U-Boot SPARC/LEON Port

SPARC/LEON U-boot specific documentation

Written by Daniel Hellström

U-BOOT-SPARC
Version 1.0.0
April 2009

Kungsgatan 12
411 19 Göteborg
Sweden

tel +46 31 7758650
fax +46 31 421407

www.aeroflex.com/gaisler

Table of Contents

1 INTRODUCTION

This document is intended as an aid getting started developing with SPARC/LEON based platforms using U-Boot. U-Boot is an universal boot loader developed by DENX Software Engineering (www.denx.de), it is able to boot SPARC Linux, RTEMS and VxWorks.

1.1 Hardware support

The SPARC/LEON U-boot port supports the hardware listed below, hardware not listed below may still be used by the booted operating system.

- LEON3 and LEON3-FT
- TSIM for LEON3
- MMU and non-MMU systems
- FPU, non-FPU, hardware MUL/DIV and software MUL/DIV support
- Interrupt controller
- APBUART UART console/terminal driver
- Timer unit
- 10/100 Ethernet networking (GRETH, LAN91C111)
- 10/100/1000 Ethernet networking (GRETH)
- USB 1.1

1.2 Software support

The SPARC U-Boot port has support for the operating systems listed below.

- VxWorks 6.5
- Linux 2.6.21.1
- RTEMS 4.8

1.3 Not supported

Below is a list of features currently not supported by the U-Boot SPARC port.

- PCI
- USB 2.0
- LEON2, AT697 etc.

1.4 Tool chain

The tool chain used to compile U-Boot for SPARC/LEON is the BCC (sparc-elf-) tool chain freely available from http://www.gaisler.com.

1.5 Sources

The U-Boot sources are freely available under the GPL License at www.denx.de. The SPARC port is maintained by Gaisler and available in the SPARC U-Boot GIT repository, git:// git.denx.de/u-boot-sparc.git. The repository contains the latest development work of the U-Boot SPARC port. It can be obtained by executing,

```
$ git clone git://git.denx.de/u-boot-sparc.git u-boot-sparc
```

The SPARC port is included in the standard U-Boot distribution, it can be downloaded from ftp.denx.de/pub/u-boot. The latest development work of U-Boot can be obtained from the main GIT repository,

```
$ git clone git://git.denx.de/u-boot.git u-boot
```

1.6 Documentation

U-Boot documentation can be found at www.denx.de.

1.7 Support

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

2 OVERVIEW

The U-Boot is a boot loader which loads an operating system kernel and optionally file systems into main memory and boot the kernel. The Kernel command line is passed to the kernel depending on the operating system, it can be set using the bootargs parameter in U-Boot.

U-Boot is stored at address zero where the SPARC processor starts from, it initializes the CPU and memory controller while running in FLASH. Once the memory has been set up U-Boot relocates it self up to main memory and jumps there.

Kernels and file systems can be downloaded from a network, serial line or USB FAT file system and put into main memory. The downloaded files can be used directly to boot or they can be stored into FLASH on a board. U-boot is controlled using a scriptable command line interpreter, it can be set up to automatically on boot download a kernel and boot it.

3 CONFIGURING U-BOOT

After downloading the U-Boot sources as described in 1.5 the U-Boot sources needs to be configured for a specific target board. There are a couple of LEON boards and GRLIB template designs that the U-Boot supports without needing any configuration. The easiest approach is to change the configuration of an existing target, the most similar to the target board.

Each board support package (BSP) has it's own configuration file located in include/configs, for example gr_cpci_ut699.h for the GR-CPCI-UT699 board. U-Boot is configured editing the C-header configuration file. U-Boot is configured from the command line specifying BSP what to build, the BSPs are available under *board/gaisler*.

```
$ make distclean              # Clean from previous config
$ make gr_cpci_ut699_config   # Select GR-CPCI-UT699 BSP
```

The boot loader configuration can be thought of as two different parts, one GRLIB/LEON board specific and one common U-Boot configuration. The common U-Boot parameters are not discussed here.

The GRLIB/LEON specific configuration options are collected in the bottom of the C-header configuration file.

3.1 System frequency

The system frequency is configured using CONFIG_SYS_CLK_FREQ.

