Document number	204911
Version	Rev. J
Issue date	2016-09-07

Sirius Breadboard User Manual

Rev. J

REVISION LOG

Rev	Date	Change description
A	2015-11-10	First Release
B	2016-03-07	Updates for new release with lots of minor corrections and clarifications.
C	2016-03-18	Version C released with the following updates: • TCM-S chapter 6 updated • UART chapter update • Spacewire router chapter 6 added. • Added GPIO chapter • Updated SCET ioctl • Corrected BSP section to be board-agnostic
D	2016-03-23	Added driver API for CCSDS
E	2016-05-01	Version E released with the following updates: • GPIO chapter updated • UART32 chapter added • TCM-S chapter updated Lots of minor corrections and fixes.
F	2016-05-03	Added missing section on TCM-S.
G	2016-06-10	Added NVRAM and PUS 1 commands. Editorial changes.
H	2016-06-30	Version H released with the following updates: • PUS 2 commands CDPU • SoC specs • TCM-S core app updates • SPW byte alignment
I	2016-09-05	Version I released with the following updates: • GPIO example • HK equations • ADC section • nandflash program bad blocks handling • Boot image information and boot order • Added section on pulse command inputs • Adjusted TCM-S section and synched with TCM-S DDD • Removed TCM-S download command as this isn't supported in this release.
J	2016-09-07	Corrected TCM-S RMAP address and command errors that snuck into release version I.

TABLE OF CONTENT

1. Introduction

This manual describes the functionality and usage of the ÅAC Sirius Breadboard. The Breadboard is a prototype board for products under development, which means that not all functions are implemented yet. The OBC-S™ and TCM-S™ functionality is described and can both run on the breadboard. The breadboard has fitted or non-fitted components and unique SoCs that give the desired functionality to match either the OBC-S™ or TCM-S™.

1.1. Applicable releases

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

OBC-S	0.6.1
TCM-S	0.6.1

1.2. Intended users

This manual is written for the software engineers using the ÅAC Sirius product suite.

1.3. Getting support

If you encounter any problem using the breadboard 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 Protocotol

2. Equipment information

The Sirius Breadboard is a prototyping platform designed to support the TCM-S^{TM}, and the OBC-S^{TM} products. The Breadboard layout is depicted in Figure 3-1.

The development board supports both a debugger interface for developing software applications and a JTAG interface for upgrading the FPGA firmware.

The FPGA firmware implements SoC based on 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), 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 differ between the products) uses the RS422 and RS485 line drivers on the board with line driver mode set by software.
- GPIOs
- Watchdog, fail-safe mechanism to prevent a system lockup
- System flash of 2 GB with EDAC-protection for storing boot images in multiple copies

For the TCM-S^{TM} 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.

The input power supply provided to the breadboard shall use a range of +4.5V to absolute max. of +16V. Nominal voltage supply level shall be set to +5V. The power consumption is highly dependent on peripheral loads and it 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 OBC-S^{TM} and TCM-S^{TM} products. The figure shows what parts are for which products and what parts are not yet implemented since the products are still under development.

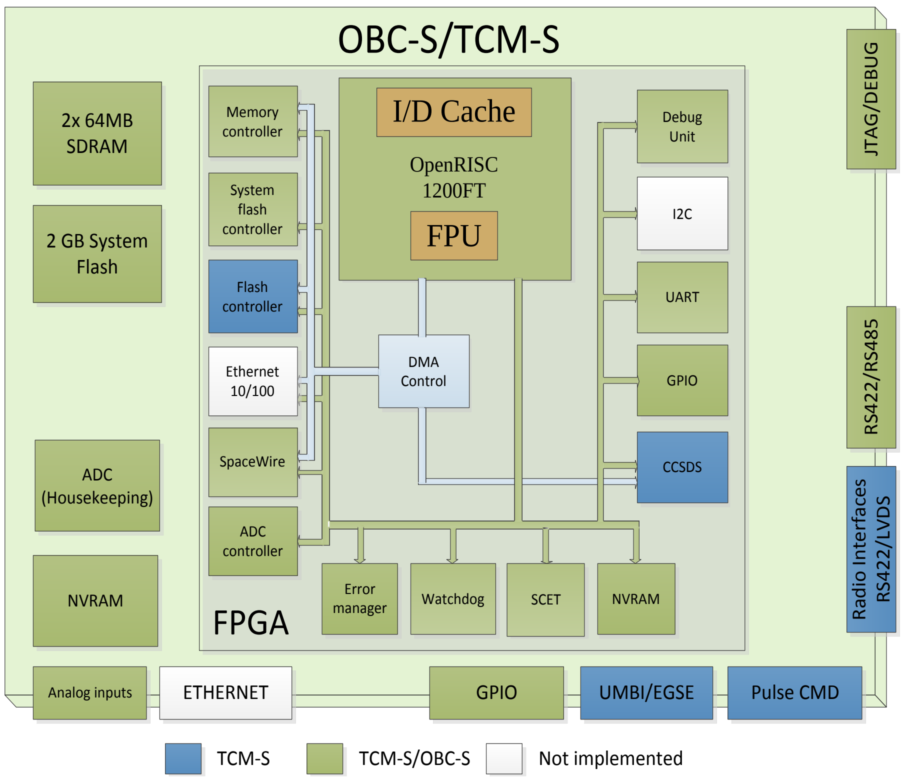


Figure 2-1 - The OBC-S^{TM} / TCM-S^{TM} SoC Overview

3. Setup and operation

3.1. User prerequisites

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

PC computer

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

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

For FPGA update capabilities

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

3.2. Connecting cables to the Sirius Breadboard

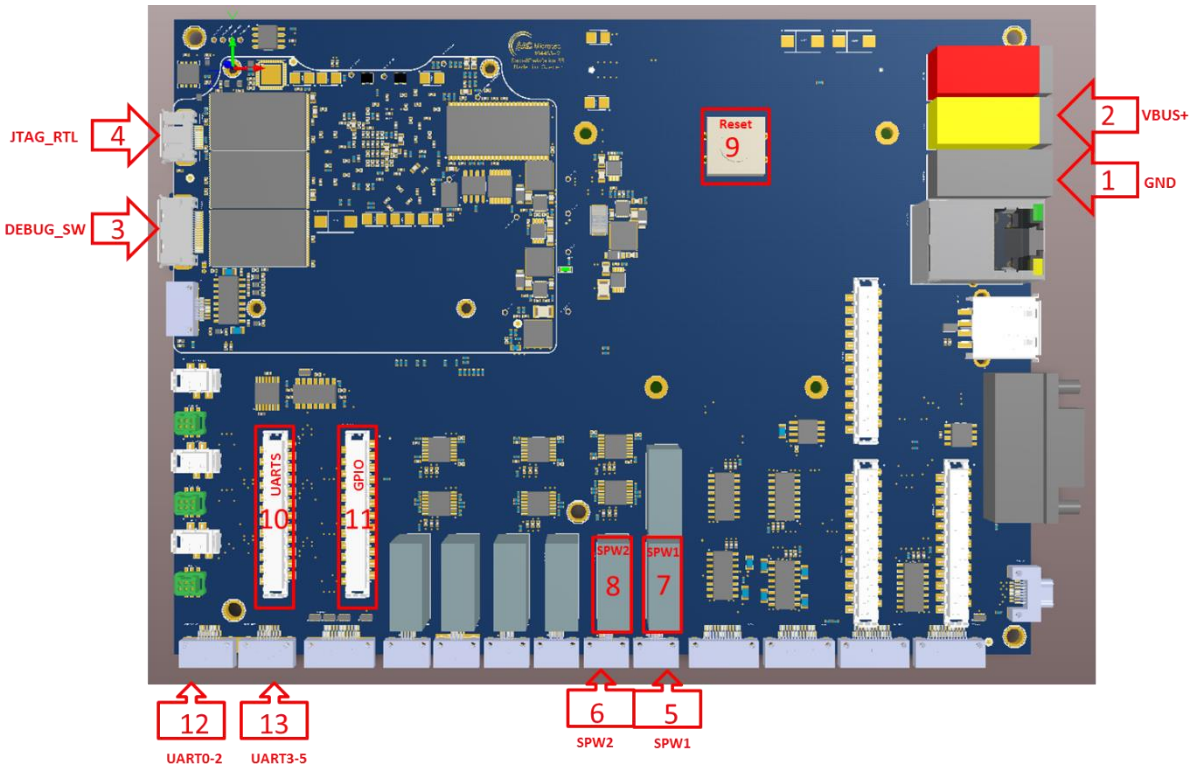


Figure 3-1 – ÅAC Sirius Breadboard with connector numbering

The Sirius Breadboard runs on a range of 4.5 to 16V DC. The instructions below refer to the connector numbering in Figure 3-1.

- Connect Ground to the black connector 1
- Connect 4.5 - 16 V DC to the yellow connector 2. The unit will nominally draw about 260-300 mA @5V DC.
- Connect the 104451 ÅAC Debugger and Ethernet adapter with the 104471 Ethernet debug unit cable to connector 3. Connect the adapter USB-connector to the host PC. The ÅAC debugger is mainly used for development of custom software for the OBC-S with monitoring/debug capabilities, but is also used for programming an image to the system flash memory. For further information refer to chapter 3.6.
- For FPGA updating only: Connect a FlashPro programmer to connector 4 using the 104470 FPGA programming cable assembly. For further information how to update the SoC refer to Chapter 9.9.
- For connecting the SpaceWire:
    - Option 1: Connect the nano-D connector to connector 5 or 6. Be careful when plugging and unplugging this connector.

- Option 2: Connect the Display port cable to connector 7 or 8 and to the 104510 Converter board. Connect your SpaceWire system to the converter board with the SpaceWire cable.

- Connecting UARTs:
  - Option 1: Connect to the nano-D number 12 (UART0-2) or 13 (UART3-5). Be careful when plugging and unplugging this connector.
  - Option 2: Connect to the debug connector 10 using a flat cable to DSUB connector harness. This can then be connected to a PC using something similar to the FTDI USB-COM485/COM422-PLUS4.

