Sirius Breadboard User Manual

B

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.

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. Intended users

This manual is written for software engineers who want to work with the ÅAC Sirius product suite.

1.2. 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

1.3. Reference documents

RD#	Document ref	Document name
RD1	http://opencores.org/openrisc,architecture	OpenRISC 1000 Architecture Manual
RD2	ECSS-E-ST-50-52C	SpaceWire – Remote memory access protocol

2. Equipment information

The Sirius Breadboard is a robust prototyping platform designed to support the TCM-S™, and the OBC-S™ products. The Breadboard layout is depicted in Figure 2.

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 [**Error! Reference source not found.**] running at a system frequency of 50 MHz and with he 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, 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™ 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 1 depicts a system overview with peripherals of the OBC-S™ and TCM-S™. The figure shows what parts are general for OBC-S™ and TCM-S™ (green), TCM-S™-specific (blue) and what parts are not yet implemented (white) since the products are still under development.

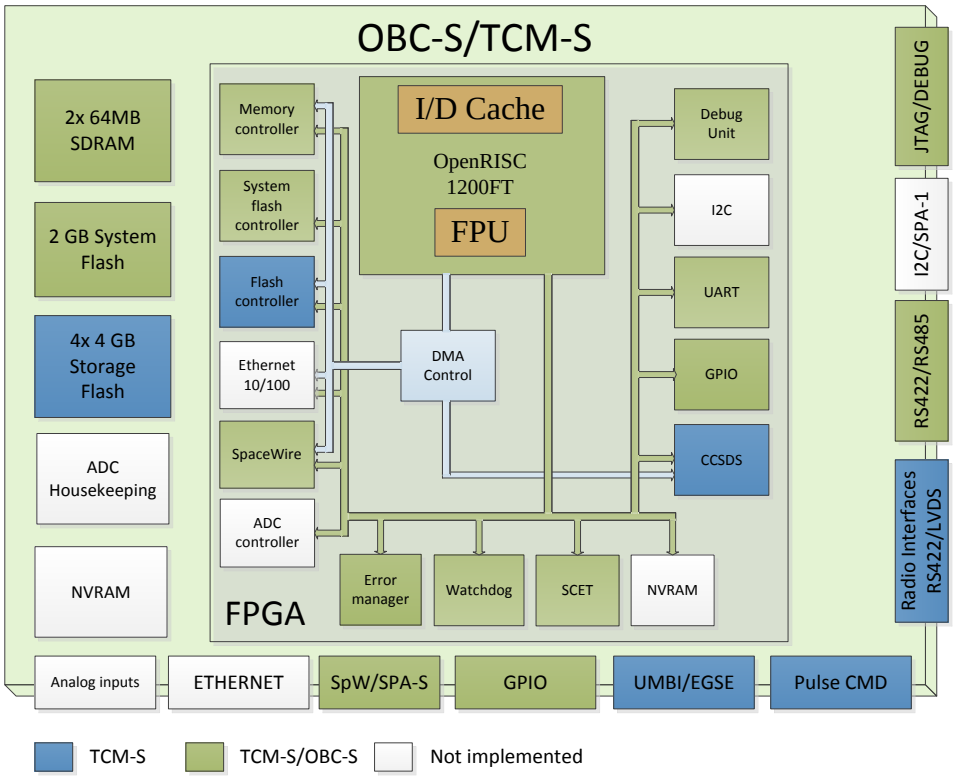

Figure 1 - The OBC-S™ / TCM-S™ SoC Overview

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 sudo 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

