# Table of Contents

1. Introduction .................................................................................................................. 1  
   1.1. Hardware ............................................................................................................. 1  
   1.2. Source code ......................................................................................................... 1  
   1.3. LEON2/3 peripherals ......................................................................................... 2  
   1.4. MMU support ...................................................................................................... 2  
2. Memory Configuration .................................................................................................. 3  
   2.1. Stack .................................................................................................................... 3  
3. Memory Management Unit ............................................................................................ 4  
   3.1. Initialization ....................................................................................................... 4  
   3.2. Adding a virtual to physical map ..................................................................... 4  
4. Booting from FLASH ..................................................................................................... 5  
   4.1. LEON Boot loader ............................................................................................. 5  
      4.1.1. BOOTCFG_FREQ_KHZ ............................................................................... 6  
      4.1.2. BOOTCFG_UART_BAUDRATE .................................................................. 6  
      4.1.3. BOOTCFG_UART_FLOWCTRL ................................................................. 6  
      4.1.4. BOOTCFG_WASH_MEM ......................................................................... 6  
      4.1.5. BOOTCFG_DSUX_ADDRESS ................................................................. 6  
      4.1.6. Memory controller configuration options ........................................... 6  
   4.2. Booting VxWorks kernel from FLASH ............................................................... 6  
      4.2.1. Selecting build rule ................................................................................... 6  
      4.3. Bootrom ........................................................................................................... 7  
      4.3.1. Configuring the boot loader ..................................................................... 7  
      4.3.2. Creating a bootrom project ................................................................. 8  
      4.3.3. Building the bootrom ............................................................................ 8  
      4.3.4. Installing and running the bootrom ..................................................... 8  
      4.3.5. Troubleshooting ....................................................................................... 9  
5. Interface ......................................................................................................................... 10  
   5.1. Function interface .......................................................................................... 10  
      5.1.1. sysOut* and sysIn* ............................................................................. 10  
6. Compilers ....................................................................................................................... 11  
7. Drivers .......................................................................................................................... 12  
8. Support .......................................................................................................................... 13
1. Introduction

This document aims to provide documentation for SPARC/LEON specific BSPs. There are two BSPs the non-MMU and the MMU BSP that comes with the Aeroflex Gaisler distribution. All BSPs can be found in the `vxworks-6.5/target/config` directory. The BSPs are similar apart from the MMU related parts.

The Board Support Package (BSP) implement the necessary functions needed by SPARC port to function. It handles memory set up, trap table, MMU, selecting what drivers will be used and so on.

The board support packages do not target a single board, they try to be general and support as many boards as possible. This is possible due to the Plug&Play support on the CPU bus, however some parameters needs to be setup. Memory parameters must be set up in order to start a VxWorks kernel, when booting from FLASH/PROM the boot loader parameters must also be set up.

1.1. Hardware

The supported hardware is summarized in the list below. For documentation about a specific core's driver please see the LEON VxWorks 6.5 Driver Manual.

- LEON2 and LEON3
- MMU and non-MMU systems
- FPU, non-FPU, hardware MUL/DIV and software MUL/DIV support
- Interrupt controller
- UART console/terminal driver
- Timer unit
- General Purpose I/O (GRGPIO)
- 10/100 Ethernet networking (GRETH, LAN91C111)
- 10/100/1000 Ethernet networking (GRETH)
- SpaceWire (GRSPW)
- DMA CAN 2.0 (GRCAN)
- non-DMA CAN 2.0 (OCCAN)
- 1553 BC, RT and MT support (B1553BRM)
- Host USB 1.1 and 2.0 (GRUSBHC)
- I2C Master (I2CMST)
- ADC/DAC controller (GRADCDAC)
- PCI support (GRPCI, PCIF)
- PCI support (AT697 PCI for non-MMU BSP)
- GR-RASTA-IO PCI Target
- GR-701 PCI Target
- GR-RASTA-ADCDAC PCI Target

1.2. Source code

All BSPs are located at `vxworks-6.5/target/config`, the two LEON BSPs are named `wrleon23_sparcleon` and `wrleon23mmu_sparcleon`. An overview of the sources of one BSP is given by the table below, all paths are given relative to one of the BSPs, not all files may be available for both BSPs.

