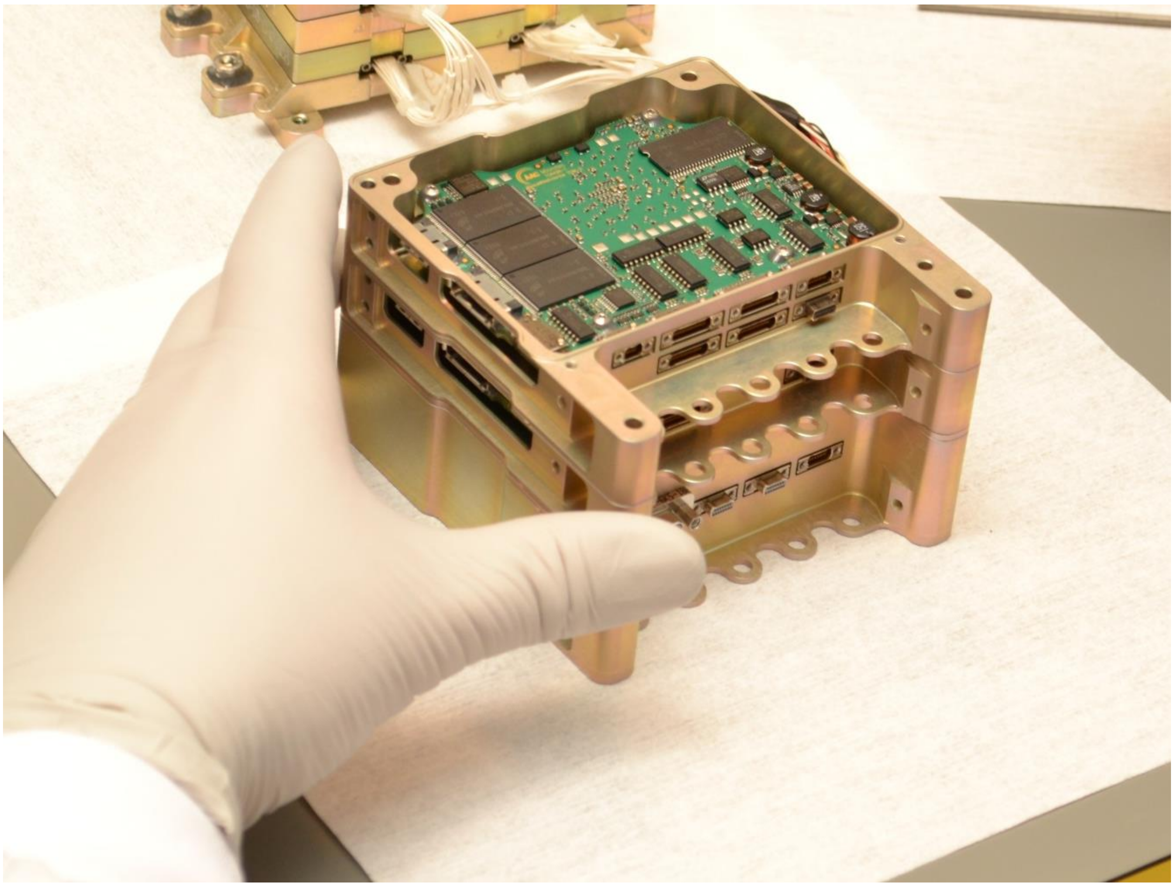
Document number	205065
Version	Rev. D
Issue date	2017-02-01

Sirius OBC and TCM User Manual

Rev. D



REVISION LOG

Rev	Date	Change description
A	2016-10-25	First release, drafted from 204911 Sirius Breadboard User Manual Rev L
B	2016-12-15	Updated after editorial updates
C	2017-01-03	Release with updates to the following sections: • Massmem (new API with DMA) • Error manager (IOCTL API) • ADC (channel table update, channel limitation) • Sirius TCM (TM/TC defaults, API updates {errno, MMStatus, TMTSStatus, }, removed limitations) • Bootrom (extended description) • SCET (extended description, new API) • UART32 (removed) • CCSDS (interrupt API deprecation) • NVRAM (EDAC/non-EDAC modes described)
D	2017-02-01	Release with updates to the following sections: • Sirius TCM (Extra info sections, TMBRSet->TMBRControl) • Mass memory (IOCTL API, error inject info) • SCET (Clarify threshold)

TABLE OF CONTENT

1. Introduction

This manual describes the functionality and usage of the ÅAC Sirius OBC and Sirius TCM products. The Sirius OBC or Sirius TCM differ in certain areas such as the SoC, interfaces etc. but can mostly be described with the same functionality and will throughout this document be referred to as “the Sirius products” when both products are referred at the same time.

1.1. Applicable releases

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

Sirius OBC	0.9.0
Sirius TCM	0.9.0

1.2. Intended users

This manual is written for software engineers using the ÅAC Sirius products. The electrical and mechanical interface is described in more detail in the electrical and mechanical ICD documents [RD10] and [RD11].

1.3. Getting support

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

Email: support@aacmicrotec.com

1.4. Reference documents

RD#	Document ref	Document name
RD1	http://opencores.org/openrisc,architecture	OpenRISC 1000 Architecture Manual
RD2	ECSS-E-ST-50-12C	SpaceWire – Links, nodes, routers and networks
RD3	ECSS-E-ST-50-52C	SpaceWire – Remote memory access protocol
RD4	ECSS-E-70-41A	Ground systems and operations – Telemetry and telecommand packet utilization
RD5	SNLS378B	PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs
RD6	AD7173-8, Rev. A	Low Power, 8-/16-Channel, 31.25 kSPS, 24-Bit, Highly Integrated Sigma-Delta ADC
RD7	Edition 4.10.99.0	RTEMS BSP and Device Driver Development Guide
RD8	CCSDS 132.0-B-2	TM Space Data Link Protocol
RD9	CCSDS 232.0-B-2	TC Space Data Link Prototcol
RD10	P2-2-96-ICD-014	Sirius OBC electrical and mechanical ICD
RD11	P2-2-96-ICD-013	Sirius TCM electrical and mechanical ICD
RD12	SS-EN 61340-5-1	Electrostatics - Part 5-1: Protection of electronic devices from electrostatic phenomena - General requirements

2. Equipment information

The Sirius OBC and Sirius OBC products are depicted in Figure 3-1 and Figure 3-2.

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 centered around a 32 bit OpenRISC Fault Tolerant processor [RD1] running at a system frequency of 50 MHz and with the following set of peripherals:

- Error manager, error handling, tracking and log of e.g. power loss and/or memory error detection.
- SDRAM 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 (Number of interfaces differs between the products) uses the 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 of 2 GB with EDAC-protection for storing boot images in multiple copies
- Pulse command inputs for reset to a specific software image
- MRAM 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.
- Mass memory of 16GB with EDAC-protection, NAND flash based, for storage of mission critical data.

For the Sirius OBC an Analog interface is included.

The input power supply provided to the Sirius products shall be between +4.5 and +16 VDC. The power consumption is highly dependent on peripheral loads and ranges from 0.8 W to 2 W.

2.1. System Overview with peripherals

Figure 2-1 depicts a System-on-Chip (SoC) overview including the related peripherals of the Sirius OBC and Sirius TCM products. The figure shows what parts that are included for which products.

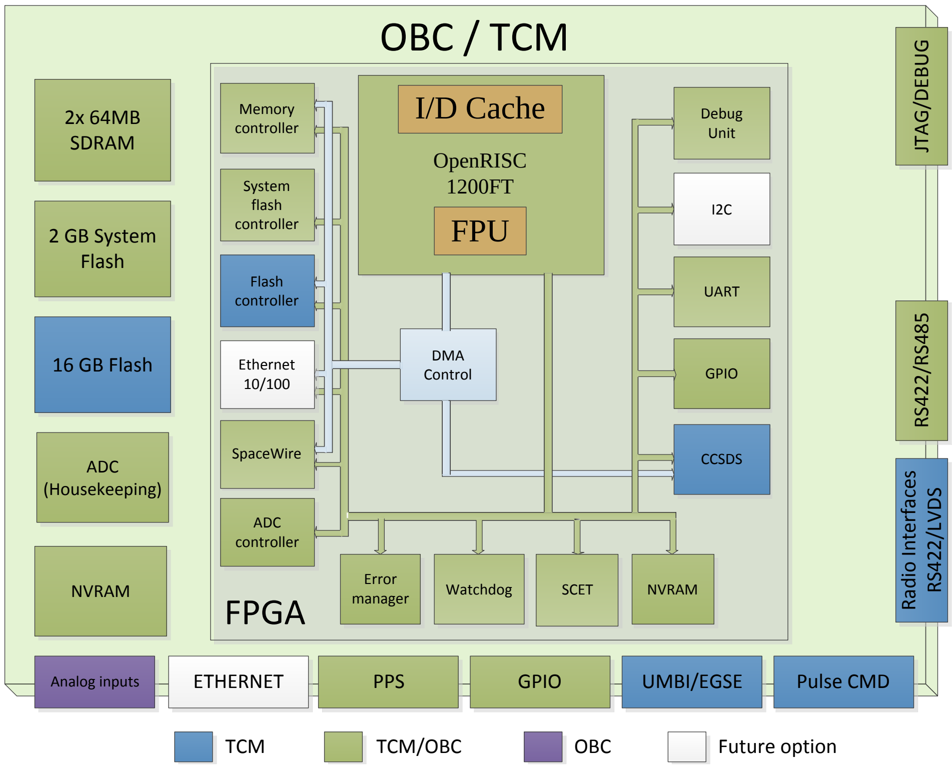


Figure 2-1 - The Sirius OBC / Sirius TCM SoC Overview

3. Setup and operation

3.1. User prerequisites

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

PC computer

- 1 GB free space for installation (minimum)
- Debian 7 or 8 64-bit with super user rights
- USB 2.0

JTAG debugger

- ÅAC JTAG debugger hardware including harness

Recommended applications and software

- Installed terminal e.g. *gtkterm* or *minicom*
- Driver for USB/COM port converter, FTDI, **www.ftdichip.com**
- Host build system, e.g. debian package build-essential
- The following software is installed by the ÅAC toolchain package
  - GCC, C compiler for OpenRISC
  - GCC, C++ compiler for OpenRISC
  - GNU binutils and linker for OpenRISC
  - Custom openocd binary designed for OpenRISC

For FPGA update capabilities

- Microsemi FlashPro Express v11.7, **http://www.microsemi.com/products/fpga-soc/design-resources/programming/flashpro#software**
- FlashPro5 programmer

3.2. Connecting cables to the Sirius products

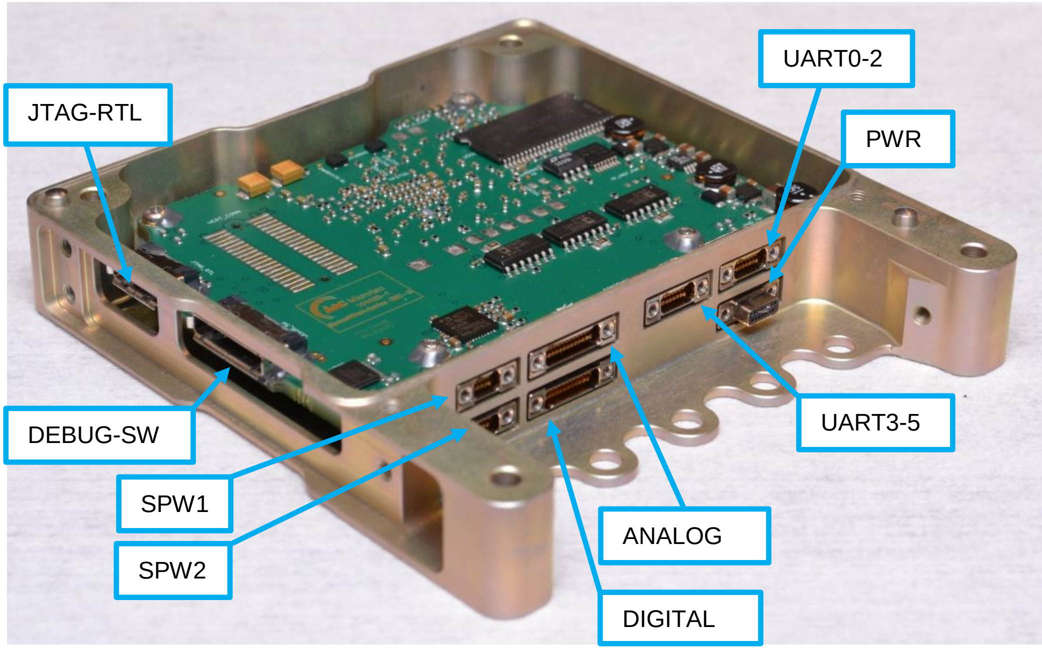


Figure 3-1 – ÅAC Sirius Sirius OBC with connector naming

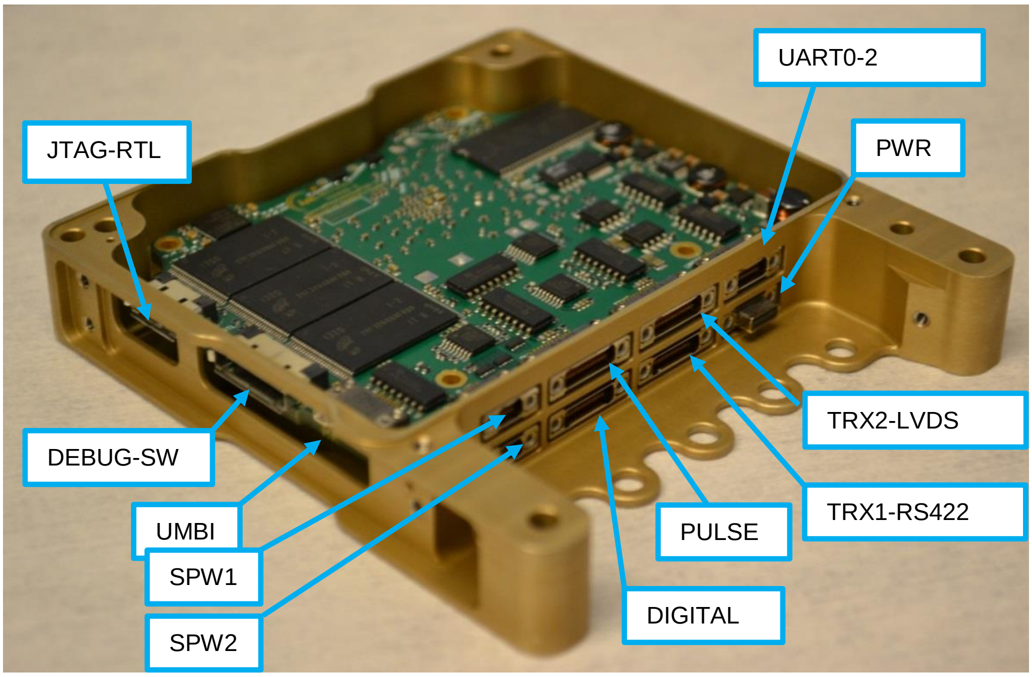


Figure 3-2 - ÅAC Sirius Sirius TCM with connector naming

The Sirius products run on a range of 4.5 to 16 VDC. The instructions below refer to the connector names in Figure 3-1 and Figure 3-2.

- 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 the [RD12] 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 ÅAC 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 Chapter 3.6. When it is to be used, connect the 104452 ÅAC Debugger to the DEBUG-SW connector. Connect the adapter USB-connector to the host PC.
- For FPGA updating only: Connect a FlashPro5 programmer to connector JTAG-RTL using the 104470 FPGA programming cable assembly. For further information how to update the SoC refer to Chapter 9.10.
- For connecting the SpaceWire interface, connect the nano-D connector to connector SPW1 or SPW2.

For more detailed information about the connectors, see Chapter 9.

3.3. Installation of toolchain

This chapter describes instructions for installing the aac-or1k-toolchain.

3.3.1. Supported Operating Systems

Debian 7 64-bit

Debian 8 64-bit

3.3.2. Installation Steps

1. Add the ÅAC Package Archive Server

   Open a terminal and execute the following command:

   ```
   sudo gedit /etc/apt/sources.list.d/aac-repo.list
   ```

   This will open a graphical editor; add the following lines to the file and then save and close it:

   ```
   deb http://repo.aacmicrotec.com/archive/ aac/
   deb-src http://repo.aacmicrotec.com/archive/ aac/
   ```

   Add the key for the package archive as trusted by issuing the following command:

```
wget -O - http://repo.aacmicrotec.com/archive/key.asc | sudo
apt-key add -
```

Terminal will echo "OK" on success.

2. Install the Toolchain Package

   Update the package cache and install the toolchain by issuing the following commands:

```
sudo apt-get update
sudo apt-get install aac-or1k-toolchain
```

   Note: The toolchain package is roughly 1GB uncompressed, downloading/installing it will take some time.

3. Setup

   In order to use the toolchain commands, the shell PATH variable needs to be set to include them, this can be done either temporarily for the current shell via

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

   or permanently by editing the ~/.profile file

```
gedit ~/.profile
```

   and adding the following snippet at the end of the file, and then save and close it:

```
# AAC OR1k toolchain PATH setup
if [ -f /opt/aac/aac-path.sh ]; then
    . /opt/aac/aac-path.sh >/dev/null
fi
```

3.4. Installing the Board Support Package (BSP)

The BSP can be downloaded from http://repo.aacmicrotec.com/bsp. Simply extract the tarball aac-or1k-xxx-x-bsp-y.tar.bz2 to a directory of your choice (xxx-x depends on your intended hardware target - Sirius OBC or Sirius TCM and y matches the current version number of that BSP).

The newly created directory aac-or1k-xxx-x-bsp now contains the drivers for both bare-metal applications and RTEMS. See the included README and chapter 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 who connect to a PC via USB. To interface the Sirius products, the Open On-Chip Debugger (OpenOCD) software is used. A script called run_aac_debugger.sh is shipped with the toolchain package which starts an OpenOCD server for gdb to connect to.

1. Connect the Sirius products according to section3.2.

2. Start the run_aac_debugger.sh script from a terminal.

3. If the printed message is according to Figure 3-3, the connection is working.

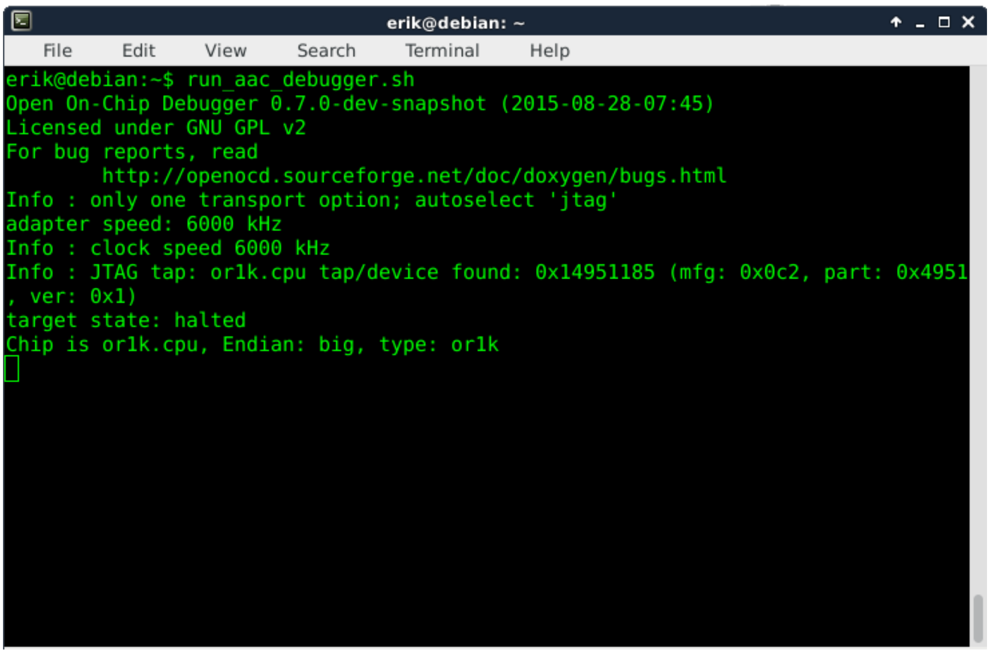


Figure 3-3 - Successful OpenOCD connection to the Sirius products

3.5.2. 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 terminal emulator 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 you have to make sure you're using the correct device number to communicate to the board's debug UART.

3.5.3. Loading an application

An application can either be loaded only to the volatile memory, which is easier and typically used during the development stages, or to NAND flash (see section 3.6). This is done using gdb.

1.a) Start gdb with the following command from a shell for a bare-metal environment

```
or1k-aac-elf-gdb
```

or

1.b) Start gdb with the following command from a shell for an RTEMS environment
`or1k-aac-rtems4.11-gdb`

2. When gdb has opened successfully, connect to the hardware through the OpenOCD server using the gdb command
`target remote localhost:50001`

3. To run an executable program in hardware, first specify its name using the gdb command file. Make sure the application is in ELF format.
`file path/to/binary_to_execute`

4. Now it needs to be uploaded onto the target RAM
`load`

5. In the gdb prompt, type `c` to start to run the application

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

This chapter describes how to program the NAND flash memory with a selected boot image. To achieve this, the boot image binary is bundled together with the NAND flash programming application during the latter's compilation. The NAND flash programming application is then uploaded to the target and started just as an ordinary application using gdb. The maximum allowed size for the boot image for 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:
Note that X is to be replaced according to what your application has been compiled against, either elf for a bare-metal application or rtems4.11 for the RTEMS variant.

   `or1k-aac-X-objcopy -O binary boot_image.elf boot_image.bin`

3. See chapter 3.4 for installing the BSP and enter
`cd path/to/bsp/aac-or1k-xxx-x-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 3.5.3. Follow the instructions on screen and when it's ready, reboot the board by a reset or power cycle.

4. Software development

Applications to be deployed on the Sirius products can either use a bare-metal approach or use the RTEMS OS. This corresponds to the two toolchain prefixes available: or1k-aac-elf-* or or1k-aac-rtems4.11-*

Drivers for both are available in the BSP, see chapter 3.4 and the BSP README for more information.

4.1. RTEMS step-by-step compilation

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

Please note that the toolchain described in chapter 3.3 needs to have been installed and the BSP unpacked as described in chapter 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-or1k-xxx-x-bsp/src/`

2. Type 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
   `export RTEMS_MAKEFILE_PATH=path/to/librtems/or1k-aac-rtems4.11/or1k-aac`

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 chapter 3.5.

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.

This source code is provided without any express or implied warranties whatsoever. Because of the diversity of conditions and hardware under which this source code may be used, no warranty of fitness for a particular purpose is offered. The user is advised to test the source code thoroughly before relying on it. The user must assume the entire risk of using the source code.

5. RTEMS

5.1. Introduction

This section presents the RTEMS drivers. The Block diagram representing driver functionality access via the RTEMS API is shown in Figure 5-1.