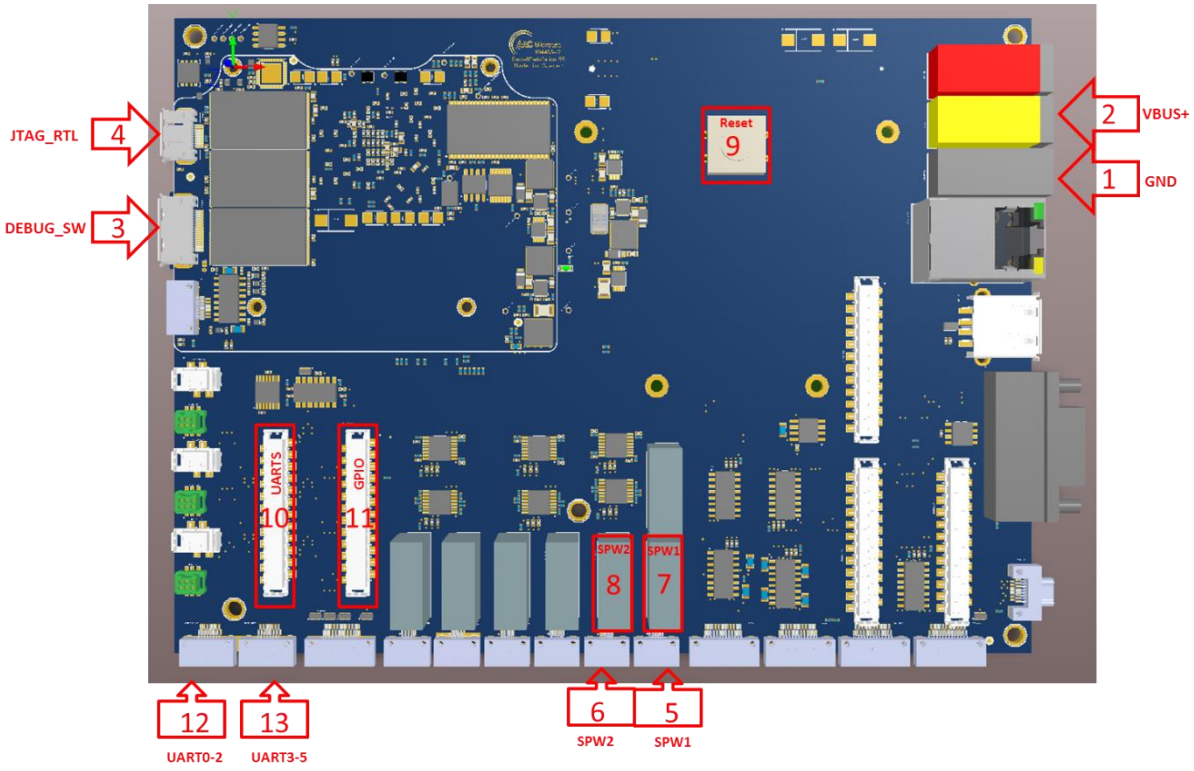

Figure 2 – ÅAC Sirius Breadboard with connector numbering

The Sirius Breadboard runs on a range of 4.5 to 16V DC. The instructions below refer to the connector numbering in Figure 2.

- 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 8.
- 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 7.3.

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-bsp-X.tar.bz2 to a directory of your choice (X matches the current version number).

The newly created directory aac-or1k-bsp now contains the drivers for both bare-metal applications and RTEMS. See the included README for more information.

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, the connection is working.