For more detailed information about the connectors, see section 8.4.

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 either be downloaded from http://repo.aacmicrotec.com/bsp or copied from the accompanying DVD. 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 - OBC-S or TCM-s and y matches the current version number of that BSP).

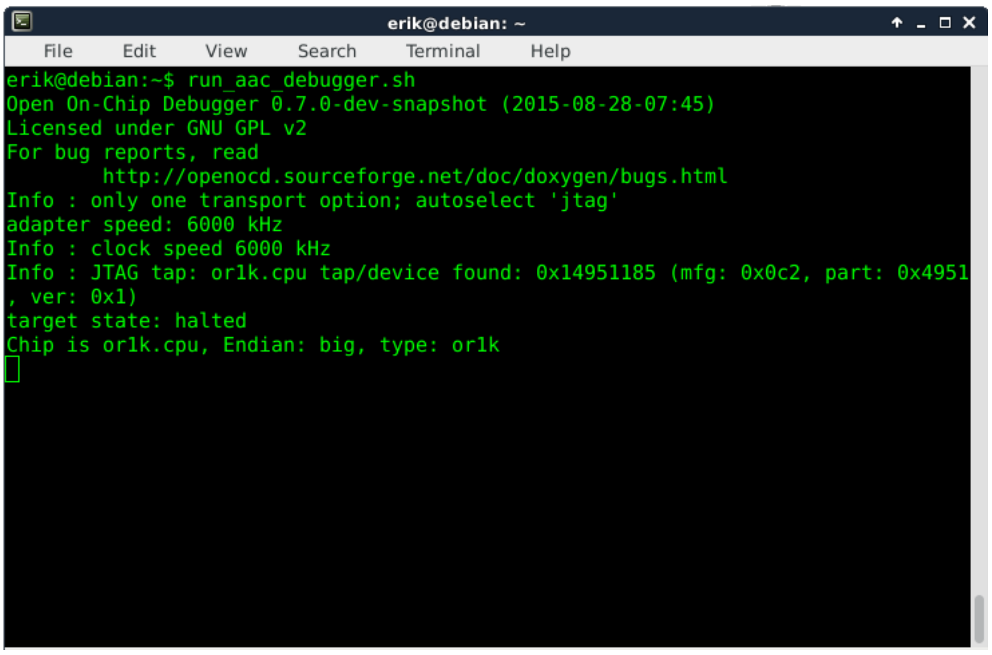
The newly created directory aac-or1k-xxx-x-bsp now contains the drivers for both bare-metal applications and RTEMS. See the included README and chapter 4.1 for build instructions.

3.5. Deploying a Sirius application

3.5.1. Establish a debugger connection to the Breadboard

The Sirius Breadboard is shipped with a debugger which connects to the PC via USB. To interface the Breadboard, 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 Breadboard according to section 3.
2. Start the run_aac_debugger.sh script from a terminal.
3. If the printed message is according to Figure 3-2, the connection is working.



```
erik@debian:~$ run_aac_debugger.sh
Open On-Chip Debugger 0.7.0-dev-snapshot (2015-08-28-07:45)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
adapter speed: 6000 kHz
Info : clock speed 6000 kHz
Info : JTAG tap: or1k.cpu tap/device found: 0x14951185 (mfg: 0x0c2, part: 0x4951
, ver: 0x1)
target state: halted
Chip is or1k.cpu, Endian: big, type: or1k
```

Figure 3-2 - Successful OpenOCD connection to the Breadboard

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 Breadboard, 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 the application

Application loading during the development stages (before programming to flash) are 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 start an executable program in hardware, first specify it's 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 and then uploaded to target just as an ordinary application is started through gdb. The maximum allowed size for the boot image for this release is 16 Mbyte. The nandflash_program application can be found in the BSP, see also instructions below.

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. Then make sure that this is in a binary-only format and not ELF. This can otherwise be accomplished with the help of the gcc tools 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. Follow the instructions on screen and when it's ready, reboot the board by resetting or power cycling.

4. Software development

Applications to be deployed on the Sirius Breadboard 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.

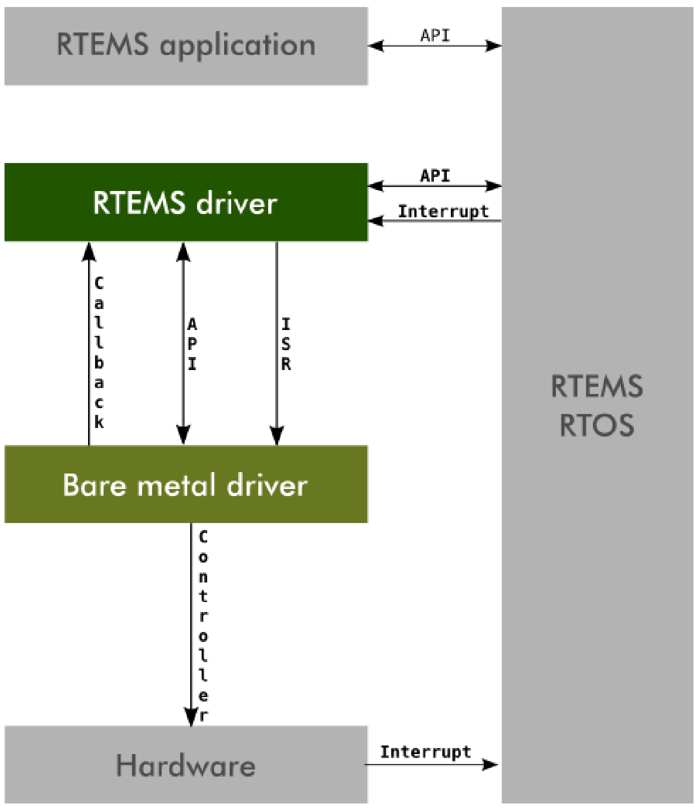


Figure 5-1 - Functionality access via RTEMS API

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	0 – 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

To enable the watchdog use the wdt_enable() function.

To disable the watchdog use the wdt_disable() function.

The watchdog must be kicked using wdt_kick() 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.

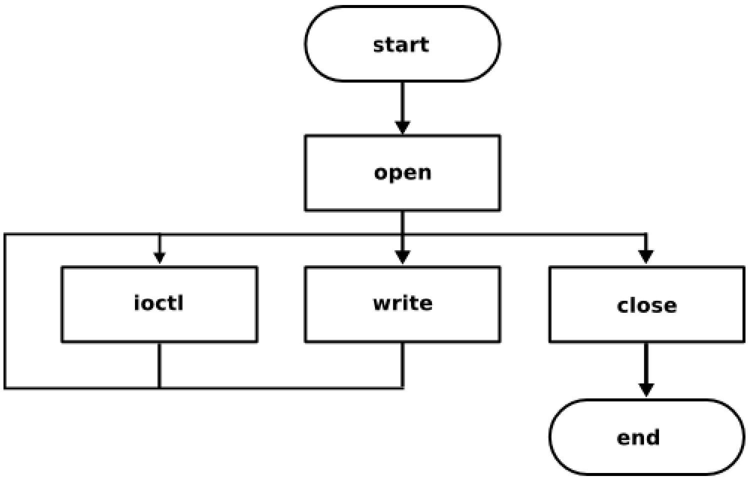


Figure 5-2 - RTEMS driver usage description

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	int	in	Command to send
val	int	in	Buffer to either read to or write from

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	Gets the next boot firmware
ERRMAN_GET_RUNFW_IOCTL	Gets the running firmware
ERRMAN_GET_SCRUBBER_IOCTL	Gets the scrubber. 1 = On, 0 = Off
ERRMAN_GET_RESET_ENABLE_IOCTL	Gets the reset enable register
ERRMAN_GET_WDT_ERRCNT_IOCTL	Gets the watchdog error count register
ERRMAN_GET_EDAC_SINGLE_ERRCNT_IOCTL	Gets the EDAC single error count register
ERRMAN_GET_EDAC_MULTI_ERRCNT_IOCTL	Gets the EDAC multiple error count register
ERRMAN_GET_CPU_PARITY_ERRCNT_IOCTL	Gets the CPU Parity error count register
ERRMAN_GET_POWER_LOSS_ENABLE_IOCTL	Gets the power loss enable state
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
ERRMAN_SET_RUNFW_IOCTL	Sets the running firmware
ERRMAN_RESET_SYSTEM_IOCTL	Performs a software reset 1 = Reset system
ERRMAN_SET_SCRUBBER_IOCTL	Sets the scrubber. 1 = Enable scrubber, 0 = Disable scrubber
ERRMAN_SET_RESET_ENABLE_IOCTL	Sets the reset enable register
ERRMAN_SET_WDT_ERRCNT_IOCTL	Sets the watchdog error count register
ERRMAN_SET_EDAC_SINGLE_ERRCNT_IOCTL	Sets the EDAC single error count register

ERRMAN_SET_EDAC_MULTI_ERRCNT_IOCTL	Sets the EDAC multiple error count register
ERRMAN_SET_CPU_PARITY_ERRCNT_IOCTL	Sets the CPU Parity error count register
ERRMAN_SET_POWER_LOSS_ENABLE_IOCTL	Sets the power loss enable state

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

5.3.2.3.1. Status register

Bit position	Name	Direction	Description
31:16	RESERVED		
15	ERRMAN_MM_MEFLG	R/W	A previous EDAC Multiple Error Reset has been detected for mass memory flash data Clear flag by write a '1'
14	ERRMAN_MM_SEFLG	R/W	A previous EDAC Single Error Reset has been detected for mass memory flash data Clear flag by write a '1'
13	ERRMAN_SYS_MEFLG	R/W	A previous EDAC Multiple Error Reset has been detected for system flash data Clear flag by write a '1'
12	ERRMAN_SYS_SEFLG	R/W	A previous EDAC Single Error Reset has been detected for system flash data Clear flag by write a '1'
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
31:9	RESERVED		
6	RESERVED		
5	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’)
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	-	

5.3.3. 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.