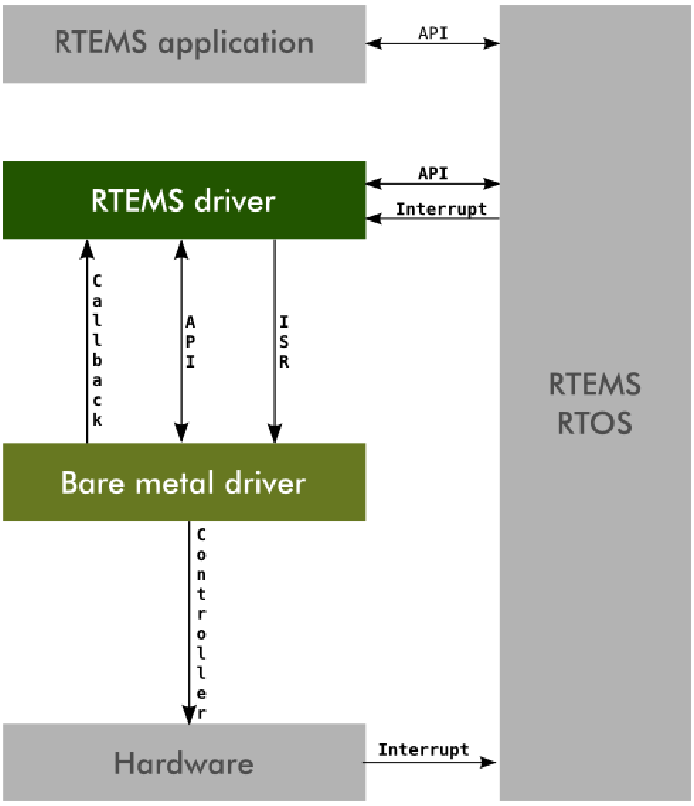


Figure 5-1 - Functionality access via RTEMS API

5.2. Watchdog

5.2.1. Description

This section describes the driver as one utility for accessing the watchdog device.

5.2.2. RTEMS API

This API represents the driver interface from a user application's perspective for the RTEMS driver.

The driver functionality is accessed through RTEMS POSIX API for ease of use. In case of failure on a function call, the errno value is set for determining the cause.

5.2.2.1. int open(…)

Opens access to the device, it can only be opened once at a time.

Argument name	Type	Direction	Description
filename	char *	in	The absolute path to the file that is to be opened. Watchdog device is defined as RTEMS_WATCHDOG_DEVICE_NAME (/dev/watchdog)
oflags	int	in	A bitwise"or" separated list of values that determine the method in which the file is to be opened (whether it should be read only, read/write).

Return value	Description
> 0	A file descriptor for the device on success
- 1	see errno values
errno values	
EALREADY	Device already opened.

5.2.2.2. int close(…)

Closes access to the device.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open

Return value	Description
0	Device closed successfully
-1	see errno values
errno values	
EPERM	Device is not open.

5.2.2.3. size_t write(…)

Any data is accepted as a watchdog kick.

Argument name	Type	Direction	Description
fd	Int	in	File descriptor received at open
buf	void *	in	Character buffer to read data from
nbytes	size_t	in	Number of bytes to write

Return value	Description
*	nNumber of bytes that were written.
- 1	see errno values
errno values	
EPERM	Device was not opened
EBUSY	Device is busy

5.2.2.4. int ioctl(…)

Ioctl allows for disabling/enabling of the watchdog and setting of the timeout.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open
cmd	int	in	Command to send
val	int	in	Data to write

Command table	Val interpretation
WATCHDOG_ENABLE_IOCTL	1 = Enables the watchdog 0 = Disables the watchdog
WATCHDOG_SET_TIMEOUT_IOCTL	1 – 255 = Number of seconds until the watchdog barks

Return value	Description
0	Command executed successfully
-1	see errno values
errno values	
EINVAL	Invalid data sent
RTEMS_NOT_DEFINED	Invalid I/O command

5.2.3. Usage

The watchdog is enabled and disabled using ioctl() calls.

The watchdog must be kicked using a write() call before the timeout occurs or else the watchdog will bark. Notice that the value shall be set between 1 and 255 seconds. Set to zero is a false value.

Default value of the watch dog is enabled. When debugged it must be set disabled otherwise the system restart occasionally.

5.2.3.1. RTEMS

The RTEMS driver must be opened before it can access the watchdog device. Once opened, all provided operations can be used as described in the RTEMS API defined in subchapter 5.2.2. And, if desired, the access can be closed when not needed.

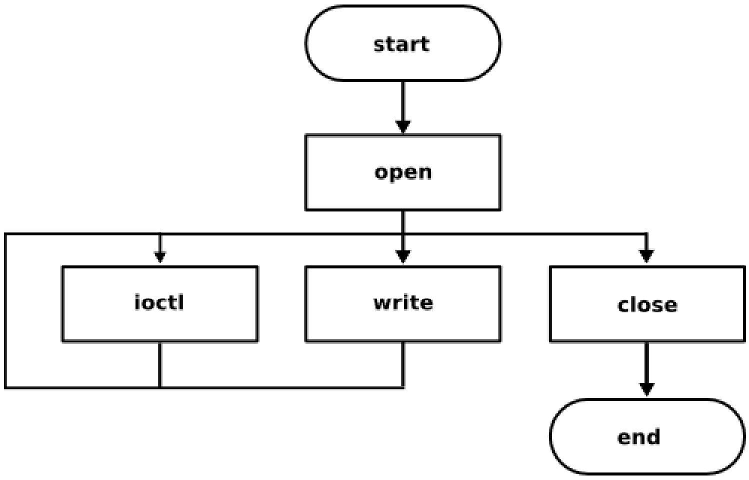


Figure 5-2 - RTEMS driver usage description

Note: All calls to the RTEMS driver are blocking calls.

5.2.3.2. RTEMS application example

In order to use the watchdog driver on the RTEMS environment, the following code structure is suggested to be used:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/wdt_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_WDT_DRIVER
#define CONFIGURE_INIT

rtems_task Init (rtems_task_argument argument);

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

rtems_task Init (rtems_task_argument argument)
{
}
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions `open`, `close`, `lseek`, `read` and `write`.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of `<bsp/wdt_rtems.h>` is required for accessing watchdog device name `RTEMS_WATCHDOG_DEVICE_NAME`.

`CONFIGURE_APPLICATION_NEEDS_WATCHDOG_DRIVER` must be defined for using the watchdog driver. By defining this as part of the RTEMS configuration, the driver will automatically be initialized at boot up.

5.3. Error Manager

5.3.1. Description

The error manager driver is a software abstraction layer meant to simplify the usage of the error manager for the application writer.

This section describes the driver as one utility for accessing the error manager device

5.3.2. RTEMS API

This API represents the driver interface from a user application's perspective for the RTEMS driver.

The driver functionality is accessed through the RTEMS POSIX API for ease of use. In case of failure on a function call, the *errno* value is set for determining the cause.

The error manager driver does not support writing nor reading to the device file. Instead, register accesses are performed using ioctls.

The driver exposes a message queue for receiving interrupt driven events such as power loss, non-fatal multiple errors generated by the RAM EDAC mechanism.

5.3.2.1. int open(…)

Opens access to the device, it can only be opened once at a time.

Argument name	Type	Direction	Description
filename	char *	in	The absolute path to the file that is to be opened. Error manager device is defined as RTEMS_ERRMAN_DEVICE_NAME.
oflags	int	in	A bitwise 'or' separated list of values that determine the method in which the file is to be opened (whether it should be read only, read/write, whether it should be cleared when opened, etc). See a list of legal values for this field at the end.

Return value	Description
fd	A file descriptor for the device on success
-1	see *errno* values

errno values	
EALREADY	Device already opened

5.3.2.2. int close(…)

Closes access to the device.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**

Return value	Description
0	Device closed successfully

5.3.2.3. int ioctl(…)

Ioctl allows for disabling/enabling functionality of the error manager, setting of the timeout and reading out counter values.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**
cmd	uint32_t	in	Command to send
val	uint32_t / uint32_t *	in / out	Value to write or a pointer to a buffer where data will be written

Command table	Description
ERRMAN_GET_SR_IOCTL	Get the status register, see 5.3.2.3.1
ERRMAN_GET_CF_IOCTL	Gets the carry flag register, see 5.3.2.3.2
ERRMAN_GET_SELFW_IOCTL	Points to which boot firmware that will be loaded and executed upon system reboot. 0x0: Programmable FW from Power on 0x1: Programmable FW, Backup copy 0x2: Programmable FW, Backup copy 0x3: Safe FW 0x4: Safe FW, Backup copy 0x5: Safe FW, Backup copy
ERRMAN_GET_RUNFW_IOCTL	Gets the currently running firmware 0x0: Programmable FW from Power on 0x1: Programmable FW, Backup copy 0x2: Programmable FW, Backup copy 0x3: Safe FW 0x4: Safe FW, Backup copy 0x5: Safe FW, Backup copy
ERRMAN_GET_SCRUBBER_IOCTL	Gets the state of the memory scrubber. 0 = Scrubber is disabled 1 = Scrubber is enabled.
ERRMAN_GET_RESET_ENABLE_IOCTL	Gets the reset enable state. 0 = Soft reset is disabled. 1 = Soft reset is enabled

ERRMAN_GET_WDT_ERRCNT_IOCTL	Gets the watchdog error count register. This register can store a value up to 15 and then wraps. After a wrap the WDT carry flag bit is set in the carry flag register. see 5.3.2.3.2
ERRMAN_GET_EDAC_SINGLE_ERRCNT_IOCTL	Gets the EDAC single error count. See 5.3.2.3.3 for interpretation of the register. After a wrap the EDAC single error count carry flag bit is set in the carry flag register. See 5.3.2.3.2
ERRMAN_GET_EDAC_MULTI_ERRCNT_IOCTL	Gets the EDAC multiple error count. See 5.3.2.3.4 for interpretation of the register. After a wrap the EDAC multiple error count carry flag bit is set in the carry flag register. See 5.3.2.3.2
ERRMAN_GET_CPU_PARITY_ERRCNT_IOCTL	Gets the CPU Parity error count register. This register can store a value up to 15 and then wraps. After a wrap the CPU parity error count carry flag bit is set in the carry flag register. See 5.3.2.3.2
ERRMAN_GET_SYS_SINGLE_ERRCNT_IOCTL	Gets the system flash single error (correctable) error count. This register is 4 bit wide and will wrap upon overflow.
ERRMAN_GET_SYS_MULTI_ERRCNT_IOCTL	Gets the system flash multiple error (un-correctable) error count. This register is 4 bit wide and will wrap upon overflow.
ERRMAN_GET_MMU_SINGLE_ERRCNT_IOCTL	Gets the mass memory single error (correctable) error count. This register is 4 bit wide and will wrap upon overflow.
ERRMAN_GET_MMU_MULTI_ERRCNT_IOCTL	Gets the mass memory multiple error (un-correctable) error count. This register is 4 bit wide and will wrap upon overflow.
ERRMAN_GET_POWER_LOSS_ENABLE_IOCTL	Gets the power loss detection enable state. 0 = Power loss detection disabled. 1 = Power loss detection enabled
ERRMAN_SET_SR_IOCTL	Sets the status register, see 5.3.2.3.1
ERRMAN_SET_CF_IOCTL	Sets the carry flag register, see 5.3.2.3.2
ERRMAN_SET_SELFW_IOCTL	Sets the next boot firmware. 0x0: Programmable FW from Power on 0x1: Programmable FW, Backup copy 0x2: Programmable FW, Backup copy 0x3: Safe FW 0x4: Safe FW, Backup copy 0x5: Safe FW, Backup copy
ERRMAN_RESET_SYSTEM_IOCTL	Performs a software reset. The reset enable state is required to be 1 (On).
ERRMAN_SET_SCRUBBER_IOCTL	Sets the state of the memory scrubber. 1 = On, 0 = Off. The scrubber is a vital part of keeping the SDRAM free from errors.
ERRMAN_SET_RESET_ENABLE_IOCTL	Sets the reset enable state. 0 = Soft reset is disabled. 1 = Soft reset is enabled
ERRMAN_SET_WDT_ERRCNT_IOCTL	Sets the watchdog error count register. The counter width is 4 bits i. e. 15 is the maximum value that can be written.
ERRMAN_SET_EDAC_SINGLE_ERRCNT_IOCTL	Sets the EDAC single error count. See 5.3.2.3.3 for register definition.
ERRMAN_SET_EDAC_MULTI_ERRCNT_IOCTL	Sets the EDAC multiple error count register. See 5.3.2.3.4 for register definition.

ERRMAN_SET_CPU_PARITY_ERRCNT_IOCTL	Sets the CPU Parity error count register. The counter width is 4 bits i. e. 15 is the maximum value that can be written.
ERRMAN_SET_SYS_SINGLE_ERRCNT_IOCTL	Sets the system flash single (correctable) error counter. This register is 4 bit wide.
ERRMAN_SET_SYS_MULTI_ERRCNT_IOCTL	Sets the system flash multiple (un-correctable) error counter. This register is 4 bit wide.
ERRMAN_SET_MMU_SINGLE_ERRCNT_IOCTL	Sets the mass memory single (correctable) error counter. This register is 4 bit wide.
ERRMAN_SET_MMU_MULTI_ERRCNT_IOCTL	Sets the mass memory multiple (un-correctable) error counter. This register is 4 bit wide.
ERRMAN_SET_POWER_LOSS_ENABLE_IOCTL	Sets the power loss enable state. 0 = Power loss detection disabled. 1 = Power loss detection enabled

5.3.2.3.1. Status register

Bit position	Name	Direction	Description
31:12	RESERVED		
11	ERRMAN_PULSEFLG	R/W	Pulse command flag bit is set. Clear flag by write a ‘1’
10	ERRMAN_POWFLG	R/W	The power loss signal has been set.
9	ERRMAN_MEMCLR	R	The memory cleared signal is set from the scrubber unit function from the memory controller. Set when the memory has been cleared and read by the bootrom to wait for image.
8	RESERVED		
7	ERRMAN_PARFLG	R/W	A previous CPU Register File Parity Error Reset has been detected Clear flag by write a ‘1’
6	ERRMAN_MEOTHFLG	R/W	A previous RAM EDAC Multiple Error Reset has been detected for non-critical data Clear flag by write a ‘1’
5	ERRMAN_SEOTHFLG	R/W	A previous RAM EDAC Single Error Reset has been detected for critical data Clear flag by write a ‘1’
4	ERRMAN_MECRIFLG	R/W	A previous RAM EDAC Multiple Error Reset has been detected for non-critical data Clear flag by write a ‘1’
3	ERRMAN_SECRIFLG	R/W	A previous RAM EDAC Single Error Reset has been detected for critical data Clear flag by write a ‘1’
2	ERRMAN_WDTFLG	R/W	A previous Watch Dog Timer Reset has been detected Clear flag by write a ‘1’
1	ERRMAN_RFLG	R/W	A previous Manual Reset has been detected Clear flag by write a ‘1’
0	ERRMAN_IFLAG	R/W	Error Manager Interrupt Flag (multiple sources i.e. read the whole status register) Read: ‘0’ – No interrupt pending ‘1’ – Interrupt pending Write: ‘0’ – Ignored ‘1’ – Clear

5.3.2.3.2. Carry flag register

Bit position	Name	Direction	Description
31:8	RESERVED		
7	ERRMAN_PARCFLG	R/W	Carry flag set when CPU Register File Parity Error counter overflow has occurred '0' – No CF set '1' – Counter overflow(Cleared by write '1')
6	ERRMAN_MEOFLG	R/W	Carry flag set when RAM EDAC multiple other error counter overflow has occurred '0' – No CF set '1' – Counter overflow(Cleared by write '1')
5	ERRMAN_SEOFLG	R/W	Carry flag set when RAM EDAC single other error counter overflow has occurred '0' – No CF set '1' – Counter overflow(Cleared by write '1')
4	ERRMAN_MECFLG	R/W	Carry flag set when RAM EDAC Multiple Error counter overflow has occurred '0' – No CF set '1' – Counter overflow (Cleared by write '1')
3	ERRMAN_SECFLG	R/W	Carry flag set when RAM EDAC Single Error counter overflow has occurred '0' – No CF set '1' – Counter overflow (Cleared by write '1')
2	ERRMAN_WDTCFLG	R/W	Carry flag set when Watch Dog Timer counter overflow has occurred '0' – No CF set '1' – Counter overflow (Cleared by write '1')
1	ERRMAN_RFCFLG	R/W	Carry flag set when Manual Reset counter overflow has occurred '0' – No CF set '1' – Counter overflow (Cleared by write '1')
0	RESERVED	-	

5.3.2.3.3. Single EDAC error register

Bit position	Name	Direction	Description
31:20	RESERVED	-	
19:16	ERRMAN_SENOCNT_SDRAM	R/W	SDRAM EDAC single error counter for non-critical errors
15:4	RESERVED	-	

3:0	ERRMAN_SECRICNT_SDRAM	R/W	SDRAM EDAC single error counter for critical errors

5.3.2.3.4. Multiple EDAC error register

Bit position	Name	Direction	Description
31:20	RESERVED	-	
19:16	ERRMAN_MENOCNT	R/W	SDRAM EDAC multiple error counter for non-critical errors
15:4	RESERVED	-	
3:0	ERRMAN_MECRICNT	R/W	SDRAM EDAC multiple error counter for critical errors

Return value	Description
0	Command executed successfully
-1	See *errno* values
errno values	
RTEMS_NOT_DEFINED	Invalid IOCTL
INVAL	Invalid value supplied to IOCTL

5.3.3. Usage

5.3.3.1. RTEMS

The RTEMS driver must be opened before it can access the error manager device. Once opened, all provided operations can be used as described in the RTEMS API defined in subchapter 5.2.2. And, if desired, the access can be closed when not needed.

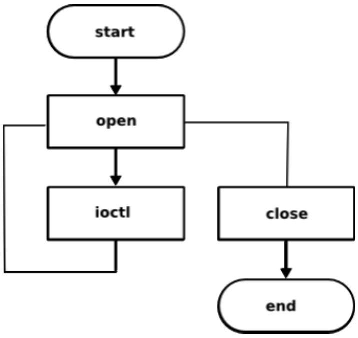


Figure 5-3 - RTEMS driver usage description

<u>Interrupt message queue</u>

The error manager RTEMS driver exposes a message queue service which can be subscribed to. The name of the queue is "'E', 'M', 'G', 'R'".
This queue emits messages upon power loss and single correctable errors.

A subscriber must inspect the message according to the following table to determine whether to take action or not. Multiple subscribers are allowed and all subscribers will be notified upon a message.

Message	Description
ERRMAN_IRQ_POWER_LOSS	A power loss has been detected
ERRMAN_IRQ_EDAC_MULTIPLE_ERR_OTHER	Multiple EDAC errors that are not critical have been detected

5.3.3.2. RTEMS application example

In order to use the error manager driver on RTEMS environment, the following code structure is suggested to be used:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/error_manager_rtems.h>

#define
CONFIGURE_APPLICATION_NEEDS_ERROR_MANAGER_DRIVER

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

#define CONFIGURE_INIT
rtems_task Init (rtems_task_argument argument);


rtems_task Init (rtems_task_argument ignored)
{}
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions: `open`, `close`, `ioctl`.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of `<bsp/error_manager_rtems.h>` is required for accessing error manager device name `RTEMS_ERROR_MANAGER_DEVICE_NAME`.

`CONFIGURE_APPLICATION_NEEDS_ERROR_MANAGER_DRIVER` must be defined for using the error manager driver. By defining this as part of RTEMS configuration, the driver will automatically be initialised at boot up.

5.3.4. Limitations

Many of the error mechanisms are currently unverifiable outside of radiation testing due to the lack of mechanisms of injecting errors in this release.

5.4. SCET

5.4.1. Description

The main purpose of the SCET IP and driver is to track the time since power on and to act as a source of timestamps. The SCET has also been enhanced with General purpose triggers and PPS signaling.

The SCET counts in seconds and subseconds, with a subsecond being 2^{-16}th of a second, roughly equivalent to 15.3 µs.

5.4.2. General purpose triggers

To be able to provide more accurate time stamping on external events, the SCET has a number of general purpose triggers. When a trigger fires, the SCET will sample a subset (24 bits) of the current clock for later software readout, matching the external event to the SCET time regardless of current software state. The exact functionality connected to each general purpose trigger and the number available is dependent on the system mapping of the SCET, e.g. in a System-On-Chip (SoC).

5.4.3. Pulse-Per-Second (PPS) signals

The SCET block is designed to be included in several different units in a system and for time synchronization between these SCETs; each SCET has the ability to receive and/or transmit PPS signals using two PPS signals which is intended for off-chip use. The first signal, pps0, is an input only and intended to be used with a time-aware component such as a GPS device for synchronizing the SCET counter to real time. The second signal, pps1, is bidirectional and intended for use in a multi-drop PPS network. One unit in a system can act as master on the multi-drop PPS network with the other units as slaves, with the ability to switch master depending on the redundancy concept used.

When the SCET synchronizes the time counter with a PPS signal, it will also monitor this PPS signal to make sure it arrives as expected within a user set timeframe (PPS threshold). If input PPS is lost, it requires software interaction to resynchronize to the incoming PPS pulse. This is to minimize the risk for sudden glitches in the SCET counter depending on the incoming PPS accuracy and availability. The PPS monitoring will issue interrupts in bare-metal or messages on the SCET message queue in RTEMS to notify the application if the PPS has arrived, been lost or been found.

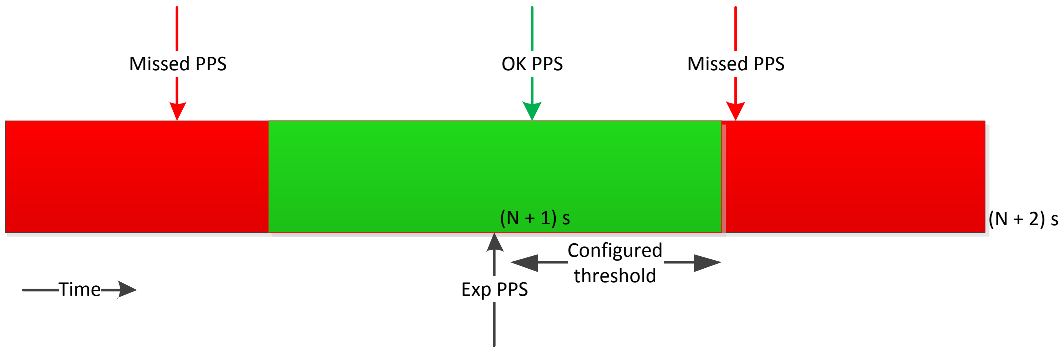


Figure 4 PPS Threshold configuration

To differentiate between the uses of the PPS signal synchronization methods, the SCET can be said to operate in a number of different modes: Free-running, Master, Master with time synchronization and Slave. Please see the explanations below and 5.4.6 for an implementation description.