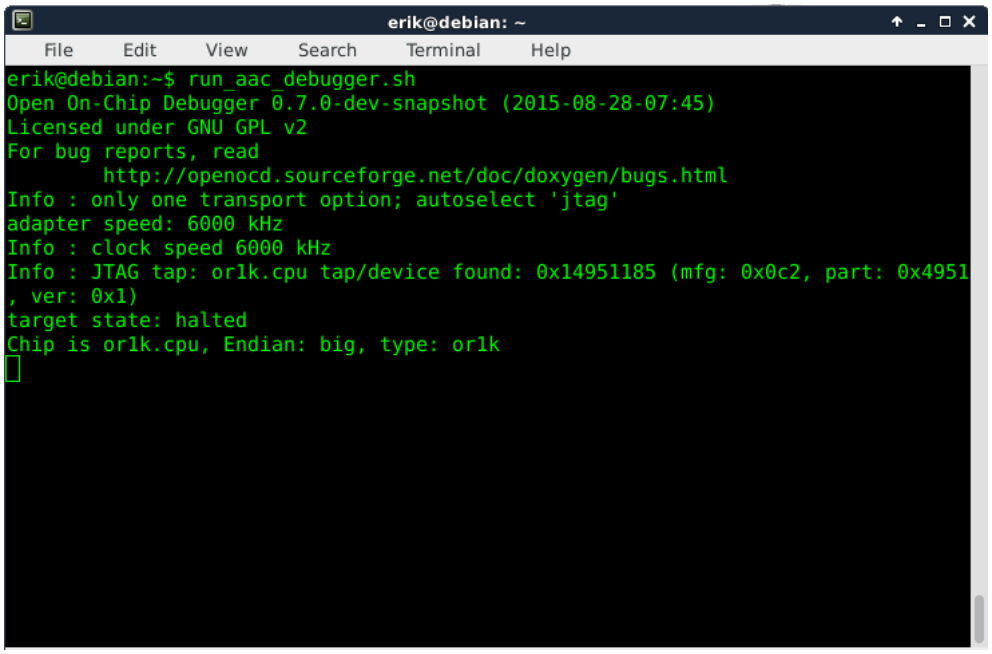

Figure 3 - Successful OpenOCD connection to the Breadboard

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
Handshaking: None

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. Start gdb with the following command from a shell
```
or1k-aac-elf-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 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-bsp/src/nandflash_programmer/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.

OBSERVE: The nandflash-program application might report bad blocks during programming. This is taken care of in the application itself, but isn't supported by the bootrom on the board. Please contact support@aacmicrotec.com for further assistance if this occurs.

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 the 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.

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-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 4.

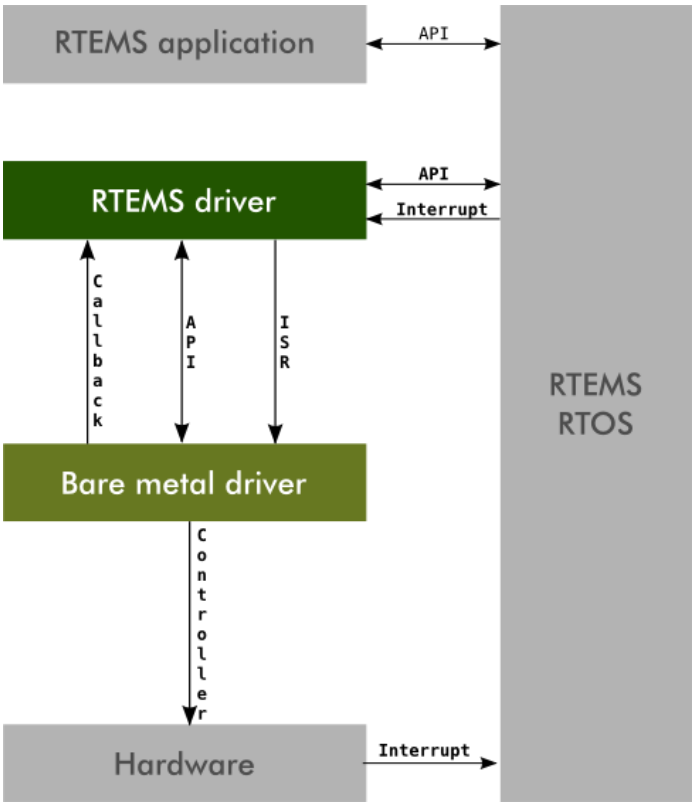

Figure 4 - Functionality access via RTEMS API

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 bare-metal 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. 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 description

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.

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.

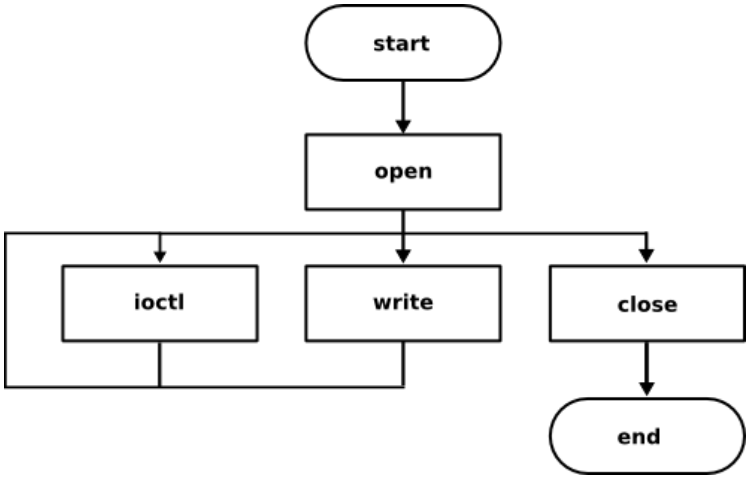

Figure 5 - RTEMS driver usage description

All calls to 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.h>


#define CONFIGURE_APPLICATION_NEEDS_WDT_DRIVER

#define CONFIGURE_INIT

rtems_task Init (rtems_task_argument argument);


#include <bsp/bsp_confdefs.h>
```

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.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 RTEMS POSIX API for ease of usage. In case of failure on a function call, *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.

5.3.2.1.int open(…)

Opens access to the low bare-metal 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. 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 of the error manager 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	Buffer to either read to or write from

Command table	Description
ERRMAN_GET_SR_IOCTL	Get the status register
ERRMAN_GET_CF_IOCTL	Gets the Carry flag register
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_SET_SR_IOCTL	Sets the status register
ERRMAN_SET_CF_IOCTL	Sets the carry flag register
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
ERRMAN_SET_SCRUBBER_IOCTL	Sets the scrubber. 1 = On, 0 = Off
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

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

5.3.3. Usage description

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.

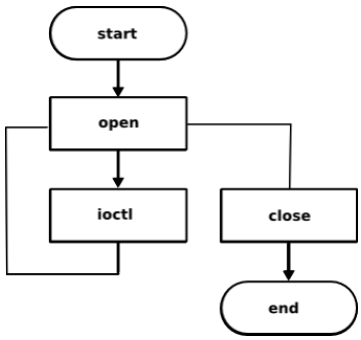

Figure 6 - RTEMS driver usage description

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.h>


#define
CONFIGURE_APPLICATION_NEEDS_ERROR_MANAGER_DRIVE
R


#define CONFIGURE_INIT
```

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

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

Inclusion of <`bsp/error_manager.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 initialized at boot up.

5.4. SCET

5.4.1. Description

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

5.4.2. 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.

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 low bare-metal driver. The device can only be opened once at a time.

Argument name	Type	Direction	Description
filename	char *	in	The absolute path to the file that is to be opened. SCET device is defined as RTEMS_SCET_DEVICE_NAME.
oflags	int	in	A bitwise 'or' separated list of values that determine the method in which the file is to be opened (whether it should be read only, read/write, 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	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	Input argument selects which GP trigger. Return value is the current value of that trigger. 0 = trigger activates on 0 to 1 transition 1 = trigger activates on 1 to 0 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 description

The main purpose of the SCET IP and driver is to track the time since power on and to act as a source of time stamps.

By utilizing the GP triggers one can trap the time stamp 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.

5.4.4. 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.

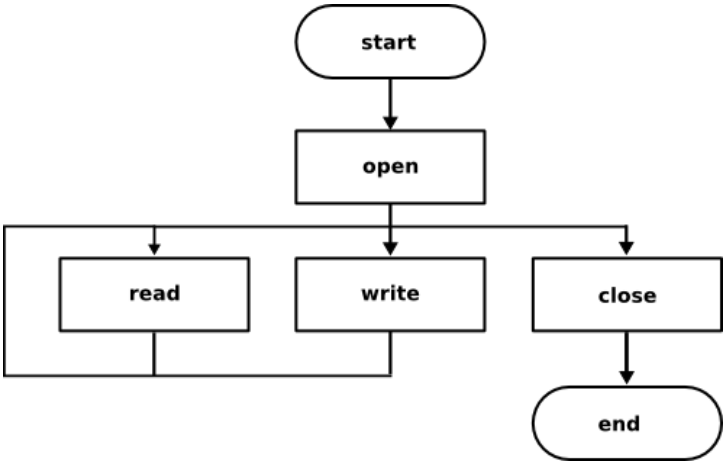

Figure 7 - RTEMS driver usage description

5.4.4.1.Time handling

Getting the current SCET time in RTEMS can be done in two ways:
1. Using read call, reading 6 bytes.
The first 4 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 write 6 bytes to the device.
1. 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.

5.4.4.2.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', P', '1' handles messages sent from the general purpose trigger 1.

Event name	Description
TRIGGER1	Trigger 1 was triggered

5.4.4.3.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.

5.4.4.4.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>
```