<table>
<thead>
<tr>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysLib.c</td>
<td>The BSP system Library implementation, this is the heart of the BSP. The code handles reboot, system initialization in two stages, MMU setup and other needed functions.</td>
</tr>
<tr>
<td>sysALib.s</td>
<td>Low level setup, such as stack, trap table. Calls usrInit().</td>
</tr>
<tr>
<td>config.h</td>
<td>Default BSP configuration, default project configuration when a project is created based upon the BSP, it can be overridden from the Kernel Configuration GUI. This file is normally customized for a particular board.</td>
</tr>
<tr>
<td>Source</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>configNet.h</td>
<td>Networking configuration, number of network interfaces, Ethernet MAC number IRQ number of SMC91C111 etc.</td>
</tr>
<tr>
<td>romInit.s</td>
<td>Boot loader implementation, used to boot the bootrom (Small VxWorks kernel with network boot capabilities) or standard kernel image. This contains the first instructions executed when starting from FLASH/PROM. It initializes hardware registers such and memory controller, CPU, FPU, wash memory etc. controlled by the configuration in bootcfg_def.h, config.h or project configuration (FOLDER_BOOTCFG in Hardware/BSP configuration variants/LEON ROM Boot setup).</td>
</tr>
<tr>
<td>bootcfg_def.h</td>
<td>Default boot loader configuration (romInit.s). It is included from config.h, the settings can be overridden from the project configurations, or a custom bootcfg_XYZ.h can be created can included instead of bootcfg_def.h.</td>
</tr>
<tr>
<td>bootloader.h</td>
<td>Used by the boot loader (romInit.s) to calculate some values such as timer SCALER from system frequency.</td>
</tr>
<tr>
<td>Makefile</td>
<td>Makefile when compiling BSP (often bootrom). One must configure which toolchain to use and the memory parameters to match config.h when creating a bootrom.</td>
</tr>
<tr>
<td>hwconf.c</td>
<td>VxBus Driver/Device configuration controlled from project settings. For LEON2 systems this also configures the hardware present apart from the basic peripherals (TIMER, UART, IRQ).</td>
</tr>
<tr>
<td>*.cdf</td>
<td>BSP configuration files for WindRiver command line tools and Workbench kernel configuration GUI.</td>
</tr>
<tr>
<td>sysI2C.c</td>
<td>GRLIB I2CMST driver implementation.</td>
</tr>
<tr>
<td>eepromI2CSlave.c</td>
<td>A simple I2C EEPROM driver. The EEPROM can be read and written, it does not support all types I2C of EEPROMs but may serve as a start point for a custom implementation.</td>
</tr>
<tr>
<td>usbPciStub.c</td>
<td>USB 1.1/2.0 driver implementation.</td>
</tr>
<tr>
<td>sysSerial.c</td>
<td>Serial UART/Console routines, called from sysLib.c</td>
</tr>
</tbody>
</table>

### 1.3. LEON2/3 peripherals

The LEON2 on-chip peripherals are assumed to reside in the default address as defined in the LEON2 VHDL model. Also the interrupt assignment of UART and Timer are assumed to retain their default values. For LEON3 systems, the address and interrupt is read from the on-chip Plug&Play information, and is assigned during run-time.

### 1.4. MMU support

One of the two BSPs support Memory Management Unit (MMU) which offers memory protection between processes and kernel. Full protection can be achieved by not allowing processes to write each others private memory and of course not to the kernel area. If such a violation occurs the kernel halts the faulting process and the developer can find out what process and what instruction went wrong.
2. Memory Configuration

The memory configuration is not auto detected in VxWorks, the user must configure the project with a valid memory configuration. The most important memory options are listed in the table below.