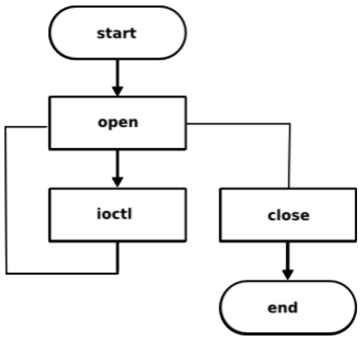


Figure 5-3 - RTEMS driver usage description

Interrupt message queue

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

This section describes the driver as a utility for accessing the SCET device.

5.4.2. 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.

SCET accesses can either be done by reading and writing to the device file. In this way the second and subsecond values can be read and/or modified.
The SCET RTEMS driver also supports a number of different IOCTLs.
Finally there is a message queue interface allowing the application to act upon different events.

5.4.2.1. 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. 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, whether it should be cleared when opened, etc).

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

5.4.2.2. 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

5.4.2.3. int ioctl(…)

Ioctl allows for disabling/enabling of the SCET 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	Value to write or a pointer to a buffer where data will be written.

Command table	Type	Direction	Description
SCET_GET_SECONDS_IOCTL	uint32_t	out	Returns the current number of seconds
SCET_GET_SUBSECONDS_IOCTL	uint32_t	out	Returns the current fraction of a second
SCET_GET_PPS_SOURCE_IOCTL	uint32_t	out	Returns the current set PPS source
SCET_GET_GP_TRIGGER_LEVEL_IOCTL	uint32_t	in/out	val input argument is the GP Trigger. Returns the currently configured level of the selected GP trigger
SCET_GET_INTERRUPT_ENABLE_IOCTL	uint32_t	out	Returns the current interrupt level register
SCET_GET_INTERRUPT_STATUS_IOCTL	uint32_t	out	Returns the current interrupt status register
SCET_GET_PPS_ARRIVE_COUNTER_IOCTL	uint32_t	out	Returns the PPS arrived counter. Bit 23:16 contains lower 8 bits of second. Bit 15:0 contains fraction of second
SCET_GET_GP_TRIGGER_COUNTER_IOCTL	uint32_t *	in/out	Pointer input argument is the GP trigger. Returns the counter of the selected GP trigger. Bit 23:16 contains lower 8 bits of second. Bit 15:0 contains fraction of second
SCET_GET_SECONDS_ADJUST_IOCTL	int32_t	out	Returns the value of the second adjust register
SCET_GET_SUBSECONDS_ADJUST_IOCTL	int32_t	out	Returns the value of the subsecond adjust register
SCET_GET_PPS_O_EN_IOCTL	uint32_t	out	Returns whether the external PPS out driver is enabled or not. 0 = Driver is disabled 1 = Driver is enabled
SCET_SET_SECONDS_IOCTL	int32_t	in	Input argument is the new second value to set
SCET_SET_SUBSECONDS_IOCTL	int32_t	in	Input argument is the new subsecond value to set
SCET_SET_INTERRUPT_ENABLE_IOCTL	uint32_t	in	Sets the interrupt enable mask register
SCET_SET_INTERRUPT_STATUS_IOCTL	uint32_t	in	Sets the interrupt status register
SCET_SET_PPS_SOURCE_IOCTL	uint32_t	in	Sets the PPS source. 0 = External PPS source 1 = Internal PPS source

SCET_SET_GP_TRIGGER_LEVEL_IOCTL	uint32_t *	in/out	Pointer input argument selects which GP trigger. Return value is the current value of that trigger. 0 = trigger activates on a rising edge transition 1 = trigger activates on falling edge transition
SCET_SET_PPS_O_EN_IOCTL	uint32_t	In	Controls if the external PPS out driver is enabled or not. 0 = Driver is disabled 1 = Driver is enabled

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

5.4.3. Usage

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.

By utilizing the GP triggers one can trap the timestamp of different events. An interrupt trigger can optionally be set up to notify the CPU of that the GP trigger has fired.

If an external PPS source is used, an interrupt trigger can be used to synchronize the SCET by reading out the SCET second and subsecond value at the time of the external PPS trigger. This value can then be subtracted from the current second and subsecond value to calculate a time difference.
This time difference can then be written to the adjustment registers to align the local time to the external pulse.

5.4.3.1. RTEMS

The RTEMS driver must be opened before it can access the SCET 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 device can be closed when not needed.

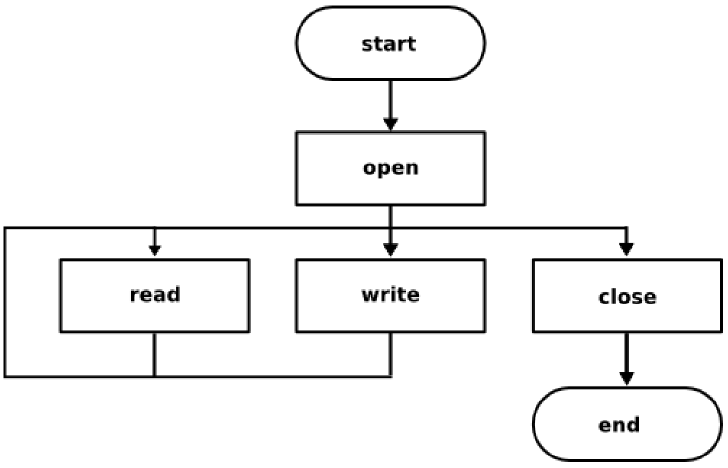


Figure 5-4 - RTEMS driver usage description

5.4.3.1.1. Time handling

Getting the current SCET time in RTEMS can be done in two ways:
1. Using read call, reading 6 bytes.
The first four bytes contains the second count.
The two last bytes contain the subsecond count.

2. Using the SCET_GET_SECONDS_IOCTL and SCET_GET_SUBSECONDS_IOCTL system calls defined in 5.4.2.3.

Adjusting the SCET time is done the same way as getting the SCET time but reversed.
You can either:
1. Write 6 bytes to the device. The first 4 bytes contains the second count **difference** to adjust with.
The last 2 bytes contains the subsecond count **difference** to adjust with.

2. Using the SCET_SET_SECONDS_IOCTL and SCET_SET_SUBSECONDS_IOCTL system calls defined in 5.4.2.3.

Negative adjustment is done by writing data in two complement notations.

5.4.3.1.2. Event callback via message queue

The SCET driver exposes three message queues.

This queue is used to emit messages 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_*

Event name	Description
PPS_ARRIVED	An external PPS signal has arrived. Use the SCET_GET_PPS_ARRIVE_COUNTER_IOCTL to get the timestamp of the external PPS signal in relation to the local SCET counter
PPS_LOST	The external PPS signal is lost
PPS_FOUND	The external PPS signal was found

‘S’, ‘G’, ‘T’, ‘0’ handles messages sent from the general purpose trigger 0.

Event name	Description
TRIGGER0	Trigger 0 was triggered

‘S’, ‘G’, T’, ‘1’ handles messages sent from the general purpose trigger 1.

Event name	Description
TRIGGER1	Trigger 1 was triggered

‘S’, ‘G’, T’, ‘2’ handles messages sent from the general purpose trigger 2.

Event name	Description
TRIGGER2	Trigger 2 was triggered

‘S’, ‘G’, T’, ‘3’ handles messages sent from the general purpose trigger 3.

Event name	Description
TRIGGER3	Trigger 3 was triggered

5.4.3.2. Typical SCET use case

A typical SCET use case scenario is to connect a GPS PPS pulse to the PPS input of the board. On every PPS_ARRIVED message the time difference is calculated and the internal SCET counter is adjusted.

5.4.3.3. RTEMS application example

In order to use the scet 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/scet_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_SCET_DRIVER
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES 20

#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/scet_rtems.h>` is required for accessing scet device name `RTEMS_SCET_DEVICE_NAME`.

`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]. As such, it is an 8 bit interface with a maximum FIFO level of 16 bytes and as such does not easily lend itself to high-speed communication exchanges for longer periods of time.

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 use. 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
Path	const char *	In	The absolute path to the file that is to be opened. See table below for uart naming.
Oflag	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
Fildes	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/psu_control	UART used for PSU communication, RS485 only
/dev/safe_bus	Safe bus UART, RS485 only

5.5.2.2. Function int close(...)

Closes access to the device and disables the line drivers.

Argument name	Type	Direction	Description
Fildes	int	In	File descriptor received at **open**

Return value	Description
0	Device closed successfully

5.5.2.3. Function int 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
Fildes	int	In	File descriptor received at **open**
Buf	void *	In	Pointer to character buffer to write data to
Nbytes	unsigned int	In	Number of bytes to read

Return value	Description
> 0	Number of bytes 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 bytes to be read

5.5.2.4. Function int write(…)

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

Argument name	Type	Direction	Description
Fildes	int	In	File descriptor received at **open**
Buf	const void *	In	Pointer to character buffer to read data from
Nbytes	unsigned int	In	Number of bytes to write

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

5.5.2.5. 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 UART.

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_SET_BITRATE_IOCTL	uint32_t	in	Sets the bitrate of the line interface: 10 = 375000 bps 9 = 153600 bps 8 = 115200 bps (default) 7 = 75600 bps 6 = 57600 bps 5 = 38400 bps 4 = 19200 bps 3 = 9600 bps 2 = 4800 bps 1 = 2400 bps 0 = 1200 bps
UART_MODE_SELECT_IOCTL	uint32_t	in	Sets the mode of the interface. 0 = RS422 (default) 1 = RS485 2 = Loopback mode (TX connected to RX internally)
UART_RX_FLUSH_IOCTL	uint32_t	in	Flushes the RX software FIFO
UART_SET_PARITY_IOCTL	uint32_t	in	Sets parity: 0 = No parity 1 = Odd parity 2 = Even parity

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

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 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/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.

5.5.3.3. Limitations

8 data bits only.
1 stop bit only.
No configuration of RX watermark level, fixed to 8.
No hardware flow control support.