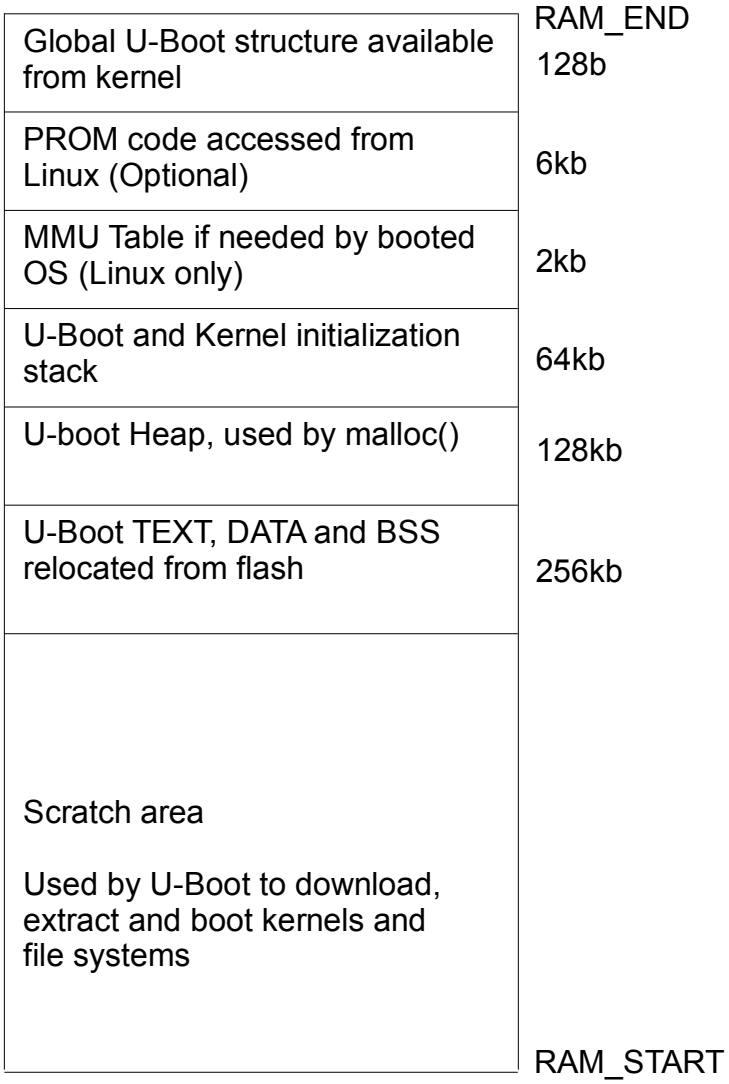
The system frequency, given in Hertz, must be configured in order for U-Boot to set up timers and UART baud rates. The system frequency is also reported to the booted operating system using different methods.

3.2 Memory configuration

Memory can be configured in different ways depending on what the board has support for. Normally one select either to use SDRAM or SRAM accessible from 0x4000000, but U-Boot can also be configured using two memory areas SDRAM at 0x60000000 and SRAM at 0x40000000. The layout is configured using CONFIG_LEON_RAM_SELECT.

The SRAM location and size is configured using the CONFIG_SYS_SRAM_BASE and CONFIG_SYS_SRAM_SIZE, the same goes for the SDRAM location and size CONFIG_SYS_SDRAM_BASE and CONFIG_SYS_SRAM_SIZE. Depending on the main memory layout one of the two areas will be used to execute U-Boot from, it is selected with the CONFIG_SYS_RAM parameter.

The memory requirement of U-Boot may vary depending on the configuration, for example adding the network stack may result in more calls to malloc(). Heap and stack sizes may be changed from the configuration file, an example main memory layout of a SPARC/LEON board is shown in the figure below.



Drawing 1: U-Boot main memory layout

Independent of how the memory layout has been configured the memory controller must be configured for U-Boot to access memory, FLASH and I/O bus. The memory controller may be different for different GRLIB designs and boards. Currently five memory controllers are supported as listed below, if a memory controller is found in Plug & Play is is set up using the configuration parameters below. Multiple memory controllers are possible, if no one is found U-Boot hangs in an infinite loop.

Memory controller	Parameters	Description
MCTRL	GRLIB_MCFG1	MCFG1 register content, bits 8 and 9 are ored with the reset value.
	GRLIB_MCFG2	MCFG2 register content
	GRLIB_MCFG3	MCFG3 register content
FTMCTRL	GRLIB_FT_MEMCFG1	MCFG1 register content, bits 8 and 9 are ored with the reset value.
	GRLIB_FT_MEMCFG2	MCFG2 register content
	GRLIB_FT_MEMCFG3	MCFG3 register content
SDCTRL	GRLIB_SDRAM	SDCFG register content
DDRSPA	GRLIB_DDR_CFG	SDCTRL register content
DDR2SPA	GRLIB_DDR2_CFG1	DDR2CFG1 register content
	GRLIB_DDR2_CFG3	DDR2CFG3 register content

Table 3.1: Memory controller options

3.3 LEON-FT options

The CPU registers and FPU registers if available are always zeroed by U-Boot during boot, zeroing the memory is however optional. Clearing all memory can take quite some time depending on the size of the memory, but may be needed for Fault Tolerant systems. The memory is washed directly after the memory controller has been set up.