Table 2.1. BSP Memory parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCAL_MEM_LOCAL_ADRS</td>
<td>Start address of main memory. Usually 0x40000000.</td>
</tr>
<tr>
<td>LOCAL_MEM_SIZE</td>
<td>Size of VxWorks available memory. Usually the memory size minus the size of the reserved memory. For example 0x3ff000 on a system with 4Mb RAM.</td>
</tr>
<tr>
<td>RAM_LOW_ADRS</td>
<td>Low RAM address, this is where the VxWorks image will be placed before jumping into sysALib.s. The trap table is installed on RAM_LOW_ADRS-0x3000. Usually the low address is start of main memory + 0x3000, 0x40003000.</td>
</tr>
<tr>
<td>RAM_HIGH_ADRS</td>
<td>High RAM address. When the bootrom is used the boot loader places the small VxWorks kernel (the bootrom) at high RAM, the RAM_LOW_ADRS..RAM_HIGH_ADRS is used by the bootrom kernel to place the network fetched VxWorks kernel into before booting. Usually set to half main memory + 0x3000, for example 0x40203000 on a system with 4Mb RAM.</td>
</tr>
</tbody>
</table>

2.1. Stack

The stack is located at top of the available main memory. Memory may be reserved over the the stack making less memory available. The stack grows downwards to lower addresses on a SPARC CPU.

The Interrupt context is executed on a separate stack, that stack is located just above the SPARC trap table.
3. Memory Management Unit

This section describes the MMU support in the LEON MMU BSP.

The wrleon23mmu_sparcleon BSP is used for systems that have MMU and wish to use it. Systems that have a MMU but does not intend to use it or systems without a MMU should use the non-MMU BSP: wrleon23_sparcleon.

In order to write device drivers independent of the MMU support physical addresses are mapped 1:1 into virtual address space.

3.1. Initialization

During most of the low level system startup the MMU is disabled. The BSP add areas that need to be mapped into virtual address space into the sysPhysMemDesc array, the most important area is of course the main memory. The areas mapped by the BSP using the sysPhysMemDesc array can be viewed by calling,

```c
void sysMemMapShow (void);
```

The AMBA Plug&Play initialization routines map all APB Slave and AHB Slave I/O address spaces into virtual space on a 1:1 basis. This means that most of the on-chip AMBA drivers does not have to take into account if a MMU is present or not.

The PCI Host driver map all configured PCI BARs into virtual address space.

3.2. Adding a virtual to physical map

Physical addresses can be accessed directly using the sysIn/sysOut functions described in Section 5.1.1. Normally physical addresses does not have to be accessed directly, instead one can map the physical page (4k large) into virtual space setting the appropriate permission bits. Pages can be mapped using platform independent VxWorks function (vmLib.h), or add the map directly into the config/BSP/sysLib.c: sysPhysMemDesc[] array.

It is possible to debug the virtual to physical mapping using the GRMON command "walk 0xVIRT_ADDR" and "vmem 0xVIRT_ADDR".
4. Booting from FLASH

A VxWorks kernel can be started from RAM or FLASH/PROM. A kernel started in RAM must be loaded into RAM by a boot loader, there are different loaders and they have their different options. The loaded VxWorks kernel expects that some hardware has been initialized, for example the memory controller. The most common loaders are listed below:

- GRMON - hardware debugger
- Boot loader - started in FLASH, initializes hardware and loads a kernel
- Bootrom - A small network VxWorks kernel booted by the boot loader
- MKPROM (not used in VxWorks projects since is has it's own loader)

In this chapter the LEON boot loader and the bootrom is discussed.

Note that when running the kernel from RAM using GRMON, it is not necessary to configure the boot loader since it is not used. GRMON itself does the basic initialization when connecting and typing `run`.