5.6. UART32

5.6.1. Description

This driver software for the UART32 IP 104 513 [RD1], handles the setup and transfer of serial data to memory. This is a high-speed receive-only UART.

5.6.2. 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.

5.6.2.1. Enum rtems_uart32_ioctl_baudrate_e

Enumerator for the baudrate of the serial link.

Enumerator	Description
UART32_IOCTL_BAUDRATE_10M	10 MBaud
UART32_IOCTL_BAUDRATE_5M	5 MBaud
UART32_IOCTL_BAUDRATE_2M	2 MBaud
UART32_IOCTL_BAUDRATE_1M	1 MBaud
UART32_IOCTL_BAUDRATE_115200	115200 Baud

5.6.2.2. Enum rtem_uart32_ioctl_endian_e

Enumerator for the endianness of the DMA transfer.

Enumerator	Description
UART32_IOCTL_ENDIAN_BIG	Big endian
UART32_IOCTL_ENDIAN_LITTLE	Little endian

5.6.2.3. Function int open(...)

Opens access to the requested UART32. Upon each open call the device interface is reset to 10MBaud and big endian mode.

Argument name	Type	Direction	Description
pathname	const char *	in	The absolute path to the UART32 to be opened. UART32 device is defined as UART32_DEVICE_NAME.
flags	int	in	Access mode flag, only O_RDONLY is supported.

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

5.6.2.4. 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

5.6.2.5. Function ssize_t read(...)

Read data from the UART32. The call block until all data has been received from the UART32 or an error has occurred.

If any error condition occurs, the read will return zero bytes.

Note! Given buffer must be aligned to `CPU_STRUCTURE_ALIGNMENT` and the size must be a multiple of `CPU_STRUCTURE_ALIGNMENT`. It is recommended to assign the buffer in the following way:

```
uint8_t CPU_STRUCTURE_ALIGNMENT buffer[BUFFER_SIZE];
```

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. Maximum number of bytes is 16777216.

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

5.6.2.6. Function int ioctl(...)

Input/output control for UART32.

Argument name	Type	Direction	Description
fd	int	in	File descriptor received at open.
cmd	int	in	Command to send.
val	uint32_t / uint32t*	in/out	Value to write or a pointer to a buffer where data will be written.

Command table	Type	Direction	Description
UART32_SET_BAUDRATE_IOCTL	uint32_t	in	Sets the baudrate for the UART32, see [5.6.2.1].
UART32_SET_ENDIAN_IOCTL	uint32_t	in	Sets the endian for the transfer, see [5.6.2.2].
UART32_GET_BURST_SIZE_IOCTL	uint32_t	out	Get the number of bytes in the burst for the UART32.

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

5.6.3. Usage description

The following #define needs to be set by the user application to be able to use the UART32:

CONFIGURE_APPLICATION_NEEDS_UART32_DRIVER

5.6.3.1. RTEMS application example

In order to use the UART32 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/uart32_rtems.h>

#define CONFIGURE_APPLICATION_NEEDS_UART32_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) {
  int read_fd;
  uint32_t buffer[4];
  ssize_t size;

  read_fd = open(UART32_DEVICE_NAME, O_RDONLY);
  size = read(read_fd, &buffer, 4);
  status = close(read_fd);
}
```

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/uart32_rtems.h>` is required for accessing the UART32.

5.6.4. Limitations

The driver has limited UART functionality and can only receive data.

Data length is always 8 bits, no parity check and only 1 stop bit is used.

The receive buffer must be aligned to `CPU_STRUCTURE_ALIGNMENT` and the size must be a multiple of `CPU_STRUCTURE_ALIGNMENT`

5.7. Mass memory

5.7.1. Description

This section describes the mass memory 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.

5.7.2.1. 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 5-1.

Return value	Description
>0	A file descriptor for the device.
- 1	see *errno* values
errno values	
ENOENT	Invalid filename
EEXIST	Device already opened.

Table 5-1 - 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(…)

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

5.7.2.3. size_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
EINVAL	The whence argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory.
EOVERFLOW	The resulting file offset would be a value which cannot be represented correctly in an object of type **off_t**.

5.7.2.4. size_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.

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.

5.7.2.5. size_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	size_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.

5.7.2.6. 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	int	in	Command defined in subchapters 5.7.2.6.1 to 5.7.2.6.9.
value	void *	in	The value relating to command operation as defined in subchapters 5.7.2.6.1 to 5.7.2.6.9.

5.7.2.6.1. Bad block check

Checks if the given block is a bad block.

Command	Type	Direction	Description
MASSMEM_IO_BAD_BLOCK_CHECK	uint32_t	in	Block number.

Return value	Description
0	Block is OK.
-1	Bad block

5.7.2.6.2. Reset mass memory device

Command	Type	Direction	Description
MASSMEM_IO_RESET			

Return value	Description
0	Always

5.7.2.6.3. Read status data

Command	Type	Direction	Description
MASSMEM_IO_READ_STATUS_DATA	uint32_t*	out	

Return value	Description
≥0	Status register value

5.7.2.6.4. Read control status data

Command	Type	Direction	Description
MASSMEM_IO_READ_CTRL_STATUS	uint8_t*	out	

Return value	Description
0	Always

5.7.2.6.5. Read EDAC register data

Command	Type	Direction	Description
MASSMEM_IO_READ_EDAC_STATUS	uint8_t*	out	

Return value	Description
0	Always

5.7.2.6.6. Read ID

Command	Type	Direction	Description
MASSMEM_IO_READ_ID	uint8_t*	out	Of type `massmem_cid_t`.

Return value	Description
0	Always

5.7.2.6.7. Erase block

Command	Type	Direction	Description
MASSMEM_IO_ERASE_BLOCK	uint32_t	in	Block number

Return value	Description
0	Always

5.7.2.6.8. Read spare area

Reads the spare area with given data.

Command	Type	Direction	Description
MASSMEM_IO_READ_SPARE_AREA	uint8_t*	in/out	Of type `massmem_ioctl_spare_area_args_t`.

Return value	Description
0	Read operation was successful.
-1	Read operation failed.

5.7.2.6.9. Program spare area

Programs the spare area from the given data

Command	Type	Direction	Description
MASSMEM_IO_PROGRAM_SPARE_AREA	uint8_t*	in/out	Of type `massmem_ioctl_spare_area_args_t`

Return value	Description
0	Program operation was successful.
-1	Program operation failed.

5.7.3. Usage

5.7.3.1. RTEMS

5.7.3.1.1. 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.

5.7.3.1.2. 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 subchapter 5.7.2. And, if desired, the access can be closed when not needed.

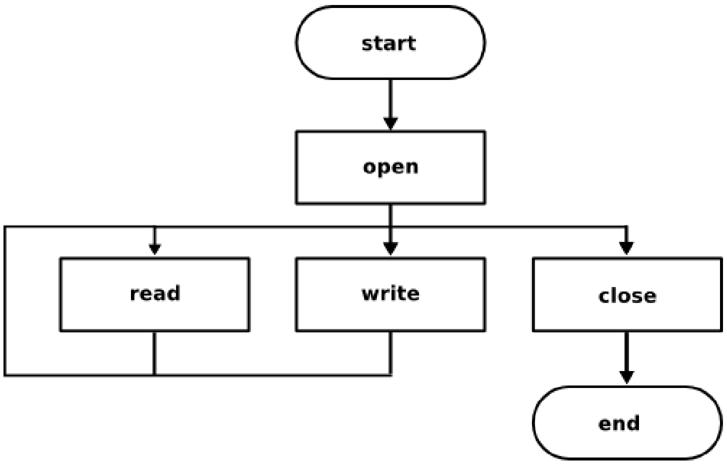


Figure 5-5 - RTEMS driver usage description

Note! All calls to RTEMS driver are blocking calls.

5.7.3.2. 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>

rtems_task Init (rtems_task_argument argument)
{
    .
    fd = open(MASSMEM_DEVICE_NAME, O_RDWR);
    .
}
```

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.

5.7.4. Limitations

The TCM mass memory interface can currently only handle multiple consecutive RMAP write commands of size 1200 bytes or below.

5.8. Spacewire

5.8.1. Description

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

5.8.2. 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 5.8.2.5.

5.8.2.1. 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 5.8.3.1.

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 5-2.

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 5-2 - Open flag symbols

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

5.8.2.2. 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

5.8.2.3. size_t read(…)

Receives a packet.

Note! Given buffer must be aligned to `CPU_STRUCTURE_ALIGNMENT`. It is recommended to assign the buffer in the following way:

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

Note! This call is blocking until a package for the logic address is received

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*.
- 1	see *errno* values
errno values	
EBADF	The file descriptor *fd* is not an open file descriptor
EINVAL	*buf* size is 0.

5.8.2.4. size_t write(…)

Transmits a packet.

Note! This call is blocking until 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.
- 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`.

5.8.2.5. 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 5.8.2.5.1
value	void *	in	The value relating to command operation as defined in subchapter 5.8.2.5.1.

