error status indication.

Reads which pass the end of the partition logical address space will automatically wrap.

17.7.29.2. Write

Writes to direct partitions needs to specify the starting address and the size via the RMAP address and RMAP data size, the size needs to be a multiple of the page size (16 Kbytes for 16 GB mass memory, or 32 Kbytes for 32 GB mass memory). If the write would overwrite existing data or write at an invalid location, an RMAP error status will be reported and no data will be written.

Writes to continuous or circular partitions needs to specify the size via the RMAP data size and must indicate use of the write pointer by setting the address to 0.

Writes which pass the end of the partition logical address space will automatically wrap.

For direct and continuous partitions, if bad blocks occur during a write which causes available blocks to run out, the remainder of the write will be discarded, and a pending copy operation will be set. In order to avoid data loss, freeing of enough data in order to provide two new unused blocks should be performed as soon as possible, which will allow the copy operation to be retried. Confirmation of the success of the copy operation should be done by verifying that the available space is equal to one block, otherwise the freeing and copy success confirmation procedure should be repeated. For circular partitions, the copy retrying is taken care of automatically.

The amount of data that was written and the amount of data that was discarded in case of a write causing available blocks to run out on direct or continuous partitions can be found by examining the data ranges.

Writing to a circular mode partition that is being downloaded is not allowed.

Writing to a TC storage partition via RMAP is not allowed.

17.7.29.3. Command and reply format

The data field of the read/write RMAP message in Table 17.62 contains raw data written to or read from the partition.

Table 17.62 - MMData data

Byte	Type	Description
0 - nn	Array of UINT8	Data

RMAP reply status (if a reply is requested):

Table 17.63 - MMData data status codes

Status code	Description
0	Success.
ENOSPC	Write: Not enough space on partition (may have been caused by bad blocks, see suggested handling above). Read: Not enough data on partition. **Note!** *It's allowed to ask for more read data than is available on the partition. Available data will be returned (stating the length in the RMAP reply packet) together with this error code.*
EINVAL	Invalid partition number, or Attempt to write partial page to direct mode partition, or Address is not 0 when writing to continuous or circular partition, or Length is greater than INT32_MAX, or Length is greater than segment size when writing to an auto-padded partition.
EIO	Uncorrectable errors when reading, data in reply contains errors.
EEXIST	Write operation to direct mode partition would overwrite existing data.
EALREADY	Write to circular partition that is being downloaded.
ENOTSUP	Write not allowed for TC storage type partition.

17.7.30. MMDataRange

This command will return all data address ranges where data is written in this partition, see Table 17.64. The range information should be interpreted differently for different partition modes.

Continuous and circular mode - Only one range will be reported, corresponding to the free and write pointers. Empty and full partitions will show the free and write pointers having the same value, use the MMPartitionSpace command to get size status.

Direct mode - This is a collection of ranges. Empty partitions will return an empty range table (RMAP reply data of length 0). The ranges will represent the start and end of each continuous data segment in the partition.

Ranges will not exactly match the currently unavailable space due to partially freed (but not yet erased) blocks.

The start address of the range is inclusive, the end address of the range is exclusive.

Table 17.64 - MMDataRange data

Byte	Type	Description
0-3	UINT32	Start address of first data range.
4-7	UINT32	End address of first data range (exclusive).
8-11	*UINT32*	*Start address of second data range (optional).*
12-15	*UINT32*	*End address of second data range (exclusive) (optional).*
. . .	. . .	. . .

RMAP reply status:

Table 17.65 - **MMDataRange** status codes

Status code	Description
0	Success.
EINVAL	Invalid partition number.

17.7.31. MMPartitionConfig

Reads the current partition configuration (see Section 11.2), the RMAP reply message data format is described in Table 17.66.

The available blocks in the flash mass memory ranges from 0 to 8191.

Table 17.66 - MMPartitionConfig data

Byte	Type	Description
0	UINT32	Starting block number of the partition.
4	UINT32	Ending block number of the partition (inclusive).
8	UINT8	Partition mode. 0 - Direct 1 - Continuous 2 - Circular 3 - Auto-padded Continuous 4 - Auto-padded Circular

Byte	Type	Description
9	UINT8	Specifies type of data stored on the partition. 0 - Space Packets 1 - Raw Data (not supported for download) 2 - TC storage
10	UINT8	Specifies which virtual channel to be used for downloading of the data in the partition. See [RD 7] for VC allocation.
11	UINT8	Segment size for the partition. 1 - 16 kbyte 2 - 32 kbyte 3 - N/A 4 - 64 kbyte
12	UINT32	The data source identifier for the partition. Can be used to set a custom identifier of a data producer to a partition. Setting of this value is not required to successfully configure a partition.

RMAP reply status:

Table 17.67 - MMPartitionConfig data status codes

Status code	Description
0	Success.
EINVAL	Invalid partition number.

17.7.32. MMPartitionSpace

Gets the amount of free space in a partition.

Note that due to the nature of the flash memory, as memory is freed, the space will become free for writing only in leaps as the free operation is used up to a block boundary. This means that a partition can have a discrepancy between reported free space and expected free space of maximum one block.

The reported space for direct partitions will correspond to the total space of every available unused page, minus any freed bytes which belongs to a block which has not yet been fully freed.

The reported space for continuous and circular partitions will correspond to the total space of every unused byte, minus the data offset in the initial write block.

For continuous/circular partitions, since the write pointer is never reset it may not be located at the beginning of a block when the initial write occurs or is about to occur, hence the amount of free space may not correspond exactly to the amount of available fully freed blocks. It is possible (but not recommended during normal operation) to re-synchronize the write pointer by writing exactly the amount needed to end up at the start of a block, and then erase up to the write pointer. This will cause the free space to be exactly equal to the amount of available blocks (or the partition maximum logical address space limit).

Table 17.68 - MMPartitionSpace data

Byte	Type	Description
0-7	UINT64	Available size in bytes.

RMAP reply status:

Table 17.69 - MMPartitionSpace status codes

Status code	Description
0	Success.
EINVAL	Invalid partition number.

17.7.33. MMDownloadPartitionData

Downloads data of the requested length from the partition using the virtual channel set in the partition configuration (see Section 11.2.5). Download commands will be processed one at a time and any prioritizations between different partitions must be handled by sending the download commands in priority order. For direct mode, all download data need to be in a continuous address area (i.e. same data range) or the download will stop when reaching the end of a continuous area even though the download ordered is larger.

In case an invalid Space packet length is encountered, or a Space CRC error occurs in a memory segment during download, the rest of the segment will be downloaded with packet errors and the download will re-synchronise at the start of the next segment.

If a download is started at the end of a partition that is simultaneously written to and the amount of data is beyond the current content of the partition from that point, the download will download only the data available at the time that the download command is issued, regardless of the data written to the partition during download.

Data will normally be downloaded in chunks equal to the segment size set for the partition. It's possible to start and end a download on an uneven segment boundary, but then it's the responsibility of the user to make sure it starts and ends on even CCSDS space packet boundaries. See also information in Section 11.2.3 on padding of data.

A download will not automatically free any data.

This command is not allowed on TC storage partitions.

The RMAP write command data format is described in Table 17.70.

Table 17.70 - MMDownloadPartitionData data.

Byte	Type	Description
0-3	UINT32	Address of the data to download
4-11	UINT64	Length in bytes to download

The RMAP reply status (if a reply is requested) will be the first error encountered during a single segment download, i.e. all segment downloads must be sent without fault for Success to be returned.

Table 17.71 - MMDownloadPartitionData data status codes

Status code	Description
0	Success.
ENOSPC	Not enough data on partition. **Note!** *It's allowed to request download of more data than is available on the partition. This error code will then be returned and to see the actual amount of data downloaded, use the MMDownloadStatus command.*
EINVAL	Invalid partition number, or invalid argument length, or length in bytes to download is zero or larger than INT64_MAX
EIO	I/O error. Failed to access storage or NVRAM.
EALREADY	A download session is already in progress on this partition.
EBADMSG	Data was not successfully downloaded on downlink.
ENOTSUP	Download not allowed for TC storage type partition.

17.7.34. MMFree

Frees memory of a partition. The MMFree operation behaves differently depending on the mode of the partition targeted.

Direct mode - The address and length given in the RMAP command together defines which memory area should be freed.

Continuous and circular mode - The free pointer position together with the length given in the RMAP command defines which memory area should be freed and the address field is ignored. This operation will also move the free pointer forward.

Trying to free more memory than is available is a valid use case and can for example be used to empty a partition by issuing an MMFree call with the maximum partition length.

If a free to a direct partition starts inside used data and not at a block boundary, the operation will free nothing and an RMAP error status will be reported, since such a free could create an illegal address gap. Freeing the whole partition is a special case and still allowed from any starting address.

Frees which pass the end of the partition logical address space will automatically wrap.

Frees may start at unused addresses.

See also Chapter 11 for an illustration of how free affects the actual amount of memory free for writes.

Note that MMFree on a partition where a download is in progress is not allowed.

This command is not allowed on TC storage partitions.

The RMAP write command data format is described in Table 17.72.

Table 17.72 - MMFree data

Byte	Type	Description
0-3	UINT32	Address of memory to free.
4-11	UINT64	Length of memory to free in bytes.

RMAP reply status (if a reply is requested):

Table 17.73 - MMFree status codes

Status code	Description
0	Success.
EINVAL	Invalid partition number, or invalid argument length, or address is not 0 for continuous/circular partition.
EEXIST	Operation could create illegal address gap inside block.

Status code	Description
EALREADY	A download is in progress on this partition.
ENOTSUP	Freeing not allowed for TC storage type partition.

17.7.35. MMDownloadStatus

Returns the amount of data downloaded for this partition during the last completed download.

This command is not allowed on TC storage partitions.

Table 17.74 - MMDownloadStatus data

Byte	Type	Description
0-7	UINT64	Number of bytes downloaded.

RMAP reply status:

Table 17.75 - MMDownloadStatus status codes

Status code	Description
0	Success.
EINVAL	Invalid partition number.
EIO	I/O error. Failed to access storage or NVRAM.
ENOTSUP	Download not allowed for TC storage type partition.

17.7.36. MMStopDownloadData

This command can be sent to stop a current download for a partition previously started by the MMDownloadPartitionData command.

This command is not supported on TC storage partitions.

RMAP reply status (if a reply is requested):

Table 17.76 - MMStopDownload status codes

Status code	Description
0	Success.
EINVAL	Invalid partition number.
ENOTSUP	Download not allowed for TC storage type partition.

17.7.37. MMGetPageSize

This command reads out the available page size and block size of the mass memory.