4.1. LEON Boot loader

The LEON boot loader is part of the BSP, in file `BSP/romInit.s`. It initializes the memory controllers, CPU, FPU, wash memory etc. The boot loader's primary task is to load the a VxWorks kernel. Some basic initialization such as memory controller parameters and system console baud rate is initialized by the boot loader before booting VxWorks. The initialization parameters are configured using the VxWorks Kernel configuration utility from the Workbench, see Figure 4.1 or by creating a custom bootcfg.h file. The default boot loader parameters are set in bootcfg_def.h, it can be found in target/config/wrleon23[mmu]_sparcLeon (BSP directory), they are included into the BSP from config.h and can be overridden by the project's configuration as it is included after bootcfg_def.h. The boot loader in romInit.s includes the configuration and behave differently depending on the settings.

![Figure 4.1. LEON Boot loader configuration](image)

As mentioned above, the boot loader settings can be set by creating a custom bootcfg.h file or editing bootcfg_def.h directly. This changes the defaults of the BSP affecting every newly created project based on that BSP, however it does not affect already created BSPs. Configuring the boot loader using the Workbench only changes that particular project's boot loader settings.
To find the correct initialization values please consult the core documentation (GR-IP documentation).

Some of the boot loader parameters are briefly described below.

4.1.1. **BOOTCFG_FREQ_KHZ**

Frequency of main AMBA bus in KHz. This setting affects the system timer interrupt frequency, baud rates calculated from the system frequency etc.

4.1.2. **BOOTCFG_UART_BAUDRATE**

The UART baud rate it is first initialized to before entering the VxWorks kernel. The default is 38400 bits/second.

4.1.3. **BOOTCFG_UART_FLOWCTRL**

Enable or disable UART flow control on start up.

4.1.4. **BOOTCFG_WASH_MEM**

Enable or disable main memory clearing. All main memory available to VxWorks is washed by setting the contents to zero. Enabling washing can be necessary for fault tolerant systems, this is because uninitialized data in the main memory have incorrect check bits and will cause a trap when accessed the first time.

4.1.5. **BOOTCFG_DSUX_ADDRESS**

Debugging the application starting from PROM without GRMON can be hard sometimes, using "grmon -ni" to connect into a target whose application has crashed may reveal much information and even more can be extracted if the DSU trace buffers are enabled first thing during boot. The DSU is enabled early and is made not to rely on the Plug&Play information requiring the user to set **BOOTCFG_DSUX_ADDRESS** to the base address of the DSU hardware registers.

The default is to disabled this feature, however when enabled the default location of the DSU is 0x90000000.

4.1.6. **Memory controller configuration options**

The memory controller options are all similar, CFG1, CFG2 and CFG3. The values are written to the memory controller registers configuration registers. Some controllers does not have three configuration registers, and some bits are ignored, typically EDAC and PROM width because they are configured from the PCB switches.

GRMON auto detects memory parameters by using different algorithms for different memory controllers, the value auto detected is often not the most optimal, the configuration can be viewed by issuing 'info reg' and 'info sys' from GRMON. Most of the bits in the MCFGx registers can be copied without modification.

To find the correct initialization values please consult the core documentation (GR-IP documentation) and the memory module documentation.

4.2. **Booting VxWorks kernel from FLASH**

VxWorks kernels can be booted by using the LEON boot loader. The loader loads a VxWorks RAM image into the RAM_LOW_ADRS, and jumps into it. The configuration parameters of the boot loader must have been set up according to Section 4.1. The build rule can be changed from default to default_romCompress to create a boot loader including a compressed RAM image, the compressed image is extracted into RAM_LOW_ADRS.

4.2.1. Selecting build rule

By using the Workbench one can set the build rule and the Workbench remembers the setting until the project is built. Choose build rule by pressing right mouse button on the project icon and selecting 'Build Options/Set Active Build Spec...'. A dialog presents all possible targets for the project:
Figure 4.2. Workbench build rule dialog

• Default - image run from directly from RAM, no boot loader included. Typically used during debugging using grmon.

• Default_rom - image with boot loader not compressed. Programmed into flash. Boot loader copy kernel to RAM and starts execution from there.