Up to two memory areas can be washed by defining CONFIG_SYS_FT_WASH_MEM and CONFIG_SYS_FT_WASH_MEM2. The CONFIG_SYS_FT_WASH_MEM_BASE and CONFIG_SYS_FT_WASH_MEM_SIZE parameters define the start and end of the memory region being cleared.

3.4 Cache options

The cache control register written by using ASI=2, can be configured using the CONFIG_SYS_CACHE_CTRL_REG parameter. The default is to enable Snooping, Instruction burst fetch, data and instruction cache.

3.5 Networking

Networking in U-Boot can be configured using either GRETH or SMC91C111 chip. Interrupt is not used by the network drivers.

One must define CONFIG_CMD_NET to include network support.

The Ethernet MAC address must be configured using CONFIG_ETHADDR, the MAC address will become available from an environment variable read by the network driver.

Files can be downloaded using the *tftpboot* command from the U-Boot command line.

3.5.1 GRETH

The GRETH driver supports the 10/100 and the 10/100/1000 MAC. The parameters listed below must be defined in order to use the GRETH driver.

Parameter	Description
CONFIG_NET_MULTI	Needed by GRETH driver
CONFIG_GRETH	Include GRETH network driver
GRETH_HWADDR_[0..5]	Set the default MAC address

Table 3.2: GRETH network driver set up

3.5.2 SMC91C111

The SMC91C111 driver supports 10/100 networking. The parameters listed below must be defined in order to use the SMC91C111 network driver on SPARC/LEON. The SMC91C111 is accessed over the I/O bus on the memory controller, this means that the I/O bus must be configured correctly by the memory controller settings, see section 3.2.

Parameter	Description
CONFIG_DRIVER_SMC91111	Include SMC91C111 driver
CONFIG_SMC91111_BASE	Address of chip, normally 0x20000300
CONFIG_SMC_USE_32_BIT	32-bit I/O bus
CONFIG_SMC_91111_EXT_PHY	External PHY

Table 3.3: SMC91C111 network driver set up

3.6 USB

USB 1.1 is supported by LEON3, the GRUSB USB Host core must be available in the system. The hardware supports UHCI 1.1. Define CONFIG_USB_UHCI to include support for the GRUSB. An example USB configuration is listed below. The USB support in U-Boot can be used to download kernels and file system from a USB mass storage device into main memory, U-Boot can read files from FAT and EXT2 filesystems.

```
#define CONFIG_USB_UHCI
#define CONFIG_CMD_FAT
#define CONFIG_CMD_EXT2
#define CONFIG_CMD_USB
#define CONFIG_USB_STORAGE
/* Enable needed helper functions */
#define CONFIG_SYS_DEVICE_DEREGISTER

#define CONFIG_USB_CLOCK      0x0001BBBB
#define CONFIG_USB_CONFIG     0x00005000
```

3.7 VxWorks support

VxWorks can be loaded by U-Boot using the ELF file format. CONFIG_CMD_ELF has to be defined in the configuration file to enable ELF support. The VxWorks ELF image is always

copied to the base address of the main memory, for example 0x40000000. The Kernel command line is located with an offset of 0x4100 from the base address. The VxWorks kernel command line location can be set by the CONFIG_SYS_VXWORKS_BOOT_ADDR parameter.

The VxWorks kernel command line is can either be set by setting the *bootargs* environment variable, or if the *bootargs* variable is undefined U-Boot tries to construct it by using the U-Boot network configuration currently available. Additional parameters can be added to the built kernel command line using the CONFIG_SYS_VXWORKS_ADD_PARAMS parameter.

CONFIG_SYS_VXWORKS_STARTTYPE can be defined to enable VxWorks start type to be passed to the kernel. The start type argument is given to U-Boot by VxWorks during warm reboot, or set to zero after a reset.