Table 17.77 - MMGetPageSize data

Byte	Type	Description
0	UINT8	Page size in bytes. 0x00 - 16 * 1024 bytes 0x01 - 32 * 1024 bytes
1	UINT8	Block size in bytes. 0x00 - 2 * 1024 * 1024 bytes 0x01 - 4 * 1024 * 1024 bytes

RMAP reply status:

Table 17.78 - MMGetPageSize status codes

Status code	Description
0	Success.

17.7.38. MMTCStorageStatus

Reads the current TC storage status information in the format described in Table 17.79.

Table 17.79 - TC Storage status information RMAP address details

Byte	Type	Description
0	UINT8	Bit 7:2 (MSB) - Reserved Bit 1 - Flag indicating if the number of rejected data chunk writes due to storage being full has reached 2^{32} and wrapped since last TCM reset (0 - has not wrapped, 1 - has wrapped). Bit 0 (LSB) - Flag indicating if a TC storage partition is configured (0 - is not configured, 1 - is configured).
1	UINT8	Partition index of TC storage partition.
2 - 3	N/A	Reserved padding.
4 - 7	UINT32	Number of stored data chunks.

Byte	Type	Description
8 - 11	UINT32	Number of rejected data chunk writes due to storage being full since last TCM reset.

If the byte 0 - bit 0 flag is not set, indicating that a TC storage partition is not configured, the rest of the status information is invalid/unspecified.

It is not possible to read partial data via this command; the read address must be the base address without any byte offset and the whole status information will be read regardless of the read size specified.

The RMAP reply status for reads via this command can be any of the values described in Table 17.80.

Table 17.80 - TC storage status information RMAP read reply status

Status code	Description
0	Success.

17.7.39. MMTCStorageClear

Clear all data and the stored data chunk count in the TC storage, the accompanying write data must use the format described in Table 17.81.

Table 17.81 - TC storage clear initiation RMAP write format

Bytes	Type	Description
0 - 3	UINT32	Range start address of data on partition.
4 - 7	UINT32	Range end address of data on partition (exclusive).

The clear will be rejected if the range does not match the range of data on the TC storage partition at the point when the clear execution is started. This means that if a new write to the TC storage has occurred, the clear will be rejected, ensuring that it is not possible to silently loose data chunks.

If the clear is accepted, all stored data chunks will be discarded.

The intended use is to first read the current TC storage partition range information via the MMDataRange command, ensure that the range information does not indicate any new data chunks which should not be cleared, and then use this range when sending the clear command.

Clearing can only clear the whole TC storage; no partial clearing is supported.

Clearing does not clear the rejected data chunks count nor the rejected data chunks count wrap flag, these items are only cleared on a TCM reset.

The RMAP reply status for writes via this command can be any of the values described in Table 17.82.

Table 17.82 - TC storage clear RMAP write reply status

Status code	Description
0	Success.
19 (ENODEV)	Rejected due to no TC storage being configured.
22 (EINVAL)	Rejected due to size of write data not being 8 bytes.
133 (ESTALE)	Rejected due to range not matching current range of data on partition.

17.7.40. MMBadBlockCount

Reads the current number of bad blocks in the Mass Memory.

Table 17.83 - MMBadBlockCount Data

Byte	Type	Description
0	UINT16	Number of Bad Blocks in the Mass Memory

RMAP reply status (if a reply is requested):

Table 17.84 - MMBadBlockCount status codes

Status code	Description
0	Success.

17.7.41. MMCombinedDataRange

Reads the address ranges for all partitions in the format described in Table 17.85.

Each configured partition will be represented by a single range in the reply. The size of the reply will vary based on the number of configured partitions.

For each continuous and circular partitions the representation is identical to MMDataRange, corresponding to all data on the partition.

For direct mode partitions the representation differs compared to MMDataRange in the following ways:

- Empty direct partitions will be represented with a single [0, 0] range.

- Direct partitions with more than one address range will be represented with a combined single range from the start address of the first range to the end address of the last range in the partition (the ordering of ranges is based on their address).

Table 17.85 - MMCombinedDataRange data

Byte	Type	Description
0-3	UINT32	Start address for partition 0.
4-7	UINT32	End address (exclusive) for partition 0.
8-11	UINT32	Start address for partition 1.
12-15	UINT32	End address (exclusive) for partition 1.
…	…	…

17.7.42. MMCombinedConfig

Reads the configuration for all partitions in the format described in Table 17.86.

The size of the reply will vary based on the number of configured partitions.

Table 17.86 - MMCombinedConfig data

Byte	Type	Description
0	UINT32	Starting block number for partition 0.
4	UINT32	Ending block number for partition 0 (inclusive).
8	UINT8	Mode for partition 0. 0 - Direct 1 - Continuous 2 - Circular 3 - Auto-padded Continuous 4 - Auto-padded Circular
9	UINT8	Type of data for partition 0. 0 - Space Packets 1 - Raw Data (not supported for download) 2 - TC storage
10	UINT8	Virtual channel used for downloading data for partition 0. See [RD7] for VC allocation.
11	UINT8	Segment size for partition 0. 1 - 16 kbyte 2 - 32 kbyte 3 - N/A 4 - 64 kbyte

Byte	Type	Description
12	UINT32	Data source identifier for partition 0. Can be used to set a custom identifier of a data producer to a partition. Setting of this value is not required to successfully configure a partition.
16	UINT32	Starting block number for partition 1.
20	UINT32	Ending block number for partition 1 (inclusive).
24	UINT8	Mode for partition 1. 0 - Direct 1 - Continuous 2 - Circular 3 - Auto-padded Continuous 4 - Auto-padded Circular
25	UINT8	Type of data for partition 1. 0 - Space Packets 1 - Raw Data (not supported for download) 2 - TC storage
26	UINT8	Virtual channel used for downloading data for partition 1. See [RD7] for VC allocation.
27	UINT8	Segment size for partition 1. 1 - 16 kbyte 2 - 32 kbyte 3 - N/A 4 - 64 kbyte
28	UINT32	Data source identifier for partition 1. Can be used to set a custom identifier of a data producer to a partition. Setting of this value is not required to successfully configure a partition.
…	…	…

17.7.43. MMCombinedSpace

Reads the amount of free space for each partition in the format described in Table 17.87.

See MMPartitionSpace for details on the concept of "free space".

The size of the reply will vary based on the number of configured partitions.

Table 17.87 - MMCombinedSpace data

Byte	Type	Description
0-7	UINT64	Number of bytes of free space for partition 0.
8-15	UINT64	Number of bytes of free space for partition 1.
...	...	...

17.7.44. MMCombinedDownloadStatus

Reads the amount of data downloaded during the last completed download for each partition in the format described in Table 17.88.

For a TC storage data type partition (which do not support downloads), the value will be undefined data.

The size of the reply will vary based on the number of configured partitions.

Table 17.88 - MMCombinedDownloadStatus data

Byte	Type	Description
0-7	UINT64	Number of bytes downloaded for partition 0.
8-15	UINT64	Number of bytes downloaded for partition 1.
...	...	...

17.7.45. SpwBackupRoutingEnableDisableSet

Enables/disables backup SpW routing.

Table 17.89 - SpwBackupRoutingEnableDisableSet data

Byte	Type	Description
0	UINT8	0x00 - Disabled 0x01 - Enabled

RMAP reply status (if a reply is requested):

Table 17.90 - SpwBackupRoutingEnableDisableSet reply status codes

Status code	Description
0	Success.
EINVAL	The argument is out of bounds
EIO	Internal RTEMS error

17.7.46. SpwBackupRoutingEnableDisableGet

Reads out the current enable/disable configuration.

Table 17.91 - SpwBackupRoutingEnableDisableGet data

Byte	Type	Description
0	UINT8	0x00 - Disabled 0x01 - Enabled

RMAP reply status (if a reply is requested):

Table 17.92 - SpwBackupRoutingEnableDisableGet reply status codes

Status code	Description
0	Success.
EINVAL	The argument is out of bounds
EIO	Internal RTEMS error

17.7.47. SpwRoutingPathSet

Configures the SpW paths. The maximum size of a path is 8 bytes, and the maximum number of paths is 20. The logic address of the receiving node must be included. It is allowed to send less data than 160 byte, but if the user tries to specify fewer paths than the highest SpW path index configured in nvram, the command will be rejected and EINVAL will be set. The length of the data must be a multiple of 8 bytes, otherwise the command will be rejected and EINVAL will be set.

NOTE All SpW paths must contain a terminating null character, otherwise the command will be rejected and EINVAL will be set.

Table 17.93 - SpwRoutingSet data

Byte	Type	Description
0 - 7	Array of UINT8	SpW path 0.
8 - 15	Array of UINT8	SpW path 1.
…	…	…
152 -159	Array of UINT8	SpW path 19.

RMAP reply status (if a reply is requested):

Table 17.94 - SpwRoutingPathSet reply status codes

Status code	Description
0	Success.
EINVAL	Invalid argument
EIO	Internal RTEMS error

17.7.48. SpwRoutingPathGet

Reads out the current SpW paths. The size of a path is 8 bytes, and the maximum number of paths is 20. If a reply is requested, the size of the data returned will always be 160 bytes.

Table 17.95 - SpwRoutingPathGet data

Byte	Type	Description
0 - 7	Array of UINT8	SpW path 0.
8 - 15	Array of UINT8	SpW path 1.
…	…	…
152 - 159	Array of UINT8	SpW path 19.

RMAP reply status (if a reply is requested):

Table 17.96 - SpwRoutingPathGet reply status codes

Status code	Description
0	Success.
EIO	Internal RTEMS error

17.7.49. SpwReplyPathSet

Configures the SpW write-reply paths. The size of a path is 8 bytes, and the maximum number of paths is 20. The logic address of the receiving node must be included. It is allowed to send less data than 160 byte, but if the user tries to specify fewer paths than the highest SpW path index configured in nvram, the command will be rejected and EINVAL will be set. The length of the data must be a multiple of 8 bytes, otherwise the command will be rejected and EINVAL will be set.

NOTE	All SpW paths must contain a terminating null character, otherwise the command will be rejected and EINVAL will be set.

Table 17.97 - SpwReplyPathSet Data

Byte	Type	Description
0 - 7	Array of UINT8	SpW write-reply path 0.
8 - 15	Array of UINT8	SpW write-reply path 1.
...	...	...
152 - 159	Array of UINT8	SpW write-reply path 19.

RMAP reply status (if a reply is requested):

Table 17.98 - SpwReplyPathSet Reply Status Codes

Status code	Description
0	Success.
EINVAL	Invalid argument
EIO	Internal RTEMS error

17.7.50. SpwReplyPathGet

Reads out the current SpW write-reply paths. The maximum size of a path is 8 bytes, and the maximum number of paths is 20. If a reply is requested, the size of the data returned will always be 160 bytes.