5.4.3.1. Free-running mode

In this mode, the SCET doesn't use any PPS signals at all. It simply counts the current time since power on without correlation with anyone else.

5.4.3.2. Master mode

In this mode, the SCET is still counting on its own, but now it also emits a pulse on pps1 for every second tick, acting as a master on the bidirectional multi-drop PPS network.

5.4.3.3. Master mode with time synchronization

This mode is the same as the previous master mode, with the addition of also synchronizing the time counter with the incoming pps0 signal. Should the PPS signal on pps0 disappear for some reason, it will revert back to normal master mode and continue issuing PPS signals on pps1.

5.4.3.4. Slave mode

In this mode, the SCET will synchronize the time counter with pps1, using the bidirectional multi-drop PPS network as an input. Should the PPS pulse disappear for some reason, it will revert to free running mode.

5.4.4. RTEMS API

This API represents the driver interface of the module from an RTEMS user application's perspective.

The driver functionality is accessed through the RTEMS POSIX API for ease of use. In case of a failure on a function call, the *errno* value is set for determining the cause.

SCET counter accesses can be done by reading or writing to the device file, modifying the second and subsecond counter values.
The SCET RTEMS driver also supports a number of different IOCTLs for other operations which isn't specifically affecting the SCET counter registers.

For event signaling, the SCET driver has a number of message queues, allowing the application to act upon different events.

5.4.4.1. Function int open(…)

Opens access to the driver. The device can only be opened once at a time.

Argument name	Type	Direction	Description
filename	char *	in	The absolute path to the file that is to be opened. SCET device is defined as RTEMS_SCET_DEVICE_NAME.
oflags	int	in	A bitwise OR-separated list of values that determine the method in which the file is to be opened (whether it should be read only, read/write, etc.).

Return value	Description
>0	A file descriptor for the device on success
-1	see *errno* values
errno values	
EALREADY	Device already opened

5.4.4.2. Function int close(…)

Closes access to the device.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**

Return value	Description
0	Device closed successfully

5.4.4.3. Function ssize_t read(…)

Reads the current SCET value, consisting of second and subsecond counters. Both counter values are guaranteed to be sampled at the same moment.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
buf	void *	in	Pointer to a 6-byte buffer where the timestamp will be stored. The first four bytes are the seconds and the last two bytes are the subseconds.
count	size_t	in	Number of bytes to read, must be set to 6.

Return value	Description
>=0	Number of bytes that were read.
-1	See *errno* values
errno values	
EPERM	Device not opened
EINVAL	Number of bytes to read, count is not 6

5.4.4.4. Function ssize_t write(…)

Adjusts the SCET value's second and subsecond counters using two's complement difference values.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
buf	const void *	in	Pointer to a 6-byte buffer where the adjustment difference values are stored. The first four bytes are the difference value for the seconds and the last two bytes are the difference value for the subseconds.
count	size_t	in	Number of bytes to write, must be set to 6.

Return value	Description
>=0	Number of bytes that were written.
-1	See *errno* values
errno values	
EPERM	Device not open
EINVAL	Number of bytes to write, count is not 6

5.4.4.5. Function int ioctl(…)

Ioctl allows for any other SCET-related operation which isn't specifically aimed at reading and/or writing the SCET time value.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**
cmd	int	in	Command to send
val	see command	in/out	Data according to the specific command.

Command table	Type	Direction	Description
SCET_SET_PPS_SOURCE_IOCTL	uint32_t	in	Sets the PPS source. 0 = External PPS source 1 = Internal PPS source (default)
SCET_GET_PPS_SOURCE_IOCTL	uint32_t	out	Gets the current PPS source 0 = External PPS source 1 = Internal PPS source (default)
SCET_SET_PPS_O_EN_IOCTL	uint32_t	in	Input argument configures if pps0 or pps1 is input and if pps1 is input or output. 0 = pps1 is input (default) 1 = pps0 is input, pps1 is output
SCET_GET_PPS_O_EN_IOCTL	uint32_t	out	Returns wheter the pps0 or pps1 is input and if pps1 is input or output. 0 = pps1 is input (default) 1 = pps0 is input, pps1 is output
SCET_SET_PPS_THRESHOLD_IOCTL	uint16_t	in	Configures the PPS threshold window where the PPS pulse is allowed to arrive without being deemed lost. Defined in number of subseconds. (0 is default)
SCET_GET_PPS_THRESHOLD_IOCTL	uint16_t	out	Returns the currently configured PPS threshold window in subseconds. (0 is default)
SCET_GET_PPS_ARRIVE_COUNTER_IOCTL	uint32_t	out	Returns 24 bits of the SCET time sampled when PPS arrived. Bit 23:16 contains lower 8 bits of second Bit 15:0 contains subseconds

SCET_SET_GP_TRIGGER_LEVEL_IOCTL	uint8_t	in	Configures the trigger level of each trigger: Bit 0 is trigger 0, Bit N is trigger N, Bit 7 is trigger 7. Bit value 0 = trigger activates on 0 to 1 transition (rising edge) Bit value 1 = trigger activates on 1 to 0 transition (falling edge). (0 is default)
SCET_GET_GP_TRIGGER_LEVEL_IOCTL	uint8_t	out	Returns the currently configured level of all the GP triggers as a bit field: Bit 0 is trigger 0, Bit N is trigger N, Bit 7 is trigger 7. Bit value 0 = trigger activates on 0 to 1 transition (rising edge) Bit value 1 = trigger activates on 1 to 0 transition (falling edge). (0 is default)
SCET_SET_GP_TRIGGER_ENABLE_IOCTL	uint8_t	in	Input argument selects which GP trigger(s) to enable. Input is a 8 bit field where bit 0 is trigger 0, bit N is trigger N, bit 7 is trigger 7. All triggers are disabled by default (0)
SCET_GET_GP_TRIGGER_ENABLE_IOCTL	uint8_t	out	Returns which GP triggers that are enabled. Input is a 8 bit field where bit 0 is trigger 0, bit N is trigger N, bit 7 is trigger 7. All triggers are disabled by default (0)
SCET_GET_GP_TRIGGER_COUNTER_IOCTL	uint32_t	in	Input argument selects which GP trigger SCET counter sample to read. Returns 24 bits of the SCET counter sampled when the GP trigger became active. Bit 23:16 contains lower 8 bits of second Bit 15:0 contains subseconds

Return value	Description
>=0	Return value on commands that return these. 0 for success in other cases.
-EINVAL	Invalid value supplied to IOCTL
-EIO	Invalid IOCTL

5.4.5. Usage

5.4.6. PPS

The four described PPS modes can be obtained by setting the PPS output enable and PPS source according to Table 5-1.

Table 5-1 Mapping between PPS modes and PPS settings

PPS mode	PPS source	PPS output enable
Free-running (default)	Internal	Input
Master	Internal	Output
Master with time synchronization	External	Output
Slave	External	Input

When PPS source is set to external and then lost, it will revert to internal setting.
Slave mode will fall back to Free-running mode and Master mode with time synchronization will revert back to Master mode.
When in PPS source is set to internal: If an incoming PPS is detected the PPS found interrupt is asserted. Typically a number of these PPS found interrupts should be investigated by the application and once the PPS is deemed stable enough the PPS source should be set to external (if external synchronization is sought after).
It is up to the application to decide and enforce if and when the external PPS source is to be used again.

5.4.6.1. PPS Threshold

The PPS threshold has a 16 bit resolution and is used to define the subsecond range within which incoming PPS that are deemed acceptable.
The range of acceptability is calculated as >= (65535 – threshold) to <= (65535 + 1 + threshold) subseconds after the previous PPS.

If the PPS threshold is configured to 0 (min value) only incoming PPS that arrive within >= subsecond 65535 of the current second to < subsecond 1 of the next second will be deemed acceptable, (>= 0.65535 to <= 1.0).
If the PPS threshold is configured to 65535 (max value) all incoming PPS are deemed acceptable. Lost events will not be detected at all.

5.4.7. Event callback via message queue

The SCET driver exposes message queues for event messaging from the driver to the application. A single subscriber is allowed for each queue.

‘S’, ‘P’, ‘P’, ‘S’ handles PPS related messages with a prefix of:
SCET_INTERRUPT_STATUS_*

Table 5-2 Driver message queue message types

Event name	Description
PPS_ARRIVED	An external PPS signal has arrived
PPS_LOST	The external PPS signal is lost
PPS_FOUND	The external PPS signal was found

‘S’, ‘G’, ‘T’, ‘n’ handles messages sent from the general purpose trigger n, with the number n ranging from 0 to up to the maximum defined for the particular SoC configuration.

Table 5-3 General purpose trigger n message queue

Event name	Description
TRIGGERn	Trigger n was triggered

5.4.8. RTEMS application example

In order to use the SCET driver in the RTEMS environment, the following code structure is suggested for use:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <assert.h>
#include <bsp/scet_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_SCET_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER

#define CONFIGURE_LIBIO_MAXIMUM_FILE_DESCRIPTORS 30
#define CONFIGURE_MAXIMUM_DRIVERS 10
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_MAXIMUM_TASKS 20
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES 20

#define CONFIGURE_INIT

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

static const int32_t secs_to_adjust = -10;
static const int16_t subsecs_to_adjust = 1000;

/* Adjust SCET time 10 seconds backwards and 1000
 * subseconds forwards */
rtems_task Init (rtems_task_argument ignored)
{
  int result;
  int scet_fd;
  uint32_t old_seconds;
  uint16_t old_subseconds;
  uint32_t new_seconds;
  uint16_t new_subseconds;
  uint8_t read_buffer[6];
  uint8_t write_buffer[6];

  scet_fd = open(RTEMS_SCET_DEVICE_NAME, O_RDWR);
  assert(scet_fd >= 0);

  result = read(scet_fd, read_buffer, 6);
  assert(result == 6);

  memcpy(&old_seconds, read_buffer,
    sizeof(uint32_t));
  memcpy(&old_subseconds, read_buffer +
    sizeof(uint32_t), sizeof(uint16_t));

  printf("\nOld SCET time is %lu.%u\n", old_seconds,
   old_subseconds);
  printf("Adjusting seconds with %ld, subseconds
    with %d\n",
```

```
    secs_to_adjust, subsecs_to_adjust);

  memcpy(write_buffer, &secs_to_adjust,
    sizeof(uint32_t));
  memcpy(write_buffer + sizeof(uint32_t),
&subsecs_to_adjust, sizeof(uint16_t));

  result = write(scet_fd, write_buffer, 6);
  assert(result == 6);

  result = read(scet_fd, read_buffer, 6);
  assert(result == 6);

  memcpy(&new_seconds, read_buffer,
    sizeof(uint32_t));
  memcpy(&new_subseconds, read_buffer +
    sizeof(uint32_t), sizeof(uint16_t));

  printf("New SCET time is %lu.%u\n", new_seconds,
    new_subseconds);

  result = close(scet_fd);
  assert(result == 0);

  rtems_task_delete(RTEMS_SELF);
}
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions: `open`, `close`, `ioctl`.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of `<bsp/scet_rtems.h>` is required for accessing SCET device name `RTEMS_SCET_DEVICE_NAME` as well as other defines.

`CONFIGURE_APPLICATION_NEEDS_SCET_DRIVER` must be defined for using the SCET driver. By defining this as part of RTEMS configuration, the driver will automatically be initialized at boot up.

5.5. UART

5.5.1. Description

This driver is using the de facto standard interface for a 16550D UART given in [RD5] and as such has an 8-bit interface, but has been expanded to provide a faster and more delay-tolerant implementation.

5.5.1.1. RX/TX buffer depth

The RX and TX FIFOs have been expanded to 128 characters compared to the original specification of 16 characters. To be backwards compatible as well as being able to utilize the larger depth of the FIFOs, a new parameter has been brought in called buffer depth. The set buffer depth decides how much of the FIFOs real depth it should base its calculations on. Buffer depth affects both RX and TX FIFOs handling in the RTEMS driver.

5.5.1.2. Trigger levels

To be able to utilize the larger FIFOs, the meaning of the trigger levels have been changed. In the specification in [RD5], it defines the trigger levels as 1 character, 4 characters, 8 characters and 14 characters. This has now been changed to instead mean 1 character, 1/4 of the FIFO is full, 1/2 of the FIFO is full and the FIFO is 2 characters from the given buffer depth top. This results in the IP being fully backwards compatible, since a buffer depth of 16 characters would yield the same trigger levels as those given in [RD5].

5.5.2. RTEMS API

This API represents the driver interface of the module from an RTEMS user application's perspective.

The driver functionality is accessed through the RTEMS POSIX API for ease of usage. In case of a failure on a function call, the *errno* value is set for determining the cause.

5.5.2.1. Function int open(...)

Opens access to the requested UART. Only blocking mode is supported.
Upon each open call the device interface is reset to 115200 bps and its default mode according to the table below.

Argument name	Type	Direction	Description
pathname	const char *	in	The absolute path to the file that is to be opened. See table below for uart naming.
flags	Int	in	A bitwise 'or' separated list of values that determine the method in which the file is to be opened (whether it should be read only, read/write etc). See below.

Flags	Description
O_RDONLY	Open for reading only.
O_WRONLY	Open for writing only.
O_RDWR	Open for reading and writing.

Return value	Description
fd	A file descriptor for the device on success
-1	See *errno* values
errno values	
ENODEV	Device does not exist
EALREADY	Device is already open

Device name	Description
/dev/uart0	Ordinary UART, default mode RS422
/dev/uart1	Ordinary UART, default mode RS422
/dev/uart2	Ordinary UART, default mode RS422
/dev/uart3	Ordinary UART, default mode RS422
/dev/uart4	Ordinary UART, default mode RS422
/dev/uart_psu_control	PSU Control, RS485 only
/dev/uart_safe_bus	Safe bus, RS485 only

5.5.2.2. Function int close(...)

Closes access to the device and disables the line drivers.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**

Return value	Description
0	Device closed successfully

5.5.2.3. Function ssize_t read(…)

Read data from the UART. The call blocks until data is received from the UART RX FIFO. Please note that it is not uncommon for the read call to return less data than requested.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**
buf	void *	in	Pointer to character buffer to write data to
count	size_t	in	Number of characters to read

Return value	Description
> 0	Number of characters that were read.
0	A parity / framing / overflow error occurred. The RX data path has been flushed. Data was lost.
- 1	see *errno* values
errno values	

EPERM	Device not open
EINVAL	Invalid number of characters to be read

5.5.2.4. Function ssize_t write(…)

Write data to the UART. The write call is blocking until all data have been transmitted.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**
buf	const void *	in	Pointer to character buffer to read data from
count	size_t	in	Number of characters to write

Return value	Description
>= 0	Number of characters that were written.
- 1	see *errno* values
errno values	
EINVAL	Invalid number of characters to be written.

5.5.2.5. Function int ioctl(…)

Ioctl allows for toggling the RS422/RS485/Loopback mode and setting the baud rate. RS422/RS485 mode selection is not applicable for safe bus and power ctrl UARTs.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at **open**
cmd	int	in	Command to send
val	int	in	Value to write or a pointer to a buffer where data will be written.

Command table	Type	Direction	Description
UART_IOCTL_SET_BITRATE	uint32_t	in	Set the bitrate of the line interface. Possible values: UART_B375000 UART_B153600 UART_B115200 (default) UART_B76800 UART_B57600 UART_B38400 UART_B19200 UART_B9600 UART_B4800 UART_B2400 UART_B1200

UART_IOCTL_MODE_SELECT	uint32_t	in	Set the mode of the interface. Possible values: UART_RTEMS_MODE_RS422 (default) UART_RTEMS_MODE_RS485 UART_RTEMS_MODE_LOOPBACK (TX connected to RX internally)
UART_IOCTL_RX_FLUSH	uint32_t	in	Flushes the RX software FIFO
UART_IOCTL_SET_PARITY	uint32_t	in	Set parity. Possible values: UART_PARITY_NONE (default) UART_PARITY_ODD UART_PARITY_EVEN
UART_IOCTL_SET_BUFFER_DEPTH	uint32_t	in	Set the FIFO buffer depth. Possible values: UART_BUFFER_DEPTH_16 (default) UART_BUFFER_DEPTH_32 UART_BUFFER_DEPTH_64 UART_BUFFER_DEPTH_128
UART_IOCTL_GET_BUFFER_DEPTH	uint32_t*	out	Get the current buffer depth.
UART_IOCTL_SET_TRIGGER_LEVEL	uint32_t	in	Set the RX FIFO trigger level. Possible values: UART_TRIGGER_LEVEL_1 = 1 character UART_TRIGGER_LEVEL_4 = 1/4 full UART_TRIGGER_LEVEL_8 = 1/2 full UART_TRIGGER_LEVEL_14 = buffer_depth - 2 (default)
UART_IOCTL_GET_TRIGGER_LEVEL	uint32_t*	out	Get the current trigger level

Return value	Description
0	Command executed successfully
-1	see *errno* values
	errno values
RTEMS_NOT_DEFINED	Invalid IOCTL
EINVAL	Invalid value supplied to IOCTL

5.5.3. Usage description

The following #define needs to be set by the user application to be able to use the UARTs:

CONFIGURE_APPLICATION_NEEDS_UART_DRIVER

5.5.3.1. RTEMS application example

In order to use the uart driver in the RTEMS environment, the following code structure is suggested to be used:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/uart_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_UART_DRIVER
#define CONFIGURE_SEMAPHORES 40

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

#define CONFIGURE_INIT
rtems_task Init (rtems_task_argument argument);

rtems_task Init (rtems_task_argument ignored){}
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions: `open, close, ioctl`.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of `<bsp/uart_rtems.h>` is required for accessing the uarts.

5.5.3.2. Parity, framing and overrun error notification

Upon receiving a parity, framing or an overrun error the read call returns 0 and the internal RX queue is flushed.

5.5.4. Limitations

8 data bits only.
1 stop bit only.
No hardware flow control support.

5.6. Mass memory

5.6.1. Description

This section describes the mass memory driver's design and usability.

This API represents the driver interface from a user application's perspective for the RTEMS driver.

The driver functionality is accessed through RTEMS POSIX API for ease of usage. In case of failure on a function call, *errno* value is set for determining the cause.

5.6.2. Data Structures

5.6.2.1. Struct massmem_cid_t

This struct is used as the target for reading the mass memory chip IDs.

Type	Name	Purpose
Array of 5 uint8_t	edac	Byte array for EDAC chip ID
Array of 5 uint8_t	chip0	Byte array for chip 0 ID
Array of 5 uint8_t	chip1	Byte array for chip 1 ID
Array of 5 uint8_t	chip2	Byte array for chip 2 ID
Array of 5 uint8_t	chip3	Byte array for chip 3 ID

5.6.2.2. Struct massmem_error_injection_t

This struct is used as a specification when manually injecting errors when writing to the the mass memory.

Type	Name	Purpose
uint8_t	edac_error_injection	Bits to be XOR:ed with generated EDAC byte
uint32_t	data_error_injection	Bits to be XOR:ed with supplied data

5.6.2.3. Struct massmem_ioctl_spare_area_args_t

This struct is used by the RTEMS API as the target when reading from spare area and data simultaneously.

Type	Name	Purpose
uint32_t	page_num	What page to read
uint32_t *	data_buf	Pointer to buffer in which the data is to be stored
uint8_t *	edac_buf	Pointer to buffer in which the EDAC data is to be stored
uint32_t	size	Size to read in bytes

5.6.2.4. Struct massmem_ioctl_error_injection_args_t

This structure is used by the RTEMS API in order to perform a special write call to inject errors into the mass memory.

Type	Name	Purpose
uint32_t	page_num	What page to write
uint8_t *	data_buf	Pointer to data to write
uint32_t	size	Size of data to write in bytes

massmem_error_injection_t *	error_injection	Pointer to error injection struct. See 5.6.2.2 for definition

5.6.3. RTEMS API

5.6.3.1. int open(…)

Opens access to the driver. The device can only be opened once at a time.

Argument name	Type	Direction	Description
filename	char *	in	The absolute path to the file that is to be opened. Mass memory device is defined as MASSMEM_DEVICE_NAME.
oflags	int	in	Device must be opened by exactly one of the symbols defined in Table 5-4.

Return value	Description
>0	A file descriptor for the device.
- 1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor
ENOENT	Invalid filename
EEXIST	Device already opened.

Table 5-4 - Open flag symbols

Symbol	Description
O_RDONLY	Open for reading only
O_WRONLY	Open writing only
O_RDWR	Open for reading and writing

5.6.3.2. int close(…)

Closes access to the device.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.

Return value	Description
0	Device closed successfully

-1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor

5.6.3.3. off_t lseek(…)

Sets page offset for read/ write operations.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.
offset	off_t	in	Page number.
whence	int	in	Must be set to SEEK_SET.

Return value	Description
offset	Page number
- 1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor
ESPIPE	*fd* is associated with a pipe, socket or FIFO.
EINVAL	*whence* is not a proper value.
EOVERFLOW	The resulting file offset would overflow **off_t**.

5.6.3.4. ssize_t read(…)

Reads requested size of bytes from the device starting from the offset set in `lseek`.

Note! For iterative read operations, `lseek` must be called to set page offset ***before*** each read operation.

Note! The character buffer location handed to `read` must be 32-bit aligned.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.
buf	void *	in	Character buffer where to store the data
nbytes	size_t	in	Number of bytes to read into *buf.*

Return value	Description
>0	Number of bytes that were read.
- 1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor
EINVAL	Page offset set in `lseek` is out of range or *nbytes* is too large and reaches a page that is out of range.
EBUSY	Device is busy with previous read/write operation.

5.6.3.5. ssize_t write(…)

Writes requested size of bytes to the device starting from the offset set in `lseek`.

Note! For iterative write operations, `lseek` must be called to set page offset before each write operation.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.
buf	void *	in	Character buffer to read data from
nbytes	ssize_t	in	Number of bytes to write from *buf*.

Return value	Description
>0	Number of bytes that were written.
- 1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor
EINVAL	Page offset set in **lseek** is out of range or *nbytes* is too large and reaches a page that is out of range.
EAGAIN	Driver failed to write data. Try again.

EIO	Failed to write data. Block should be marked as a bad block.

5.6.3.6. int ioctl(…)

Additional supported operations via POSIX Input/Output Control API.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.
cmd	ioctl_command_t	in	Command specifier
(varies)	(varies)	(varies)	Command-specific argument

The following return and errno values are common for all commands except.

Return value	Description
0	Operation successful (or block is marked ok in case of bad block check)
-EBUSY	Device is busy with previous read/write operation.
-1	See errno values
errno values	
ENODEV	Internal RTEMS error
EIO	Internal RTEMS error

5.6.3.6.1. Reset mass memory device

Resets the mass memory device.

Command	Value type	Direction	Description
MASSMEM_IO_RESET	n/a	n/a	n/a

5.6.3.6.2. Read status data

Reads the status register value.

Command	Value type	Direction	Description
MASSMEM_IO_READ_DATA_STATUS	uint32_t*	out	Pointer to variable in which status data is to be stored.

5.6.3.6.3. Read control status data

Reads the control status register value.