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. 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 usage. In case of a failure on a function call, the *errno* value is set for determining the cause.

5.5.2.1.Function int open(...)

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

Argument name	Type	Direction	Description
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/uart_psu_control	PSU Control, RS485 only
/dev/uart_safe_bus	Safe bus, RS485 only

5.5.2.2.Function int close(...)

Closes access to the device and disables the line drivers.

Argument name	Type	Direction	Description
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.
- 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 mode and setting the baud rate.
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: 7 = 115200 bps (default) 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

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

5.5.3. Usage description

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

CONFIGURE_APPLICATION_NEEDS_UART_DRIVER

5.5.3.1.1. 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.

5.5.4. Limitations

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

5.6. Mass memory

5.6.1. Description

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

5.6.2. 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 usage. In case of failure on a function call, *errno* value is set for determining the cause.

5.6.2.1. 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

5.6.2.2. 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

5.6.2.3. 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**.

5.6.2.4.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.

5.6.2.5. 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.

5.6.2.6.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 5.6.2.6.1 to 5.6.2.6.9.
value	void *	in	The value relating to command operation as defined in subchapters 5.6.2.6.1 to 5.6.2.6.9.

5.6.2.6.1. Bad block check

Checks if the given block is a bad block.

Command	Value type	Direction	Description
MASSMEM_IO_BAD_BLOCK_CHECK	uint32_t	in	Block number.

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

5.6.2.6.2. Reset mass memory device

Resets the mass memory device.

Command	Value type	Direction	Description
MASSMEM_IO_RESET			

Return value	Description
0	Always

5.6.2.6.3. Read status data

Reads the status register value.

Command	Value type	Direction	Description
MASSMEM_IO_READ_STATUS_DATA	uint32_t*	out	

Return value	Description
≥0	Status register value

5.6.2.6.4. Read control status data

Reads the control status register value.

Command	Value type	Direction	Description
MASSMEM_IO_READ_CTRL_STATUS	uint8_t*	out	

Return value	Description
0	Always

5.6.2.6.5. Read EDAC register data

Reads the EDAC register value.

Command	Value type	Direction	Description
MASSMEM_IO_READ_EDAC_STATUS	uint8_t*	out	

Return value	Description
0	Always

5.6.2.6.6. Read ID

Reads the ID

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

Return value	Description
0	Always

5.6.2.6.7. Erase block

Erases a block

Command	Value type	Direction	Description
MASSMEM_IO_ERASE_BLOCK	uint32_t	in	Block number

Return value	Description
0	Always

5.6.2.6.8. Read spare area

Reads the spare area with given data.

Command	Value 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.

5.6.2.6.9. Program spare area

Programs the sapre area from the given data

Command	Value 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.

5.6.3. Usage description

5.6.3.1.RTEMS

5.6.3.1.1. 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.

5.6.3.1.2. 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 5.6.2. And, if desired, the access can be closed when not needed.

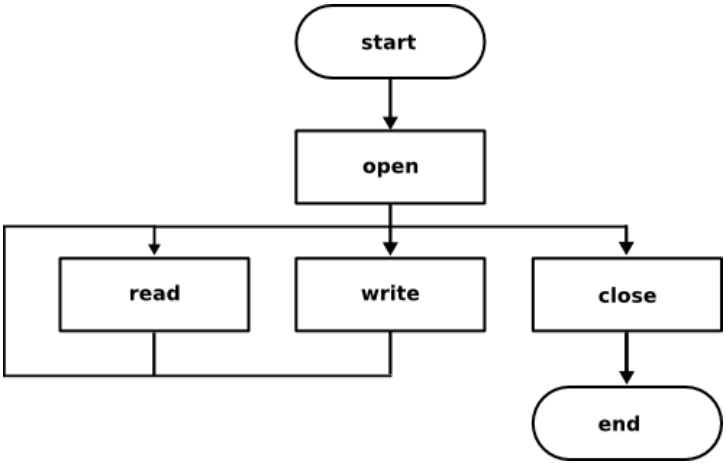

Figure 8 - RTEMS driver usage description

Note! All calls to RTEMS driver are blocking calls.

5.6.3.1.3. 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 ignored)
{
  .
  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. Spacewire

5.7.1. Description

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

5.7.2. RTEMS API

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

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

5.7.2.1.int open(…)

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

Argument name	Type	Direction	Description
filename	char *	in	Device name to register to for data transaction.
oflags	int	in	Device must be opened by exactly one of the symbols defined in Table 5.2.

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

5.7.2.2.int close(…)

Deregisters the device name from data transactions.