5.8.2.5.1. 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	SPWN_IOCTL_MODE_NORMAL for normal mode or SPWN_IOCTL_MODE_LOOPBACK for loopback mode

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

5.8.3. Usage

5.8.3.1. RTEMS

5.8.3.1.1. 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 5.8.3.2. 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 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];
```

5.8.3.1.2. 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 5.8.2. Additionally, if desired, the access can be closed when not needed.

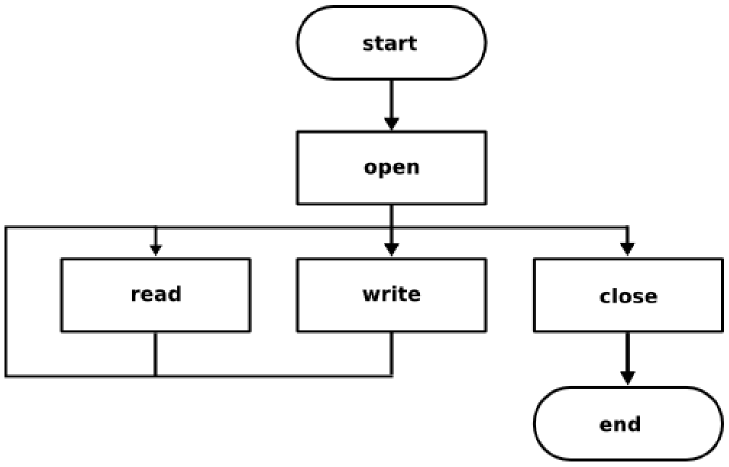


Figure 5-6 - RTEMS driver usage description

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.

5.8.3.2. 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 CPU_STRUCTURE_ALIGNMENT buf_rx[PACKET_SIZE];
uint8_t CPU_STRUCTURE_ALIGNMENT buf_tx[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.

5.8.4. Limitations

Currently, default transmission/reception bit rate is set to 50 MBAUD and cannot be altered during operation. This functionality is planned to be added in a future release.

A packet must be of a size of at **least** 4 bytes.

5.9. GPIO

5.9.1. 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.

5.9.1.1. 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.

5.9.1.2. 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.

5.9.1.3. 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).

5.9.1.4. 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.

5.9.2. 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.

5.9.2.1. 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

5.9.2.2. 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

5.9.2.3. 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 5.9.2.5.

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

5.9.2.4. 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

5.9.2.5. 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

5.9.3. Usage description

5.9.3.1. 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`, `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.

5.9.4. Limitations

Differential mode works in output only.

5.10. CCSDS

5.10.1. 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.

5.10.2. 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

5.10.2.1. 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

5.10.2.2. 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

5.10.2.3. 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

5.10.2.4. 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

5.10.2.5. Data type tc_config_t

This datatype is a struct for configuration of the TM 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

5.10.2.6. 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.

5.10.2.7. 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

5.10.2.8. 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

5.10.2.9. 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

5.10.2.10. 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

5.10.2.11. 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 5.10.2.3
CCSDS_GET_TM_CONFIG	/dev/ccsds-tm	tm_config_t *	Returns the configuration of the TM path. See 5.10.2.3
CCSDS_SET_TC_CONFIG	/dev/ccsds-tc	tc_config_t	Sets a configuration of the TC path. See 5.10.2.5
CCSDS_GET_TC_CONFIG	/dev/ccsds-tc	tc_config_t *	Returns the configuration of the TC path. See 5.10.2.5
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_IE_CONFIG	/dev/ccsds	uint32_t	Enables/Disables interrupts in the CCSDS IP.
CCSDS_GET_IE_CONFIG	/dev/ccsds	uint32_t*	Gets the configuration of the enabled/disabled interrupts.
CCSDS_SET_DMA_DESC	/dev/ccsds-tm	dma_descriptor_t	Configures a DMA-descriptor in the range (0-31). See 5.10.2.3
CCSDS_GET_DMA_DESC	/dev/ccsds-tm	dma_descriptor_t*	Returns the configuration of a DMA-descriptor in the range (0-31). See 5.10.2.3
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 the CCSDS IP.
CCSDS_SET_CLCW	/dev/ccsds-tm	uint32_t	Sets the CLCW of TM frames
CCSDS_GET_CLCW	/dev/ccsds-tm	uint32_t*	Gets the CLCW of the TM Frames

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

5.10.3. Usage description

5.10.3.1. 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. Enable the different interrupts to be generated by ioctl CCSDS_SET_IE_CONFIG on device “/dev/ccsds”.
3. Prepare DMA-descriptors by ioctl CCSDS_SET_DMA_DESC on device “/dev/ccsds-tm”.
4. Enable DMA by ioctl CCSDS_ENABLE_DMA
5. Enable TM by ioctl CCSDS_ENABLE_TM on device “/dev/ccsds-tm”.
6. Prepare the content in SDRAM that will be fetched by DMA-transfer by writing to “/dev/ccsds-tm”

5.10.3.2. 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. Enable the different interrupts to be generated by ioctl CCSDS_SET_IE_CONFIG
3. Do a read from “/dev/ccsds-tc”. This call will block until a new TC has been received.

5.10.3.3. 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.

5.11. ADC

5.11.1. Description

This section describes the driver for accessing the ADC device when reading the house-keeping (HK) data. The following ADC channels contain housekeeping information:

Parameter	Abbreviation	ADC channel
Temperature	Temp	9
Input current	Iin	8
Input voltage	Vin	7
Regulated 3.3V	3V3	6
Regulated 2.5V	2V5	5
Regulated 1.2V	1V2	4

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)

5.11.2. 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.

5.11.2.1. 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 200.8
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

5.11.2.2. 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 already exists
EALREADY	Device is already open
EINVAL	Invalid options

5.11.2.3. 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	
EEXISTS	Device already exists
EALREADY	Device is already open

EINVAL	Invalid options

5.11.2.4. 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 as this is the minimum return element from a read, see bit definition table below.

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

5.11.2.5. 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	int / int*	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	Set 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

5.11.3. Usage

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

CONFIGURE_APPLICATION_NEEDS_ADC_DRIVER

5.11.3.1. 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;

  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);
  status = close(read_fd);
}
```

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.

5.11.4. Limitations

Only one enabled channel at a time is supported in current implementation.

Only the default divisor value is supported in the current implementation which set the SPI frequency for read out of the buffered ADC data. Default val is ~200kHz.

5.12. 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.

5.12.1. 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. The system memory start at SPI RAM address 0x100 and the user memory start at SPI RAM address 0x200.

5.12.2. 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.

5.12.2.1. 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.

5.12.2.2. 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

5.12.2.3. 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	
EINVAL	Invalid options

5.12.2.4. 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

5.12.2.5. 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

5.12.2.6. 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

5.12.2.7. 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 [5.12.2.1].
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

5.12.3. 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

5.12.3.1. 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 OBC-S^{TM} and TCM-S^{TM} 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.

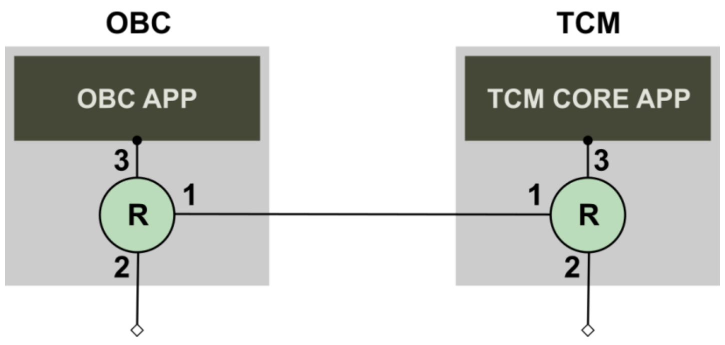


Figure 6-1 Integrated router location

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

In addition to this, each end node, OBC-S^{TM} or TCM-S^{TM}, 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 OBC-S^{TM} to the TCM-S^{TM} it needs to be prepended with 0x01 0x03 XX.
0x01 routes the packet to port 1 of the OBC-S^{TM} router.
0x03 routes the packet to port 3 of the TCM-S^{TM} router.
XX is the logical address of the recipient application/service on the TCM-S.

7. TCM-S^{TM}

7.1. Description

The TCM-S^{TM} 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 TCM-S^{TM} software. Any command not addressing the TCM-S^{TM} will be routed to other nodes on the SpaceWire network according to the current TCM-S^{TM} configuration.

TM is received from other nodes on the SpaceWire network. The TCM-S^{TM} 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 TCM-S^{TM} 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.

7.2. Block diagram

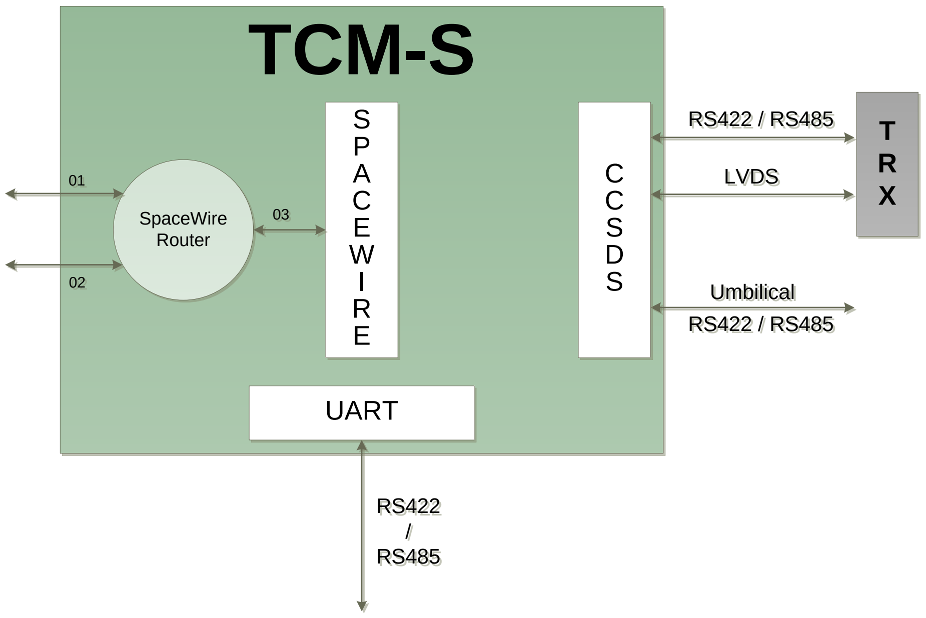


Figure 7.1 – TCM-S^{TM} functionality layout

7.3. 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 TCM-S^{TM} according to Table 7-1 to separate between configuration commands and mass memory storage of data (partition handling).

Table 7-1: 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 TCM-S^{TM} to compensate for topology external to the TCM-S^{TM} and the embedded SpaceWire router. As can be seen Figure 7.1, if the TCM-S^{TM} were to be addressed from SpaceWire port 1, the example addresses below must be added to the routing addresses in the RMAP header.

Table 7-2: RMAP predefined fields for routing

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

7.3.1. Input

The RMAP commands supported by the TCM-S^{TM} are specified in table below. See chapter 7.4 for details on each specific command.