Table 17.99 - SpwReplyPathGet Data

Byte	Type	Description
0 - 7	Array of UINT8	SpW write-reply path 0.
8 - 15	Array of UINT8	SpW write-reply path 1.
...	...	...
152 - 159	Array of UINT8	SpW write-reply path 19.

RMAP reply status (if a reply is requested):

Table 17.100 - SpwReplyPathGet Reply Status Codes

Status code	Description
0	Success.
EIO	Internal RTEMS error

17.7.51. SpwBackupRoutingTimeoutSet

Configures the maximum amount of time the TCM SW will wait for a write-reply from an SpW node. If SpW backup routing is enabled, and an RMAP command has been sent from the TCM SW to a SpW node, and the write-reply does not arrive to the TCM SW before the timeout, the TCM will switch to SpW backup routing and try to send this packet once again.

NOTE	Since the granularity of the system is 10 ms, values not divisible by 10 ms will be truncated to the nearest multiple if 10 ms. Setting a timeout less than 10 ms will result in a timeout of 0 ms.

Table 17.101 - SpwBackupRoutingTimeoutSet data

Byte	Type	Description
0 - 1	UINT16	The timeout in milliseconds, max value 65535.

RMAP reply status (if a reply is requested):

Table 17.102 - SpwBackupRoutingTImoutSet reply status codes

Status code	Description
0	Success.
EINVAL	The argument is out of bounds.
EIO	Internal RTEMS error

17.7.52. SpwBackupRoutingTimeoutGet

Reads out the maximum amount of time the TCM SW will wait for a write-reply from an external SpW node.

Table 17.103 - SpwBackupRoutingTimeoutGet data

Byte	Type	Description
0 - 1	UINT16	The timeout in milliseconds, max value 65535.

RMAP reply status (if a reply is requested):

Table 17.104 - SpwBackupRoutingTImoutGet reply status codes

Status code	Description
0	Success.
EIO	Internal RTEMS error

17.7.53. RIRPTransactionStatus

Read the status of ongoing, finished, and timed out commands from the transaction status buffer.

The RIRPTransactionStatus command will return data in the format described in Table 17.105.

Table 17.105 - RIRPTransactionStatus Data

Byte	Type	Description
0 - 3	UINT32	Number of transaction status entries in buffer.
4 - 7	UINT32	Transaction buffer full status. 0x00 - Buffer not full 0x01 - Buffer full
8 - 11	-	Transaction status entry for first command.
12 - 15	-	Transaction status for second command.
..		
NN - (NN+3)	-	Transaction status for last command.

The format of each transaction status entry is described in Table 17.106.

Table 17.106 - RIRPTransactionStatus transaction status entry data

Byte	Type	Description
0 - 1	UINT16	Transaction ID
2	UINT8	Operation state: 0x00 - Ongoing 0x01 - Timed out 0x02 - Finished
3	UINT8	Execution status for finished commands. Will contain the same status as non-RIRP replies.

Reading the transaction status entry of a finished or timed out command fully will clear it from the transaction status buffer. When one or more transaction status entries are cleared, the remaining transaction status entries will be shifted towards the beginning of the buffer to remove any gaps.

17.7.54. GPIOGetConfig

Gets the configuration of the addressed GPIO.

Table 17.107 - GPIOGetConfig data

Byte	Type	Description
0	UINT8	Direction: 0 - Output 1 - Input
1	UINT8	Mode: 0 - Single ended 1 - Differential

RMAP reply status (if a reply is requested):

Table 17.108 - GPIOGetConfig status codes

Status code	Description
0	Success.
ENOSPC	The addressed GPIO does not exist/is not configured
EIO	I/O error. The GPIO device cannot be accessed

17.7.55. GPIOSetConfig

Sets the configuration of the addressed GPIO. Differential mode means that a pair of pins is used together for a differential output signal. The pins are paired in sequence, so [0|1], [2|3] and so on, and each pair is controlled by setting the lower numbered pin (i.e. if pin 0 is set to differential output, pin 1 will automatically be set to match).

IMPORTANT

An RMAP command to change configuration for a lower numbered pin has no effect on the higher numbered pin when both pins are in differential mode. As differential mode is only valid for output, a reply with status code EINVAL will be sent to the initiator if Direction is set to input and Mode to differential.

NOTE

If a pin pair that shares the same value enters differential mode, the pins will keep their initial values until the lower pin is explicitly set.

Table 17.109 - GPIOSetConfig data

Byte	Type	Description
0	UINT8	Direction: 0 - Output 1 - Input

Byte	Type	Description
1	UINT8	Mode: 0 - Single ended 1 - Differential (Note: Differential mode is only valid for output pins)

RMAP reply status (if a reply is requested):

Table 17.110 - GPIOSetConfig status codes

Status code	Description
0	Success.
ENOSPC	The addressed GPIO does not exist/is not configured
EINVAL	Invalid argument length or invalid value or combination of values in configuration
EIO	I/O error. The GPIO device cannot be accessed

17.7.56. GPIOGetValue

Gets the value of the addressed GPIO. Reading out the value of the higher numbered pin of a differential pair will show the actual value of that pin.

Table 17.111 - GPIOGetValue data

Byte	Type	Description
0	UINT8	Value 0 - Pin is low 1 - Pin is high

RMAP reply status (if a reply is requested):

Table 17.112 - GPIOGetValue status codes

Status code	Description
0	Success.
ENOSPC	The addressed GPIO does not exist/is not configured
EIO	I/O error. The GPIO device cannot be accessed

17.7.57. GPIOSetValue

Sets the value of the addressed GPIO. In a differential pair it is only valid to set the value of the lower numbered pin.

Table 17.113 - GPIOSetValue data

Byte	Type	Description
0	UINT8	Value 0 - Set pin low 1 - Set pin high

RMAP reply status (if a reply is requested):

Table 17.114 - GPIOSetValue status codes

Status code	Description
0	Success.
ENOSPC	The addressed GPIO does not exist/is not configured
EINVAL	Invalid argument length **or** Invalid value **or** Trying to set the higher numbered pin in a differential pair
EIO	I/O error. The GPIO device cannot be accessed

17.8. RMAP output address details

17.8.1. TCCommand

A CCSDS space packet that is being routed via SpaceWire according to its APID, see Table 6.10 for details regarding the APID routing configuration.

17.8.2. UARTData

Routed data from UART. See also Section 6.1.2 for the configuration.

Table 17.115 - UARTData data

Byte	Type	Description
0 - nn	Array of UINT8	Data received on UART

18. Spacewire Backup Routing

The TCM provides a “Spacewire Backup Routing” service, this is a service that will resend a command packet over an alternative Spacewire path if the first transmission of the command packet fails.

When SBR is enabled and a command message is redistributed to a SpW node by the TCM SW, the SpW node must send a reply to the TCM SW if the command message was received properly. If the TCM SW has not received a valid reply within the user-specified time period, the TCM SW will switch to using the backup SpW path and try to send the packet once again. After that TCM SW will send command messages using the original routing path to avoid switching paths due to temporal errors. If a write-reply arrives after the user specified time period, or no matching timer is found, the write-reply will be ignored.

When the TCM SW is requesting a write-reply from an external SpW node, the TCM SW must provide the path for the write reply. Since it is not possible to determine the reply path from the corresponding backup path, the user must also provide one write-reply path for every defined SpW path.

Enabling SpW backup routing, setting the primary and backup SpW paths, setting SpW write-reply paths, and configuration of the duration of the timeout can be done by configuring NVRAM. This is described in Chapter 6. These parameters can also be configured or read out by sending RMAP commands to the TCM, the RMAP commands are described in Section 17.7.38 - Section 17.7.53.These RMAP commands make it possible to configure these parameters during flight, since a debugger must be connected to the TCM when configuring NVRAM.

Altering the SpW routing configuration via RMAP commands does not trigger any write actions to NVRAM, the RMAP commands only alter local copies of the NVRAM parameters in the TCM SW. Upon reboot, all SpW routing configurations (enable/disable, paths and timeout) set by RMAP commands are lost. After reboot, the TCM SW will use the SpW routing configurations and paths set by the NVRAM configuration.

$$timeout[s] \cdot packetrate\left[\frac{n}{s}\right] \leq \frac{maxbuffers}{2}[n]$$

The linear relationship in the equation above should be used as a rule of thumb when selecting write reply timeout to avoid running out of resources. A maximum of half the number of available buffers (internal buffers of the TCM SW, here used for holding the data contents of RMAP commands while waiting for a write reply), 64/2=32 buffers, should be allowed to be occupied waiting for timeouts or write replies. Different SpW-networks and different sizes of TC/uart packets require different minimum timeouts therefore care must be taken so that the timeout is set high enough for the packets to be sent properly.

It is allowed to update backup routing parameters via RMAP during ongoing SBR transactions, but updating the SpW reply paths, the enable/disable parameter or the timeout duration parameter will not affect already started transactions. For example if SBR is enabled and an RMAP command requesting a reply is sent to an external node, and SBR is disabled before the TCM SW has received a reply or the timer has timed out, then it is possible that the RMAP command will be resent on its backup path although SBR is disabled. If SBR is enabled and an RMAP command requesting a reply is sent to an external node, and the user updates the SpW routing paths before the TCM SW has received a reply or the timer has fired, the new updated paths will be used for the possible resend of that RMP command packet.

19. SpaceWire router

In both Sirius OBC and Sirius TCM products, a small router is integrated in the SoCs. The routers use path addressing (see [RD13]) and given the topology illustrated in Figure 19.1, the routing addressing can be easily calculated.

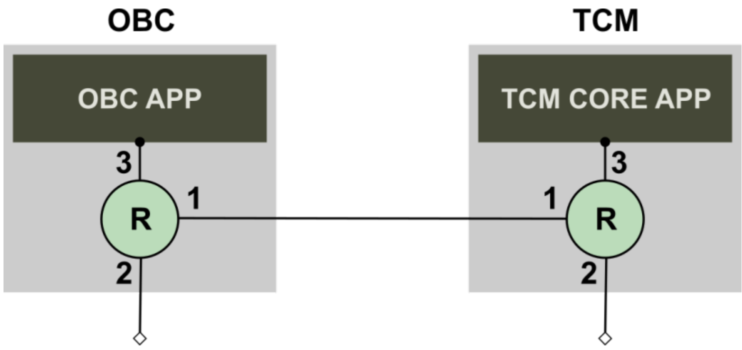

Figure 19.1 - Integrated router location

In the topology above, sending a package from the OBC to the TCM or vice versa, the routing address will be 1-3. Each end node, Sirius OBC or Sirius TCM, also has one or more logical address(es) to help distinguish between different applications or services running on the same node. The logical address complements the path address and must be included in a SpaceWire packet.