• Default_romCompress - image with boot loader able to decompress kernel. Programmed into flash. Boot loader decompress kernel into RAM and jump to start of kernel thereafter.

• Default_romResident - image intended to run from flash memory. Programmed into flash memory. Kernel is not copied to RAM. Used where RAM space is an issue.

The same targets apply for building projects with the command line tools. The build rule is passed as an environment variable ROM during project building. If ROM is left unset the default target will be built, that is the RAM image. Setting it to rom, romCompress or romResident selects one of the build rules described above.

4.3. Bootrom

This section describes the main steps in creating a VxWorks bootrom. The bootrom is typically used during the development process to download VxWorks kernels from a host on target reset and power on. The host can distribute VxWorks kernels over the network using FTP, TFTP and TSFS host file system. See the VxWorks documentation for more information.

A bootrom can be created using either the command line tools or the Workbench GUI. The make targets vxworks-bsp-compile and vxworks-bsp-res-compile has been prepared for compiling the bootrom using the command line tools.

The default bootrom is configured with networking and GRETH driver, the driver can be changed by editing BSP/config.h.

4.3.1. Configuring the boot loader

The bootrom is configured using the target/config/ wrleon23[mmu]_sparcleon/ config.h file for component selection and bootcfg_def.h for custom boot loader parameters. From config.h and Makefile it is possible to do the memory configuration needed, see the defines ROM_SIZE, ROM_BASE_ADRS, RAM_LOW_ADRS, RAM_HIGH_ADRS, LOCAL_MEM_LOCAL_ADRS, LOCAL_MEM_SIZE in Chapter 2. The memory configuration must be the same for the boot loader and for VxWorks kernels loaded.
Depending on the boot method the component selection may vary. If networking is not needed or the LanChip driver must be used rather than the GRETH driver, it can be changed from within config.h. The bootrom network and boot settings can be changed by editing DEFAULT_BOOT_LINE. The boot line argument definition can be found in the VxWorks documentation, below is a typical boot line, booting the image VxWorks from host neptune (192.168.0.47) using the anonymous FTP service, the target itself has the IP-address 192.168.0.184 and name grxc3s1500.

```
greth0(0,0)neptune:vxWorks e=192.168.0.184 h=192.168.0.47 g=192.168.0.1 u=ftp pw=user f=0x04 tn=grxc3s1500
```

Hardware registers are initialized by the boot loader before the bootrom starts, in romInit.s values from bootloader.h and bootcfg_def.h are used to set up the registers. One must create a custom bootcfg_def.h in order for the boot loader to successfully initialize the system. The boot loader parameters are described in Section 4.1.

### 4.3.2. Creating a bootrom project

A standard bootrom is created from the Workbench with the project creation guide, File -> New -> VxWorks Boot Loader Project. After giving the project a name and selecting the non-MMU or the MMU BSP depending on the hardware configuration, build target to Compressed or Resident and format to ELF.

The boot loader options are not available through the GUI as when creating VxWorks image projects, the configuration has to be done by hand, editing BSP/config.h.

### 4.3.3. Building the bootrom

The memory configuration in BSP/config.h must match the memory configuration in BSP/Makefile. The toolchain used when building the the bootrom is controlled from the TOOL variable in BSP/Makefile, the TOOL parameter can be changed to gnu/gnuv8/sfgnu/sfgnuv8 or diab/sfdiab.

Building the bootrom using the workbench is similar to building any other project, by pressing Build. The bootrom can be built using different build targets bootrom, bootrom_uncmp, bootrom_res and bootrom_res_high. The default is bootrom. The bootrom target produces an image that will uncompress itself into RAM and run from RAM. The build target is selected when creating the project or by selecting "Project -> Build Options -> Set Active Build Spec". The bootrom ELF-file is created in the project directory, named bootrom.