Note! The TCM-S^{TM} uses the RMAP Transaction ID to separate between outstanding replies to different units. When several nodes are addressing the TCM-S^{TM}, they need to be assigned a unique transaction id range to ensure a correct system behaviour. To allow for a similar transaction identification throughout the system, the TCM-S^{TM} uses the Transaction ID range `0x0000-0x0FFF` in all outgoing communication.

Table 7-3: RMAP commands to TCM

Name	Ext. Addr	Address	Cmd	Description
MMData	0x00-0x0F	0x00000000	R/W	Reads and writes data of a partition.
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	Configures telemetry clock frequency.
TMOCFControl	0xFF	0x00000A00	W	Enable/Disable inclusion of Operational Control field in TM Frames.

TMTSControl	0xFF	0x00000B00	W	Configures Timestamp of telemetry.
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.
ErrorStatus	0xFF	0x02000300	W	Reads error status.
UARTCommand	0xFF	0x0400020x	W	Sends a command to a specific UART device. 0 - UART0 1 - UART1 2 - UART2 3 - S-Band / RS422 interface 4 - X-Band / LVDS interface 5 - PSU Ctrl 6 - Safe Bus.
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.
MMVolReadPointerReset	0xFF	0x0500060x	W	Sets the partition x volatile readpointer to the same position as the readpointer stored in NVRAM. Only valid for partition type static circular.

7.3.2. Output

The TCM-S^{TM} 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 7-4: Published data from TCM

Name	Ext. Addr.	Address	Cmd	Description
TCCommand	0x00[1]	0x00000000	W	Routed Telecommands
UARTData	0x00[1]	0x0400000x	W	Data received on specified UART 0 - UART0 1 - UART1 2 - UART2 3 - S-Band / RS422 interface 4 - X-Band / LVDS interface 5 - PSU Ctrl 6 - Safe Bus

7.4. RMAP address details

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

[1] This value is scheduled to be changed to 0xFF in the next release.

7.4.1. MMData

Read or write data from/to a partition.

Table 7-5: MMData data

Byte	Type	Description
0 - nn	Array of UINT8	Data

7.4.2. TMStatus

Reads the latest telemetry status.

Table 7-6: TMStatus data

Byte	Type	Description
0	UINT8	0x00 – No Error 0x01 – FIFO error.
1	UINT8	0x00 – No transfer in progress. 0x00 – Transfer in progress.

7.4.3. TMConfig

Reads the telemetry configuration.

Table 7-7: 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 Frame Control 0x00 – Disabled 0x01 – Enabled
4	UINT8	Idle Frame Control 0x00 – Disabled 0x01 – Enabled
5	UINT8	Convolutional Encoding Control 0x00 – Disabled 0x01 – Enabled
6	UINT8	Pseudo Randomization Control 0x00 – Disabled 0x01 – Enabled
7	UINT8	CLCW Control 0x00 – Disabled 0x01 - Enabled

7.4.4. TMControl

Enables/disables generation of telemetry.

Table 7-8: TMControl data

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

7.4.5. TMFEControl

Controls Frame Error Control Field inclusion for transfer frames.

Table 7-9: TMFEControl data

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

7.4.6. TMMCFCControl

Controls the Master Channel Frame Counter generation for transfer frames.

Table 7-10: TMMFControl data

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

7.4.7. TMIFControl

Controls the Idle Frame generation for transfer frames.

Table 7-11: TMIFControl data

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

7.4.8. TMPRControl

Controls the Pseudo Randomization for transfer frames.

Table 7-12: TMPRControl data

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

7.4.9. TMOCFControl

Controls Operational Control Field inclusion in TM Transfer frames.

Table 7-13: CLCW data

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

7.4.10. 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 7-14: TMCEControl data

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

7.4.11. TMBRControl

Configures the telemetry clock frequency.

The telemetry clock is fed to the radio. The frequency of the telemetry clock is the system clock (50 MHz) divided by the divisor. I.e. if the divisor value is set to 16, the telemetry clock frequency is *3.125 MHz.*
Note! If the convolutional encoding is **enabled**, as defined in subchapter 7.4.10, the telemetry clock is multiplied by two, i.e. 6.25 MHz from example above, to keep the net datarate the same.

Table 7-15: TMBRControl data

Byte	Type	Description
0	UINT8	Bitrate divisor value (default 0x16). Minimum divisor is 4.

7.4.12. TMTSControl

Configures the timestamping for transfer frames.

Table 7-16: 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

7.4.13. TMSend

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

Table 7-17: TMSend data

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

7.4.14. TCStatus

Reads current telecommand status.

Table 7-18: TCStatus data

Byte	Type	Description
0	UINT32	CLCW word of the last received telecommand.
4	UINT8	Missed frames counter. Wrapped at 0xFF.
5	UINT8	CPDU command rejections counter. Wrapped at 0xFF.
6	UINT8	Telecommand rejections counter. Wrapped at 0xFF.
7	UINT8	Parity errors counter. Wrapped at 0xFF.
8	UINT8	Telecommand receptions counter. Wrapped at 0xFF.
9	UINT16	Last CPDU pulse command output. Bit 15:4 – Line 11:0 Bit 3:0 – Unused
11	UINT8	Accepted CPDUs counter. Wrapped at 0xFF.
12	UINT8	Derandomizer setting 0x00 – Disabled. 0x01 – Enabled.
13	UINT16	Length of the last received TC frame

7.4.15. TCDRControl

Configures derandomization for telecommand frames.

Table 7-19: TCDRControl data

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

7.4.16. HKData

Reads the housekeeping data.

Table 7-20: HKData data

Byte	Type	Description
0	UINT16	Input voltage [mV]
2	UINT16	Regulated 3V3 voltage [mV]
4	UINT16	Regulated 2V5 voltage [mV]
6	UINT16	Regulated 1V2 voltage [mV]
8	UINT16	Input current [mA]
10	UINT16	Temperature [mºC]
12	UINT32	SCET seconds
16	UINT8	S/W version.
17	UINT8	CPU Parity Errors
18	UINT8	Watchdog trips
19	UINT8	Critical SDRAM EDAC Single Errors
20	UINT8	Other SDRAM EDAC Single Errors
21	UINT8	Critical SDRAM EDAC Multiple Errors
22	UINT8	Other SDRAM EDAC Multiple Errors

7.4.17. SCETTime

Reads/sets the SCET time. Any adjustment to SCET time will have different effects depending on the SCET mode (see subchapter 7.4.18).

Free-running and Master 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 7.21: 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, this must be a 2's complementary value that shall be added to the subseconds counter.

7.4.18. 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 7.22: SCETConfig data

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

7.4.19. ErrorStatus

Reads the error status data.

Table 7-23: ErrorStatus data

Byte	Type	Description
0	UINT8	CPU Parity Errors
1	UINT8	Watchdog trips
2	UINT8	Critical SDRAM EDAC Single Errors
3	UINT8	Other SDRAM EDAC Single Errors
4	UINT8	Critical SDRAM EDAC Multiple Errors
5	UINT8	Other SDRAM EDAC Multiple Errors

7.4.20. UARTCommand

Send a command on the specified UART interface.

Table 7-24: UARTCommand data

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

7.4.21. MMStatus

Reads mass memory status.

Table 7-25: MMStatus data

Byte	Type	Description
0	UINT8	Chip 3 status Bit 7 - WP# (write protect) Bit 6 - RDY (Ready/Busy) Bit 5 - ARDY (Ready/Busy Array) Bit 1 - FAILC (Pass/Fail – set if the previous operation (program) failed) Bit 0 - FAIL (Pass/Fail – 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	UINT8	Controller status 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
6	UINT8	Chip ID: Chip 3
7	UINT8	Chip ID: Chip 2
8	UINT8	Chip ID: Chip 1
9	UINT8	Chip ID: Chip 0
10	UINT8	Chip ID: EDAC

7.4.22. MMWritePointer

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

Table 7-26: MMWritePointer data

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

7.4.23. MMReadPointer

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

Table 7-27: MMReadPointer data

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

7.4.24. MMPartitionConfig

The partition configuration of the specified partition is read by this command.

Table 7-28: 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. 1 – Circular. Oldest data is overwritten when full. 2 – Static Circular. The current readpointer of the partition is not stored non-volatile during read accesses of the partition. On a power-cycle of the TCM-S, the readpointer will be reset to an initial value stored in NVRAM-
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.

7.4.25. 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 Mbyte).

Table 7-29 MMPartitionSpace data

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

7.4.26. MMVolReadPointerReset

For static circular partitions a volatile readpointer is used when reading from the partition. The volatile readpointer can be reset to an initial value stored on NVRAM. This command resets the volatile readpointer to the initial value

7.4.27. TCCommand

A fully formed PUS packet according to [RD4].

7.4.28. UARTData

Routed data from UART.

Table 7-30: UARTData data

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

7.5. Telemetry

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

7.6. Telecommands

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

7.7. ECSS standard service

The TCM-S^{TM} supports a subset of the services described in [RD4]

7.7.1. Telecommand verification service

The TCM-S^{TM} 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 Table 7-31. 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 7-31: Telecommand Acceptance Report – Failure (1,2)

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

7.7.2. PUS-2 Device Command Distribution Service

The TCM-S^{TM} supports the Command Pulse Distribution Unit (CPDU) pulse commands in hardware as defined in 7.2.2 in [RD3]. 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 7-32 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 7-33 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.

7.8. Limitations

For the current release, the TCM-S only support PUS packets for download with a 32-bit aligned size.

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

When entering AD mode, the TCM-S will issue a false error notification. However, AD mode is operational and this message can be disregarded.

8. System-on-Chip definitions

The ÅAC Sirius products include two boards built around the OR1200 fault tolerant processor, the OBC-S^{TM} and the TCM-S^{TM}. Below are the peripherals, memory sections and interrupts defined for the SoC for these two boards. Some of these might not be equipped in this development release.