Command	Value type	Direction	Description
MASSMEM_IO_READ_CTRL_STATUS	uint8_t*	out	Pointer to variable in which control status data is to be stored.

5.6.3.6.4. Read EDAC register data

Reads the EDAC register value.

Command	Value type	Direction	Description
MASSMEM_IO_READ_EDAC_STATUS	uint8_t*	out	Pointer to variable in which control status data is to be stored.

5.6.3.6.5. Read ID

Reads the chip IDs

Command	Value type	Direction	Description
MASSMEM_IO_READ_ID	massmem_cid_t.*	out	Pointer to struct in which ID is to be stored, see 5.6.2.1.

5.6.3.6.6. Erase block

Erases a block

Command	Value type	Direction	Description
MASSMEM_IO_ERASE_BLOCK	uint32_t	in	Block number

Return value	Description
-EINVAL	The block number is out of range
-EIO	Failed to erase block. Block should be marked as a bad block

5.6.3.6.7. Read spare area

Reads the spare area with given data.

Command	Value type	Direction	Description
MASSMEM_IO_READ_SPARE_AREA	massmem_ioctl_spare_area_args_t*	in/out	Pointer to struct with input page number specifier, and destination buffers where spare area data is to be stored, see 5.6.2.3

Return value	Description
-EINVAL	Indicates one or more of: • The page number is out of range • Size is 0 • Size is larger than page size • Size is not a multiple of 4 • The data or EDAC buffer is NULL The data or EDAC buffer is not 4-byte aligned
-EIO	Reading timed out or read status indicated failure.

5.6.3.6.8. Bad block check

Reads the factory bad block status from a block.

Note that this only gives information about factory bad blocks; subsequent bad block status is not included in this information.

Command	Value type	Direction	Description
MASSMEM_IO_BAD_BLOCK_CHECK	uint32_t	in	Block number.

Return value	Description
0	Block is marked ok.
1	Block is marked as bad.
-EINVAL	The page number is out of range, buffers are NULL or not 4-byte aligned.

5.6.3.6.9. Error Injection

Injects errors in page write command call. The purpose is to test error corrections (EDAC).

Command	Value type	Direction	Description
MASSMEM_IO_ERROR_INJECTION	massmem_ioctl_error_injection_args_t*	in	Pointer to struct with program page arguments as defined in 5.6.2.4

Return value	Description
-EINVAL	Indicates one or more of: • The page number is out of range • Size is 0 • Size is larger than page size • Size is not a multiple of 4 • The data or EDAC buffer is NULL The data buffer is not 4-byte aligned
-EIO	The mass memory write operation failed, the block should be marked as a bad block

5.6.4. Usage

5.6.4.1. RTEMS

5.6.4.1.1. Overview

The RTEMS driver accesses the mass memory by the reference a page number. There are MASSMEM_BLOCKS blocks starting from block number 0 and MASSMEM_PAGES_PER_BLOCK pages within each block starting from page 0. Each page is of size MASSMEM_PAGE_SIZE bytes.

When writing new data into a page, the memory area must be in its reset value. If there is data that was previously written to a page, the block where the page resides must first be erased in order to clear the page to its reset value. **Note** that the whole block is erased, not only the page.

It is the user application’s responsibility to make sure any data the needs to be preserved after the erase block operation must first be read and rewritten after the erase block operation, with the new page information.

5.6.4.1.2. Usage

The RTEMS driver must be opened before it can access the mass memory flash device. Once opened, all provided operations can be used as described in the subchapters 5.6.3.3. to 5.6.3.6. And, if desired, the access can be closed when not needed.

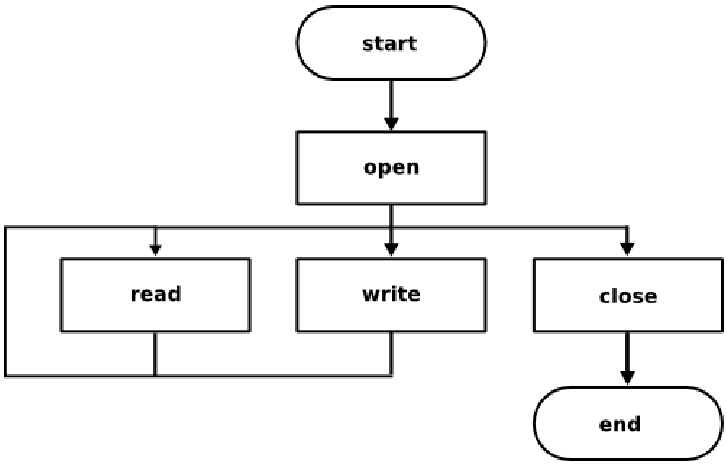


Figure 5-5 - RTEMS driver usage description

Note! All calls to RTEMS driver are blocking calls.

5.6.4.2. RTEMS application example

In order to use the mass memory flash driver in RTEMS environment, the following code structure is suggested to be used:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/massmem_flash_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_MASS_MEMORY_FLASH_DRIVER
.
.
#define CONFIGURE_INIT
rtems_task Init (rtems_task_argument argument);

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

static uint8_t buf[MASSMEM_PAGE_SIZE];

rtems_task Init (rtems_task_argument ignored)
{
  .
  fd = open(MASSMEM_DEVICE_NAME, O_RDWR);
  .
  s = lseek(fd, page_number, SEEK_SET);
  .
  sz = write(fd, buf, MASSMEM_PAGE_SIZE);
  .
  lseek(fd, page_number, SEEK_SET)
  .
  sz = read(fd, buf, MASSMEM_PAGE_SIZE);
  .
}
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions `open`, `close`, `lseek`, `read` and `write` and `ioctl` functions for accessing driver.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of <`bsp/massmem_flash_rtems.h`> is required for driver related definitions .

Inclusion of <`bsp/bsp_confdefs.h`> is required to initialise the driver at boot up.

`CONFIGURE_APPLICATION_NEEDS_MASSMEM_FLASH_DRIVER` must be defined for using the driver. This will automatically initialise the driver at boot up.

5.6.5. Error injection

Error injection is used to verify the EDAC capabilities of the IP.
The IP always writes/reads 8 32-bit data words. If less or an uneven amount of data is requested from the application the drivers pads this internally.
To ensure that the memory can withstand a full byte corruption of data the 8 words of data are interleaved over the mass memory chips. This is done transparently from the user perspective except when writing the error injection vector.
Looking at the massmem_error_injection_t struct defined in 5.6.2.2:
the data_error_injection member is an uint32_t.
Bit 0 of byte 0, 1, 2, 3 affects the first data word.
Bit 1 of byte 0, 1, 2, 3 affects the second data word.
…
Bit 7 of byte 0, 1, 2, 3 affects the eight data word.

To inject a correctible error in the third data word flip either bit 2, 10, 18 or 26.
To inject an uncorrectible in the third data word flip two bits of either 2, 10, 18, 26.

5.6.6. Limitations

The mass memory flash driver may only have one open file descriptor at a time.

5.7. Spacewire

5.7.1. Description

This section describes the SpaceWire driver's design and usability.

5.7.2. RTEMS API

This API represents the driver interface from a user application's perspective for the RTEMS driver.

The driver functionality is accessed through RTEMS POSIX API for ease of use. In case of failure on a function call, *errno* value is set for determining the cause. Additional functionalities are supported via POSIX Input/Output Control API as described in subchapter 5.7.2.5.

5.7.2.1. int open(…)

Registers the application to the device name for data transactions. Although multiple accesses for data transaction is allowed, only one access per unique device name is valid. Device name must be set with a logical number as described in usage description in subchapter 5.7.3.1.

Argument name	Type	Direction	Description
filename	char *	in	Device name to register to for data transaction.
oflags	int	in	Device must be opened by exactly one of the symbols defined in Table 5-5.

Return value	Description
>0	A file descriptor for the device.
- 1	see *errno* values
errno values	
ENOENT	Invalid device name
EEXIST	Device already opened.
EEGAIN	Opening of device failed due to internal error. Try again.

Table 5-5 - Open flag symbols

Symbol	Description
O_RDONLY	Open for reading only
O_WRONLY	Open writing only
O_RDWR	Open for reading and writing

5.7.2.2. int close(…)

Deregisters the device name from data transactions.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.

Return value	Description
0	Device name deregistered successfully
-1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor

5.7.2.3. size_t read(…)

Reads a packet when available.

Note! This call is blocked until a package for the logic address is received. In addition, only **one** task must access one file descriptor at a time. Multiple task accessing the same file descriptor is not allowed.

Note! buf reference **must** be aligned to a 32 bit aligned address.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.
buf	void *	in	Character buffer where to store the packet
nbytes	size_t	in	Packet size in bytes. Must be between 0 and `SPWN_MAX_PACKET_SIZE bytes`.

Return value	Description
>0	Received size of the actual packet. Can be less than *nbytes*.
0	Buffer size was lower than received packet size. Errno value is set to EOVERFLOW.
- 1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor
EINVAL	Packet size is 0 or larger than `SPWN_MAX_PACKET_SIZE`
ETIMEDOUT	Timeout received. Received packet is incomplete.

5.7.2.4. size_t write(…)

Transmits a packet.

Note! This call is blocked till the package is transmitted.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at `open`.
buf	void *	in	Character buffer containing the packet.
nbytes	size_t	in	Packet size in bytes. Must be between 0 and `SPWN_MAX_PACKET_SIZE bytes`.

Return value	Description
>0	Number of bytes that were transmitted.
≤0	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor
EINVAL	Packet size is 0 or larger than `SPWN_MAX_PACKET_SIZE`.
ETIMEDOUT	Failed to transmit the complete packet.
EIO	Internal error

5.7.2.5. int ioctl(…)

Additional supported operations via POSIX Input/Output Control API.

Argument name	Type	Direction	Description
fd	int	in	A file descriptor received at `open`.
cmd	int	in	Command defined in subchapter 5.7.2.5.1
value	void *	in	The value relating to command operation as defined in subchapter 5.7.2.5.1.

5.7.2.5.1. Mode setting

Sets the device into the given mode.

Note! The mode setting affects the SpaceWire device and therefore all file descriptors registered to it.

Command	Type	Direction	Description
SPWN_IOCTL_MODE_SET	uint32_t	in	Modes available: • SPWN_IOCTL_MODE_OFF: Turns off the node. • SPWN_IOCTL_MODE_LOOPBACK: Internal loopback mode SPWN_IOCTL_MODE_NORMAL: Normal mode.

Return value	Description
0	Given mode was set
- 1	see *errno* values
errno values	
EINVAL	Invalid mode.

5.7.3. Usage description

5.7.3.1. Overview

The driver provides SpaceWire link setup and data transaction via the SpaceWire device. Each application that wants to communicate via the SpaceWire device must register with a logical address.

The logical address is tied to a device number. To register to the device, the application must use the predefined string SPWN_DEVICE_0_NAME_PREFIX with a chosen logical address to register itself to the driver. See code example in subchapter 5.7.3.3. The registration is done by function `open` and deregistered by the function `close`.

Only one logical address can be registered at a time yet multiple logical addresses can be used at the same time within an application.

Logical addresses between 0 – 31 and 255 are reserved by the ESA's ECSS SpaceWire standard [RD2] and cannot be registered to.

Note! A reception packet buffer must be aligned to 4 bytes in order to handle the packet's reception correctly. It is therefore recommended to assign the reception buffer in the following way:

```
uint8_t __attribute__ ((aligned (SPWN_RX_PACKET_ALIGN_BYTES))
buf_rx[PACKET_SIZE];
```

5.7.3.2. Usage

The application must first register to a device name before it can be accessed for data transaction. Once registered via function `open`, all provided operations can be used as described in the subchapter 5.7.2. If desired, the access can be closed when not needed.

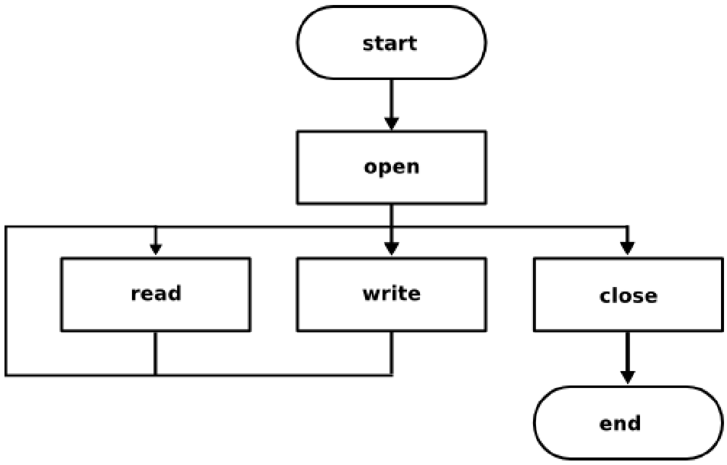


Figure 5-6 - RTEMS driver usage description

Note! All calls to RTEMS driver are blocking calls.

Note! Data rate is dependent on the maximum packet size and packet transmission rate that is limited by SpaceWire IP core. This simply results in effect to that the packet size is proportionate to data rate i.e. the larger the packet size, the higher the data rate.

5.7.3.3. RTEMS application example

In order to use the driver in RTEMS environment, the following code structure is suggested to be used:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/spacewire_node_rtems.h>

.
.
#define CONFIGURE_APPLICATION_NEEDS_SPACEWIRE_DRIVER

#define RESOURCES_MEM_SIZE (512*1024) /* 1 Mb */
#define CONFIGURE_EXECUTIVE_RAM_SIZE RESOURCES_MEM_SIZE
#define CONFIGURE_MAXIMUM_TIMERS 1 /* Needed by driver */
.
.
#define CONFIGURE_INIT
rtems_task Init (rtems_task_argument argument);

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

uint8_t __attribute__ ((aligned (SPWN_RX_PACKET_ALIGN_BYTES)))
    buf_rx[SPWN_MAX_PACKET_SIZE];
uint8_t buf_tx[SPWN_MAX_PACKET_SIZE];