When building using the command line tools, the make targets vxworks-bsp-compile and vxworks-bsp-res-compile has been prepared for compiling the bootrom using the command line tools, the "MMU=yn" variable selects the BSP which is compiled. It is also possible to create a bootrom using the command line tools manually as below.

```
$ make execute
sh-3.00$ cd vxworks-6.5/target/config/wrleon23_sparcleon/
sh-3.00$ make bootrom
```

### 4.3.4. Installing and running the bootrom

The bootrom can be installed into FLASH using GRMON. The FLASH must be erased before written as below.

```
$ grmon -eth -ip 192.168.0.55
grmon> flash unlock all
grmon> flash erase 0 0x200000
grmon> flash load bootrom.rom
```

After a successful configuration the bootrom is booted after reset and power on, it can also be started from GRMON as follows,

```
grmon> run 0
```
The bootrom uses the serial terminal with the default settings as indicated by the table below. A terminal emulator can be started from within the workbench, "Window -> Show View -> Terminal". Other terminal emulators can also be used, for example minicom or hyper terminal. It is also possible to get the console output directly to GRMON's console if the flag -u is given when starting GRMON.

Table 4.1. Default terminal settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud rate</td>
<td>38400</td>
</tr>
<tr>
<td>Data bits</td>
<td>8</td>
</tr>
<tr>
<td>Stop bits</td>
<td>1</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
</tr>
<tr>
<td>Flow control</td>
<td>None</td>
</tr>
</tbody>
</table>

### 4.3.5. Troubleshooting

When running the bootrom from GRMON works, but not when power cycling the board it often is caused by bad memory controller settings. GRMON sets up them correctly, however the boot loader forgets to initializes them and everything seems to work.

Note that GRMON auto detects the memory controller configuration by using previous register contents for some bits. This means that if a faulty boot loader has been loaded, the boot loader may destroy the memory configuration and GRMON will not be able to auto detect memory parameters no more. To avoid this from happening the Gaisler boards have a `break` button which can be pressed during power cycling and reset that will prevent the CPU from start executing, it will break on reset.

One can compare the memory configuration with the configuration that GRMON auto detects, by issuing 'info sys' and 'info reg'. To avoid that GRMON initializes the registers on start up (overwriting the boot loader's memory configuration) the -ni flag can be used.

The `BOOTCFG_DSUX_ADDRESS` can be used to enable the DSU trace buffer on startup, enabling this during debugging may be helpful, when the boot loader crashes or hangs due to a faulty configuration, the last instructions may be viewed even though the application wasn't started by GRMON. When connecting with GRMON, the -ni flag can be used so that GRMON doesn't overwrite registers and memory, the instruction trace can be viewed by typing 'inst', the back trace can be helpful but it requires that the symbols are loaded with 'sym bootrom' for C function names to appear.
5. Interface

This section describes some of the LEON specific functions exported by the BSPs. The intention is not to document all functions. Many functions are documented in BSP generic documents provided by WindRiver.

5.1. Function interface

5.1.1. sysOut* and sysIn*

The sysIn and sysOut functions can be used to load and store a 8-bit, 16-bit or 32-bit data to and from a physical address (the address is not translated), the loads and stores are guaranteed to occur in order. However, the LEON Level-1 cache is write-through meaning all stores will be written directly no matter (no cache effects as on some other platforms). sysIn* functions always skip cache by using the LDA instruction forcing a cache miss, this can be convenient sometimes when a core is doing DMA and the system does not have snooping, as a cache flush can be avoided.

For a non-MMU system all addresses are physical, on a MMU system the address will not be translated into a physical and the permission bits for that page will not have an effect, it will not cause a MMU trap when accessing a invalid page. For more information about mapping see Section 3.2.
6. Compilers

The board support packages support both the DIAB and the GNU GCC compiler.

For information about the compilers please see the LEON VxWorks compiler manual.
7. Drivers

For information about a specific driver or hardware please see the LEON VxWorks Driver manual.
8. Support

For Support, contact the Aeroflex Gaisler support team at support@gaisler.com.