Example: If a packet is to be sent from Sirius OBC to the Sirius TCM it needs to be prepended with 0x01 0x03 XX:

- 0x01 routes the packet to port 1 of the Sirius OBC router.
- 0x03 routes the packet to port 3 of the Sirius TCM router.
- XX is the logical address (0x20 – 0xFE) of the recipient application/service on the Sirius TCM.

20. NVRAM areas

The system flash bad block table located at 0x0E00 – 0x11FF is used by the bootrom, the Software upload library and nandflash program.

The TCM SW configuration described in Chapter 6 is stored in two copies, one in the safe area for the safe SW images to use and one copy in the update area for the update images to use. The boot procedure is described further in Chapter 21. When configuring NVRAM with the nv_config library, EDAC mode (described further in the NVRAM section in [RD6]) is used. Therefore Table 20.1 lists addresses as how they are used when EDAC is enabled.

The mass memory bad block table is used by the TCM SW and it is updated during runtime when new bad blocks are discovered. The TCM SW has a reserved area for storing operation markers during runtime.

Table 20.1 - NVRAM Areas

Area	Area type	Board type	Range	Description
TCM SW Configuration	Safe	TCM	`0x0000 - 0x0DFF`	nv_config: Configuration parameters for TCM SW.
SF_BAD_BLOCKS	Safe	OBC and TCM	`0x0E00 - 0x0FFF`	Bad-block information for System Flash
SF_BAD_BLOCKS	Update	OBC and TCM	`0x1000 - 0x11FF`	Bad-block information for System Flash.
TCM SW Configuration	Update	TCM	`0x1200 - 0x1FFF`	nv_config: Configuration parameters for TCM-S SW.
MM_BAD_BLOCKS	Update	TCM	`0x2000 - 0x23FF`	Bad-block information for Mass Memory.
TCM SW Parameters	Update	TCM	`0x2400 - 0x25FF`	Reserved area for operation markers of the TCM SW.
Free space	Update		`0x2600 - 0x3FFF`	Currently unused area.

21. Boot procedure

21.1. Description

The bootrom is a small piece of software built into a read-only memory inside the SoC. Its main function is to load a software image from the system flash to RAM and start it by jumping to the reset vector. To make the system fault tolerant, there are two logical images of the main software, designated Updated and Safe. Each logical image is stored in three physical copies distributed over the system flash. By default, the bootrom will first try to load the Updated image and if that fails fall back to the Safe image. Boot order of the logical images and their physical copies is shown in Figure 21.1.

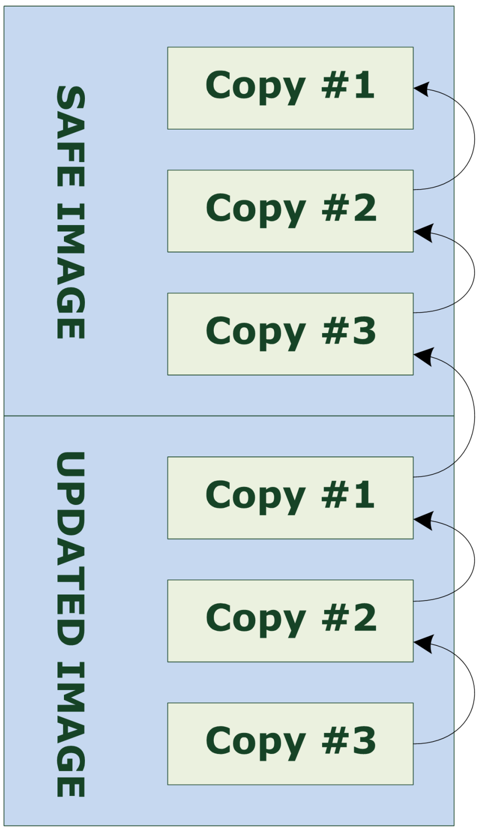

Figure 21.1 - Software images in flash

21.2. Usage description

The locations in the system flash where the bootrom looks for software images are given in Table 21.1. The first two 32-bit words of the image are expected to be a header with image size and an XOR checksum, see Table 21.2. If the size falls within the accepted range, the bootrom loads the image to RAM while verifying the checksum. Both the image size check and the checksum verification are performed in addition to the EDAC built into the System Flash. The System Flash EDAC is handled by hardware and calculates one extra byte of redundancy data for each true data byte written to flash.

The bootrom loads the system flash bad-block table from an NVRAM offset described in Table 20.1. If a flash block within the range to load from is marked as bad in the table, that block is assumed to have been skipped when the image was programmed, so the bootrom continues reading from the next block. If the image could be loaded from flash without error and its checksum is correct, the bootrom jumps to the reset vector in RAM. If there is a flash error when loading, if the checksum is incorrect, or if the image has an invalid size, the bootrom steps to the next image by changing the *Next FW* field in the Error Manager and doing a soft reset. If the image being loaded is the last available the bootrom will ignore errors and attempt to start it anyway, in order to always have a chance of a working system. To indicate to the software which image and copy is loaded, the *Running FW* field in the Error Manager is updated before handing over execution. The boot loader will also update the Error Manager Latest Boot Status register to indicate where it is in the boot process, so that more information can be retrieved in case of a failed boot, see Section 17.7.22.

Reading out that register in orbit requires a subsequent successful boot. Therefore, if multiple image copies fail to boot, the register information that is saved will be from the first failed attempt.

21.3. Limitations

If the image size is out of range for Safe image copy #1 (the final fallback image), the bootrom will not be able to load it and the fallback option of handing execution to a damaged software image if no other is available cannot be used.

Table 21.1 - Software image locations

Image number	Description	Flash page number
0x0	Updated copy #3	`0xC0000`
0x1	Updated copy #2	`0xA0000`
0x2	Updated copy #1	`0x80000`
0x3	Safe copy #3	`0x40000`
0x4	Safe copy #2	`0x20000`
0x5	Safe copy #1	`0x00000`

Table 21.2 - Software image header

Field	Size	Description
Image size	32 bits	The size in bytes of the software image, not including the header, stored as a 32-bit unsigned integer. A software image can be 264 Bytes - 63 MB.
Checksum	32 bits	A cumulative XOR of all 32-bit words in the image including the size, so that a cumulative XOR of the whole image and header (including checksum) shall evaluate to 0.

21.4. Cause of last reset

There is an RMAP command for reading out the cause of last reset from the TCM, see Section 17.7.21 for details.

21.5. Pulse commands

The pulse command inputs to the Sirius products can be used to force a board to reboot from a specific image. Paired with the ability of the Sirius TCM to decode PUS CPDU telecommands without software interaction and issue pulse commands, this provides a means to reset malfunctioning boards by direct telecommand from ground as a last resort.

Each board has two pulse command inputs. Input 0 resets the board and loads the updated image while input 1 resets the board and loads the safe image. Both require an active-high pulse length between 20 - 40 ms to be valid. If, for some reason, both pulse command inputs would be active at the same time, the pulse on input 0 takes precedence.

22. Software Upload

22.1. Description

During the lifetime of a satellite, the on-board software might need adjustments as bugs are detected or the mission parameters adjusted. This module tries to solve that by providing a means for updating the on-board software in orbit. The OBC and the TCM are both prepared for this functionality by having two software images, where writing to the first one requires the debugger to be connected, thus making only the second one available for updates in orbit.

The process for updating a flight software image is described below:

- The actual data transfer and commanding from earth performing the software upload needs to be compliant with the CCSDS standard for TC. In this description, it is assumed that the TCM is the initial recipient of TC, regardless of the end target. See Chapter 8.
- The TCM acts as a router in this case, routing the Telecommand to the intended target based on the APID and the TCM routing table (i.e. potentially to the TCM itself). See Table 6.10. Alternatively, the telecommands can be saved temporarily in TC Storage or in the TC queue for retrieval by an OBC.
- If the TCM is the intended target, then the software upload packets must comply to the PUS extension of the CCSDS standard (see [RD11]), and follow Section 22.2.
- All the individual telecommand frames, containing one data fragment each, need to be assembled into a full or partial image for update with verification.
- Finally, the actual update of the physical flash image can take place, where the uploaded image is written to the system flash.

The API described in the following chapter takes care of the last two steps.

The picture in Figure 22.1 shows the intended control flow when commanding the software update from ground.

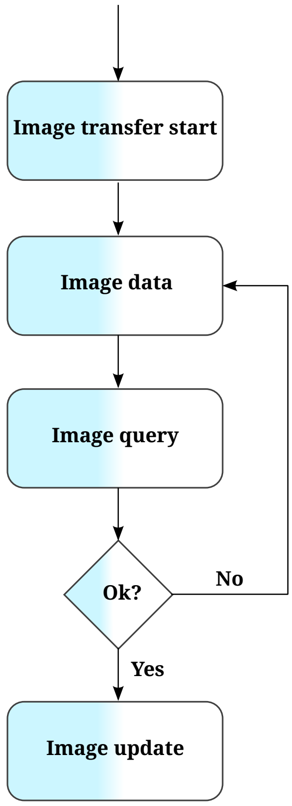

Figure 22.1 - The intended software upload command flow

22.2. CCSDS API – custom PUS service 130

22.2.1. Description

This service is provided to allow updates to the flight software on a node in a data handling system using Sirius components, but can be used for any type of on-board computer. The subtypes consist of a set of commands.

All service subtypes will report telecommand acceptance as PUS service [1,1] / [1,2] and telecommand execution complete as PUS services [1,7] / [1,8] (see [RD11]) if requested in the telecommand PUS header. See [RD7] for information on the allocated virtual channel for sending PUS reports. Recommended usage is to always request acceptance and execution complete reports so that the Ground Segment can keep track of the upload process.

All checksum parameters in the service are CRC32 with polynomial `0x04C11DB7` and seed value `0`.

The Telecommand Acceptance Report - Failure will use the standard error codes according to Table 22.1 without any parameters (see [RD11]).

Telecommand Execution Completed Report - Failure values are listed under each subtype heading. Errors noted as 'critical' will cause the whole software upload process to be aborted.

Table 22.1 - Telecommand acceptance failure error types

Error code	Data type	Error description
0	UINT8	Illegal APID (PAC error)
1	UINT8	Incomplete or invalid length packet
2	UINT8	Incorrect checksum
3	UINT8	Illegal packet type
4	UINT8	Illegal packet subtype
5	UINT8	Illegal or inconsistent application data
6	UINT8	Illegal PUS version

The numerical values of error codes returned in execution failure report are shown in Table 22.2 below.