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

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

5.7.2.3.size_t read(…)

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 CPU_STRUCTURE_ALIGNMENT buf_rx[PACKET_SIZE];
```

Note! This call is blocked till 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	*buf* size in 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.

5.7.2.4.size_t write(…)

Transmits 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 CPU_STRUCTURE_ALIGNMENT buf_rx[PACKET_SIZE];
```

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

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

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

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.

5.7.2.5.int ioctl(…)

Additional supported operations via POSIX Input/Output Control API.

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

5.7.2.5.1. Mode setting

Sets the device into the given mode.

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

Command	Value 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.

5.7.3. Usage description

5.7.3.1.RTEMS

5.7.3.1.1. 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 5.7.3.1.3. The registration is done by function `open` and deregistered by the function `close`.

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

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

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

```
uint8_t CPU_STRUCTURE_ALIGNMENT buf_rx[PACKET_SIZE];
```

5.7.3.1.2. 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 5.6.2. And, if desired, the access can be closed when not needed.

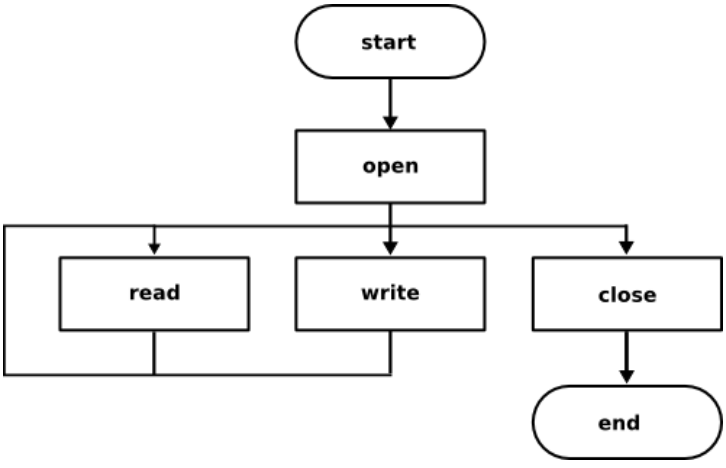

Figure 9 - RTEMS driver usage description

Note! All calls to RTEMS driver are blocking calls.

5.7.3.1.3. RTEMS application example

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

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

.
.
#define CONFIGURE_APPLICATION_NEEDS_SPACEWIRE_DRIVER

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

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

uint8_t 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.

5.7.4. 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.

6. TCM-S™

6.1. Description

The TCM-S™ handles receiving and decoding of telecommands (TC) transferred as Telemetry Transfer Frames on a baseband path from a transceiver. If the actual TC is a pulse command, it is decoded in HW and a HW signal is set according to the parameters in the pulse command. Other types of telecommands are handled in SW by the TCM-S™ application. If the actual telecommand is not addressed for the TCM-S™, the telecommand is routed to other nodes in the S/C on the SpW-network. Telecommands addressed for the TCM-S™ are handled by the TCM-S™

Telemetry (TM) from other nodes on the SpW-network is received by the TCM-S™. Telemetry is stored on a Mass Memory or downloaded to ground during ground-passes.

The Mass Memory is organized in different partitions. Every partition is intended for a certain type of telemetry. The partitions of the Mass Memory are configurable in terms of size, mode of buffer, etc. During ground-passes, the content of the partitions are downloaded to ground as Telemetry Transfer Frames. The downloading priority of the partitions is configurable.

SW Images are stored on a system flash. The image to boot is configurable by a Telecommand. If a boot of an image fails, the TCM-S™ will boot from a default, fallback image. An image can be updated by a PUS-service.

Configuration data and meta-data of partitions of Mass Memory are stored in a NVRAM.

In addition to SpW-interfaces, the TCM-S™ also has interfaces for SPI, I2C, RS 422/485 and Ethernet.

6.2. RMAP

To access sub-systems in the TCM-S™ from SpW, a simplified protocol with some similarities with RMAP is supported. Write and read-commands as described in **Error! eference source not found.** are supported with the following deviations:

- The Reply Address length is 0
- No buffering of received commands is done, so the TCM handles one command at a time.
- The TCM-S™ does not support verification of data or increment.

According to **Error! Reference source not found.**, a 40-bits address map consisting of an -bit Extended Address field and a 32-bit Address field is supported in the TCM-S™.

Extended Address Field	Description
0x00	Configuration
0x01-0xFF	Partitions on Mass Memory

An overview of the commands is given in the table below. Commands that are yet not implemented are marked as To Be Implemented (TBI) in the table below.

Extended Address Field	Address	Command	Comment
0x00	0x00000000	Telemetry status	Read command. (TBI)
0x00	0x000001xx	Partition Virtual Channel	Read/Write command (TBI)
0x00	0x00000200	Downlink Baseband Processing	Read/Write command (TBI)
0x00	0x00000300	Timestamp control	Read/Write command (TBI)
0x00	0x00000400	Bitrate	Read/write command (TBI)
0x00	0x01000000	Telecommand status	Read command (TBI)
0x00	0x01000100	Pulsecommand status	Read command (TBI)
0x00	0x02000000	Housekeeping status	Read command
0x00	0x03000000	Error status	Read command (TBI)
0x00	0x04000000	Housekeeping (TRX)	Read command (TBI)
0x00	0x04000100	GetRxStatus (TRX)	Read command (TBI)
0x00	0x04000200	GetTxStatus (TRX)	Read command (TBI)
0x00	0x04000300	TranscevierCommand (TRX)	Write command (TBI)
0x00	0x05000000	MassMemoryStatus,	Read command (TBI)
0x00	0x050001xx	WritePointer,	Read/Write command. xx-Partition Number
0x00	0x050002xx	ReadPointer,	Read/Write command. xx-Partition Number
0x00	0x05000300	PartitionConfiguration. (Read/Write)	Read/Write command.
0x01-0xFF	0x00000000-0xFFFFFFFF	Mass Memory PartitionData	Read/Write command

6.2.1. Housekeeping Interface

To get Housekeeping data from TCM-S™, a read command with an Extended Address Field and Address Field as described below shall be sent.