8.1. Memory mapping

Table 8-1 - Sirius memory structure definition

Memory Base Address	Function
0xF0000000	Boot ROM
0xE0000000	CCSDS (TCM-S^{TM} only)
0xCB000000	Watchdog
0xCA000000	SpaceCraft Elapsed Time
0xC1000000	SoC info
0xC0000000	Error Manager
0xBD000000 - 0xBF000000	Reserved
0xBC000000	Reserved for SPI interface 1
0xBB000000	SPI interface 0
0xBA000000	GPIO
0xB6000000	Reserved for ADC controller 1
0xB5000000	ADC controller 0
0xB4000000	Reserved
0xB3000000	Mass memory flash controller (TCM-S^{TM} 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	Ethernet MAC
0x9C000000 -0x9F000000	Reserved
0x9B000000	I2C interface 1
0x9A000000	I2C interface 0
0x99000000	Reserved
0x98000000	UART 7 (Safe bus functionality, RS485)
0x97000000	UART 6 (PSU control functionality, RS485)
0x96000000	UART 5 (OBC-S^{TM} only, High speed UART w. DMA)
0x95000000	UART 4 (Routed to LVDS HK on TCM-S^{TM})
0x94000000	UART 3 (Routed to RS422 HK on TCM-S^{TM})
0x93000000	UART 2
0x92000000	UART 1
0x91000000	UART 0
0x90000000	UART Debug (LVTTL)
0x80000000 - 0x8F000000	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	ADC measurement completed
12	-	Ready to use (reserved for ADC)
13	i2c 0	Master/slave transaction complete/req
14	-	Ready to use (reserved for i2c)
15	-	Ready to use (reserved for i2c)
16	-	Ready to use (reserved for i2c)
17	SCET	SCET interrupt signal
18	Error manager	Error manager interrupt
19	-	Reserved for redundant spacewire
20	System flash	System flash controller interrupt
21	Mass memory	Mass memory flash controller interrupt
22	Spacewire	Spacewire interrupt
23	CCSDS	CCSDS interrupt
24	Ethernet	Ethernet MAC interrupt signal
25	GPIO	GPIO interrupt
26	SPI 0	Serial Peripheral interface
27	-	Ready to use (reserved for SPI 1)
28	-	Ready to use (reserved for custom adaptation)
29	-	Ready to use (reserved for custom adaptation)
30	-	Ready to use (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 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 (TCM-S^{TM} 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. Boot order of the logical images and their physical copies is shown in Figure 8-1.

8.4.2. Block diagram

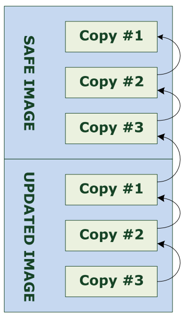


Figure 8-1 Software images in flash

8.4.3. Usage description

The locations in the system flash where the bootrom looks for software images is 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 the computed checksum evaluated to zero, the bootrom jumps to the reset vector in RAM. If there is a flash error when loading, the checksum is incorrect or 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 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.

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 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 an asynchronous reset with an internal synchronous release, i.e. synchronizing each reset release to its clock domain.

All clock domain crossings are either handled via FIFOs or synchronized into the other clock domain. Two serial flip-flops are used to reduce possible metastability effects.

8.6. Pulse command inputs

The pulse command inputs on the breadboard can be used to force the board to reboot from a specific image. Paired with the ability of the TCM-S 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 safe image while input 1 resets the board and loads the updated image. Both require a 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.

8.7. SoC information map

The information included in the SoC info block for the Sirius products have the following contents in Table 8-6. This information must be used from gdb prompt and can be used as a control of which SoC version that is flashed on the board. In the terminal window that you have opened `or1k-aac-elf-gdb` type:

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

and you will get the below information presented.

Table 8-6 Sirius SoC info

Base address number	Function	Description	
0x0	TIME_STAMP	When building the SoC, a Unix timestamp is taken and put into the system. A 32 bit vector indicating seconds since 1970-01-01 (UTC).	
0x4	PRODUCT_ID	• 0x00 • 0x01 • 0x02 • … • 0x0F	OBC S BB OBC S OBC SR – With SPW router 3 ports
		• 0x10 • 0x11 • 0x12 • … • 0x1F	TCM S BB TCM S TCM S R – With SPW router 3 ports
		• 0x20-0xFF	Reserved
0x8	SOC_VERSION	Follows the methodology of release 0.1.0 = Release-X.Y.Z, , where X represent a major number. 8bits , where Y represent a minor number. 8bits , where Z is patch number. 8bits Representated in 32 bits. Example: 0x000101FF = 1.1.255 First eight bits are reserved	

9. Connector interfaces

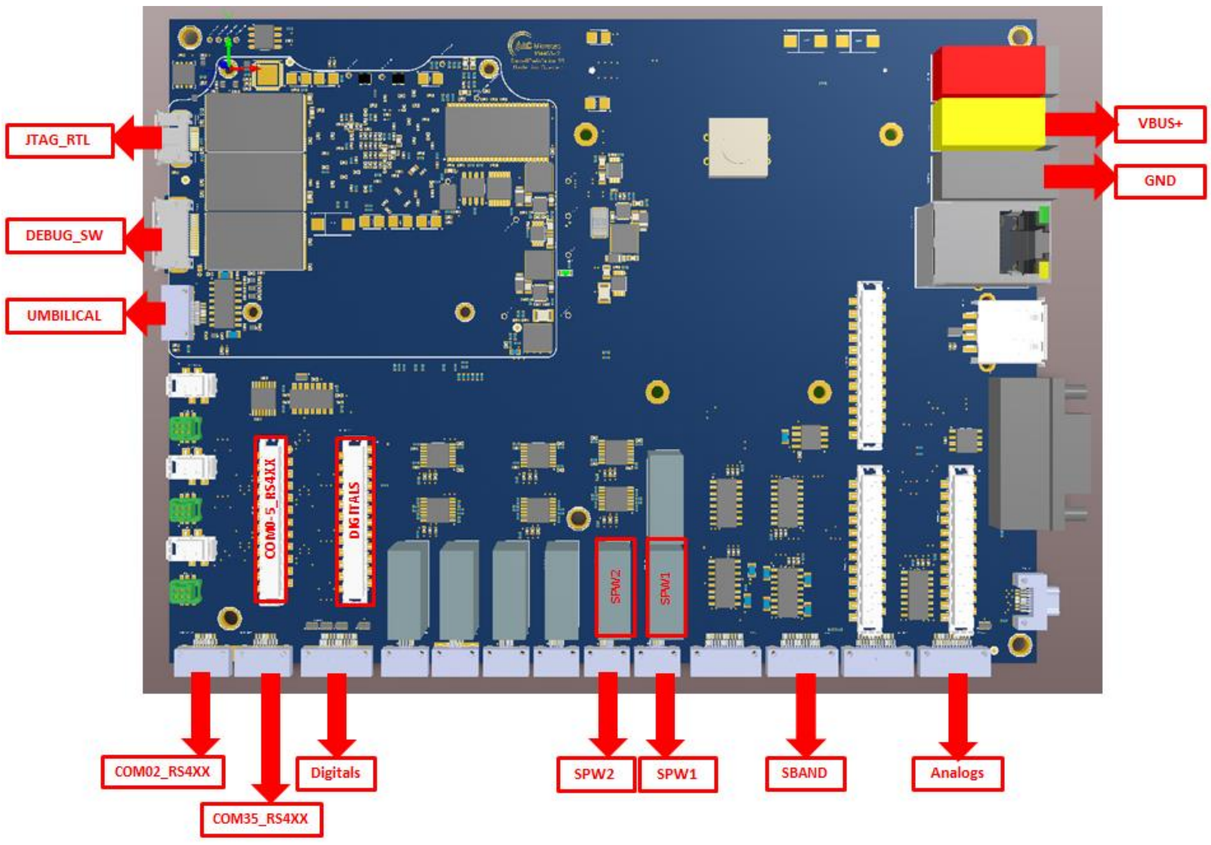


Figure 9-1 - Sirius ports

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	GND	Ground
Pin 18	DEBUG-JTAG-TCK	Debug Test clock

9.3. SPW1 - Spacewire

The following pins are available on the nano-D9 socket connector, see Table 9-3

Table 9-3 - SPW1 pin-outs

Pin #	Signal name	Description
Pin 1	SPW1_DIN_LVDS_P	SpaceWire data in positive, pair with p6
Pin 2	SPW1_SIN_LVDS_P	SpaceWire strobe in positive, pair with p7
Pin 3	Shield	Cable shielded, connected to chassis
Pin 4	SPW1_SOUT_LVDS_N	SpaceWire strobe out negative, pair with p8
Pin 5	SPW1_DOUT_LVDS_N	SpaceWire data out negative, pair with p9
Pin 6	SPW1_DIN_LVDS_N	SpaceWire data in negative, pair with p1
Pin 7	SPW1_SIN_LVDS_N	SpaceWire strobe in negative, pair with p2
Pin 8	SPW1_SOUT_LVDS_P	SpaceWire strobe out positive, pair with p4
Pin 9	SPW1_DOUT_LVDS_P	SpaceWire data out positive, pair with p5

9.4. SPW2 - Spacewire

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

Table 9-4 – SPW2 pin-outs

Pin #	Signal name	Description
Pin 1	SPW2_DIN_LVDS_P	SpaceWire data in positive, pair with p6
Pin 2	SPW2_SIN_LVDS_P	SpaceWire strobe in positive, pair with p7
Pin 3	Shield	Cable shielded, connected to chassis
Pin 4	SPW2_SOUT_LVDS_N	SpaceWire strobe out negative, pair with p8
Pin 5	SPW2_DOUT_LVDS_N	SpaceWire data out negative, pair with p9
Pin 6	SPW2_DIN_LVDS_N	SpaceWire data in negative, pair with p1
Pin 7	SPW2_SIN_LVDS_N	SpaceWire strobe in negative, pair with p2
Pin 8	SPW2_SOUT_LVDS_P	SpaceWire strobe out positive, pair with p4
Pin 9	SPW2_DOUT_LVDS_P	SpaceWire data out positive, pair with p5