Table 22.2 - Error code numerical values

Error code	Numeric value
ENOENT	2
EIO	5
EBUSY	16
EINVAL	22
ENOSPC	28
ENODATA	61
EALREADY	120

22.2.2. Subtype 1 – Image transfer start

A telecommand using this subtype must be sent first before sending any image data and will set up for a new image upload. It can also be used to abort an existing upload transaction during the data transfer phase, by simply initializing a new one. The data format is specified in Table 22.3 below.

Minimum image size is currently 272 bytes including header, and maximum image size is 16 Mbyte.

Table 22.3 - Image transfer start command data structure

Total number of bytes in image	Reserved (zero)	Reserved (zero)
UINT32	UINT32	UINT32

A telecommand execution complete report (if requested in the PUS header) will return the values listed in Table 22.4 in case of a failure.

Table 22.4 - Image transfer start telecommand execution failure codes

Error code	Data type	Error description
EINVAL	UINT8	Invalid image size
EBUSY	UINT8	Unable to open System Flash for writing or processing queue for requests is full.

22.2.3. Subtype 2 – Image data

This subtype transports data segments of the actual flight software image. Each segment can carry data with a maximum length of 900 bytes (to avoid splitting packets over several frames) and all segments except the last must have data of maximum length. The data in each segment is preceded by a 2-byte segment number and a 2-byte segment length, see Table 22.5 below.

Table 22.5 - Image data command structure

Segment number	Segment length	Segment data			
UINT16	UINT16	UINT8	UINT8	UINT8	...

A telecommand execution complete report (if requested in the PUS header) will return the values listed in Table 22.6 in case of a failure.

Table 22.6 - Image data telecommand execution failure codes

Error code	Data type	Error description
EALREADY	UINT8	This segment number has already been added.
EINVAL	UINT8	Segment number or segment length is out of bounds.
EIO	UINT8	Read/write error in intermediate storage area of flash (critical.)
ENOSPC	UINT8	Out of non-bad blocks in intermediate storage area of flash (critical.)
ENOENT	UINT8	No upload in progress
EBUSY	UINT8	Processing queue for requests is full.

22.2.4. Subtype 3 – Verify uploaded image

This subtype calculates and compares the checksum of the uploaded software image with the checksum set in the command's payload data, see Table 22.7.

Table 22.7 - Verify uploaded image argument

Checksum
UINT32

A telecommand execution complete report (if requested in the PUS header) will return the values listed in Table 22.8 in case of a failure.

Table 22.8 - Verify uploaded image telecommand execution failure codes

Error code	Data type	Error description
EINVAL	UINT8	Checksum argument doesn't match image checksum.
ENOENT	UINT8	No upload in progress.
ENODATA	UINT8	Segments missing.
EBUSY	UINT8	Processing queue for requests is full.

22.2.5. Subtype 4 – Write uploaded image

To do the updating of the flight image, this command is sent to the service provider which will then write the image to flash. To safeguard against accidental update commanding, a correct CRC is required as input argument for this command, see Table 22.9.

Table 22.9 - Write image command argument

Checksum
UINT32

A telecommand execution complete report (if requested in the PUS header) will return the values listed in Table 22.10 in case of a failure.

Table 22.10 - Write image telecommand execution failure codes

Error code	Data type	Error description
EINVAL	UINT8	Checksum argument doesn't match image checksum.
ENOSPC	UINT8	Out of non-bad blocks in flash(critical.)
ENOENT	UINT8	No upload in progress

Error code	Data type	Error description
EIO	UINT8	Read/write error in intermediate storage area of flash (critical.)
EBUSY	UINT8	Processing queue for requests is full.

22.2.6. Subtype 5 – Calculate CRC in flash

This command allows the CRC calculation of an image copy stored in flash. This can be used for extra verification after update of an image, or whenever the flight image copies need verification. The telecommand takes the image copy number as argument (max value 6), see Table 22.11. Image copy numbers 1 - 3 are for the (non-updateable) safe image and 4 - 6 cover the updated image copies.

Table 22.11 - Calculate CRC in flash command argument

Image copy number
UINT8

A telecommand execution complete report (if requested in the PUS header) will return the values listed in Table 22.12 in case of a failure.

Table 22.12 - Calculate flash CRC telecommand execution failure codes

Error code	Data type	Error description
EINVAL	UINT8	Image number too high (maximum 6).
EBUSY	UINT8	Unable to open System Flash device or processing queue for requests is full.
EIO	UINT8	Read/write error in intermediate storage area of flash (critical.)

Furthermore, upon execution completed, a report will be generated using the same type and subtype as for the telecommand. This report will contain the calculated checksum, see Table 22.13.

Table 22.13 - Calculated flash CRC report

Image copy number	Checksum
UINT8	UINT32

22.3. Limitations

The maximum size of an image for upload is 16 Mbytes.

23. Death Reports

23.1. Description

When an exception occurs, a death report consisting of a SCET timestamp, relevant process registers and further information about the trap is written to the death report area on persistent NVRAM. There are five available death report slots in the NVRAM. If the table is full and a new trap occurs, no new report will be added to the table, it is left unchanged.

When an unexpected trap has occurred, the watchdog will not be kicked and the TCM will reset. Death reports for the TCM can be read and also cleared via RMAP. See Section 17.7.23. Note that FPU traps are disabled in the TCM core application, and thus no death reports will be generated for them.

23.2. Trap types

Table 7-1 in [RD14] describes the implemented traps for LEON3FT. Table 23.1 shows the implementation for Sirius and which unexpected traps that will/will not result in Death Reports. When an exception has occurred, the trap type can be determined by reading the tt-field in the death report entry field. See Table 17.50 and Table 23.1 for details.

Table 23.1 - Sirius Trap Types

Trap	tt-value	Pri	Description	Class	Comment
~~reset~~	~~00~~	~~1~~	~~Power-on reset~~	~~Interrupting~~	Expected trap
data store error	0x2b	2	Write buffer error during data store	Interrupting	
instruction access exception	0x01	3	Error or MMU page fault during instruction fetch	Precise	
privileged instruction	0x03	4	Execution of privileged instruction in user mode	Precise	
illegal instruction	0x02	5	UNIMP or other un-implemented instruction	Precise	
fp disabled	0x04	6	FP instruction while FPU disabled	Precise	

Trap	tt-value	Pri	Description	Class	Comment
~~cp disabled~~	~~0x24~~	~~6~~	~~CP instruction while Co-processor disabled~~	~~Precise~~	No co-processor in current implementation
~~watchpoint detected~~	~~0x0B~~	~~7~~	~~Hardware breakpoint match~~	~~Precise~~	Expected trap
window overflow	0x05	8	SAVE into invalid window	Precise	
window underflow	0x06	8	RESTORE into invalid window	Precise	
~~r register access error~~	~~0x20~~	~~9~~	~~Register file EDAC error (LEON3FT only)~~	~~Interrupting~~	Not present in current implementation
mem address not aligned	0x07	10	Memory access to un-aligned address	Precise	
fp exception	0x08	11	FPU Exception	Deferred	
~~cp exception~~	~~0x28~~	~~11~~	~~Co-processor exception~~	~~Deferred~~	No co-processor in current implementation
data access exception	0x09	13	Access error during data load, MMU page fault	Precise	
tag overflow	0x0A	14	Tagged arithmetic overflow	Precise	
division by zero	0x2A	15	Divide by zero	Precise	

24. TM/TC-structure and COP-1

24.1. SCID

For commanding the spacecraft, a 10-bit Spacecraft Identifier is needed. For every mission, a mission specific SCID is configured in the TCM.

24.2. APID

The application running on the TCM has a unique identifier, Application Process Identifier, that is configurable for every mission by a parameter stored in the NVRAM on the TCM.

24.3. Virtual Channel Allocation

See [RD7] for VC allocation.

24.4. Uplink Channel Coding, Randomization and Synchronization

24.4.1. Channel Coding

The Telecommand Code Block is BCH (63, 56) and supports Single Error Correction mode.

24.4.2. Randomization

Derandomization of telecommands can be enabled/disabled by a configuration in NVRAM (see Section 6.1.9) or by a RMAP command.

24.4.3. Channel Synchronization

The 2-byte start sequence of Telecommands is `0xEB90`. The 8-byte tail sequence of Telecommands is `0xC5C5C5C5C5C5C579`.

24.5. Downlink Channel Coding, Randomization and Synchronization

24.5.1. Channel Coding

Reed-Solomon encoding by a RS (255, 223) encoder with an interleaving depth of 5, resulting in a Telemetry Transfer Frame length of 1115 octets.

Convolutional encoding is according to [RD15] section 3.3 (code rate 1/2 bit per symbol; constraint length 7 bits; polynomial generators G1=171 octal and G2=133 octal; inversion on G2).

It can be enabled/disabled by a configuration in NVRAM (see Section 6.1.12) or by a RMAP command.

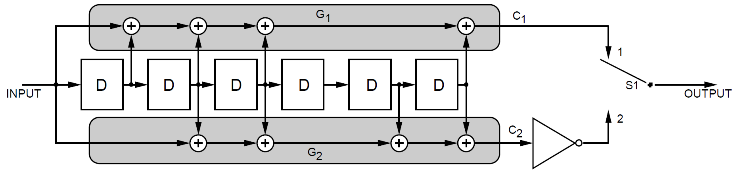

Figure 24.1 - Convolutional Encoder Block Diagram

24.5.2. Randomization

Randomization of Telemetry Transfer Frames can be enabled/disabled by a configuration in NVRAM (see Section 6.1.12) or by a RMAP command.

24.5.3. Synchronization

The 4-byte synchronization pattern prepended to the Reed-Solomon code block is `0x1ACFFC1D`.

24.6. Telecommand format

This chapter describes the format of the TC Transfer frames and TC Packets the TCM handles.

24.6.1. Telecommand Transfer Frame

The Telecommand Transfer Frame conforms to the format described in [RD10] and shown below.