4 BUILDING U-BOOT

Building U-Boot is done after configuring the U-Boot for a specific BSP. In the example below the GR-CPCI-UT699 BSP is used when building U-Boot.

```
$ make distclean              # Clean from previous config
$ make gr_cpci_ut699_config   # Select GR-CPCI-UT699 BSP
$ make                        # Build U-Boot
```

The resulting file is named *u-boot*. It is an ELF-file containing the boot loader.

5 INSTALLING U-BOOT ON TARGET BOARD

After building U-Boot the ELF-image u-boot is found in the root of the U-Boot sources. U-Boot can be downloaded to the target board's FLASH with GRMON, as in the example below,

```
$ grmon -jtag
grmon> flash unlock all
grmon> flash er 0 0x40000
grmon> flash lo u-boot
grmon> run 0
```

U-Boot can be controlled using a serial terminal, the default terminal settings are 38400baud, 8N1.

Below is an execution example booting VxWorks from FLASH and from Network.

```
U-Boot 2009.03-00151-g2344bb8-dirty (Apr 06 2009 -
09:57:54)GAISLER LEON3 GR-CPCI-UT699

CPU: LEON3
Board: GR-CPCI-UT699
FLASH: 32 MB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   GRETH 10/100

Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  0
=> print
bootcmd=run flash_self
bootdelay=5
baudrate=38400
ethaddr=00:00:7a:cc:00:14
ipaddr=192.168.0.206
serverip=192.168.0.47
preboot=echo;echo Type "run flash_nfs" to mount root
filesystem over NFS;echo
rootpath=/export/rootfs
gatewayip=192.168.0.1
netmask=255.255.255.0
hostname=ut699
bootfile=/uImage
netdev=eth0
nfsargs=setenv bootargs console=ttyS0,38400 root=/dev/nfs rw
nfsroot=${serverip}:${rootpath}
ramargs=setenv bootargs console=ttyS0,${baudrate}
root=/dev/ram rw addip=setenv bootargs ${bootargs} ip=$
{ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:$
{netdev}:off pa1
flash_nfs=run nfsargs addip;bootm ${kernel_addr}
flash_self=run ramargs addip;bootm ${kernel_addr} $
{ramdisk_addr}
getkernel=tftpboot $(scratch) $(bootfile)
bootargs=console=ttyS0,38400 root=/dev/nfs rw
nfsroot=192.168.0.20:/export/rootfs
ip=192.168.0.206:192.168.0.200
getkernelVx=tftpboot $(scratch) $(bootfile)Vx
bootargsVx=setenv bootargs greth(0,0)gx620:vxWorks e=$
{ipaddr} h=${serverip} g=${gatewayip} u=anonymous pw=usert
net_nfs=tftp 40000000 ${bootfile};run nfsargs addip;bootm
scratch=41000000
stdin=serial
stdout=serial
```

```
stderr=serial
ethact=GRETH 10/100

Environment size: 1206/32764 bytes
=> iminfo 200000

## Checking Image at 00200000 ...
   Legacy image found
   Image Name:   SPARC/LEON VxWorks-6.5
   Image Type:   SPARC VxWorks Kernel Image (gzip compressed)
   Data Size:    647260 Bytes = 632.1 kB
   Load Address: 40800000
   Entry Point:  40800000
   Verifying Checksum ... OK
=> run bootargsVx
=> print
bootcmd=run flash_self
bootdelay=5
baudrate=38400
ethaddr=00:00:7a:cc:00:14
ipaddr=192.168.0.206
serverip=192.168.0.47
preboot=echo;echo Type "run flash_nfs" to mount root
filesystem over NFS;echo
rootpath=/export/rootfs
gatewayip=192.168.0.1
netmask=255.255.255.0
hostname=ut699
bootfile=/uImage
netdev=eth0
nfsargs=setenv bootargs console=ttyS0,38400 root=/dev/nfs rw
nfsroot=${serverip}:${rootpath}
ramargs=setenv bootargs console=ttyS0,${baudrate}
root=/dev/ram rw
addip=setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:$
{gatewayip}:${netmask}:${hostname}:${netdev}:off pa1
flash_nfs=run nfsargs addip;bootm ${kernel_addr}
flash_self=run ramargs addip;bootm ${kernel_addr} $
{ramdisk_addr}
getkernel=tftpboot $(scratch) $(bootfile)
getkernelVx=tftpboot $(scratch) $(bootfile)Vx
bootargsVx=setenv bootargs greth(0,0)gx620:vxWorks e=$
{ipaddr} h=${serverip} g=${gatewayip} u=anonymous pw=usert
net_nfs=tftp 40000000 ${bootfile};run nfsargs addip;bootm
scratch=41000000
stdin=serial
stdout=serial
stderr=serial
ethact=GRETH 10/100
bootargs=greth(0,0)gx620:vxWorks e=192.168.0.206
h=192.168.0.47 g=192.168.0.1 u=anonymous pw=user f=0x04
```

```
tn=vxTt

Environment size: 1173/32764 bytes
=> bootm 200000
## Booting kernel from Legacy Image at 00200000 ...
   Image Name:   SPARC/LEON VxWorks-6.5