rtems_task Init (rtems_task_argument ignored)
{
  .
  fd = open(SPWN_DEVICE_0_NAME_PREFIX"42", O_RDWR);
  .
}
```

The above code registers the application for using the unique device name with the logical address 42 (`SPWN_DEVICE_0_NAME_PREFIX"42")` for data transaction.

Two buffers, `buf_tx` and `buf_rx`, are aligned with `CPU_STRUCTURE_ALIGNMENT` for correctly handling DMA access regarding transmission and reception of a SpaceWire packet.

Inclusion of `<fcntl.h>` and `<unistd.h>` are required for using the POSIX functions `open`, `close`, `read` and `write` and `ioctl` functions for accessing the driver.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of `<bsp/spacewire_node_rtems.h>` is required for driver related definitions.

Inclusion of `<bsp/bsp_confdefs.h>` is required to initialise the driver at boot up.

`CONFIGURE_APPLICATION_NEEDS_SPACEWIRE_DRIVER` must be defined for using the driver. This will automatically initialise the driver at boot up.

`CONFIGURE_EXECUTIVE_RAM_SIZE` must also be defined for objects needed by the driver.

5.8. GPIO

5.8.1. Description

This driver software for the GPIO IP handles the setting and reading of general purpose input/output pins. It implements the standard set of device file operations according to [RD7].

The GPIO IP has, apart from logical pin and input/output operations, also a number of other features.

5.8.1.1. Falling and rising edge detection

Once configured, the GPIO IP can detect rising or falling edges on a pin and alert the driver software by the means of an interrupt.

5.8.1.2. Time stamping in SCET

Instead, or in addition to the interrupt, the GPIO IP can also signal the SCET to sample the current timer when a rising or falling edge is detected on a pin. Reading the time of the timestamp requires interaction with the SCET and exact register address depends on the current board configuration. One SCET sample register is shared by all GPIOs.

5.8.1.3. RTEMS differential mode

In RTEMS finally, a GPIO pin can also be set to operate in differential mode on output only. This requires two pins working in tandem and if this functionality is enabled, the driver will automatically adjust the setting of the paired pin to output mode as well. The pins are paired in logical sequence, which means that pin 0 and 1 are paired as are pin 2 and 3 etc. Thus, in differential mode it is recommended to operate on the lower numbered pin only to avoid confusion. Pins can be set in differential mode on specific pair only, i.e. both normal single ended and differential mode pins can operate simultaneously (though not on the same pins obviously).

5.8.1.4. Operating on pins with pull-up or pull-down

For scenarios when one or multiple pins are connected to a pull-up or pull-down (for e.g. open-drain operation), it's recommended that the output value of such a pin should always be set to 1 for pull-down or 0 for pull-up mode. The actual pin value should then be selected by switching between input or output mode on the pin to comply with the external pull feature.

5.8.2. RTEMS API

This API represents the driver interface of the module from an RTEMS user application's perspective.

The driver functionality is accessed through the RTEMS POSIX API for ease of use. In case of a failure on a function call, the *errno* value is set for determining the cause.

5.8.2.1. Function int open(...)

Opens access to the specified GPIO pin, but do not reset the pin interface and instead retains the settings from any previous access.

Argument name	Type	Direction	Description
pathname	const char *	in	The absolute path to the GPIO pin to be opened. All possible paths are given by "/dev/gpioX" where X matches 0-31. The actual number of devices available depends on the current hardware configuration.
flags	int	in	Access mode flag, O_RDONLY, O_WRONLY or O_RDWR.

Return value	Description
Fildes	A file descriptor for the device on success
-1	See *errno* values
errno values	
EALREADY	Device is already open
EINVAL	Invalid options

5.8.2.2. Function int close(...)

Closes access to the GPIO pin.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.

Return value	Description
0	Device closed successfully
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.8.2.3. Function ssize_t read(...)

Reads the current value of the specified GPIO pin. If no edge detection have been enabled, this call will return immediately. With edge detection enabled, this call will block with a timeout until the pin changes status such that it triggers the edge detection. The timeout can be adjusted using an ioctl command, but defaults to zero - blocking indefinitely, see also 5.8.2.5.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.

buf	void*	in	Pointer to character buffer to put the read data in.
count	size_t	in	Number of bytes to read, must be set to 1.

Return value	Description
>=0	Number of bytes that were read.
-1	See *errno* values
errno values	
EINVAL	Invalid options
ETIMEDOUT	Driver timed out waiting for the edge detection to trigger

5.8.2.4. Function ssize_t write(...)

Sets the output value of the specified GPIO pin. If the pin is in input mode, the write is allowed, but its value will not be reflected on the pin until it is set in output mode.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
buf	const void*	in	Pointer to character buffer to get the write data from.
count	size_t	in	Number of bytes to write, must be set to 1.

Return value	Description
>=0	Number of bytes that were written.
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.8.2.5. Function int ioctl(...)

The input/output control function can be used to configure the GPIO pin as a complement to the simple data settings using the read/write file operations.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
cmd	int	in	Command to send.
val	void *	in/out	Data according to the specific command.

Command table	Type	Direction	Description
GPIO_IOCTL_GET_DIRECTION	uint32_t	out	Get input/output direction of the pin. '0' output mode '1' input mode
GPIO_IOCTL_SET_DIRECTION	uint32_t	in	Set input/output direction of the pin. '0' output mode '1' input mode

GPIO_IOCTL_GET_FALL_EDGE_DETECTION	uint32_t	out	Get falling edge detection status of the pin. '0' detection disabled '1' detection enabled
GPIO_IOCTL_SET_FALL_EDGE_DETECTION	uint32_t	in	Set falling edge detection configuration of the pin. '0' detection disabled '1' detection enabled
GPIO_IOCTL_GET_RISE_EDGE_DETECTION	uint32_t	out	Get rising edge detection status of the pin. '0' detection disabled '1' detection enabled
GPIO_IOCTL_SET_RISE_EDGE_DETECTION	uint32_t	in	Set rising edge detection configuration of the pin. '0' detection disabled '1' detection enabled
GPIO_IOCTL_GET_TIMESTAMP_ENABLE	uint32_t	out	Get timestamp enable status of the pin. '0' timestamp disabled '1' timestamp enabled
GPIO_IOCTL_SET_TIMESTAMP_ENABLE	uint32_t	in	Set timestamp enable configuration of the pin. '0' timestamp disabled '1' timestamp enabled
GPIO_IOCTL_GET_DIFF_MODE	uint32_t	out	Get differential mode status of the pin. '0' normal, single ended, mode '1' differential mode
GPIO_IOCTL_SET_DIFF_MODE	uint32_t	in	Set differential mode configuration of the pin. '0' normal, single ended, mode '1' differential mode
GPIO_IOCTL_GET_EDGE_TIMEOUT	uint32_t	out	Get the edge trigger timeout value in ticks. Defaults to zero which means wait indefinitely.
GPIO_IOCTL_SET_EDGE_TIMEOUT	uint32_t	in	Set the edge trigger timeout value in ticks. Zero means wait indefinitely.

Return value	Description
0	Command executed successfully
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.8.3. Usage description

5.8.3.1. RTEMS application example

The following #define needs to be set by the user application to be able to use the GPIO:

CONFIGURE_APPLICATION_NEEDS_GPIO_DRIVER

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/gpio_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_GPIO_DRIVER

#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_USE_IMFS_AS_BASE_FILESYSTEM

#define CONFIGURE_MAXIMUM_DRIVERS 15
#define CONFIGURE_MAXIMUM_SEMAPHORES 20
#define CONFIGURE_LIBIO_MAXIMUM_FILE_DESCRIPTORS 30

#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_MAXIMUM_TASKS 20

#define CONFIGURE_INIT

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

rtems_task Init (rtems_task_argument argument) {
  rtems_status_code status;
  int gpio_fd;
  uint32_t buffer;
  uint32_t config;
  ssize_t size;

  gpio_fd = open("/dev/gpio0", O_RDWR);
  config = GPIO_DIRECTION_IN;
  status = ioctl(gpio_fd, GPIO_IOCTL_SET_DIRECTION,
                 &config);
  size = read(gpio_fd, &buffer, 1);
  status = close(gpio_fd);
}
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions: `open`, `close, read, write and ioctl`.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of `<bsp/gpio_rtems.h>` is required for accessing the GPIO.

5.8.4. Limitations

Differential mode works on output only.

5.9. CCSDS

5.9.1. Description

This section describes the driver as a utility for accessing the CCSDS IP.

On the telemetry, the frames are encoded with Reed Solomon encoding that conforms to the CCSDS standard with a (255-223) RS encoder implementation and an interleaving depth of 5. That makes a total frame length of 1115 bytes. The standard RS polynomial is used.

On the telecommands the BCH decoder (63-56) supports the error correcting mode.

The driver can be configured to handle all available interrupts from the CCSDS IP:

- Pulse commands (CPDU)
- Timestamping of telemetry sent on virtual channel 0
- DMA transfer finished.
- Telemetry transfer frame error.
- Telecommand rejection due to error in the incoming telecommand.
- Telecommand frame buffer errors.
- Telecommand frame buffer overflow.
- Telecommand successfully received.

5.9.2. RTEMS API

This API represents the driver interface from a user application's perspective for the RTEMS driver.

The driver functionality is accessed through the RTEMS POSIX API for ease of use. In case of failure on a function call, *errno* value is set for determining the cause.

Access to the CCSDS-driver from an application is provided by three different device-files:

- "/dev/ccsds" that is used for configuration and status for common TM and TC functionality in the IP. Is defined as CCSDS_NAME

- "/dev/ccsds-tm" that is used for functions related to handling of Telemetry. Is defined as CCSDS_NAME_TM

- "/dev/ccsds-tc" that is used for functions related to handling of Telecommands. Is defined as CCSDS_NAME_TC

During initialization of the CCSDS driver, a default setup described in 7.1 is configured. All available interrupts are enabled.

5.9.2.1. Datatype struct tm_frame_t

This datatype is a struct representing a telemetry transfer frame. The elements are described in the table below:

Element	Size(in bits)	Description
transfer_frame_version_no	2	The transfer frame version number
scid	10	The SCID
vcid	3	The virtual channel id of the TM frame

vcf_flag	1	The OCF-flag
mcfc	8	The master channel frame counter
vcfc	8	The virtual channel frame counter
tr_frame_sec_head_flag	1	The transfer frame secondary header flag
tr_frame_sync_flag	1	The transfer frame sync flag
tr_frame_packet_ord_flag	1	The transfer frame packet order flag
segment_length_id	2	The segment length id
first_header_pointer	11	The first header pointer
data_field	1103*8	The data field of the TM frame
clcw	32	The CLCW
crc	16	The CRC

5.9.2.2. Datatype struct tc_frame_t

This datatype is a struct representing a telecommand transfer frame. The elements are described in the table below:

Element	Size (in bits)	Description
transfer_frame_version_no	2	The transfer frame version number
bypass_flag	1	The bypass flag
control_command_flag	1	The control command flag
Spare	2	Reserved for future use
Scid	10	The SCID
Vcid	6	The virtual channel id
frame_length	10	The TC frame length
data_field	1017*8	The data field of the TC frame
Crc	16	The CRC

5.9.2.3. Data type dma_descriptor_t

This datatype is a struct for DMA descriptors. The elements of the struct are described below:

Element	Type	Description
desc_no	uint32_t	The descriptor number (0-31)
desc_config	uint32_t	The configuration of the DMA descriptor
desc_adress	uint32_t	The configuration of the DMA address descriptor

5.9.2.4. Data type tm_config_t

This datatype is a struct for configuration of the TM path. The elements of the struct are described below:

Element	Type	Description
clk_divisor	uint8_t	The divisor of the clock
tm_enabled	uint8_t	Enable/disable of telemetry 0 - Disable 1 - Enable
fecf_enabled	uint8_t	Enable/disable of FECF 0 - Disable 1 - Enable

mc_cnt_enabled	uint8_t	Enable/Disable of master channel frame counter 0 - Disable 1 - Enable
idle_frame_enabled	uint8_t	Enable/disable of generation of Idle frames 0 - Disable 1 - Enable
ocf_clcw_enabled	uint8_t	Enable/disable of OCF/CLCW in TM Transfer frames 0 – Disable 1 – Enable
tm_conv_bypassed	uint8_t	Bypassing of the TM convolutional encoder 0 - No bypass 1 - Bypass
tm_pseudo_rand_bypassed	uint8_t	Bypassing of the TM pseudo randomizer encoder 0 - No bypass 1 - Bypass
tm_rs_bypassed	uint8_t	Bypassing of the TM Reed Solomon encoder 0 - No bypass 1 - Bypass

5.9.2.5. Data type tc_config_t

This datatype is a struct for configuration of the TC path. The elements of the struct are described below:

Element	Type	Description
tc_derandomizer_bypassed	uint8_t	Bypassing of TC derandomizer. 0 - No bypass 1 - Bypass

5.9.2.6. Data type tc_status_t

This datatype is a struct to store status parameters of the TC path. The elements of the struct are described below:

Element	Type	Description
tc_frame_cnt	uint8_t	Number of received TC frames. The counter will wrap around after 2^8-1.
tc_buffer_cnt	uint16_t	Actual length on the read TC buffer data in bytes. MAX val 1024 bytes.
cpdu_line_status	uint16_t	Bits 0-11 show if the corresponding pulse command line was activated by the last command.
cpdu_bypass_cnt	uint8_t	Indicates the number of accepted commands. Wraps at 15.

5.9.2.7. int open(…)

Opens the devices provided by the CCSDS RTEMS driver. The device can only be opened once at a time.

Argument name	Type	Direction	Description
Filename	char *	in	The absolute path to the file that is to be opened. Shall be CCSDS_NAME, CCSDS_NAME_TM or CCSDS_NAME_TC
Oflags	int	in	A bitwise 'or' separated list of values that determine the method in which the file is to be opened (whether it should be read only, read/write, whether it should be cleared when opened, etc). See a list of legal values for this field at the end.

Return value	Description
≥0	A file descriptor for the device on success
- 1	see *errno* values
errno values	
EBUSY	If device already opened
EPERM	If wrong permissions
ENOENT	Bad file descriptor

5.9.2.8. int close(…)

Closes access to the device.

Argument name	Type	Direction	Description
Fd	int	in	File descriptor received at **open**

Return value	Description
0	Device closed successfully
-1	see *errno* values
errno values	
ENOENT	Bad file descriptor

5.9.2.9. size_t write(…)

To send a Telemetry Transfer frame a write-operation on device “/dev/ccsds-tm” shall be used. The TM frame to send is passed as a pointer to a variable of type tm_frame_t.

Argument name	Type	Direction	Description
Fd	int	in	File descriptor received at **open**
Buf	void *	in	Character buffer to read data from
Nbytes	size_t	in	Number of bytes to write to the device.

Return value	Description
≥0	number of bytes that were written.
- 1	see *errno* values
errno values	
EINVAL	Wrong arguments
EIO	A physical access on the device failed

5.9.2.10. size_t read(…)

To read a Telecommand Transfer frame a read-operation on device "/dev/ccsds-tc" shall be used. The read Telecommand Transfer frame is passed as a pointer to a variable of type tc_frame_t. This call is blocking until a Telecommand Transfer Frame is received.

Argument name	Type	Direction	Description
Fd	int	in	File descriptor received at **open**
Buf	void *	in	Character buffer where read data is returned
Nbytes	size_t	in	Number of bytes to write from the

Return value	Description
≥0	Number of bytes that were read.
- 1	see *errno* values
errno values	
EINVAL	Wrong arguments
EIO	A physical access on the device failed

5.9.2.11. int ioctl(…)

The devices provided by the CCSDS driver support different IOCTL's.

Argument name	Type	Direction	Description
Fd	int	in	File descriptor received at **open**
Cmd	int	in	Command to send
Val	void *	in	The parameter to pass is depended on which IOCTL is called. Is described in table below.

Command table	Device	Parameter type	Description
CCSDS_SET_TM_CONFIG	/dev/ccsds-tm	tm_config_t	Sets a configuration of the TM path. See 5.9.2.4
CCSDS_GET_TM_CONFIG	/dev/ccsds-tm	tm_config_t *	Returns the configuration of the TM path. See 5.9.2.4
CCSDS_SET_TC_CONFIG	/dev/ccsds-tc	tc_config_t	Sets a configuration of the TC path. See 5.9.2.5
CCSDS_GET_TC_CONFIG	/dev/ccsds-tc	tc_config_t *	Returns the configuration of the TC path. 5.9.2.5
CCSDS_SET_DMA_CONFIG	/dev/ccsds-tm	uint32_t	Set a configuration of the DMA register.

CCSDS_GET_DMA_CONFIG	/dev/ccsds-tm	uint32_t*	Returns a configuration of the DMA register.
CCSDS_SET_DMA_DESC	/dev/ccsds-tm	dma_descriptor_t	Configures a DMA-descriptor in the range (0-31). See 5.9.2.3
CCSDS_GET_DMA_DESC	/dev/ccsds-tm	dma_descriptor_t*	Returns the configuration of a DMA-descriptor in the range (0-31). See 5.9.2.3
CCSDS_GET_TM_STATUS	/dev/ccsds-tm	uint32_t*	Gets status of TM path.
CCSDS_GET_TM_ERR_CNT	/dev/ccsds-tm	uint32_t*	Gets the TM error counter.
CCSDS_GET_TC_ERR_CNT	/dev/ccsds-tc	uint32_t*	Gets the TC error counter.
CCSDS_GET_TC_STATUS	/dev/ccsds-tc	tc_status_t*	Gets status of TC path.
CCSDS_SET_TC_BUF_CTRL	/dev/ccsds-tc	uint32_t	Set the TC buffer control register.
CCSDS_ENABLE_TM	/dev/ccsds-tm	N.A	Enables TM.
CCSDS_DISABLE_TM	/dev/ccsds-tm	N.A	Disable TM.
CCSDS_ENABLE_DMA	/dev/ccsds-tm	N.A.	Enables DMA transfers.
CCSDS_DISABLE_DMA	/dev/ccsds-tm	N.A	Disables DMA transfers.
CCSDS_INIT	/dev/ccsds	N.A.	Sets a default configuration of CCSDS IP.
CCSDS_SET_CLCW	/dev/ccsds-tm	uint32_t	Sets the CLCW of TM frames
CCSDS_GET_CLCW	/dev/ccsds-tm	uint32_t*	Get the CLCW of TM frames.

Return value	Description
0	Command executed successfully
-1	see *errno* values
errno values	
ENOENT	Bad file descriptor
EINVAL	Invalid I/O command

5.9.3. Usage description

5.9.3.1. Send Telemetry

1. Open the device “/dev/ccsds-tm”, “/dev/ccsds-tc” and “/dev/ccsds”. Set up the TM path by ioctl-call CCSDS_SET_TM_CONFIG on device “/dev/ccsds-tm” or ioctl CCSDS_INIT on device “/dev/ccsds”
2. Prepare DMA-descriptors by ioctl CCSDS_SET_DMA_DESC on device “/dev/ccsds-tm”.
3. Enable TM by ioctl CCSDS_ENABLE_TM on device “/dev/ccsds-tm”.
4. Prepare the content in SDRAM that will be fetched by DMA-transfer by writing to “/dev/ccsds-tm”

5.9.3.2. Receive Telecommands

1. Open the device “/dev/ccsds-tm”, “/dev/ccsds-tc” and “/dev/ccsds”. Set up the TC path by ioctl-call CCSDS_SET_TC_CONFIG on device “/dev/ccsds-tc” or or ioctl CCSDS_INIT on device “/dev/ccsds”
2. Do a read from “/dev/ccsds-tc”. This call will block until a new TC has been received.

5.9.3.3. Application configuration

Inclusion of <fcntl.h> and <unistd.h> are required for using the POSIX functions open(), close(), read(), write() and ioctl() to access the CCSDS device.

Inclusion of <errno.h> is required for retrieving error values on failures.

Inclusion of <bsp/ccsds_rtems.h> is required for data-types, definitions of IOCTL of device CCSDS.

CONFIGURE_APPLICATION_NEEDS_CCSDS_DRIVER must be defined to use the CCSDS driver from the application.

5.10. ADC

5.10.1. Description

This section describes the driver for accessing the ADC device.

The following ADC channels are available for the Sirius OBC:

Parameter	Abbreviation	ADC channel
Analog input	ADC in 0	0
Analog input	ADC in 1	1
Analog input	ADC in 2	2
Analog input	ADC in 3	3
Analog input	ADC in 4	4
Analog input	ADC in 5	5
Analog input	ADC in 6	6
Analog input	ADC in 7	7
Regulated 1.2V	1V2	8
Regulated 2.5V	2V5	9
Regulated 3.3V	3V3	10
Input voltage	Vin	11
Input current	Iin	12
Temperature	Temp	13

The following ADC channels are available for the Sirius TCM:

Parameter	Abbreviation	ADC channel
Regulated 1.2V	1V2	8
Regulated 2.5V	2V5	9
Regulated 3.3V	3V3	10
Input voltage	Vin	11
Input current	Iin	12
Temperature	Temp	13

The TCM-S FM board does not contain any input ADC channels.

When data is read from a channel, the lower 8 bits contains the channel status information, and the upper 24 bits contains the raw ADC data.

To convert the ADC value into mV, mA or m°C, the formulas specified in the table below shall be used. Note that this assumes a 24 bit ADC value which is what the ADC IP returns on read. Should the raw bit value be truncated or scaled down, the scale factor (2^24) in the equations need to be adjusted as well. Note also that the temperature equation require the 3V3 [mV] value.

HK channel	Formula
Temp [m°C]	Temp_mV = (ADC_value*2500)/2^24 Temp_mC = (1000*(3V3_mV - Temp_mV) - Temp_mV*1210) / 0.00385*(Temp_mV - 3300)
Iin [mA]	Iin_mA = (ADC_value*5000)/(2^24)
Vin [mV]	Vin_mV = (ADC_value*20575)/(2^24)
3V3 [mV]	3V3_mV = (ADC_value*5000)/(2^24)
2V5 [mV]	2V5_mV = (ADC_value*5000)/(2^24)
1V2 [mV]	1V2_mV =(ADC_value*2525)/(2^24)

5.10.2. RTEMS API

This API represents the driver interface of the module from an RTEMS user application's perspective.

The driver functionality is accessed through the RTEMS POSIX API for ease of usage. In case of a failure on a function call, the *errno* value is set for determining the cause.

5.10.2.1. Enum adc_ioctl_sample_rate_e

Enumerator for the ADC sample rate.

Enumerator	Description
ADC_IOCTL_SPS_31250	SPS 31250
ADC_IOCTL_SPS_15625	SPS 15625
ADC_IOCTL_SPS_10417	SPS 10417
ADC_IOCTL_SPS_5208	SPS 5208
ADC_IOCTL_SPS_2597	SPS 2597
ADC_IOCTL_SPS_1007	SPS 1007
ADC_IOCTL_SPS_503_8	SPS 503.8
ADC_IOCTL_SPS_381	SPS 381
ADC_IOCTL_SPS_200_3	SPS 200.3
ADC_IOCTL_SPS_100_5	SPS 100.5
ADC_IOCTL_SPS_59_52	SPS 59.52
ADC_IOCTL_SPS_49_68	SPS 49.68
ADC_IOCTL_SPS_20_01	SPS 20.01
ADC_IOCTL_SPS_16_63	SPS 16.63
ADC_IOCTL_SPS_10	SPS 10
ADC_IOCTL_SPS_5	SPS 5
ADC_IOCTL_SPS_2_5	SPS 2.5
ADC_IOCTL_SPS_1_25	SPS 1.25

5.10.2.2. Function int open(…)

Opens access to the ADC. Only one instance can be open at any time, only read access is allowed and only blocking mode is supported.

Argument name	Type	Direction	Description
Pathname	const char *	in	The absolute path to the ADC to be opened. ADC device is defined as ADC_DEVICE_NAME.
Flags	int	in	Access mode flag, only O_RDONLY is supported.

Return value	Description
Fd	A file descriptor for the device on success
-1	See *errno* values
errno values	
EEXISTS	Device not opened
EALREADY	Device is already open
EINVAL	Invalid options

5.10.2.3. Function int close(…)

Closes access to the device.

Argument name	Type	Direction	Description
Fd	int	in	File descriptor received at open.

Return value	Description
0	Device closed successfully
-1	See *errno* values
errno values	
EFAULT	Device not opened

5.10.2.4. Function ssize_t read(…)

This is a blocking call to read data from the ADC.

Note! The size of the given buffer must be a multiple of 32 bit.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
buf	void*	in	Pointer to buffer to write data into.
count	size_t	in	Number of bytes to read. Only 4 bytes is supported in this implementation.

Return value	Description
>= 0	Number of bytes that were read.
- 1	see *errno* values
errno values	
EPERM	Device not open
EINVAL	Invalid number of bytes to be read

ADC data buffer bit definition	Description
31:8	ADC value
7:4	ADC status
3:0	Channel number

5.10.2.5. Function int ioctl(…)

Ioctl allows for more in-depth control of the ADC IP like setting the sample mode, clock divisor etc.

Argument name	Type	Direction	Description
Fd	int	in	File descriptor received at open
Cmd	int	in	Command to send
Val	uint32_t / uint32_t*	in/out	Value to write or a pointer to a buffer where data will be written.

Command table	Type	Direction	Description
ADC_SET_SAMPLE_RATE_IOCTL	uint32_t	in	Set the sample rate of the ADC chip, see [RD6].
ADC_GET_SAMPLE_RATE_IOCTL	uint32_t	out	Get the sample rate of the ADC chip, see [RD6].
ADC_SET_CLOCK_DIVISOR	uint32_t	in	Set the clock divisor of the clock used for communication with the ADC chip. Minimum 0 and maximum 255.
ADC_GET_CLOCK_DIVISOR	uint32_t	out	Get the clock divisor of the clock used for communication with the ADC chip.
ADC_ENABLE_CHANNEL	uint32_t	in	Enable specified channel number to be included when sampling. Minimum 0 and maximum 15.
ADC_DISABLE_CHANNEL	uint32_t	in	Disable specified channel number to be included when sampling. Minimum 0 and maximum 15.

Return value	Description
0	Command executed successfully
-1	see *errno* values
errno values	

RTEMS_NOT_DEFINED	Invalid IOCTL
EINVAL	Invalid value supplied to IOCTL

5.10.3. Usage description

The following #define needs to be set by the user application to be able to use the ADC:

CONFIGURE_APPLICATION_NEEDS_ADC_DRIVER

5.10.3.1. RTEMS application example

In order to use the ADC driver on RTEMS environment, the following code structure is suggested to be used:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/adc_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_ADC_DRIVER

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

#define CONFIGURE_INIT
rtems_task Init (rtems_task_argument argument);

rtems_task Init (rtems_task_argument argument) {
  rtems_status_code status;
  int read_fd;
  uint32_t buffer;
  ssize_t size;

  read_fd = open(ADC_DEVICE_NAME, O_RDONLY);
  status = ioctl(read_fd, ADC_ENABLE_CHANNEL_IOCTL, 4);
  size = read(read_fd, &buffer, 4);
  status = ioctl(read_fd, ADC_DISABLE_CHANNEL_IOCTL, 4);
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions: `open`, `close`, `ioctl`.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of `<bsp/adc_rtems.h>` is required for accessing the ADC.

5.10.4. Limitations

Only one ADC channel can be enabled at a time. To switch channels, disabling the old and enabling the new channel is required.

5.11. NVRAM

The NVRAM on the OBC and TCM is a 262,144-bit magnetoresistive random access memory (MRAM) device organized as 32,768 bytes of 8 bits. EDAC is implemented on a byte basis meaning that half the address space is filled with checksums for correction. It's a strong correction which corrects 1 or 2 bit errors on a byte and detects multiple. The table below presents the address space defined as words (**16,384** bytes can be used). The address space is divided into two sub groups as product- and user address space.

5.11.1. Description

This driver software for the SPI RAM IP, handles the initialization, configuration and access of the NVRAM.

The NVRAM is divided into a system memory area and a user memory area.
System memory area is protected and must be unlocked before each write.

5.11.2. EDAC mode

When in EDAC mode, which is the normal mode of operation, all write and read transactions are protected by EDAC algorithms. All NVRAM addresses containing EDAC are hidden by the IP. The address space is given by the table below:

Area	Range start	Range end
System	0x100	0x1FC
User	0x200	0x7FFC

5.11.3. Non-EDAC mode

Non-EDAC mode is a debug mode that allows the user to examine the EDAC bytes.
The purpose of this mode is to be able to insert errors into the memory for testing of the EDAC algorithm.
When in Non-EDAC mode net data and EDAC data is interleaved on an 8 bit basis.
I.e. when reading a 32 bit word byte, 0, 2 contains the net data and byte 1, 3 contains EDAC data. The address space is doubled when compared to EDAC mode, as is shown with the table below:

5.11.4. RTEMS API

This API represents the driver interface of the module from an RTEMS user application's perspective.

The driver functionality is accessed through the RTEMS POSIX API for ease of usage. In case of a failure on a function call, the *errno* value is set for determining the cause.

5.11.4.1. Enum rtems_spi_ram_edac_e

Enumerator for the error correction and detection of the SPI RAM.

Enumerator	Description
SPI_RAM_IOCTL_EDAC_ENABLE	Error Correction and Detection enabled.
SPI_RAM_IOCTL_EDAC_DISABLE	Error Correction and Detection disabled.

5.11.4.2. Function int open(...)

Opens access to the requested SPI RAM.

Argument name	Type	Direction	Description
pathname	const char *	in	The absolute path to the SPI RAM to be opened. SPI RAM device is defined as SPI_RAM_DEVICE_NAME.
flags	int	in	Access mode flag.

Return value	Description
fd	A file descriptor for the device on success
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.11.4.3. Function int close(...)

Closes access to the device.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.

Return value	Description
0	Device closed successfully
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.11.4.4. Function ssize_t read(...)

Read data from the SPI RAM. The call block until all data has been received from the SPI RAM.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
buf	void*	in	Pointer to character buffer to write data into.
count	size_t	in	Number of bytes to read. Must be a multiple of 4.

Return value	Description
>=0	Number of bytes that were read.
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.11.4.5. Function ssize_t write(...)

Write data into the SPI RAM. The call block until all data has been written into the SPI RAM.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
buf	void*	in	Pointer to character buffer to read data from.
count	size_t	in	Number of bytes to write. Must be a multiple of 4.

Return value	Description
>=0	Number of bytes that were written.
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.11.4.6. Function int lseek(...)

Set the address for the read/write operations.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
offset	void*	in	SPI RAM read/write byte offset. Must be a multiple of 4.
whence	int	in	SEEK_SET and SEEK_CUR are supported.

Return value	Description
>=0	Byte offset
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.11.4.7. Function int ioctl(...)

Input/output control for SPI RAM.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
cmd	int	in	Command to send.
val	int	in/out	Value to write or a pointer to a buffer where data will be written.

Command table	Type	Direction	Description
SPI_RAM_SET_EDAC_IOCTL	uint32_t	in	Configures the error correction and detection for the SPI RAM, see [5.11.4.1.]
SPI_RAM_SET_DIVISOR_IOCTL	uint32_t	in	Configures the serial clock divisor.

SPI_RAM_GET_EDAC_STATUS_IOCTL	uint32_t	out	Get EDAC status for previous read operations.
SPI_RAM_UNLOCK_MEMORY_IOCTL	uint32_t	in	Unlocks system memory for writing. The input value is ignored. Must be called before every write operation (4 bytes) of the system memory.

EDAC Status	Description
SPI_RAM_EDAC_STATUS_MULT_ERROR	Multiple errors detected.
SPI_RAM_EDAC_STATUS_DOUBLE_ERROR	Double error corrected.
SPI_RAM_EDAC_STATUS_SINGLE_ERROR	Single error corrected.

Return value	Description
0	Command executed successfully
-1	See *errno* values
errno values	
EINVAL	Invalid options

5.11.5. Usage description

The following #define needs to be set by the user application to be able to use the SPI RAM:

CONFIGURE_APPLICATION_NEEDS_SPI_RAM_DRIVER

5.11.5.1. RTEMS application example

In order to use the SPI RAM driver on RTEMS environment, the following code structure is suggested to be used:

```
#include <bsp.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <bsp/spi_ram_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_SPI_RAM_DRIVER

#include <bsp/bsp_confdefs.h>
#include <rtems/confdefs.h>

#define CONFIGURE_INIT
rtems_task Init (rtems_task_argument argument);

rtems_task Init (rtems_task_argument argument){
  rtems_status_code status;
  int dsc;
  uint8_t buf[8];
  ssize_t cnt;
  off_t offset;

  dsc = open(SPI_RAM_DEVICE_NAME, O_RDWR);
  offset = lseek(dsc, 0x200, SEEK_SET);
  cnt = write(dsc, &buf[0], sizeof(buf));
  offset = lseek(dsc, 0x200, SEEK_SET);
  cnt = read(dsc, &buf[0], sizeof(buf));
  status = close(dsc);
}
```

Inclusion of `<fcntl.h> and <unistd.h>` are required for using the POSIX functions: `open`, `close, ioctl`.

Inclusion of `<errno.h>` is required for retrieving error values on failures.

Inclusion of <`bsp/spi_ram_rtems.h`> is required for accessing the SPI_RAM.

6. Spacewire router

In both Sirius OBC and Sirius TCM products, a smaller router is integrated onto their relative SoCs. The routers all use path addressing (see [RD2]) and given the topology illustrated in Figure 6-1, the routing addressing can be easily calculated.

Figure 6-1 Integrated router location

In reference to the topology above, sending a package from the Sirius OBC to the Sirius TCM™ or vice versa, the routing address will be 1-3.

In addition to this, each end node, Sirius OBC or Sirius TCM, 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 of the recipient application/service on the Sirius TCM.

7. Sirius TCM

7.1. Description

The Sirius TCM handles receiving of Telecommands (TCs) and Telemetry (TM) as well as Spacewire communication using the RMAP protocol.

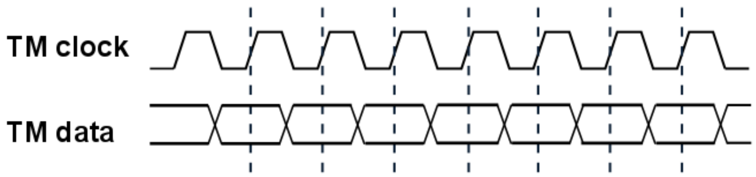
TC, received from ground, can be of two command types; a pulse command or a Telecommand. A pulse command is decoded directly in the hardware and the hardware then sets an output pin according to the pulse command parameters. All other commands are handled by the Sirius TCM software. Any command not addressing the Sirius TCM will be routed to other nodes on the SpaceWire network according to the current Sirius TCM configuration.

TM is received from other nodes on the SpaceWire network. The Sirius TCM supports both live TM transmissions directly to ground as well as storage of TM to the Mass Memory for later retrieval or download to ground during ground passes.

The Sirius TCM is highly configurable to be adaptable to different customer needs and missions and currently supports SpaceWire (SpW) using the Read Memory Access Protocol (RMAP), UART interfaces, pulse commands as well as Telecommand and Telemetry using CCSDS frame encodings and ECSS PUS packets.

Radio Clock

The TM clock and data symbol towards the radio has the following signal output shown in the figure below:



The default configuration of the TM downlink is:

- The divisor of the TM clock is set to 25
- TM is enabled
- OCF/CLCW is included in TM Transfer frames
- FECF is included in TM Transfer Frames
- Idle Frame generation is enabled
- Convolutional encoding is disabled
- Pseudo randomization is enabled
- Reed-Solomon encoding is enabled

The default configuration of the TC uplink is:

- Derandomization is disabled.

7.2. Block diagram

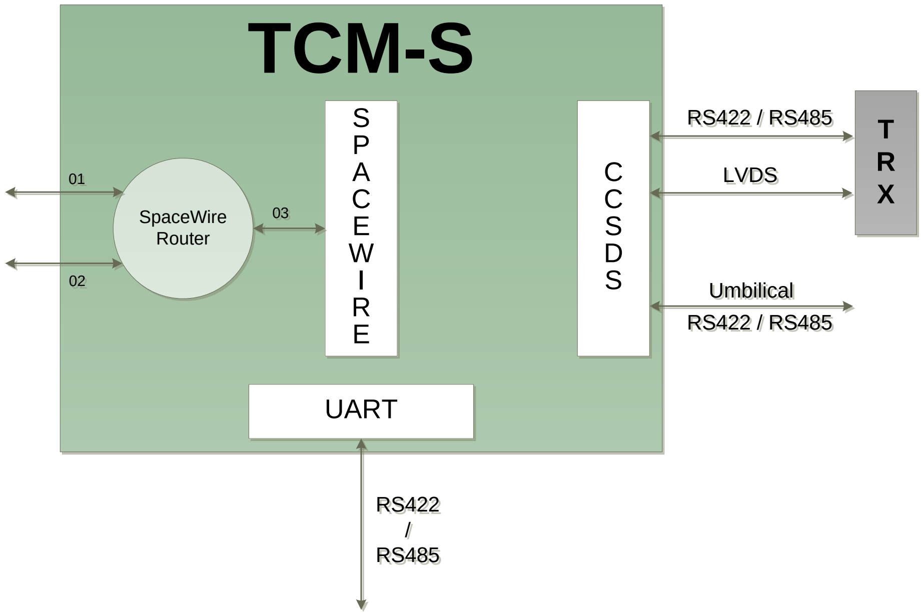


Figure 7-1 – Sirius TCM functionality layout

7.3. TCM-S application overview

The TCM-S application is partitioned into several software modules who each handles a specific functional part. An overview of the software architecture of the TCM-S is presented in Figure 7.2. A main design driver of the TCM-S software architecture is the ability to pass along data between the different handlers without copying, since that would quickly decrease the performance and throughput of the system. To help with the no-copy policy, each peripheral handling larger amounts of data have DMA functionality, off-loading the CPU from mere datashuffling tasks while at the same time increasing performance by at least a magnitude. Data coming in on the SpaceWire interface intended for the mass memory will thus be stored in RAM only once - in the handoff between the SpaceWire and mass memory handlers.

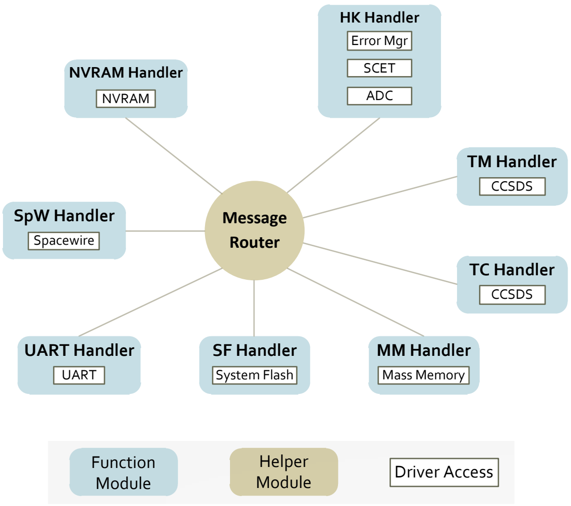


Figure 7.2 TCM-S software application overview

7.4. Configuration

The TCM-S can be configured for specific missions by parameters in a NVRAM described in Table 7-1 to Table 7-4.

Data from different sources can be routed on the SpW-network. Routing info is set by format specified in Table 7-1

Table 7-1: UART_ROUTING

Data	Type	Description
uart	UINT8	Source of message 0 - UART0 1 - UART1 2 - UART2 3 - UART3 4 - UART4 5 - PSU Ctrl 6 - Safe Bus 50 – Telecommands 51 – CPDU report 52 – TM Error 53 – TC Error
address	UINT32	The RMAP-address UART info is routed to
ext address	UINT8	The extended RMAP-address UART info is routed to
Path	UINT16	The index of the SpW-path for the routing. See Table 7-3

Configuration of UART-devices is done by Table 7-2 below.

Table 7-2: UART_CONFIG

Data	Type	Description
uart	UINT8	The UART device. 0 - UART0 1 - UART1 2 - UART2 3 - UART3 4 - UART4 5 - PSU Ctrl 6 - Safe Bus
Bitrate	UINT8	UART bitrate: 10 = 375000 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
Reserved	UINT8	Reserved for padding and future use

Paths on SpW-network are specified by table Table 7-3 below:

Table 7-3: NVRAM SpW path storage

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

Telecommand can be routed to nodes on the SpW by APID as specified in Table 7-4 below:

Table 7-4: 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: 0 – Ext. APID, 1 – TCM-S APID Bit13:11 Not used Bit10:0 – APID
2-3	UINT16	Upper APID in APID range Bit15: 0 – Single APID Routing, 1 – APID range Bit14: 0 – Ext. APID, 1 – TCM-S APID Bit13:11 – Not used Bit10:0 - APID
4-5	UINT16	The index of the SpW-path of the APID. See Table 7-3
6-7	UINT16	Reserved for future use and padding.

7.5. Telemetry

Telemetry is sent to a transceiver through the RS422/RS485 and LVDS interfaces. VC 0 and VC 1 are supported for TM Data and VC 7 is reserved for Idle-frames. The CCSDS HW generates complete TM Transfer Frames from TM PUS packets. If a TM PUS packet does not fit in one TM Transfer Frame, the CCSDS module sends the TM PUS packet in several TM Transfer Frames. If a TM PUS packet not does fill the whole TM Transfer Frame, an Idle-packet is padded to the TM PUS packet. The following functionality of telemetry is configurable by RMAP-commands (see 7.13):

- Divisor of TM Clock
- Inclusion of CLCW of TM Transfer Frames if enabled
- Inclusion of Frame Error Control Field of TM Transfer Frames if enabled
- Updating of Master Channel Frame Counter if enabled.
- If idle frame generation is enabled Idle frames are sent on VC7 when no data is send on VC0 or VC1
- Convolutional encoding of telemetry can be enabled/disabled

- Pseudo randomization of telemetry can be enabled/disabled

The TCM-S supports a format of TM Transfer Frames described in [RD8].

7.6. Telecommands

Telecommands can be received on the TRX1 or the UMBI interfaces. The TCM also support an alternative transmitter and hence have a TRX2 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 7.3

- All incoming signals on the inputs are synchronized to the system clock domain.
- 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.
- 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.
- The BCH interface does not "see" the data/clock until the start pattern is decoded correctly and the enable signal is set.

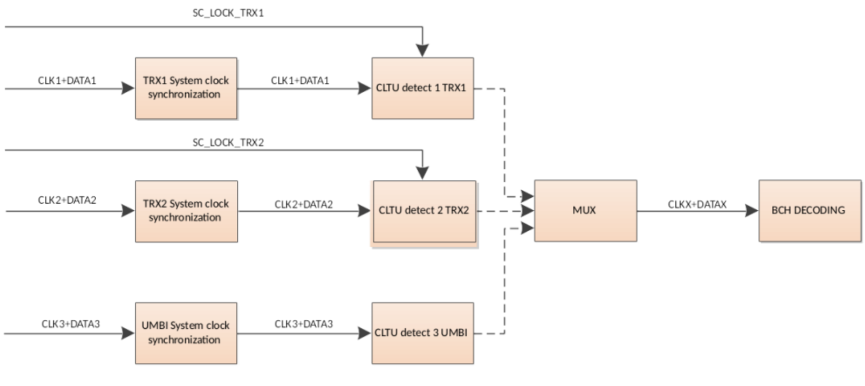


Figure 7.3 – Telecommand Input Multiplexer

Derandomization of TC can be enable/disabled by RMAP command (see 7.13)

The TCM-S supports a format of TC Transfer Frames described in [RD9].

7.7. 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 the hardware and the hardware 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 be utilized to reset modules on SC and choose which software image to boot, either the latest or the safe image. The last executed pulse command can be read from the telecommand status data field.

For details about format of pulse command, see 7.12.2

7.8. 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.

7.9. Time Management

The TCM-S has an internal SCET timer that can be configured in three different modes Free-running, Master or Slave settable by a RMAP command. See 7.14.1.18 for further info.

The default mode is Free-running, which means no external synchronization, the SCET is free-running using the internal oscillator as reference and outputs no PPS.

A time stamp can be generated when a TM Transfer Frame is send on VC0. The rate of time stamp generation is configurable by RMAP. The last time stamp is readable by RMAP. See 7.14.1.11 and 7.14.1.12 for further info.

7.10. Error Management and System Supervision

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

- EDAC single error count
- EDAC multiple error count
- Watchdog trips
- CPU Parity errors.

Error Manager related information and housekeeping data is available by RMAP. See 7.14.1.16

The status of the TM Downlink and TC Uplink are available through RMAP. See 7.14.1.14 and 7.14.1.1

A watchdog is enabled in the TCM-S that must be kicked by the TCM-S Application or a reset will occur.

7.11. Mass Memory Handling

The mass memory in TCM-S is intended for storage of telemetry data while awaiting transfer to ground. To simplify divisions between different types of data with different configurations, the mass memory is divided into logical partitions. All data stored to mass memory is in the form of PUS packets ready for transmit to ground and are grouped together into segments. Each segment is of a fixed size (configurable for each partition) and is defined as starting with a PUS packet. If stored PUS packets won't match the segment size exactly, the remaining bytes in each segment must be padded with a bit pattern of 0xF5 which will be removed during download.

Writing/Reading of data to/from a partition is provided by RMAP-commands. Downloading of data from a partition to ground is provided by a RMAP-command. See 7.14.1.20 and **Error! Reference source not found.** for further info.

When a partition is full, there is no possibility to write data. When a partition is full, reading or downloading of data from the partition must occur before new attempts to write data.

Operations to modify read and write pointers of the partitions are provided by RMAP. See 7.14.1.22 and 7.14.1.23 for further info.

As the mass memory is flash-based, 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 all partitions should be created with some extra space to account for this phenomenon.

7.12. ECSS standard service

The TCM-S supports a subset of the services described in [RD4]

7.12.1. PUS-1 Telecommand verification service

The TCM-S performs a verification of APID of the incoming TC. If the verification fails, the telecommand is rejected and a Telecommand Acceptance Failure - report (1,2) is generated as described in [RD4]. On successful verification, the command is routed to the receiving APID. The receiving APID performs further verification of packet length, checksum of packet, packet type, packet subtype and application data and generates reports accordingly (1,1) or (1,2). If specified by the mission, the APID shall implement services for Telecommand Execution Started, Telecommand Execution Progress and Telecommand Execution Complete.

Table 7-5: Telecommand Acceptance Report – Failure (1,2)

Packet ID	Packet Sequence Control	Code
UINT16	UINT16	UINT8. 0 – Illegal APID

7.12.2. PUS-2 Device Command Distribution Service

The TCM-S supports the command pulse distribution unit (CPDU) pulse commands in hardware as defined in 7.2.2 in [RD4].
The CPDU listens on virtual channel 2, APID 2.
It has 12 controllable (0-11) output lines and can be toggled to supply different pulse lengths according to the following scheme:

Table 7-6 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 7-7 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.

7.13. Spacewire RMAP

According to [RD3], 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 Sirius TCM according to Table 7-8 to separate between configuration commands and mass memory storage of data (partition handling).

Table 7-8: RMAP predefined fields

Field	Value
Initiator Logical Address	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 Figure 7-1, 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 7-9: RMAP predefined fields for routing

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

7.13.1. Input

The RMAP commands supported by the Sirius TCM are specified in the table below. See chapter 7.14.1 for details on each specific command.

Note! The Sirius TCM uses the RMAP Transaction ID to separate between outstanding replies to different units. When several nodes are addressing the Sirius 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 Sirius TCM uses the Transaction ID range `0x0000-0x0FFF` in all outgoing communication.

Table 7-10: RMAP commands to TCM

Name	Ext. Addr	Address	Cmd	Description
TMStatus	0xFF	0x00000000	R	Reads latest telemetry status.
TMConfig	0xFF	0x00000200	R	Reads telemetry configuration.
TMControl	0xFF	0x00000300	W	Enable/Disable telemetry.
TMFEControl	0xFF	0x00000400	W	Enable/Disable Frame Error Control Field for TM Transfer Frames.
TMMCFCControl	0xFF	0x00000500	W	Enable/Disable Master Channel Frame Counter Control for TM Transfer Frames.
TMIFControl	0xFF	0x00000600	W	Enable/Disable Idle Frames.
TMPRControl	0xFF	0x00000700	W	Enable/Disable Pseudo Randomization for telemetry.
TMCEControl	0xFF	0x00000800	W	Enable/Disable Convolutional Encoding for telemetry.
TMBRControl	0xFF	0x00000900	W	Sets telemetry clock frequency divisor.
TMOCFControl	0xFF	0x00000A00	W	Enable/Disable inclusion of Operational Control field in TM Frames.
TMTSControl	0xFF	0x00000B00	R/W	Configures Timestamp of telemetry.
TMTSStatus	0xFF	0x00000C00	R	Latest timestamp of telemetry on virtual channel 0.
TMSend	0xFF	0x00001000	W	Sends telemetry on virtual channel 0.
TCStatus	0xFF	0x01000000	R	Reads latest telecommand status.
TCDRControl	0xFF	0x01000100	W	Enables/Disables Derandomizer of telecommands.
HKData	0xFF	0x02000000	R	Reads Houskeeping data.
SCETTime	0xFF	0x02000100	R/W	Reads/Configures SCET time.
SCETConfig	0xFF	0x02000200	R/W	Reads/Configures SCET configuration.
UARTCommand	0xFF	0x0400010x	W	Sends a command to a specific UART device. 0 - UART0 1 - UART1 2 - UART2 3 - UART3 4 - UART45 - PSU Ctrl 6 - Safe Bus.
MMData	0x00-0x0F	0x00000000	R/W	Reads and writes data of a partition.
MMStatus	0xFF	0x05000000	R	Reads mass memory device status.
MMWritePointer	0xFF	0x0500010x	R/W	Position of the writepointer for partition x
MMReadPointer	0xFF	0x0500020x	R/W	Position of the readpointer for partition x

MMPartitionConfig	0xFF	0x0500030x	R	Configuration of partition x
MMPartitionSpace	0xFF	0x0500040x	R	Reads available space in partition x.
MMDownloadPartitionData	0xFF	0x0500050x	W	Downloads partition x data via telemetry.

7.13.1. Output

The TCM-S publishes data to other nodes according to the address map below:

Note! All outgoing communication will use the Transaction ID range of `0x0000-0x0FFF`.

Table 7-11: Published data from TCM

Name	Ext. Addr.	Address	Cmd	Description
TCCommand	0xFF	0x00000000	W	Routed Telecommands
UARTData	0xFF	0x0400000x	W	Data received on specified UART 0 - UART0 1 - UART1 2 - UART2 3 - UART3 4 - UART4 5 - PSU Ctrl 6 - Safe Bus

7.13.2. Status code in reply messages

In the status field of write/read, the values in table below can be returned.

Table 7-12: Status code

Code	Numeric value	Comment
	0	Operation success
EIO	5	Internal error occurred.
EINVAL	22	A provided parameter in command is invalid
ENOSPC	28	No space left on a partition or no data available on a partition.
EALREADY	37	Operation already in progress

7.14. RMAP address details

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

7.14.1. Input

7.14.1.1. TMStatus

Reads the latest telemetry status.

Table 7-13: TMStatus data

Byte	Type	Description
0	UINT8	0x00 – No Error

		0x01 – FIFO error.
1	UINT8	0x00 – No transfer in progress. 0x01 – Transfer in progress.

RMAP reply status:

Table 7-14: 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

7.14.1.2. TMConfig

Reads the telemetry configuration.

Table 7-15: TMConfig data

Byte	Type	Description
0	UINT8	Bitrate divisor value
1	UINT8	Telemetry Control 0x00 – Disabled 0x01 – Enabled
2	UINT8	Frame Error Counter Field Control 0x00 – Disabled 0x01 – Enabled
3	UINT8	Master Channel Frame Count Control 0x00 – Disabled 0x01 – Enabled
4	UINT8	Idle Frame Control 0x00 – Disabled 0x01 – Enabled
5	UINT8	CLCW Control 0x00 – Disabled 0x01 - Enabled
6	UINT8	Convolutional Encoding Control 0x00 – Disabled 0x01 – Enabled
7	UINT8	Pseudo Randomization Control 0x00 – Disabled 0x01 – Enabled

RMAP reply status:

Table 7-16: 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

7.14.1.3. TMControl

Controls generation of telemetry.

Table 7-17: TMControl data

Byte	Type	Description
0	UINT8	0x00 – Disable 0x01 – Enable (Default)

RMAP reply status (if a reply is requested):

Table 7-18: TMControl status codes

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

7.14.1.4. TMFEControl

Controls Frame Error Control Field inclusion for transfer frames.

Table 7-19: TMFEControl data

Byte	Type	Description
0	UINT8	0x00 – Disable 0x01 – Enable (Default)

RMAP reply status (if a reply is requested):

Table 7-20: TMFEControl status codes

Status code	Description
0	Success.
EINVAL	The driver for the TM device has not been initialized or the argument is out of range

EIO	I/O error. The TM device cannot be accessed

7.14.1.5. TMMCFCControl

Controls the Master Channel Frame Counter generation for transfer frames.

Table 7-21: TMMCFCControl data

Byte	Type	Description
0	UINT8	0x00 – Disable 0x01 – Enable (Default)

RMAP reply status (if a reply is requested):

Table 7-22: TMMCFCControl status codes

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

7.14.1.6. TMIFControl

Controls the Idle Frame generation for transfer frames.

Table 7-23: TMIFControl data

Byte	Type	Description
0	UINT8	0x00 – Disable 0x01 – Enable (Default)

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

Table 7-24: TMIFControl status codes

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

7.14.1.7. TMPRControl

Controls the Pseudo Randomization for transfer frames.

Table 7-25: TMPRControl data

Byte	Type	Description
0	UINT8	0x00 – Disable (Default) 0x01 – Enable

RMAP reply status (if a reply is requested):

Table 7-26: TMPRControl status codes

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

7.14.1.8. TMOCFControl

Controls Operational Control Field inclusion in TM Transfer frames.

Table 7-27: TMOCFControl data

Byte	Type	Description
0	UINT8	0x00 – Disable 0x01 – Enable (Default)

RMAP reply status (if a reply is requested):

Table 7-28: TMOCFControl status codes

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

7.14.1.9. 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 7-29: TMCEControl data

Byte	Type	Description
0	UINT8	0x00 – Disable (Default) 0x01 – Enable

RMAP reply status (if a reply is requested):

Table 7-30: TMCEControl status codes