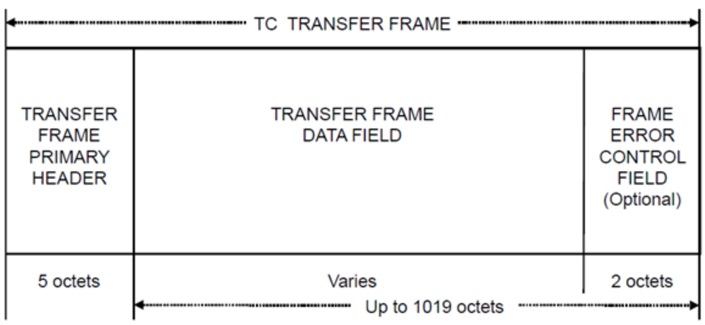

Figure 24.2 - TC Transfer Frame

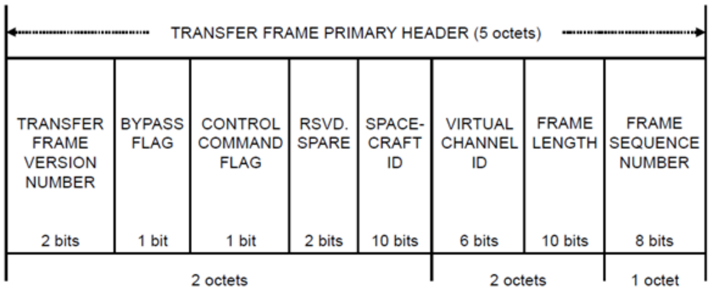

Figure 24.3 - Transfer Frame Header

24.6.2. Transfer Frame Header

Table 24.1 - Transfer Frame Header

Field	Description	Comment
VERSION NUMBER	Shall be set to `00`	
BYPASS FLAG	Set to `0` to set Type-A of frame. Set to 1 to set Type-B of frame.	When this flag is set to `0`, the frame will be subject to Frame Acceptance Check of the FARM on TCM. When this flag is set to 1, the Frame Acceptance Check will be bypassed on the TCM.
CONTROL COMMAND FLAG	Set to `0` to indicate the Transfer Frame Data Field contains a Frame Data Unit (Type-D) Set to 1 to indicate the Transfer Frame Data field contains control information (Type-C)	In conjunction with BYPASS FLAG, the frame types Type-AD, Type-BD and Type-BC are supported by the TCM.
RESERVED SPARE		
SPACECRAFT ID	Contains the mission-specific spacecraft identifier (SCID)	If the SCID of the TC Transfer Frame is not same as the SCID configured on the TCM, the TC Transfer Frame will be rejected.
Virtual Channel ID	Virtual channel ID of Telecommand	See [RD7] for VC allocation, and also Section 6.1.11.

Field	Description	Comment
FRAME LENGTH	Shall be set to total number of octets in the TC Transfer Frame - 1	The maximum number of octets in the TC Transfer Frame is 1024.
FRAME SEQUENCE NUMBER	The number of the TC Transfer Frame	The Frame sequence number enables the FARM to check sequence of incoming Type-A transfer frames

24.6.3. Transfer Frame Data Field

TC Transfer Frames sent to the TCM are expected to contain the Frame Error Control Field, which results in a maximum length of 1017 octets of the Transfer Frame Data Field. The Transfer Frame Data Field shall contain either a Frame Data Unit (for Type-D Transfer Frame) or a control command (for Type-C Transfer Frames).

For Transfer Frames carrying a Frame Data Unit, a Segment Header follows the Transfer Frame Primary Header, see Figure 24.4. For Frame Data Units, the user data shall contain a complete packet, see Section 24.6.5.

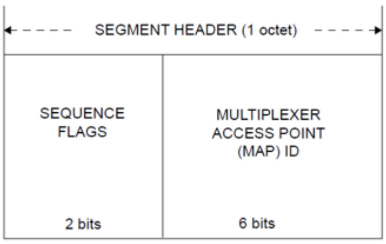

Figure 24.4 - Segment Header

Table 24.2 - Segment Header

Field	Description	Comment
SEQUENCE FLAGS	Shall be set to 11 since no segmentation is supported on TCM	
MAP ID	Shall be set to 0.	Only MAP ID 0 is supported on TCM

Two control commands are supported by TCM: Unlock and Set V(R). The Unlock Control Command consists of a single octet containing “all zeroes”. The Set V(R) Control Command shall consist of three octets with the following values:

10000010 00000000 XXXXXXXX

where XXXXXXXX is the value the FARM shall set the Receiver Frame Sequence Number, V(R).

24.6.4. Frame Error Control Field

The Frame Error Control holding an error detection code (checksum) shall always be included in the telecommand transfer frames, which allows the receiving application to verify the integrity of the telecommand frame data.

The checksum shall be calculated using CRC with polynomial `0x8408`, LSB first (reverse of `0x1021`, MSB first); and initial value `0xFFFF` over the whole TC Transfer Frame except the two last octets.

24.6.5. Telecommand Packet

All Telecommand Packets in Frame Data Units received by the TCM must be CCSDS Space Packets according to [RD16].

Telecommand packets directed to the TCM unit itself must additionally be ECSS PUS Packets (a subset of CCSDS Space Packets) according to [RD11].

The format of the CCSDS Space Packet is shown in Figure 24.5 and Figure 24.6 below.

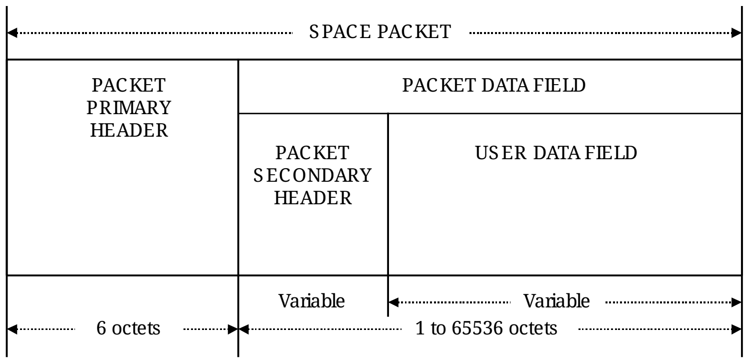

Figure 24.5 - CCSDS Space Packet (reproduced from [RD16])

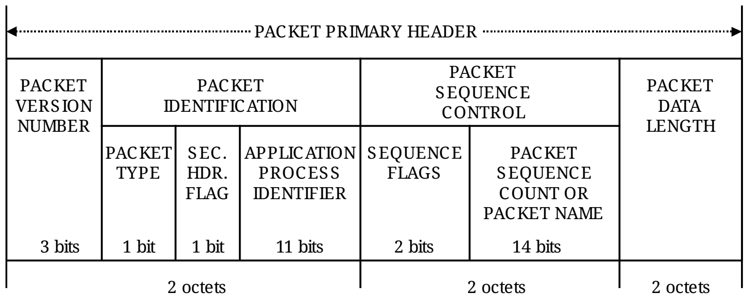

Figure 24.6 - CCSDS Space Packet Primary Header (reproduced from [RD16])

Table 24.3 - CCSDS Space Packet Primary Header description

Field	Description	Comment
Packet Version Number		Must be set to 0.
Packet Type	Indicates if the packet is a telemetry or telecommand packet.	Must be set to 1 for telecommand packets.
Secondary Header Flag	Flag indicating the presence of a secondary header.	Must be set to 1 for PUS packets directed to the TCM unit (except for CPDU command packets).
Application Process ID	Sets the destination on-board application for the telecommand packet. Often abbreviated to "APID".	
Sequence Flags	Flags indicating if this packet is a continuation, first, last, or stand-alone packet	Must be set to `0b11` (stand-alone) for PUS packets directed to the TCM unit.
Packet Sequence Count or Packet Name	Identifier provided to be able to track a specific packet.	
Packet Data Length	Specifies number of octets within the packet data field. The encoded value is the number of octets in the packet data field - 1.	

The format of the secondary header for PUS telecommand packets directed to the TCM unit (except CPDU command packets) is shown below.

TC packet PUS version number	acknowledgement flags	message type ID		source ID	spare
		service type ID	message subtype ID		
enumerated (4 bits)	enumerated (4 bits)	enumerated (8 bits)	enumerated (8 bits)	enumerated (16 bits)	fixed-size bit-string
					optional

Figure 24.7 - PUS Telecommand Packet Secondary Header (reproduced from [RD11])

Table 24.4 - PUS Telecommand Packet Secondary Header description

Field	Description	Comment
TC packet PUS version number		Must be set to 2.
acknowledgement flags	Specifies level of reporting to ground by the receiving Application Process.	The TCM unit sends acceptance success reports and execution completion success reports based on the ack flags.
service type ID	Indicates the service to which the message relates.	
message subtype ID	Indicates the message subtype within the service.	
Source ID	Source identifier for issuer of request.	Ignored by the TCM unit.
Spare	Optional padding	Must be omitted for PUS packets directed to the TCM unit.

The format of the user data field for PUS telecommand packets directed to the TCM unit (including CPDU command packets) is shown below.

application data	spare	packet error control
deduced	fixed-size bit-string (deduced)	fixed-size bit-string (16 bits)
	optional	

Figure 24.8 - PUS Telecommand Packet User Data Field (reproduced from [RD11])

Table 24.5 - PUS Telecommand Packet User Data Field description

Field	Description	Comment
application data	Data payload.	
spare	Optional padding.	Not needed by the TCM unit.
packet error control	Whole packet checksum.	Must use the CRC defined in Annex B of [RD11] (same algorithm as described in Section 24.6.4) for PUS packets directed to the TCM unit.

24.6.6. Carrier Lock and Subcarrier Lock

In the radio interface connectors on the TCM there are two input signals called Carrier Lock and Subcarrier Lock. These need to be active for the TCM to process the incoming telecommand data. The state of the signals is reflected in the CLCW flags “No RF Available” and “No Bit Lock”, see Section 24.7.4.

24.7. Telemetry Format

This chapter describes the format of TM Transfer Frames and TM Packets sent from the TCM to ground.

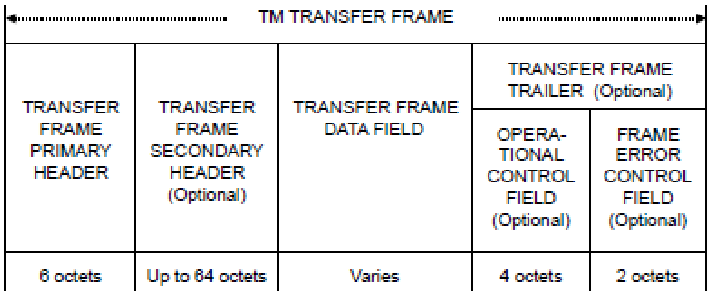

Figure 24.9 - Telemetry Transfer Frame

24.7.1. Transfer Frame Primary Header

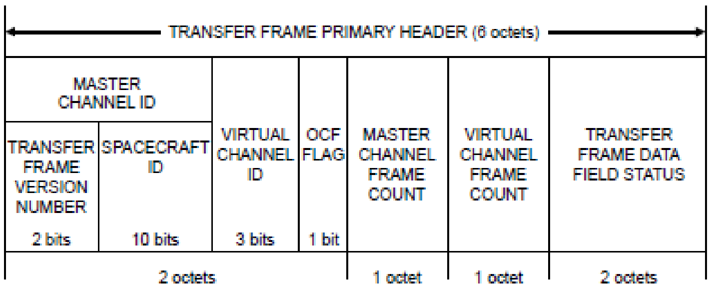

Figure 24.10 - Telemetry Transfer Frame Primary Header

Table 24.6 - Telemetry Transfer Frame Primary Header

Field	Description	Comment
TRANSFER FRAME VERSION NUMBER	Set to `00`.	
SPACECRAFT ID	Mission specific identifier of the spacecraft.	
VIRTUAL CHANNEL ID	See [RD7] for VC allocation.	