   Image Type:   SPARC VxWorks Kernel Image (gzip compressed)
   Data Size:    647260 Bytes = 632.1 kB
   Load Address: 40800000
   Entry Point:  40800000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
## Ethernet MAC address not copied to NV RAM
Loading .text @ 0x40003000 (1432192 bytes)
Loading .init$00 @ 0x40160a80 (12 bytes)
Loading .init$99 @ 0x40160a8c (8 bytes)
Loading .fini$00 @ 0x40160a94 (12 bytes)
Loading .fini$99 @ 0x40160aa0 (8 bytes)
Loading .data @ 0x40160ac0 (53056 bytes)
Clearing .bss @ 0x4016da00 (129728 bytes)
## Using bootline (@ 0x40004100): greth(0,0)gx620:vxWorks
e=192.168.0.206 h=192.168.0.47 g=192.168.0.1 u=anonymt
## Starting vxWorks at 0x40003000 ...
amba_init: irqctrl: 0x80000200
amba_init: gptimer: 0x80000300
uart: Note, no hardware flow ctrl enabled, disable it in
minicom as well
10/100 GRETH Ethermac at [0x80000e00] irq 14. Running 100
Mbps full duplex
PHY: Vendor 0x4de    Device 0xe    Revision 2
Failed.
Error opening gx620:vxWorks.sym: status = 0x2b0001


                VxWorks

Copyright 1984-2007  Wind River Systems, Inc.

             CPU: Gaisler SPARC/LEON3 non-MMU BSP
    Runtime Name: VxWorks
 Runtime Version: 6.5
     BSP version: 2.0/9
         Created: Apr  2 2009, 07:07:53
ED&R Policy Mode: Deployed
   WDB Comm Type: WDB_COMM_END
             WDB: Ready.

-> cmd
[vxWorks *]# version
VxWorks (for Gaisler SPARC/LEON3 non-MMU BSP) version 6.5.
Kernel: WIND version 2.10.
```

```
Made on Apr  2 2009, 07:07:53.
Boot line:
greth(0,0)gx620:vxWorks e=192.168.0.206 h=192.168.0.47
g=192.168.0.1 u=anonymous pw=user f=0x04 tn=vxTarget
[vxWorks *]# reboot


U-Boot 2009.03-00151-g2344bb8-dirty (Apr 06 2009 -
09:57:54)GAISLER LEON3 GR-CPCI-UT699

CPU: LEON3
Board: GR-CPCI-UT699
FLASH: 32 MB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   GRETH 10/100

Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  0
=>run bootargsVx
=> run getkernelVx
Using GRETH 10/100 device
TFTP from server 192.168.0.47; our IP address is 192.168.0.206
Filename '/uImageVx'.
Load address: 0x41000000
Loading:
#################################################################
#################################################################
#####
done
Bytes transferred = 647324 (9e09c hex)
=> iminfo

## Checking Image at 41000000 ...
   Legacy image found
   Image Name:   SPARC/LEON VxWorks-6.5
   Image Type:   SPARC VxWorks Kernel Image (gzip compressed)
   Data Size:    647260 Bytes = 632.1 kB
   Load Address: 40800000
   Entry Point:  40800000
   Verifying Checksum ... OK
=> bootm
## Booting kernel from Legacy Image at 00200000 ...
   Image Name:   SPARC/LEON VxWorks-6.5
   Image Type:   SPARC VxWorks Kernel Image (gzip compressed)
   Data Size:    647260 Bytes = 632.1 kB
   Load Address: 40800000
```

```
   Entry Point:  40800000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
## Ethernet MAC address not copied to NV RAM
Loading .text @ 0x40003000 (1432192 bytes)
Loading .init$00 @ 0x40160a80 (12 bytes)
Loading .init$99 @ 0x40160a8c (8 bytes)
Loading .fini$00 @ 0x40160a94 (12 bytes)
Loading .fini$99 @ 0x40160aa0 (8 bytes)
Loading .data @ 0x40160ac0 (53056 bytes)
Clearing .bss @ 0x4016da00 (129728 bytes)
## Using bootline (@ 0x40004100): greth(0,0)gx620:vxWorks
e=192.168.0.206 h=192.168.0.47 g=192.168.0.1 u=anonymt
## Starting vxWorks at 0x40003000 ...
amba_init: irqctrl: 0x80000200
amba_init: gptimer: 0x80000300
uart: Note, no hardware flow ctrl enabled, disable it in
minicom as well
10/100 GRETH Ethermac at [0x80000e00] irq 14. Running 100
Mbps full duplex
PHY: Vendor 0x4de    Device 0xe    Revision 2
Failed.
Error opening gx620:vxWorks.sym: status = 0x36


                VxWorks

Copyright 1984-2007  Wind River Systems, Inc.

             CPU: Gaisler SPARC/LEON3 non-MMU BSP
    Runtime Name: VxWorks
 Runtime Version: 6.5
     BSP version: 2.0/9
         Created: Apr  2 2009, 07:07:53
ED&R Policy Mode: Deployed
   WDB Comm Type: WDB_COMM_END
             WDB: Ready.

->
```