Extended Address Field	Address Field	Description
0x00	0x02000000	Housekeeping data. (Read)

The data parameters of the read command is described below:

Data	Type	Description
InputVoltage[V]	FLOAT32	The input voltage to the TCM-S™
RegulatedVoltage[V]	FLOAT32	The regulated voltage of the TCM-S™
InputCurrent[A]	FLOAT32	The input current to the TCM-S™
Temperature[degC]	FLOAT32	The temperature

6.2.2. Mass Memory Interface

To read status and configuration of the partitions of the TCM-S™, read and write commands with an Extended Address Field and Address Field as described below shall be sent.

Extended Address Field	Address Field	Description
0x00	0x050001xx	WritePointer, xx-Partition Number (Read/Write)
0x00	0x050002xx	ReadPointer, xx-Partition Number (Read/Write)
0x00	0x05000300	PartitionConfiguration. (Read/Write)

The data parameter of WritePointer command is described below:

Data	Type	Description
WritePointer	UINT64	The write pointer of the selected partition

The data parameter of ReadPointer command is described below:

Data	Type	Description
ReadPointer	UINT64	The read pointer of the selected partition

The data parameter of PartitionConfiguration command is described below:

Data	Type	Description
ConfigurationArray	Array of UINT8	The partition configuration

Byte	Type	Description
0	uint8_t	The partition number
1:8	uint64_t	Size in bytes. Must be in multiples of block size (128 pages * 16384 bytes)
9:12	uint32_t	The offset in blocks of the partition
13	uint8_t	The mode of the partition. 1: FIFO 2: Circular 3: Static Circular
14	uint8_t	The data source identifier for the partition.

An example of a command for reading the read pointer of partition 1 is shown below:

Target Logic Address 0x30	Protocol Identifier 0x01	Instruction 0x48	Key 0x00
Initiator Logic Address 0x40	Transaction Id. (MS) 0x01	Transaction Id. (LS) 0x02	Extended Address 0x00
Address (MS) 0x05	Address 0x00	Address 0x02	Address (LS) 0x01
Data Length (MS) 0x00	Data Length 0x00	Data Length (LS) 0x08	Header CRC 0x00

The response for the command above is:

Initiator Logic Address 0x40	Protocol Identifier 0x01	Instructtion 0x08	Status 0x00
Target Logic Address 0x30	Transaction Id. (MS) 0x01	Transaction Id. (LS) 0x02	Reserved 0x00
Data Length (MS) 0x00	Data Length 0x00	Data Length (LS) 0x08	Header CRC 0x00
Data 0x00	Data 0x00	Data 0x00	Data 0x00
Data 0x01	Data 0x02	Data 0x03	Data 0x04
Data CRC 0x00			

6.2.3. Mass Memory Partition Data

To command for writing/reading data to/from a partition is described below.

Extended Address Field	Address Field	Description
0x01-0xFF	0x00000000-0xFFFFFFFF	Reads or writes data to/from a partition. The extended address field states which partition to access and the address field states how many bytes to read/write from/to the partition

The data parameter of Read/Write Data command is described below:

Data	Type	Description
DataArray	Array of UINT8	The written or read bytes

An example of a command for writing 7 bytes to partition 1 is shown in below:

Target Logic Address 0x30	Protocol Identifier 0x01	Instruction 0x68	Key 0x00
Initiator Logic Address 0x40	Transaction Id. (MS) 0x01	Transaction Id. (LS) 0x02	Extended Address 0x01
Address (MS) 0x00	Address 0x00	Address 0x00	Address (LS) 0x00
Data Length (MS) 0x00	Data Length 0x00	Data Length (LS) 0x07	Header CRC 0x00
Data 0x10	Data 0x20	Data 0x30	Data 0x40
Data 0x50	Data 0x60	Data 0x70	Data CRC 0x00

The response to the command above is:

Initiator Logic Address 0x40	Protocol Identifier 0x01	Instructtion 0x28	Status 0x00
Target Logic Address 0x30	Transaction Id. (MS) 0x01	Transaction Id. (LS) 0x02	Header CRC 0x00

7. System-on-Chip definition

The ÅAC Sirius products include two boards built around the OR1200 fault tolerant processor, the OBC-S™ and the TCM-S™. 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.