Field	Description	Comment
OCF FLAG	Indicates presence of Operation Control Field (OCF) in TM Transfer Frames. It shall be 1 if the OCF is present. It shall be `0` if the OCF is not present.	This is configurable by a setting in NVRAM for the TCM. It can also be set by a RMAP-command.
MASTER CHANNEL FRAME COUNT	An 8-bit sequential binary count (modulo 256).	
VIRTUAL CHANNEL FRAME COUNT	An 8-bit sequential binary count (modulo 256).	
TRANSFER FRAME DATA FIELD STATUS	See below	

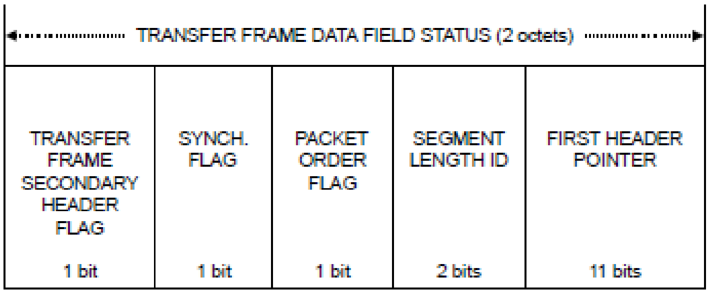

Figure 24.11 - Transfer Frame Data Field Status

Table 24.7 - Transfer Frame Data Field Status

Field	Description	Comment
TRANSFER FRAME SECONDARY HEADER FLAG	Shall be 1 if Transfer Frame Secondary Header is present. Shall be `0` if Transfer Frame Secondary Header is not present.	In the TCM, the Transfer Frame Secondary Header is not used, so this field is always set to `0`.
SYNCHRONIZATION FLAG	Indicates type of data inserted in the Transfer Frame Data Field. It shall be ´0´if octet-synchronized, 1 otherwise.	In the TCM, data is always inserted octet-synchronized, so this field is always set to `0`.
PACKET ORDER FLAG	Packet Order Flag.	Always set to `0` in TCM.
SEGMENT LENGTH ID	Shall be set to 11 if Synchronization Flag is set to `0`.	Set to 11 in TCM.

Field	Description	Comment
FIRST HEADER POINTER	If the Synchronization Flag is set to `0`, the First Header Pointer shall contain the position of the first octet of the first Packet that starts in the Transfer Frame Data Field. When valid data exist in frame, but no packet/segment header is present the First Header Pointer is set to 11111111111. If the frame contains only idle data, the First Header Pointer is set to `11111111110`.	

24.7.2. Transfer Frame Secondary Header

The Transfer Frame Secondary Header is not used by the TCM.

24.7.3. Transfer Frame Data Field

The Transfer Frame Data Field contains an integral number of octets of data formatted as TM Packets. The length of this field is fixed but can be different for different configurations depending on inclusion of OCF and FECF. The maximum length of this field is 1109 octets (1115 – 6), and the minimum length is 1103 octets (1115 - 6 - 4 - 2).

24.7.4. Operational control field

The Operational Control Field contains a Communications Link Control Word as described in [RD10] section 4.2.

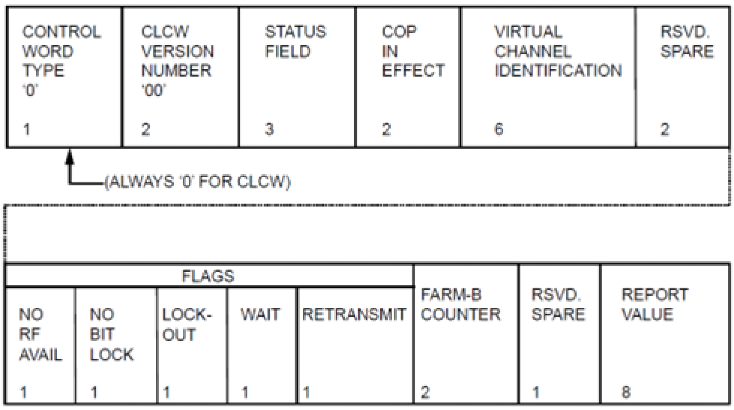

Figure 24.12 - Command Link Control Word

Table 24.8 - Command Link Control Word

Field	Description	Comment
CONTROL WORD TYPE	Is set to `0`.	
CLCW VERSION NUMBER	Is set to `00`.	
STATUS FIELD	Can be used for Mission-specific status.	No specific setting by TCM.
COP IN EFFECT	Set to `01`.	
VIRTUAL CHANNEL IDENTIFICATION	Virtual Channel Identifier.	
RESERVED SPARE	Set to `00`.	
NO RF AVAIL	Set to `0` if Physical Layer Available. Set to 1 if Physical Layer is not available.	Controlled by physical input signal, see Section 24.6.6
NO BIT LOCK	Set to `0` when bit lock has been achieved. Set to 1 when bit lock has not been achieved.	Controlled by physical input signal, see Section 24.6.6
LOCK-OUT	Shows Lockout status of the FARM. Set to `0` when FARM is not in Lockout. Set to 1 when FARM is in Lockout.	

Field	Description	Comment
WAIT	Set to 1 (Wait) indicates that all further Type-A Transfer Frames on that virtual channel will be rejected by FARM until the condition cleared. Set to `0` indicates TCM is able to accept and process incoming Type-A Transfer Frames.	
RETRANSMIT	Set to 1 indicates that one or more Type-A Transfer Frames have been rejected. Set to `0` indicates no outstanding Type-A Transfer Frame rejections so far.	
FARM-B COUNTER	Contains two least significant bits of FARM-B Counter.	
RESERVED SPARE	Set to `0`.	
REPORT VALUE	Contains the value of the Next Expected Frame Sequence Number, N(R).	

24.7.5. Frame Error Control Field

If used, the checksum of the Frame Error Control Field shall be calculated using CRC with polynomial `0x8408`, LSB first (reverse of `0x1021`, MSB first); and initial value `0xFFFF` over the whole TM Transfer Frame except the two last octets.

24.7.6. Telemetry Packet

The TCM unit will generate PUS reports as telemetry packets in response to incoming PUS request telecommand packets. See [RD11] for more information.

The format of the PUS telemetry packet used for these PUS reports is shown in Figure 24.13 and Figure 24.14 below.

<table>
<tr><td colspan="7">packet primary header</td><td colspan="2">packet data field</td></tr>
<tr><td rowspan="2">packet version number</td><td colspan="3">packet ID</td><td colspan="2">packet sequence control</td><td rowspan="2">packet data length</td><td rowspan="2">packet secondary header</td><td rowspan="2">user data field</td></tr>
<tr><td>packet type</td><td>secondary header flag</td><td>application process ID</td><td>sequence flags</td><td>packet sequence count or packet name</td></tr>
<tr><td>3 bits</td><td>1 bit</td><td>1 bit</td><td>11 bits</td><td>2 bits</td><td>14 bits</td><td>16 bits</td><td>variable</td><td>variable</td></tr>
<tr><td colspan="4">2 octets</td><td colspan="2">2 octets</td><td>2 octets</td><td colspan="2">1 to 65536 octets</td></tr>
</table>

Figure 24.13 - PUS Telemetry Packet (reproduced from [RD11])

Table 24.9 - PUS Telemetry Packet

Field	Description	Comment
Packet Version Number		Will be set to 0.
Packet Type	Indicates if the packet is a telemetry or telecommand packet.	Will be set to 0 for telemetry packets.
Secondary Header Flag	Flag indicating the presence of a secondary header.	Will be set to 1 for PUS packets originating from the TCM unit.
Application Process ID	Indicates the origin on-board application for the telemetry packet.	Will be set to the configured APID for the TCM unit for packets originating from the TCM unit, see Section 6.1.10.
Sequence Flags	Flags indicating if this packet is a continuation, first, last, or stand-alone packet	Will be set to `0b11` (stand-alone) for PUS packets originating from the TCM unit.
Packet Sequence Count or Packet Name	Identifier provided to be able to track a specific packet.	
Packet Data Length	Specifies number of octets within the packet data field. The encoded value is the number of octets in the packet data field - 1.	

The format of the secondary header for PUS telemetry packets originating from the TCM unit is shown below.

TM packet PUS version number	spacecraft time reference status	message type ID		message type counter	destination ID	time	spare
		service type ID	message subtype ID				
enumerated (4 bits)	enumerated (4 bits)	enumerated (8 bits)	enumerated (8 bits)	unsigned integer (16 bits)	enumerated (16 bits)	absolute time	fixed-size bit-string
							optional

Figure 24.14 - PUS Telemetry Secondary Header (reproduced from [RD11])

Table 24.10 - PUS Telecommand Packet Secondary Header description

Field	Description	Comment
TM packet PUS version number		Will be set to 2.
spacecraft time reference status	Status of the on-board time reference used when time tagging the telemetry packet.	The TCM unit does not support reporting of the on-board time reference status and will set this field to 0.
service type ID	Indicates the service to which the message relates.	
message subtype ID	Indicates the message subtype within the service.	
message type counter	Counter for generated messages per destination	The TCM unit does not support counting message type per destination and will set this field to 0.
destination ID	User identifier of the application process addressed by the report.	The TCM unit assumes that all outgoing telemetry is directed to a single ground destination and will set this field to 0.
time	Time tag of report.	The TCM unit uses an explicit CUC time format with Packet field Format Code (PFC) 0, see Table 24.11.
Spare	Optional padding	Not used by the TCM unit.

The format of the secondary header time field for PUS telemetry packets originating from the TCM unit is based on the CUC time format in [RD17] and is shown below.

Table 24.11 - PUS Telemetry Packet Secondary Header Time Field

Byte	Type	Description
0	UINT8	Explicit P-field. The TCM unit will set this to `0x2F` corresponding to an "agency-defined epoch", 4 octets of basic time, and 3 octets of fractional time.
1-4	UINT32	Basic/coarse time. The TCM unit will set this field to the seconds counter of the SCET.
5-7	UINT16 + UINT8	Fractional/fine time. The TCM unit will set the most significant 16 bits of this field to the subseconds counter of the SCET and will set the least significant 8 bits to 0.