9.5. ANALOGS, Analog input and 4xGPIO (OBC-S)

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

Table 9-5 – ANALOGS, 4xGPIO pin-outs

Pin #	Signal name	Description
Pin 1	ADC_IN_0	Analog input to ADC with buffer
Pin 2	ADC_IN_1	Analog input to ADC with buffer
Pin 3	ADC_IN_2	Analog input to ADC with buffer
Pin 4	ADC_IN_3	Analog input to ADC with buffer
Pin 5	ADC_IN_4	Analog input to ADC with buffer
Pin 6	ADC_IN_5	Analog input to ADC with buffer
Pin 7	ADC_IN_6	Analog input to ADC with buffer
Pin 8	ADC_IN_7	Analog input to ADC with buffer
Pin 9	ADC_IN_8	Analog input to ADC with buffer
Pin 10	ADC_IN_9	Analog input to ADC with buffer
Pin 11	GPIO12	Digital input/output
Pin 12	GPIO13	Digital input/output
Pin 13	GPIO14	Digital input/output
Pin 14	GND	Board ground
Pin 15	GND	Board ground
Pin 16	GND	Board ground
Pin 17	GND	Board ground
Pin 18	GND	Board ground
Pin 19	GND	Board ground

Pin 20	GND	Board ground
Pin 21	GND	Board ground
Pin 22	GND	Board ground
Pin 23	GND	Board ground
Pin 24	GPIO15	Digital input/output
Pin 25	GND	Board ground

9.6. DIGITALS, 3x I2C, PPS and 12xGPIO

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

Table 9-6 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	SPI_MISO	SPI Master-In-Slave-Out
Pin 15	SPI_MOSI	SPI Master-out-Slave-In
Pin 16	SPI_CLK	SPI clock
Pin 17	I2C_SCL0	I2C bus 0, clock
Pin 18	I2C_SDA0	I2C bus 0, data
Pin 19	I2C_SCL1	I2C bus 1, clock
Pin 20	I2C_SDA1	I2C bus 1, data
Pin 21	I2C_SCL2	I2C bus 2, clock
Pin 22	I2C_SDA2	I2C bus 2, data
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

9.7. COM02_RS4XX, 3xRS422/485

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

Table 9-7 COM02_RS4XX pinouts

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

9.8. COM35_RS4XX, RS422/485

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

Table 9-8 COM35_RS4XX pin-outs

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

Pin 13	UART5_TX_RS4XX_P	UART Port 5 TX
Pin 14	UART5_TX_RS4XX_N	
Pin 15	GND	Ground

9.9. CCSDS RS422, S-BAND TRX (TCM-S)

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

Table 9-9 S-BAND TRX pin-outs

Pin #	Signal name	Description
Pin 1	SBAND_DOUT_RS422_P	Baseband data out, RS422
Pin 2	SBAND_DOUT_RS422_N	
Pin 3	SBAND_COUT_RS422_P	Baseband clock out, RS422
Pin 4	SBAND_COUT_RS422_N	
Pin 5	SBAND_DIN_RS422_P	Baseband data in, RS422
Pin 6	SBAND_DIN_RS422_N	
Pin 7	SBAND_CIN_RS422_P	Baseband clock in, RS422
Pin 8	SBAND_CIN_RS422_N	
Pin 9	SBAND_SC_LOCK_IN_RS422_P	Sub-carrier lock in
Pin 10	SBAND_SC_LOCK_IN_RS422_N	
Pin 11	SBAND_C_LOCK_IN_RS422_P	Carrier lock in
Pin 12	SBAND_C_LOCK_IN_RS422_N	
Pin 13	GND	
Pin 14	SBAND_HKCTRL1_TX_RS422_P	TRX control & housekeeping signaling
Pin 15	SBAND_HKCTRL1_TX_RS422_N	
Pin 16	SBAND_HKCTRL2_TX_RS422_P	TRX control & housekeeping signaling
Pin 17	SBAND_HKCTRL2_TX_RS422_N	
Pin 18	SBAND_HKCTRL3_TX_RS422_P	TRX control & housekeeping signaling
Pin 19	SBAND_HKCTRL3_TX_RS422_N	
Pin 20	SBAND_HKCTRL4_TX_RS422_P	TRX control & housekeeping signaling
Pin 21	SBAND_HKCTRL4_TX_RS422_N	
Pin 22	SBAND_HKCTRL1_RX_RS422_P	TRX control & housekeeping signaling
Pin 23	SBAND_HKCTRL1_RX_RS422_N	
Pin 24	EXTRA TX_RS422_P (reserved)	
Pin 25	EXTRA TX_RS422_N (reserved)	

9.10. UMBI – Baseband Umbilical (TCM-S^{TM})

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

Table 9-10 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	GND	Ground (reference)
Pin 14	GND	
Pin 15	GND	

9.11. Pulse Command Outputs

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

Table 9-10 Pulse command pin-outs

Pin #	Signal name	Description
Pin 1	PULSE0_O_RS422_P	
Pin 2	PULSE0_O_RS422_N	
Pin 3	PULSE1_O_RS422_P	
Pin 4	PULSE1_O_RS422_N	
Pin 5	PULSE2_O_RS422_P	
Pin 6	PULSE2_O_RS422_N	
Pin 7	PULSE3_O_RS422_P	
Pin 8	PULSE3_O_RS422_N	
Pin 9	PULSE4_O_RS422_P	
Pin 10	PULSE4_O_RS422_N	
Pin 11	PULSE5_O_RS422_P	
Pin 12	PULSE5_O_RS422_N	
Pin 13	GND	
Pin 14	PULSE6_O_RS422_P	
Pin 15	PULSE6_O_RS422_N	
Pin 16	PULSE7_O_RS422_P	
Pin 17	PULSE7_O_RS422_N	
Pin 18	PULSE8_O_RS422_P	
Pin 19	PULSE8_O_RS422_N	
Pin 20	PULSE9_O_RS422_P	
Pin 21	PULSE9_O_RS422_N	
Pin 22	PULSE10_O_RS422_P	
Pin 23	PULSE10_O_RS422_N	
Pin 24	PULSE11_O_RS422_P	
Pin 25	PULSE11_O_RS422_N	

10. Updating the Sirius FPGA

To be able to update the SoC on the OBC-S^{TM} and TCM-S^{TM} 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 three files:
   a. The actual FPGA file with an .stp file ending
   b. The programmer file with a .pro file ending
   c. The programmer script file with a .tcl file ending
4. Execute the following command:
   FPExpress script:fileWithTclEnding.tcl

   Please note that you either need to launch FPExpress with super user rights or change the user rights to the usb node.
5. If the programming was successful one of the last commands should be:
   programmer: Chain programming PASSED.
6. 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.

The outline in the left upper corner of the drawing below corresponds to the FM version of the TCM-S^{TM} and OBC-S^{TM} boards.

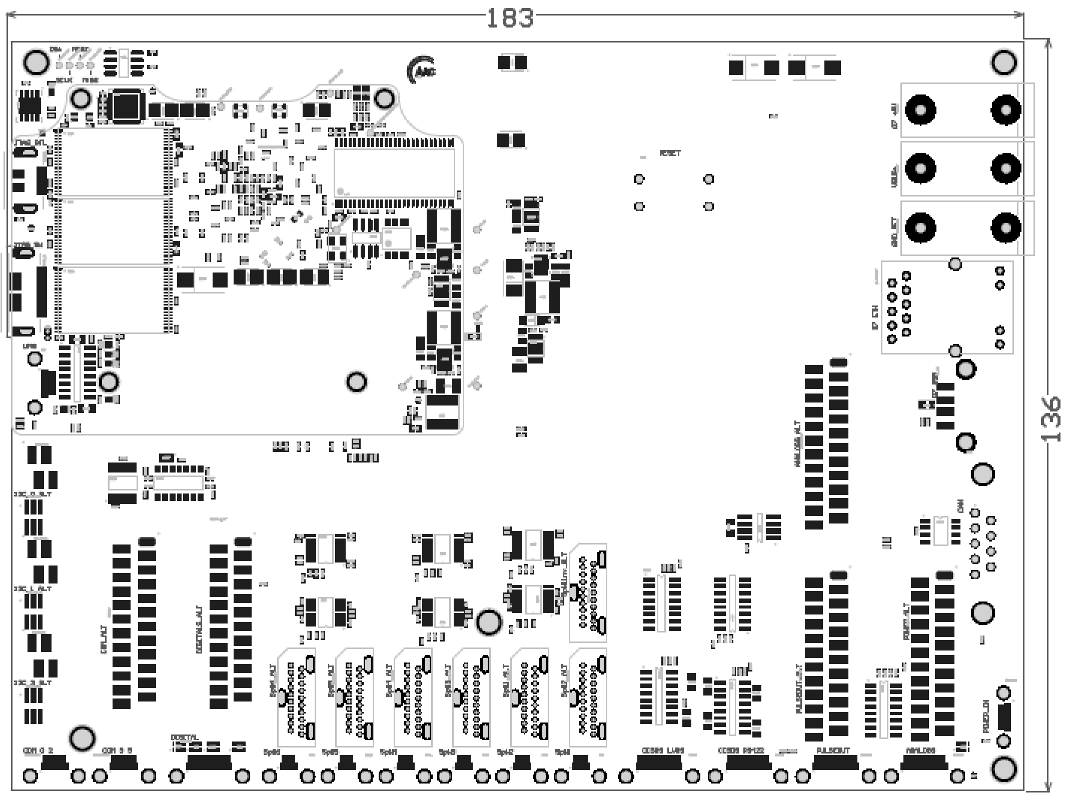


Figure 11-1 - The Sirius board mechanical dimensions

12. Environmental information

The Sirius Breadboard is an engineering model and as such it is only intended for office usage.

Table 12-1 - Environmental temperature ranges

Environment	Range
Operating temperature EM	0-40 ºC
Storage temperature EM	0-40 ºC

13. 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