7.1. Memory mapping

Table 7.1 - Sirius memory structure definition

Memory Base Address	Function
0xF0000000	Boot ROM
0xE0000000	CCSDS (TCM-S™ only)
0xCB000000	Watchdog
0xCA000000	Space Craft 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™ 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™ only, High speed UART w. DMA)
0x95000000	UART 4 (OBC-S™ only)
0x94000000	UART 3 (OBC-S™ only)
0x93000000	UART 2
0x92000000	UART 1
0x91000000	UART 0
0x90000000	UART Debug (LVTTL)
0x80000000 - 0x8F000000	Customer IP
0x00000000	SDRAM memory including EDAC (64 MB)

7.2. Interrupt sources

The following interrupts are available to the processor:

Table 7.2 - 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)

7.3. Peripherals/ports

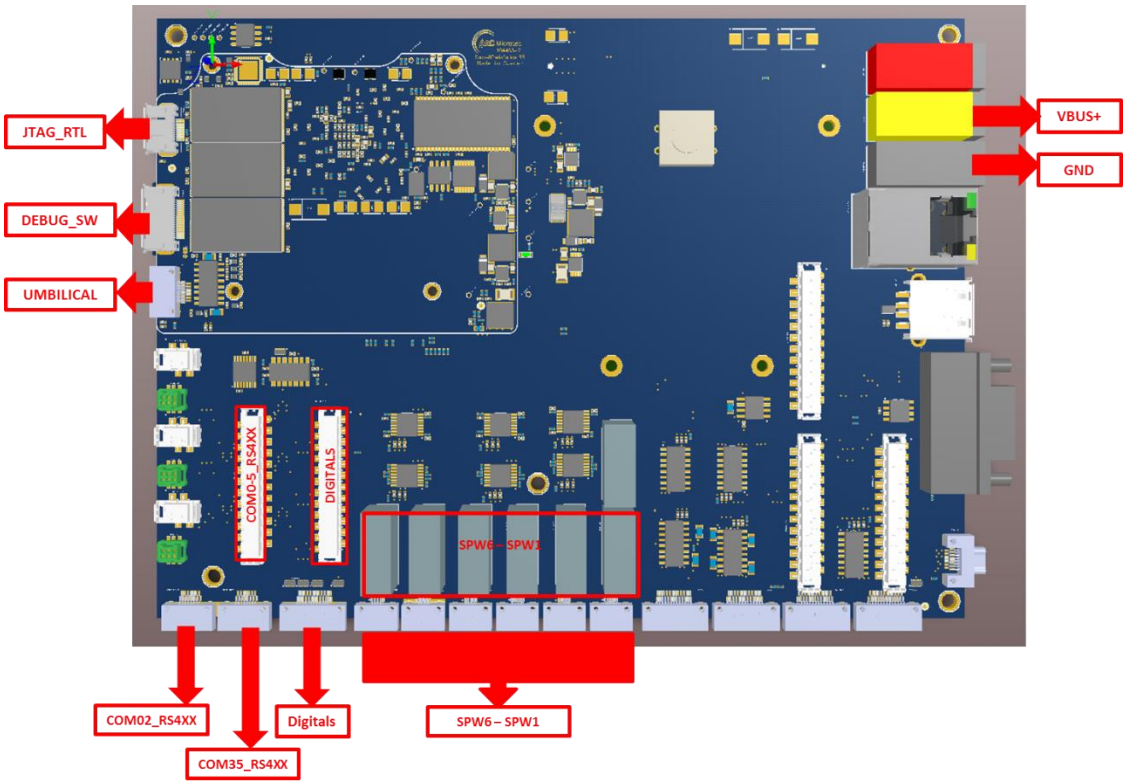

Figure 10 - Sirius ports

7.3.1. JTAG_RTL

The following pins are available on the JTAG_RTL, Hirose ST60-10P, connector. See Table 7.3.

Table 7.3 - 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

7.3.2. Debug SW

The following pins are available on the DEBUG SW, Hirose ST60-18P, connector. See Table 7.4.

Table 7.4 - 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

7.3.3. Spacewire/SPA-S (SPW1-6)

The following pins are available on the SPW1-6 connectors, Glenair Nano-D 891-013-9SA2-BRST. See Table 7.5

Table 7.5 - 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

7.3.4. DIGITALS, 3x I2C / SPA-1, PPS and 12xGPIO.

The following pins are available on the DIGITALS connector, Connector_nanoD_25_Socket

Table 7.6 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	
Pin 15	SPI_MOSI	
Pin 16	SPI_CLK	
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

7.3.5. UART RS422/485-1

The following pins are available on the COM02_RS4XX connector, Glenair Nano-D 891-013-15SA2-BRST. SeeTable 7.5.

Table 7.7 COM02_RS4XX pinouts

Pin #	Signal name	Description
Pin 1	COM0_RX_RS4XX_P	Com Port 0 RX
Pin 2	COM0_RX_RS4XX_N	
Pin 3	COM0_TX_RS4XX_P	Com Port 0 TX

Pin 4	COM0_TX_RS4XX_N	
Pin 5	GND	Ground
Pin 6	GND	
Pin 7	COM1_RX_RS4XX_P	COM Port 1 RX
Pin 8	COM1_RX_RS4XX_N	
Pin 9	COM1_TX_RS4XX_P	COM Port 1 TX
Pin 10	COM1_TX_RS4XX_N	
Pin 11	COM2_RX_RS4XX_P	COM Port 2 RX
Pin 12	COM2_RX_RS4XX_N	
Pin 13	COM2_TX_RS4XX_P	COM Port 2 TX
Pin 14	COM2_TX_RS4XX_N	
Pin 15	GND	Ground

7.3.6. UART RS422/485-2

The following pins are available on the COM35_RS4XX connector, Glenair Nano-D 891-013-15SA2-BRST. See Table 7.8.