6 CREATING U-BOOT KERNEL IMAGES

U-Boot reads kernel images of different formats, independent of format all kernels must be encapsulated in a so called U-Boot image. An U-Boot image is created using the *mkimage* application that comes with U-Boot , in the *tools* directory.

6.1 VxWorks

VxWorks images can be booted using either the U-Boot command *vxboot* or *bootm*. Booting a VxWorks image is divided into five steps,

1. Creating the VxWorks Image
2. Downloading the kernel to FLASH or RAM, using GRMON or U-Boot networking
3. Extracting the compressed content of the boot able image in to a readable VxWorks ELF-image
4. Loading the ELF image to 0x40000000
5. Setting up kernel command line, start type and jumping to the ELF-entry point

The address to where the ELF image is extracted depends on the U-Boot image creation in step 3. This address must not be to the same address as the ELF-image is to be loaded, or else the ELF image will be overwritten by the content of the ELF image.

Below is an example script creating an U-Boot image including a GZIP compressed stripped VxWorks ELF image. The ELF image will be extracted to 0x40800000.

```
ENTRY=0x40800000
LOAD_ADR=0x40800000
FILE=VxWorks

rm -f $FILE.*
sparc-elf-strip -s -o $FILE.stripped $FILE
gzip -9 $FILE.stripped
tools/mkimage -A sparc -O vxworks -T kernel -C gzip -a $
{LOAD_ADR} -e ${ENTRY} -n 'SPARC/LEON VxWorks-6.5' -d
$FILE.stripped.gz $FILE.uImg
```

The image can be downloaded using the networking facilities in U-Boot and booted by the command line "bootm ADR" where ADR is the address in hex the image was downloaded to.

7 DEBUGGING U-BOOT

7.1 Using GRMON

Since U-Boot relocate itself software breakpoints can not be used in all situations since the relocation procedure will overwrite the breakpoints with instructions. Instead hardware breakpoints may be used, in GRMON the hbreak command can be used to set hardware breakpoints.

7.2 Running U-Boot in RAM

When debugging U-Boot it might be convenient to start U-Boot from RAM instead of FLASH. The process is still the same, U-Boot configures hardware and relocates itself from RAM to Top of RAM. The load address can be changed by setting TEXT_BASE to base of RAM, for example

```
TEXT_BASE = 0x40000000
```

The configuration parameter is found in *board/gaisler/BSP/config.mk*.

7.3 Relocating symbols

After U-Boot has initialized memory controllers it relocates U-Boot to RAM, this makes the symbols in the u-boot to become invalid. To avoid this problem the symbols in the resulting u-boot file can be relocated, it can be done as in the example below. The main memory layout must taken into account when calculating the U-Boot RAM base address, it can also be printed on the U-Boot console by executing *bdinfo*.

```
$ IN=u-boot
$ OUT=u-boot-reloc
$ ADR=0x47f8e000  # Running from FLASH
#$ ADR=0x03f8e000 # Running from RAM
$ sparc-elf-objdcopy \
  --change-section-address .text+$ADR \
  --change-section-address .u_boot_cmd+$ADR \
  --change-section-address .data+$ADR \
  --change-section-address .got+$ADR \
  --change-section-address .bss+$ADR \
  $IN $OUT
$
```

8 SUPPORT

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