The format of the user data field for PUS telemetry packets originating from the TCM unit is shown below.

source data	spare	packet error control
deduced	fixed-size bit-string (deduced)	fixed-size bit-string (16 bits)
	optional	*optional*

Figure 24.15 - PUS Telemetry Packet User Data Field (reproduced from [RD11])

Table 24.12 - PUS Telemetry Packet User Data Field description

Field	Description	Comment
source data	Data payload.	
spare	Optional padding.	Not used by the TCM unit.
packet error control	Whole packet checksum.	Will use the CRC defined in Annex B of [RD11] (same algorithm as described in Section 24.6.4) for PUS packets originating from the TCM unit.

24.7.7. Idle Data

In the TCM, `0x5A` is the data sent for Idle Frames and Idle Packets.

24.8. FARM-parameters

COP-1 is supported on the TCM.

24.8.1. FARM_Sliding_Window_Width(W)

In the TCM, the parameter W is fixed to 128.

24.8.2. FARM_Positive_Window_Width(PW)

In the TCM, the parameter PW is fixed to 64.

24.8.3. FARM_Negative_Window_Width(NW)

In the TCM, the parameter NW is fixed to 64.

25. Updating the Sirius FPGA

To be able to update the SoC on the Sirius OBC and Sirius TCM you need the following items:

Prerequisite hardware:

- Microsemi FlashPro5 unit
- 104470 FPGA programming cable assembly

Prerequisite software:

- Microsemi FlashPro Express v11.8 or later
- The updated FPGA firmware

25.1. Generation of encryption key

When AAC Clyde Space is supporting a customer, files with sensitive data to be transferred between AAC and customers can be encrypted/decrypted by GPG.

1. Generate a key by:
   `gpg --gen-key`
2. Select option “DSA and Elgamal” and a keysize of 2048 bits
3. After successful generation of the key, export the key by:
   `gpg --export -a -o your_pub.key`
4. The generated key, `your_pub.key`, in example above is to be sent to AAC if needed.

25.2. Step-by-step guide

The following instructions show the necessary steps that need to be taken in order to upgrade the FPGA firmware:

1. Connect the FlashPro5 programmer via the 104470 FPGA programming cable assembly to the JTAG-RTL connector in Figure 25.1.
2. Connect the power cables according to Figure 25.1.
3. The updated FPGA firmware delivery from AAC should contain at least two files:
   a. The actual FPGA file with an .stp file ending
   b. The programmer file with a .pro file ending
4. Start the FlashPro Express application, click “Open…” in the “Job Projects” box (see Figure 25.1) and select the supplied .pro file.

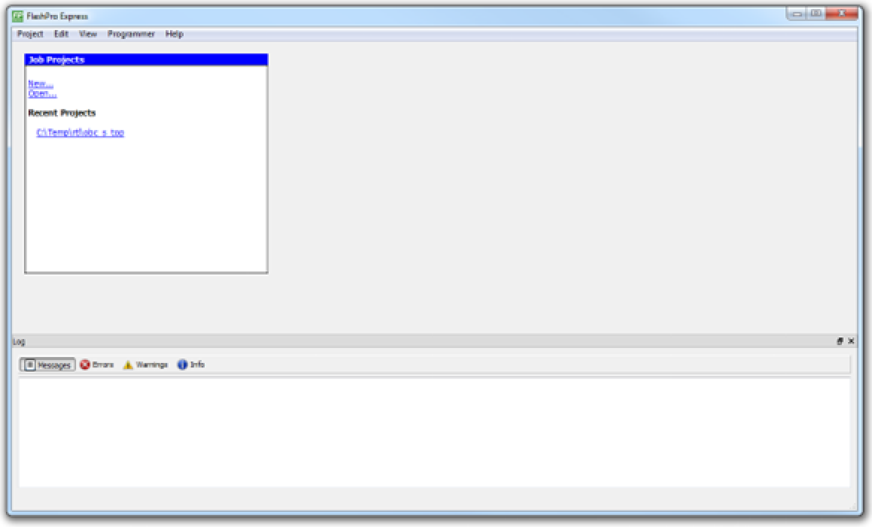

Figure 25.1 - Startup view of FlashPro Express

5. Once the file has loaded (warnings might appear), click RUN (see Figure 25.2). Please note that the connected FlashPro5 programmed ID should be shown.

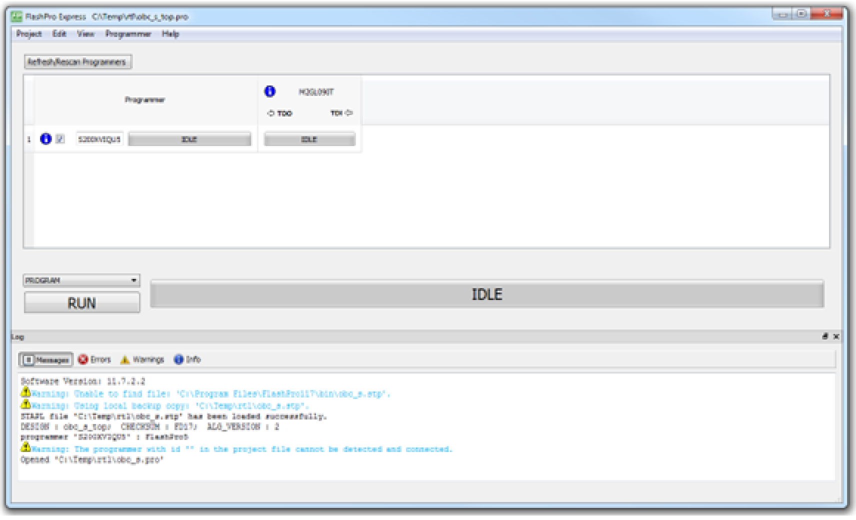

Figure 25.2 - View of FlashPro Express with project loaded.

6. The FPGA should now be loaded with the new firmware, which might take a few minutes. Once it is finalized the second last message should be "Chain programming PASSED", see Figure 25.3.

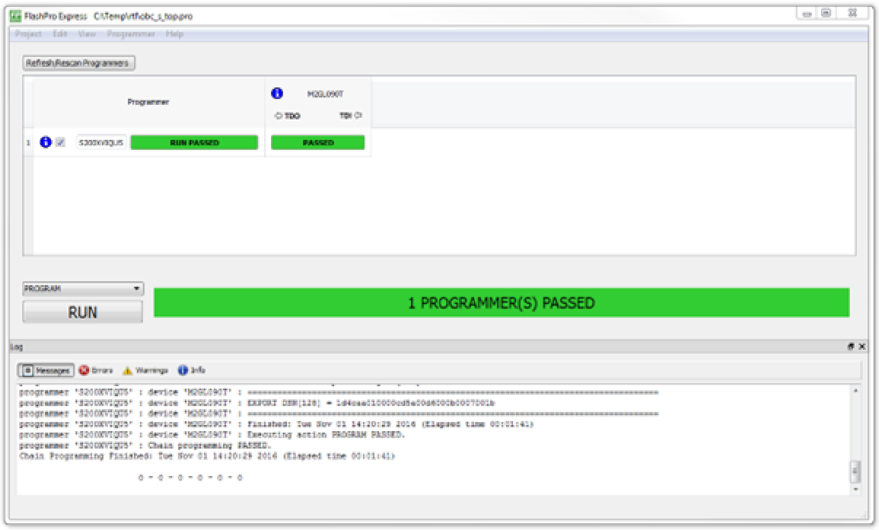

Figure 25.3 - View of FlashPro Express after program passed.

The Sirius FPGA image is now updated.

26. Mechanical data

Please refer to the mechanical and Electrical ICD [RD2].

27. Glossary

Acronym	Description
ABI	Application Binary Interface
ADC	Analog Digital Converter
API	Application Programming Interface
APID	Application Process ID
BCH	Bose-Chaudhuri-Hocquenghem code, a type of error correction code
BSP	Board Support Package
CCSDS	The Consultative Committee for Space Data Systems
CLCW	Command Link Control Word, see [RD8] and [RD10]
COP	Communications Operation Procedure, see [RD8] and [RD10]
CPDU	Command Pulse Distribution Unit
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
ECC	Error Correction Code
EDAC	Error Detection and Correction
EM	Engineering model
ESD	Electrostatic Discharge
FARM	Frame Acceptance and Reporting Mechanism, see [RD10]
FECF	Frame Error Control Field, see [RD8] and [RD10]
FIFO	First In First Out
FLASH	Flash memory is a non-volatile computer storage chip that can be electrically erased and reprogrammed
FPGA	Field Programmable Gate Array
FW	Firmware
GCC	GNU Compiler Collection program (type of standard in Unix)
GDB	GNU Debugger
GPIO	General Purpose Input/Output
Gtkterm	A terminal emulator that drives serial ports

Acronym	Description
I^2C	Inter-Integrated Circuit, generally referred as “two-wire interface” is a multi-master serial single-ended computer bus invented by Philips.
IP (core)	Intellectual property core, reusable functional logic block used e.g. in a FPGA
JTAG	Joint Test Action Group, interface for debugging the PCBs
LVTTL	Low-Voltage TTL
LSB	Least significant bit/byte
MCFC	Master Channel Frame Counter
Minicom	Is a text based modem control and terminal emulation program
MSB	Most significant bit/byte
NA	Not Applicable
NVRAM	Non Volatile Random Access Memory
OBC	On Board Computer
OCF	Operational Control Field, see [RD8] and [RD8
OS	Operating System
PCB	Printed Circuit Board
PCBA	Printed Circuit Board Assembly
POSIX	Portable Operating System Interface
PPS	Pulse-Per-Second
PSU	Power Supply Unit
PUS	Packet Utilization Standard
RAM	Random Access Memory, however modern DRAM has not random access. It is often associated with volatile types of memory
RMAP	Remote Memory Access Protocol
ROM	Read Only Memory
RTEMS	Real-Time Executive for Multiprocessor Systems
SCET	SpaceCraft Elapsed Timer
SCID	SpaceCraft ID
SDRAM	Synchronous Dynamic Random Access Memory

Acronym	Description
SoC	System-on-Chip
SPI	Serial Peripheral Interface Bus is a synchronous serial data link which sometimes is called a 4-wire serial bus.
SpW	SpaceWire
SW	Software
TC	Telecommand
TCL	Tool Command Language, a script language
TCM	Telemetry, Tracking and Command Control Module
TM	Telemetry
TMR	Triple Modular Redundancy
TTL	Transistor Transistor Logic, digital signal levels used by IC components
UART	Universal Asynchonous Receiver Transmitter that translates data between parallel and serial forms.
USB	Universal Serial Bus, bus connection for both power and data
VC	Virtual Channel
WDT	WatchDog Timer