Table 7.8 COM35_RS4XX pin-outs

Pin #	Signal name	Description
Pin 1	COM3_RX_RS4XX_P	Com Port 3 RX
Pin 2	COM3_RX_RS4XX_N	
Pin 3	COM3_TX_RS4XX_P	Com Port 3 TX
Pin 4	COM3_TX_RS4XX_N	
Pin 5	GND	Ground
Pin 6	GND	
Pin 7	COM4_RX_RS4XX_P	COM Port 4 RX
Pin 8	COM4_RX_RS4XX_N	
Pin 9	COM4_TX_RS4XX_P	COM Port 4 TX
Pin 10	COM4_TX_RS4XX_N	
Pin 11	COM5_RX_RS4XX_P	COM Port 5 RX
Pin 12	COM5_RX_RS4XX_N	
Pin 13	COM5_TX_RS4XX_P	COM Port 5 TX
Pin 14	COM5_TX_RS4XX_N	
Pin 15	GND	Ground

7.3.7. Digital I/O

The following pins are available on the DIGITALS connector, Glenair Nano-D 891-013-25SA2-BRST. See Table 7.9 Table 7.5.

Table 7.9 DIGITALS pin-outs

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	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-0 Clock
Pin 18	I2C_SDA0	I2C-0 Data
Pin 19	I2C_SCL1	I2C-1 Clock
Pin 20	I2C_SDA1	I2C-1 Data
Pin 21	I2C_SCL2	I2C-2 Clock
Pin 22	I2C_SDA2	I2C-2 Data
Pin 23	PPS_INPUT_RS422_N	Optional Pulse Per Second input
Pin 24	PPS_INPUT_RS422_P	
Pin 25	GND	Ground

8. Updating the Sirius FPGA

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

8.1. Prerequisite hardware

- Microsemi FlashPro5 unit
- 104470 FPGA programming cable assembly

8.2. Prerequisite software

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

8.3. 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 2
2. Connect the power cables according to Figure 2
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

9. 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™ and OBC-S™ boards.

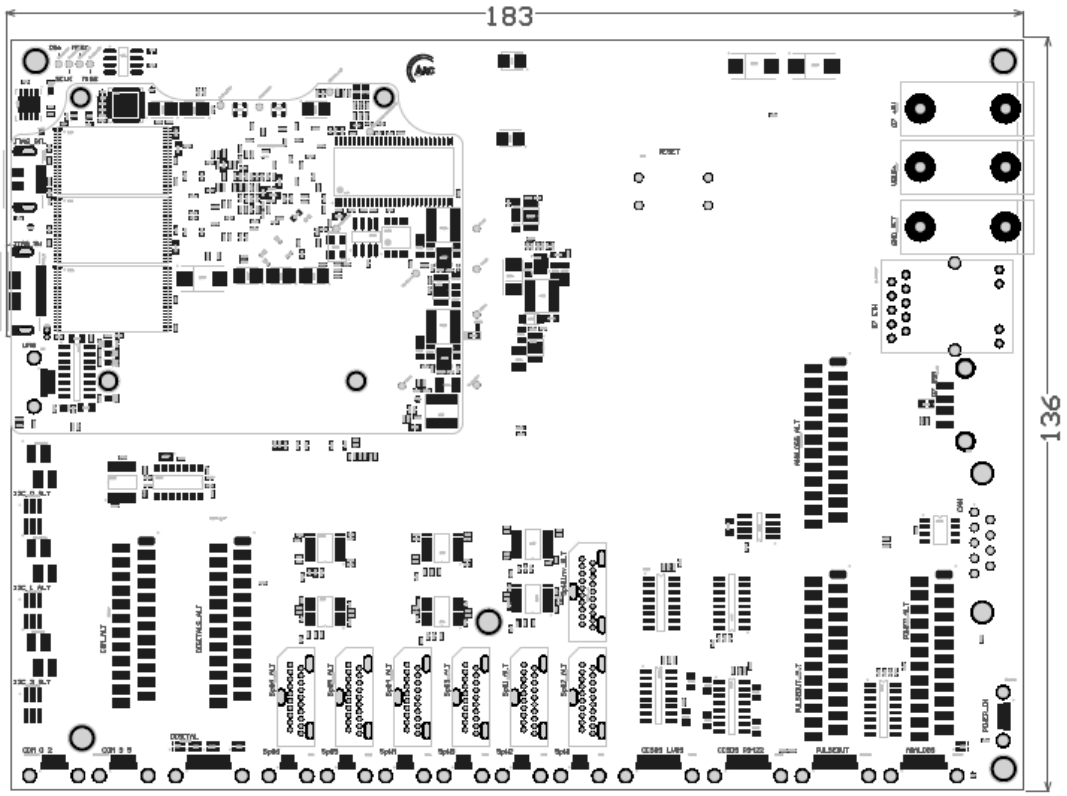

Figure 11 - The Sirius board mechanical dimensions

10. Environment information

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

Table 10.1 - Environment temperature ranges

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

11. Glossary

ADC	Analog Digital Converter
BSP	Board Support Package
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
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
OBC	On Board Computer
OS	Operating System
PCB	Printed Circuit Board
PCBA	Printed Circuit Board Assembly
POSIX	Portable Operating System Interface
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