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

7.14.1.10. TMBRControl

Sets the telemetry clock frequency divisor.

The telemetry clock is fed to the radio. The frequency of the telemetry clock is the system clock (50 MHz) divided by the divisor. E.g. if the divisor value is set to 25, the telemetry clock frequency is 2 MHz
Note! If the convolutional encoding is **disabled**, as defined in subchapter 7.14.1.9, the telemetry clock is divided by two, i.e. 1 MHz from example above, to keep the net data rate the same.

Table 7-31: TMBRControl data

Byte	Type	Description
0	UINT8	Bitrate divisor value (default 25). Minimum divisor is 4.

RMAP reply status (if a reply is requested):

Table 7-32: TMBRControl 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

7.14.1.11. TMTSControl

Configures the timestamping for transfer frames.

Table 7-33: TMTSControl data

Byte	Type	Description
0	UINT8	0x00 – No timestamping (Default) 0x01 – Take a timestamp every time frame sent 0x02 – Take a timestamp every 2nd time frame sent … 0xFF – Take a timestamp every 255th time frame sent

RMAP reply status (if a reply is requested):

Table 7-34: TMTSControl 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.

7.14.1.12. TMTSStatus

The latest timestamp of telemetry send on virtual channel 0. Timestamping needs to be enabled before timestamps can be read. See7.14.1.14.

Table 7-35: 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 7-36: TMTSStatus status codes

Status code	Description
0	Success.
EINVAL	Timestamping is not enabled. See 7.14.1.11
EIO	I/O error. The TM device cannot be accessed

7.14.1.13. TMSend

Sends telemetry to the TM path on virtual channel 0. The data must contain **at least one** telemetry PUS Packet.

Table 7-37: TMSend data

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

RMAP reply status (if a reply is requested):

Table 7-38: TMSend 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

7.14.1.14. TCStatus

Reads current telecommand status.

Table 7-39: 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 7-40: 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

7.14.1.15. TCDRControl

Configures derandomization for telecommand frames.

Table 7-41: TCDRControl data

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

RMAP reply status (if a reply is requested):

Table 7-42: TCDRControl 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

7.14.1.16. HKData

Reads the housekeeping data.

Table 7-43: HKData data

Byte	Type	Description
0	UINT16	Input voltage
2	UINT16	Regulated 3V3 voltage
4	UINT16	Regulated 2V5 voltage
6	UINT16	Regulated 1V2 voltage
8	UINT16	Input current
10	UINT16	Temperature
12	UINT32	SCET Seconds
16	UINT8	S/W version 0-padding
17	UINT8	S/W major version
18	UINT8	S/W minor version
19	UINT8	S/W patch version
20	UINT8	CPU Parity Errors
21	UINT8	Watchdog trips

22	UINT8	SDRAM EDAC Single Errors on instructions
23	UINT8	SDRAM EDAC Single Errors on data
24	UINT8	SDRAM EDAC Multiple Errors on instructions
25	UINT8	SDRAM EDAC Multiple Errors on data

RMAP reply status:

Table 7-44: 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

7.14.1.17. SCETTime

Reads/sets the SCET time. Any adjustment to SCET time will have different effects depending on the SCET mode (see subchapter 7.14.1.18).

Free-running mode: The SCET seconds and subseconds adjustment will happen immediately.

Slave mode: The SCET seconds will be adjusted at the next PPS edge and the subseconds adjustment will thus be ignored.

Table 7.45: SCETTime data

Byte	Type	Description
0	UINT32	SCETSeconds when reading. When writing a value to SCETSeconds, this must be a 2's complementary value that shall be added to the seconds counter.
4	UINT16	SCETSubSeconds when reading. When writing a value to SCETSubSeconds a 2's complementary value shall be added to the subseconds counter.

RMAP reply status (if a reply is requested):

Table 7-46: SCETTime status codes

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

7.14.1.18. SCETConfig

The SCET can be configured in three modes: Free-running (default), Master or Slave.

Free-running mode: No external synchronization, the SCET is free-running using the internal oscillator as reference and outputs no PPS.

Master mode: No external synchronization, the SCET is free-running using the internal oscillator as reference and outputs a PPS at integer seconds.

Slave mode: The SCET is synchronized to an external PPS and outputs no PPS.

Table 7.47: SCETConfig data

Byte	Type	Description
0	UINT32	Configuration of SCET mode, see above 0 - Free-running mode (default) 1 – Master mode 2 - Slave mode

RMAP reply status (if a reply is requested):

Table 7-48: SCETConfig status codes

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

7.14.1.19. UARTCommand

Send a command on the specified UART interface.

Table 7-49: UARTCommand data

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

RMAP reply status (if a reply is requested):

Table 7-50: UARTCommand status codes

Status code	Description
0	Success.

EINVAL	The driver for the UART device has not been initialized.
EIO	I/O error. The UARTdevice cannot be accessed

7.14.1.20. MMData

Read or write data from/to a partition

Table 7-51: MMData data

Byte	Type	Description
0 - nn	Array of UINT8	Data

RMAP reply status (if a reply is requested):

Table 7-52: MMData data status codes

Status code	Description
0	Success.
ENOSPC	Write: Not enough space on partition. Read: Not enough data on partition.
EINVAL	Invalid argument. E.g. invalid partition number
EIO	I/O error. E.g. failed to access storage or nvram.

7.14.1.21. MMStatus

Reads mass memory status.

Table 7-53: MMStatus data

Byte	Type	Description
0	UINT8	Chip 3 status Bit 7 - WP# (Write protect Off: 1/On: 0) Bit 6 - RDY (Ready: 1/Busy: 0) Bit 5 - ARDY (Ready: 1/Busy Array: 0) Bit 1 - FAILC (Fail: 1/Pass: 0 – set if the previous operation (program) failed) Bit 0 - FAIL (Fail: 1/Pass: 0 – set if the most recently finished operation (program, erase) on the selected die failed). Bits 2, 3, and 4 are reserved.
1	UINT8	Chip 2 status, see above
2	UINT8	Chip 1 status, see above
3	UINT8	Chip 0 status, see above
4	UINT8	EDAC-chip status, see above
5-6	UINT16	Controller status Bits 9-15 are reserved. Bit 8 – Set feature done Bit 7 - Busy (command in progress when high) Bit 6 - Reserved Bit 5 - Reset done Bit 4 - Read ID done Bit 3 - Erase block done Bit 2 - Read page setup done Bit 1 - Read status done Bit 0 - Program page done
7-11	Array of 5 UINT8	Chip ID: EDAC
12-16	Array of 5 UINT8	Chip ID: Chip 0
17-21	Array of 5 UINT8	Chip ID: Chip 1
22-26	Array of 5 UINT8	Chip ID: Chip 2
27-31	Array of 5 UINT8	Chip ID: Chip 3

RMAP reply status:

Table 7-54: MMStatus status codes

Status code	Description
0	Success.
EINVAL	Invalid argument. E.g. invalid partition number
EIO	I/O error. E.g. failed to access storage.

7.14.1.22. MMWritePointer

The writepointer of the specified partition is set/read by this command.

Table 7-55: MMWritePointer data

Byte	Type	Description
0-7	UINT64	Pointer's byte position in the partition.

RMAP reply status (if a reply is requested):

Table 7-56: MMWritePointer status codes

Status code	Description
0	Success.
EINVAL	Invalid argument. E.g. invalid partition number
EIO	I/O error. E.g. failed to access storage or nvram.

7.14.1.23. MMReadPointer

The readpointer of the specified partition is set/read by this command.

Table 7-57: MMReadPointer data

Byte	Type	Description
0-7	UINT64	Pointer's byte position in the partition.

RMAP reply status (if a reply is requested):

Table 7-58: MMReadPointer status codes

Status code	Description
0	Success.
EINVAL	Invalid argument. E.g. invalid partition number
EIO	I/O error. E.g. failed to access storage or nvram.

7.14.1.24. MMPartitionConfig

The partition configuration of the specified partition is read by this command. The partition configuration is set using the nvconfig utility.

Table 7-59: MMPartitionConfig data

Byte	Type	Description
0	UINT64	Size in bytes. Must be in multiples of mass memory block size (2097152 bytes)
8	UINT32	The offset in blocks from the first block of the Mass Memory.
12	UINT8	Partition mode 0 – FIFO. Newest data is discarded when full.
13	UINT8	Specifies type of data stored on the partition 0 – PUS Packets 1 – Raw Data
14	UINT8	Specifies which Virtual Channel to be used for downloading of the data in the partition.
15	UINT8	Priority during download. (0 – Highest priority)
16	UINT16	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 7-60: MMPartitionConfig data status codes

Status code	Description
0	Success.
EINVAL	Invalid argument. E.g. invalid partition number
EIO	I/O error. E.g. failed to access storage or nvram.

7.14.1.25. MMPartitionSpace

Reads the space available in the specified partition. Please note that due to the nature of the flash memory, as the read pointer advances, the space will become free only in leaps as the read pointer crosses a flash block edge. This means that a partition can have a discrepancy between reported free space and expected free space of maximum 1 block (2 MiB).

Table 7-61 MMPartitionSpace data

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

RMAP reply status:

Table 7-62: MMPartitionSpace status codes

Status code	Description
0	Success.
EINVAL	Invalid argument. E.g. invalid partition number

7.14.1.26. MMDownloadPartitionData

Downloads the data from the specified partition. The Virtual Channel used for the download is set by the partition configuration. The command is finished when as much data as possible of the requested length has been downloaded. If an error occurred during download, the command is aborted. If a download is already in progress when this command is received, the current download session continues and a reply message with status EALREADY is returned.

Table 7-63 MMDownloadPartitionData data.

Byte	Type	Description
0-3	UINT32	Length in bytes to download

RMAP reply status (if a reply is requested):

Table 7-64 MMDownloadPartitionData data status codes

Status code	Description
0	Success.
ENOSPC	Not enough data on partition.
EINVAL	Invalid argument. E.g. invalid partition number
EIO	I/O error. E.g. failed to access storage or nvram.
EALREADY	A download session is already in progress.
EBADMSG	Data was not successfully downloaded on downlink.

7.14.2. Output

7.14.2.1. TCCommand

A fully formed PUS packet according to [RD4] containing a TC packet to be routed.

7.14.2.2. UARTData

Routed data from UART.

Table 7-65: UARTData data

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

7.15. Handling of Rd/Wr-pointers and wrap-flags for partitions

When a power-loss occurs or the Sirius TCM is reset, the state of the Rd/Wr-pointers and wrap-flags of the partitions must be restored to a state so access of the partitions data can continue after the power-loss or reset.

During initialization of the Sirius TCM partition configuration, Rd/Wr-pointers and wrap-flags are read from NVRAM and populate an area in RAM storing partition information.

Access to a partition of the Mass Memory is handled in two steps. In the first step data is written or read to/from the partition. If the access is successful the Rd/Wr-pointers and wrap-flags are updated in RAM.

For read-accesses, the read-pointers and wrap-flags in RAM are also always stored to NVRAM.

Write accesses to a partition utilize a write-cache in RAM. When the write cache is full, the content of the write-cache is written to a page on the Mass Memory. During write-accesses to a partition the write-pointers and wrap-flags in RAM are also stored to NVRAM when a page is written to the Mass Memory. Note that the pointer and wrap information in RAM contains values for a previous access at this moment. By this strategy, the write-pointers stored in NVRAM will always contain a value of a successful and complete write-operation where all data has been stored on Mass Memory.

During initialization of the Sirius TCM, the write pointer and wrap flags of a partition are read from NVRAM to point to the last successful write access prior the restart of the Sirius TCM. The read-pointer of the partition is read from NVRAM. The read pointer is rewound to the same position as the write pointer if the read pointer read from NVRAM points to data that has not been written yet. The rewind might occur if a read-access occurred from the write cache prior the restart of the Sirius TCM. The purpose of the rewinding is to restore every partition in a pristine state, where pointers point to the last position data was written or read from the Mass Memory.

When the values of the write- and read-pointers and wrap-flags have been restored, a read-modify-write operation in the block of the last write operation for every partition is done to prepare the write-cache for every partition.

7.16. Limitations

For performance reasons, the current Sirius TCM release calculates checksums on neither the incoming nor the outgoing RMAP/SpaceWire packets.

8. System-on-Chip definitions

In this section the peripherals, memory sections and interrupts defined for the SoC for the Sirius OBC and Sirius TCM are described. Some of these might not be equipped in this release.

8.1. Memory mapping

Table 8-1 - Sirius memory structure definition

Memory Base Address	Function
0xF0000000	Boot ROM
0xE0000000	CCSDS (Sirius TCM only)
0xCB000000	Watchdog
0xCA000000	SpaceCraft Elapsed Time
0xC1000000	SoC info
0xC0000000	Error Manager
0xBD000000 - 0xBF000000	Reserved
0xBC000000	Reserved for SPI interface 1
0xBB000000	Reserved for SPI interface 0
0xBA000000	GPIO
0xB6000000	Reserved for ADC controller 1
0xB5000000	ADC controller 0
0xB4000000	Reserved
0xB3000000	Mass memory flash controller (Sirius TCM only)
0xB2000000	System flash controller
0xB1000000	Reserved
0xB0000000	NVRAM controller
0xAC000000	Reserved for PCIe
0xAB000000	Reserved for CAN
0xAA000000	Reserved for USB
0xA9000000 -0xA3000000	Reserved
0xA2000000	Reserved for redundant SpaceWire
0xA1000000	SpaceWire
0xA0000000	Reserved for Ethernet MAC
0x9C000000 -0x9F000000	Reserved
0x9B000000	Reserved for I2C interface 1
0x9A000000	Reserved for I2C interface 0
0x99000000	Reserved
0x98000000	UART 7 (Safe bus functionality, RS485)
0x97000000	UART 6 (PSU control functionality, RS485)
0x96000000	Reserved for High speed UART w. DMA
0x95000000	UART 4 (Routed to LVDS HK on Sirius TCM)
0x94000000	UART 3 (Routed to RS422 HK on Sirius TCM)
0x93000000	UART 2
0x92000000	UART 1
0x91000000	UART 0
0x90000000	UART Debug (LVTTL)
0x80000000 - 0x8F000000	Reserved for customer IP
0x00000000	SDRAM memory including EDAC (64 MB)

8.2. Interrupt sources

The following interrupts are available to the processor:

Table 8-2 - Sirius interrupt assignment

Interrupt no.	Function	Description
0-1	Reserved	Internal use
2	UART Debug	UART interrupt signal
3	UART 0	UART interrupt signal
4	UART 1	UART interrupt signal
5	UART 2	UART interrupt signal
6	UART 3	UART interrupt signal
7	UART 4	UART interrupt signal
8	UART 5	UART interrupt signal
9	UART 6	UART interrupt signal
10	UART 7	UART interrupt signal
11	ADC controller 0	ADC controller 0 interrupt signal
12	-	Avaliable (reserved for ADC controller 1)
13	-	Avaliable (reserved for I2C interface 0)
14	-	Avaliable (reserved for I2C interface 1)
15	-	Avaliable
16	-	Avaliable
17	SCET	SCET interrupt signal
18	Error manager	Error manager interrupt signal
19	-	Available (reserved for redundant SpaceWire)
20	System flash	System flash controller interrupt signal
21	Mass memory	Mass memory flash controller interrupt signal
22	Spacewire	SpaceWire interrupt signal
23	CCSDS	CCSDS interrupt signal
24	-	Available (reserved for Ethernet)
25	GPIO	GPIO interrupt signal
26	-	Available (reserved for SPI 0)
27	-	Available (reserved for SPI 1)
28	-	Avaliable (reserved for custom adaptation)
29	-	Avaliable (reserved for custom adaptation)
30	-	Avaliable (reserved for custom adaptation)

8.3. SCET timestamp trigger sources

Some of the peripherals in the SoC have the capability of sending a timestamp trigger signal on specific events. These signals are routed to the SCET which has a number of general purpose trigger registers (GP) where a snapshot of the SCET counter is stored for later retrieval by application software, see chapter 5.4. The tables below detail the mapping between the trigger signals and the general purpose trigger registers in the two products.

Table 8-3 General purpose trigger map

GP number	Trigger source	Description
0	power_loss	Triggered when the voltage drops below a certain level, i.e. power is lost to the board
1	ccsds	Triggered when telemetry sending on virtual channel 0 starts (Sirius TCM only)
2	gpio	Triggered when one of the pins input changes states and edge detection and timestamping are enabled
3	adc	Triggered when an ADC conversion is started

8.4. Boot images and boot procedure

8.4.1. Description

The bootrom is a small piece of software built into a read-only memory inside the System-on-Chip. Its main function is to load a software image from the system flash to RAM and start it by jumping to the reset vector (0x100). 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. The image to load can also be selected by setting the *Next FW* register in the Error Manager and doing a soft reset (see section 5.3 for more details). Boot order of the logical images and their physical copies is shown in Figure 8-1.

8.4.2. Block diagram

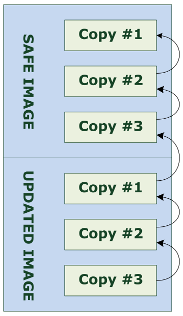


Figure 8-1 Software images in flash

8.4.3. Usage description

The locations in the system flash where the bootrom looks for software images are given in Table 8.4. The first two 32-bit words of the image are expected to be a header with image size and an XOR checksum, see Table 8.5. If the size falls within the accepted range, the bootrom loads the image to RAM while verifying the checksum.

The bootrom loads a table of bad blocks from the NVRAM. 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.

Table 8.4 Software image locations

Image	Flash page number
Safe copy #1	0x00000

Safe copy #2	0x20000
Safe copy #3	0x40000
Updated copy #1	0x80000
Updated copy #2	0xA0000
Updated copy #3	0xC0000

Table 8.5 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.

8.4.4. 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.

8.5. Reset behaviour

The SoC has a clock and reset block that synchronizes the external asynchronous reset to each clock domain. The internal soft reset, which can be commanded by software, follows the same design philosophy i.e. is also synchronized into the clock domain where it's used.

8.6. General synchronize method

All signals passing clock domain crossings are either handled via asynchronous two port FIFOs or synchronized into the other clock domain. Two flip-flops in series are used to reduce possible metastability effects. All external signals are synchronized into its clock domain following the above method.

8.7. Pulse command inputs

The pulse command inputs on 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-2 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.

8.8. SoC information map

The information included in the SoC info block for the Sirius products can be found in Table 8-6. This information must be fetched from the gdb prompt and can be used as a check of which SoC version that is flashed on the board. In a connected gdb prompt type:

```
x/3xw 0xC1000000
```

Table 8-6 Sirius SoC info

<table>
<tr><th>Base address number</th><th>Function</th><th colspan="2">Description</th></tr>
<tr><td>0x0</td><td>TIME_STAMP</td><td colspan="2">When building the SoC, a Unix timestamp is taken and put into the system. It is made as a 32 bit vector indicating seconds since 1970-01-01 (UTC).</td></tr>
<tr><td rowspan="5">0x4</td><td rowspan="5">PRODUCT_ID</td><td>• 0x00
• 0x01</td><td>OBC S BB
OBC SR – With SPW router 3 ports</td></tr>
<tr><td>• 0x08
• 0x09</td><td>OBC S FM
OBC SR FM – With SPW router 3 ports</td></tr>
<tr><td>• 0x10
• 0x11</td><td>TCM S BB
TCM S R – With SPW router 3 ports</td></tr>
<tr><td>• 0x18
• 0x19</td><td>TCM S FM
TCM SR FM – With SPW router 3 ports</td></tr>
<tr><td>• 0x20-0xFF</td><td>Reserved</td></tr>
<tr><td>0x8</td><td>SOC_VERSION</td><td colspan="2">Follows the methodology release 0.1.0 = Release-X.Y.Z,
First eight bits are reserved: 0x00XXYYZZ
X represents a major number, 8 bits
Y represents a minor number, 8 bits
Z represents a patch number, 8 bits
Representated in 32 bits.

Example: 0x00010203 = 1.2.3
Major version 1
Minor version 2
Patch number 3</td></tr>
</table>

9. Connector interfaces

9.1. JTAG-RTL, FPGA-JTAG connector

The following pins are available on the ST60-10P connector, see Table 9-1.

Table 9-1 - JTAG pin-outs

Pin #	Signal name	Description
Pin 1	GND	Ground
Pin 2	RTL-JTAG-TDI	Test Data In, data shifted into the device.
Pin 3	RTL-JTAG-TRSTB	Test Reset
Pin 4	VCC_3V3	Power supply
Pin 5	VCC_3V3	Power supply
Pin 6	RTL-JTAG-TMS	Test Mode Select
Pin 7	Not connected	-
Pin 8	RTL-JTAG-TDO	Test Data Out, data shifted out of the device
Pin 9	GND	Ground
Pin 10	RTL-JTAG-TCK	Test Clock

9.2. DEBUG-SW

The following pins are available on the ST60-18P, connector. See Table 9-2.

Table 9-2 - Debug SW pin-outs

Pin #	Signal name	Description
Pin 1	ETH-DEBUG-RESET	Reset
Pin 2	GND	Ground
Pin 3	ETH-DEBUG-SYNC	Not available
Pin 4	ETH-DEBUG-TX	Not available
Pin 5	ETH-DEBUG-RX	Not available
Pin 6	ETH-DEBUG-MDC	Not available
Pin 7	ETH-DEBUG-MDIO	Not available
Pin 8	ETH-DEBUG-CLK	Not available
Pin 9	GND	Ground
Pin 10	DEBUG-JTAG-TDI	Debug Test data in
Pin 11	DEBUG-JTAG-RX	Debug UART RX
Pin 12	DEBUG-JTAG-TX	Debug UART TX
Pin 13	VCC_3V3	Power supply
Pin 14	DEBUG-JTAG-TMS	Debug Test mode select
Pin 15	VCC_3V3	Power supply
Pin 16	DEBUG-JTAG-TDO	Debug Test data out
Pin 17	ETH-DEBUG-DETECT	Detect signal for the debugger

Pin 18	DEBUG-JTAG-TCK	Debug Test clock

9.3. PWR – Power

The following pins are available on the nano-D9 socket connector, see Table 9-4

Table 9-3 - Power pin-outs

Pin #	Signal name	Description
Pin 1	VBUS+	Power input
Pin 2	VBUS+	
Pin 3	UART7_RXTX_RS485_P	Safebus
Pin 4	UART7_RXTX_RS485_N	
Pin 5	PPS_RS422_P	PPS Distribution
Pin 6	PPS_RS422_N	
Pin 7	UART6_RXTX_RS485_P	PSU Control Interface
Pin 8	UART6_RXTX_RS485_N	
Pin 9	GND	Ground
Pin 10	GND	
Pin 11	GND	
Pin 12	PULSE0_I_RS422_P	Pulse Command 0
Pin 13	PULSE0_I_RS422_N	
Pin 14	PULSE1_I_RS422_P	Pulse Command 1
Pin 15	PULSE1_I_RS422_N	

9.4. SPW1 – Spacewire 1

The following pins are available on the nano-D9 socket connector, see Table 9-4

Table 9-4 - SPW1 pin-outs

Pin #	Signal name	Description
Pin 1	SPW1_DIN_LVDS_P	Data in, Positive
Pin 2	SPW1_SIN_LVDS_P	Strobe in, Positive
Pin 3	CGND	Chassis ground
Pin 4	SPW1_SOUT_LVDS_N	Strobe out, Negative
Pin 5	SPW1_DOUT_LVDS_N	Data out, Negative
Pin 6	SPW1_DIN_LVDS_N	Data in, Negative
Pin 7	SPW1_SIN_LVDS_N	Strobe in, Negative
Pin 8	SPW1_SOUT_LVDS_P	Strobe out, Positive
Pin 9	SPW1_DOUT_LVDS_P	Data out, Positive

9.5. SPW2 – Spacewire 2

The following pins are available on the nano-D9 socket connector, see Table 9-5

Table 9-5 – SPW2 pin-outs

Pin #	Signal name	Description
Pin 1	SPW2_DIN_LVDS_P	Data in, Positive
Pin 2	SPW2_SIN_LVDS_P	Strobe in, Positive
Pin 3	CGND	Chassis ground
Pin 4	SPW2_SOUT_LVDS_N	Strobe out, Negative
Pin 5	SPW2_DOUT_LVDS_N	Data out, Negative
Pin 6	SPW2_DIN_LVDS_N	Data in, Negative
Pin 7	SPW2_SIN_LVDS_N	Strobe in, Negative
Pin 8	SPW2_SOUT_LVDS_P	Strobe out, Positive
Pin 9	SPW2_DOUT_LVDS_P	Data out, Positive

9.6. ANALOG, Analog input and 3xGPIO (Sirius OBC only)

The following pins are available on the nanoD25 socket connector, see Table 9-5

Table 9-6 – ANALOGS, 4xGPIO pin-outs

Pin #	Signal name	Description
Pin 1	ADC_IN_0	Analog input channel
Pin 2	ADC_IN_1	Analog input channel
Pin 3	ADC_IN_2	Analog input channel
Pin 4	ADC_IN_3	Analog input channel
Pin 5	ADC_IN_4	Analog input channel
Pin 6	ADC_IN_5	Analog input channel
Pin 7	ADC_IN_6	Analog input channel
Pin 8	ADC_IN_7	Analog input channel
Pin 9	BIAS	2.5V bias voltage
Pin 10	BIAS	2.5V bias voltage
Pin 11	GPIO12	3.3V Digital I/O
Pin 12	GPIO13	3.3V Digital I/O
Pin 13	GPIO14	3.3V Digital I/O
Pin 14	GND	Ground
Pin 15	GND	Ground
Pin 16	GND	Ground
Pin 17	GND	Ground
Pin 18	GND	Ground
Pin 19	GND	Ground
Pin 20	GND	Ground

Pin 21	GND	Ground
Pin 22	BIAS	2.5V bias voltage
Pin 23	BIAS	2.5V bias voltage
Pin 24	GPIO15	3.3V Digital I/O
Pin 25	GND	Ground

9.7. DIGITAL, PPS input and 12xGPIO

The following pins are available on the nanoD25 socket connector, see Table 9-6

Table 9-7 DIGITALS pinouts

Pin #	Signal name	Description
Pin 1	GPIO0	Digital input/output
Pin 2	GPIO1	Digital input/output
Pin 3	GPIO2	Digital input/output
Pin 4	GPIO3	Digital input/output
Pin 5	GPIO4	Digital input/output
Pin 6	GPIO5	Digital input/output
Pin 7	GPIO6	Digital input/output
Pin 8	GPIO7	Digital input/output
Pin 9	GPIO8	Digital input/output
Pin 10	GPIO9	Digital input/output
Pin 11	GPIO10	Digital input/output
Pin 12	GPIO11	Digital input/output
Pin 13	GND	Board ground
Pin 14	Not used	Not used
Pin 15	Not used	Not used
Pin 16	Not used	Not used
Pin 17	Not used	Not used
Pin 18	Not used	Not used
Pin 19	Not used	Not used
Pin 20	Not used	Not used
Pin 21	Not used	Not used
Pin 22	Not used	Not used
Pin 23	PPS_INPUT_RS422_N	Pulse per second, differential RS422 signal for time synchronization
Pin 24	PPS_INPUT_RS422_P	
Pin 25	GND	Board ground

9.8. UART0-2 - RS422/485

The following pins are available on the nanoD15 socket connector, see Table 9-7

Table 9-8 COM02_RS4XX pinouts

Pin #	Signal name	Description
Pin 1	UART0_RX_RS4XX_P	UART 0 RX
Pin 2	UART0_RX_RS4XX_N	
Pin 3	UART0_TX_RS4XX_P	UART 0 TX
Pin 4	UART0_TX_RS4XX_N	
Pin 5	GND	Ground
Pin 6	GND	
Pin 7	UART1_RX_RS4XX_P	UART 1 RX
Pin 8	UART1_RX_RS4XX_N	
Pin 9	UART1_TX_RS4XX_P	UART 1 TX
Pin 10	UART1_TX_RS4XX_N	
Pin 11	UART2_RX_RS4XX_P	UART 2 RX
Pin 12	UART2_RX_RS4XX_N	
Pin 13	UART2_TX_RS4XX_P	UART 2 TX
Pin 14	UART2_TX_RS4XX_N	
Pin 15	GND	Ground

9.9. UART3-5 - RS422/485 (Sirius OBC only)

The following pins are available on the nanoD15 socket connector, see Table 9-8

Table 9-9 COM35_RS4XX pin-outs

Pin #	Signal name	Description
Pin 1	UART3_RX_RS4XX_P	UART 3 RX
Pin 2	UART3_RX_RS4XX_N	
Pin 3	UART3_TX_RS4XX_P	UART 3 TX
Pin 4	UART3_TX_RS4XX_N	
Pin 5	GND	Ground
Pin 6	GND	
Pin 7	UART4_RX_RS4XX_P	UART 4 RX
Pin 8	UART4_RX_RS4XX_N	
Pin 9	UART4_TX_RS4XX_P	UART 4 TX
Pin 10	UART4_TX_RS4XX_N	
Pin 11	UART5_RX_RS4XX_P	UART 5 RX
Pin 12	UART5_RX_RS4XX_N	
Pin 13	UART5_TX_RS4XX_P	UART 5 TX
Pin 14	UART5_TX_RS4XX_N	

Pin 15	GND	Ground

9.10. TRX1 - RS422 Transceiver interface (Sirius TCM only)

The following pins are available on the nano-D25, socket connector, see Table 9-9. This connector can, for instance, be used for an S-BAND radio.

Table 9-10 TRX1 pin-outs

Pin #	Signal name	Description
Pin 1	TRX1_DOUT_RS422_P	RS422 output
Pin 2	TRX1_DOUT_RS422_N	
Pin 3	TRX1_COUT_RS422_P	RS422 output
Pin 4	TRX1_COUT_RS422_N	
Pin 5	TRX1_DIN_RS422_P	RS422 Input
Pin 6	TRX1_DIN_RS422_N	
Pin 7	TRX1_CIN_RS422_P	RS422 Input
Pin 8	TRX1_CIN_RS422_N	
Pin 9	TRX1_SCAL_IN_RS422_P	RS422 Input
Pin 10	TRX1_SCAL _IN_RS422_N	
Pin 11	TRX1_C_LOCK_IN_RS422_P	RS422 Input
Pin 12	TRX1_C_LOCK_IN_RS422_N	
Pin 13	GND	Ground
Pin 14	GND	Ground
Pin 15	GND	Ground
Pin 16	GND	Ground
Pin 17	GND	Ground
Pin 18	GND	Ground
Pin 19	GND	Ground
Pin 20	UART4_TX_RS422_P	RS422 output
Pin 21	UART4_TX_RS422_N	
Pin 22	UART4_RX_RS422_P	RS422 input
Pin 23	UART4_RX_RS422_N	
Pin 24	TRX1_DETECT	Transceiver detect input
Pin 25	GND	Ground

9.11. TRX2 - LVDS Transceiver interface (Sirius TCM only)

The following pins are available on the nano-D25, socket connector, see Table 9-9. This connector can, for instance, be used for an X-BAND or S-BAND radio.

Table 9-11 TRX2 pin-outs

Pin #	Signal name	Description
Pin 1	TRX2_DOUT_LVDS_P	Baseband data out, LVDS
Pin 2	TRX2_DOUT_LVDS_N	
Pin 3	TRX2_COUT_LVDS_P	Baseband clock out, LVDS

Pin 4	TRX2_COUT_LVDS_N	
Pin 5	TRX2_DIN_LVDS_P	Baseband data in, LVDS
Pin 6	TRX2_DIN_LVDS_N	
Pin 7	TRX2_CIN_LVDS_P	Baseband clock in, LVDS
Pin 8	TRX2_CIN_LVDS_N	
Pin 9	TRX2_SCAL_IN_LVDS_P	Sub-carrier lock in, LVDS
Pin 10	TRX2_SCAL_IN_LVDS_N	
Pin 11	TRX2_CAL_IN_LVDS_P	Carrier lock in, LVDS
Pin 12	TRX2_CAL_IN_LVDS_N	
Pin 13	GND	Ground
Pin 14	UART3_TX_LVDS_P	LVDS output
Pin 15	UART3_TX_LVDS_N	
Pin 16	UART3_RX_LVDS_P	LVDS input
Pin 17	UART3_RX_LVDS_N	
Pin 18	GND	Ground
Pin 19	GND	
Pin 20	UART3_TX_RS422_P	RS422 output
Pin 21	UART3_TX_RS422_N	
Pin 22	UART3_RX_RS422_P	RS422 input
Pin 23	UART3_RX_RS422_N	
Pin 24	TRX2_DETECT	Transceiver detect input
Pin 25	GND	Ground

9.12. UMBI – Baseband Umbilical (Sirius TCM only)

The following pins are available on the nano-D15 socket connector, see Table 9-10

Table 9-12 UMBI pin-outs

Pin #	Signal name	Description
Pin 1	UMBI_DOUT_RS422_P	Baseband data out
Pin 2	UMBI_DOUT_RS422_N	
Pin 3	UMBI_COUT_RS422_P	Baseband clock out
Pin 4	UMBI_COUT_RS422_N	
Pin 5	UMBI_DIN_RS422_P	Baseband data in
Pin 6	UMBI_DIN_RS422_N	
Pin 7	UMBI_CIN_RS422_P	Baseband clock in
Pin 8	UMBI_CIN_RS422_N	
Pin 9	UMBI_SC_LOCK_IN_RS422_P	Sub-carrier lock in
Pin 10	UMBI_SC_LOCK_IN_RS422_N	
Pin 11	UMBI_C_LOCK_IN_RS422_P	Carrier lock in
Pin 12	UMBI_C_LOCK_IN_RS422_N	

Pin 13	UMBI_DETECT	Umbilical detect input
Pin 14	GND	Ground (reference)
Pin 15	GND	Ground (reference)

9.13. Pulse Command Outputs (Sirius TCM only)

The following pins are available on the nano-D25, socket connector, see Table 9-10

Table 9-13 Pulse command pin-outs

Pin #	Signal name	Description
Pin 1	PULSE1_O_RS422_P	RS422 output
Pin 2	PULSE1_O_RS422_N	
Pin 3	PULSE2_O_RS422_P	RS422 output
Pin 4	PULSE2_O_RS422_N	
Pin 5	PULSE3_O_RS422_P	RS422 output
Pin 6	PULSE3_O_RS422_N	
Pin 7	PULSE4_O_RS422_P	RS422 output
Pin 8	PULSE4_O_RS422_N	
Pin 9	PULSE5_O_RS422_P	RS422 output
Pin 10	PULSE5_O_RS422_N	
Pin 11	PULSE6_O_RS422_P	RS422 output
Pin 12	PULSE6_O_RS422_N	
Pin 13	GND	Ground
Pin 14	PULSE7_O_RS422_P	RS422 output
Pin 15	PULSE7_O_RS422_N	
Pin 16	PULSE8_O_RS422_P	RS422 output
Pin 17	PULSE8_O_RS422_N	
Pin 18	PULSE9_O_RS422_P	RS422 output
Pin 19	PULSE9_O_RS422_N	
Pin 20	PULSE10_O_RS422_P	RS422 output
Pin 21	PULSE10_O_RS422_N	
Pin 22	PULSE11_O_RS422_P	RS422 output
Pin 23	PULSE11_O_RS422_N	
Pin 24	PULSE12_O_RS422_P	RS422 output
Pin 25	PULSE12_O_RS422_N	

10. Updating the Sirius FPGA

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

10.1. Prerequisite hardware

- Microsemi FlashPro5 unit
- 104470 FPGA programming cable assembly

10.2. Prerequisite software

- Microsemi FlashPro Express v11.7 or later
- The updated FPGA firmware

10.3. 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 connector 4 in Figure 3-1
2. Connect the power cables according to Figure 3-1
3. The updated FPGA firmware delivery from ÅAC 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 10-1) and select the supplied .pro file.

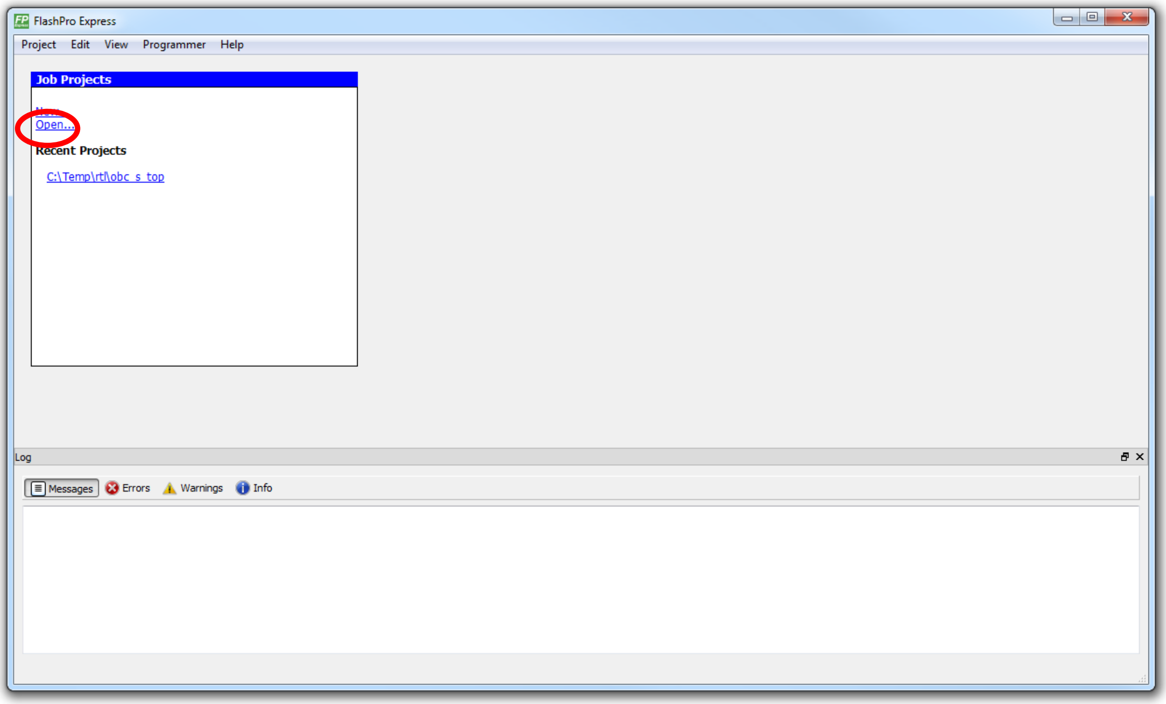


Figure 10-1 - Startup view of FlashPro Express

5. Once the file has loaded (warnings might appear), click RUN (see Figure 10-2). Please note that the connected FlashPro5 programmed ID should be shown.

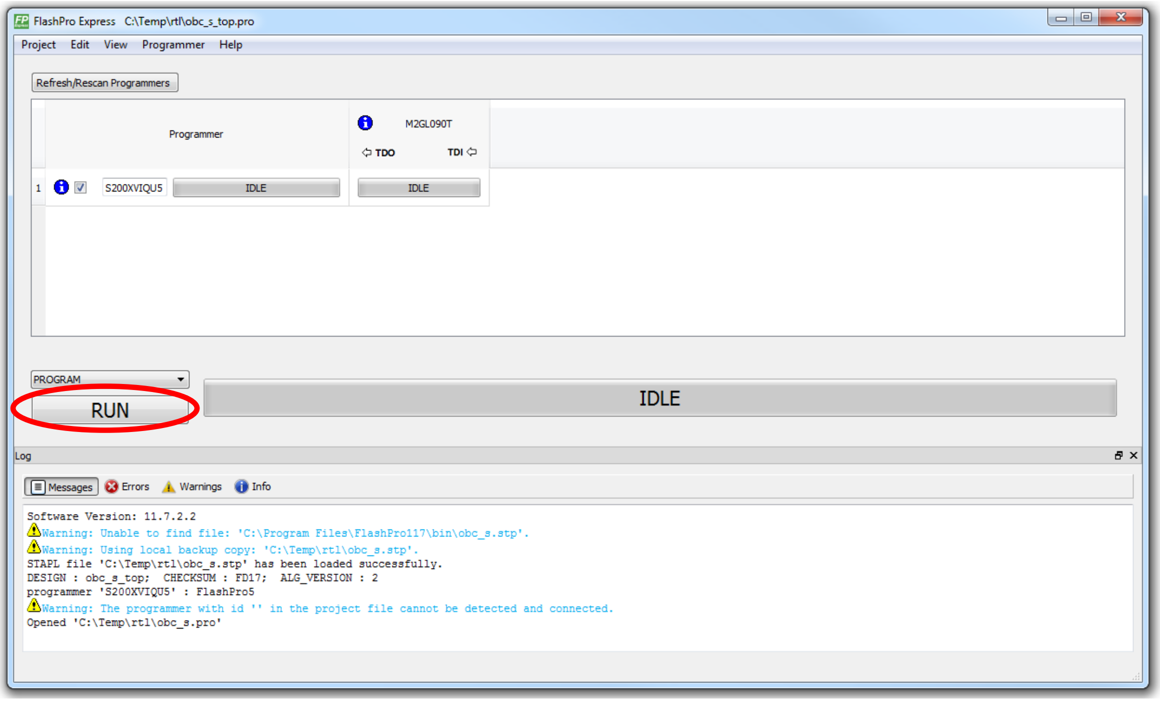


Figure 10-2 - View of FlashPro Express with project loaded.

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 10-3.

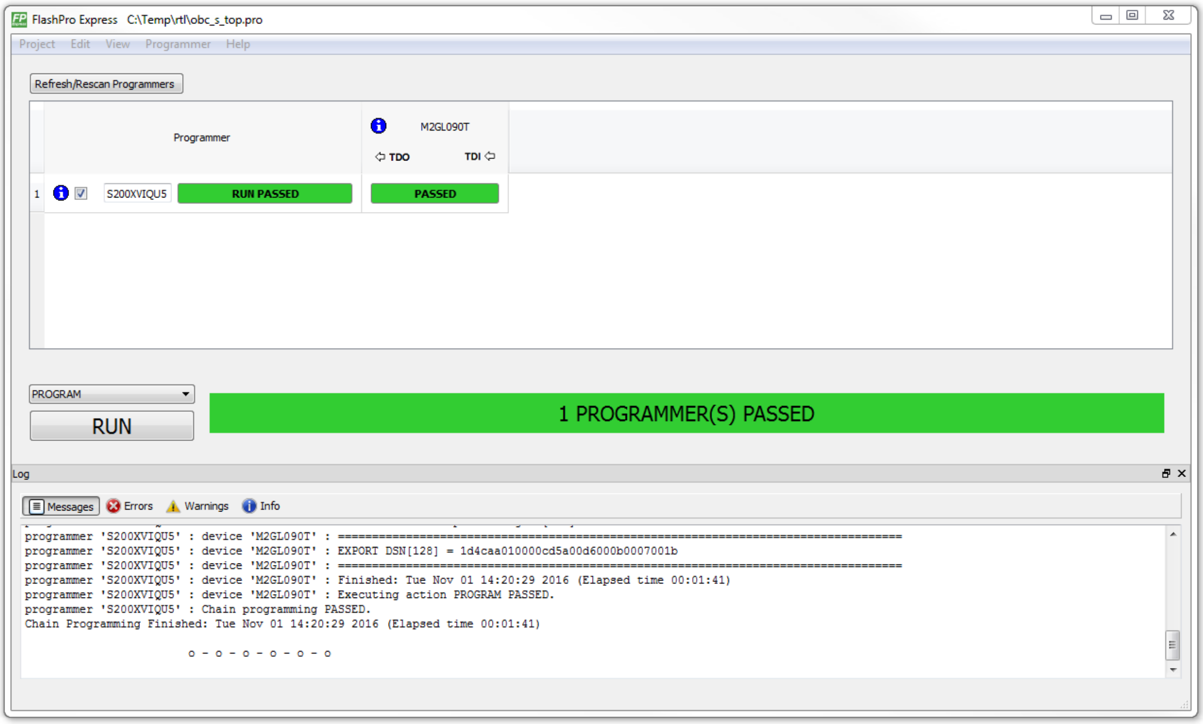


Figure 10-3 - View of FlashPro Express after program passed.

The Sirius FPGA image is now updated

11. Mechanical data

The total size of the Sirius board is 183x136 mm.

Mounting holes are ø3.4 mm with 4.5 mm pad size.

12. Glossary

ADC	Analog Digital Converter
APID	Application Process ID
BSP	Board Support Package
CCSDS	The Consultative Committee for Space Data Systems
EDAC	Error Detection and Correction
EM	Engineering model
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
GCC	GNU Compiler Collection program (type of standard in Unix)
GPIO	General Purpose Input/Output
Gtkterm	Is a terminal emulator that drives serial ports
I^2C	Inter-Integrated Circuit, generally referred as “two-wire interface” is a multi-master serial single-ended computer bus invented by Philips.
JTAG	Joint Test Action Group, interface for debugging the PCBs
LVTTL	Low-Voltage TTL
Minicom	Is a text based modem control and terminal emulation program
NA	Not Applicable
NVRAM	Non Volatile Random Access Memory
OBC	On Board Computer
OS	Operating System
PCB	Printed Circuit Board
PCBA	Printed Circuit Board Assembly
POSIX	Portable Operating System Interface
PUS	Packet Utilization Standard
RAM	Random Access Memory, however modern DRAM has not random access. It is often associated with volatile types of memory
ROM	Read Only Memory
RTEMS	Real-Time Executive for Multiprocessor Systems
SCET	SpaceCraft Elapsed Timer
SoC	System-on-Chip
SPI	Serial Peripheral Interface Bus is a synchronous serial data link which sometimes is called a 4-wire serial bus.
TC	Telecommand
TCL	Tool Command Language, a script language
TCM	Mass memory
TM	Telemetry
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

Headquarters

NASA ARP office

ÅAC Microtec AB
Dag Hammarskjölds väg 48
751 83 Uppsala
Sweden
T: +46 18 560 130
E: info@aacmicrotec.com
W: www.aacmicrotec.com

ÅAC Microtec Inc.
NASA Ames Research Park Bldg 19
Moffett Field CA 94035
USA
T: +1 844 831-7158
E: info@aacmicrotec.com
W: www.aacmicrotec.com