Technical Report

UCAM-CL-TR-869
ISSN 1476-2986

Number 869

UNIVERSITY OF
CAMBRIDGE

Computer Laboratory

Bluespec Extensible RISC Implementation: BERI Software reference

Robert N. M. Watson, David Chisnall,
Brooks Davis, Wojciech Koszek,
Simon W. Moore, Steven J. Murdoch,
Peter G. Neumann, Jonathan Woodruff

April 2015

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Approved for public release; distribution is unlimited.
Sponsored by the Defense Advanced Research Projects
Agency (DARPA) and the Air Force Research Laboratory
(AFRL), under contract FA8750-10-C-0237 ("CTSRD") as
part of the DARPA CRASH research program. The views,
opinions, and/or findings contained in this report are those of
the authors and should not be interpreted as representing the
official views or policies, either expressed or implied, of the
Department of Defense or the U.S. Government. Portions of
this work were sponsored by Google, Inc.

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

The *BERI Software Reference* documents how to build and use the FreeBSD operating system on the Bluespec Extensible RISC Implementation (BERI) developed by SRI International and the University of Cambridge. The reference is targeted at hardware and software programmers who will work with BERI or BERI-derived systems.

Acknowledgments

The authors of this report thank other current and past members of the CTSRD team, and our past and current research collaborators at SRI and Cambridge:

Ross J. Anderson	Jonathan Anderson	Ruslan Bukin	Gregory Chadwick
Nirav Dave	Khilan Gudka	Jong Hun Han	Alex Horsman
Alexandre Joannou	Asif Khan	Myron King	Ben Laurie
Patrick Lincoln	Anil Madhavapeddy	Ilias Marinos	A. Theodore Markettos
Ed Maste	Andrew Moore	Will Morland	Alan Mujumdar
Prashanth Mundkur	Robert Norton	Philip Paeps	Michael Roe
Colin Rothwell	John Rushby	Hassen Saidi	Hans Petter Selasky
Muhammad Shahbaz	Stacey Son	Richard Uhler	Philip Withnall
Bjoern A. Zeeb			

The CTSRD team wishes to thank its external oversight group for significant support and contributions:

Lee Badger	Simon Cooper	Rance DeLong	Jeremy Epstein
Virgil Gligor	Li Gong	Mike Gordon	Steven Hand
Andrew Herbert	Warren A. Hunt Jr.	Doug Maughan	Greg Morrisett
Brian Randell	Kenneth F. Shotting	Joe Stoy	Tom Van Vleck
Samuel M. Weber			

Finally, we are grateful to Howie Shrobe, MIT professor and past DARPA CRASH program manager, who has offered both technical insight and support throughout this work. We are also grateful to Robert Laddaga, who has succeeded Howie in overseeing the CRASH program.

Contents

Chapter 1

Introduction

This is the *Software Reference* for the Bluespec Extensible RISC Implementation (BERI) prototype. The *Software Reference* describes the software development environment on the BERI processor – especially, as relates using the FreeBSD operating system on FPGA-synthesized BERI systems. The reference is intended to address the needs of hardware and software developers who are prototyping new hardware features, bringing up operating systems, language runtimes, and compilers on BERI, rather than literal end users. It complements the *BERI Hardware Reference*, which describes the BERI physical platform, the *CHERI Architecture Document*, which describes our CHERI ISA extensions, and the *CHERI Programmer's Guide*, which describes our software extensions for CHERI.

1.1 Bluespec Extensible RISC Implementation (BERI)

The Bluespec Extensible RISC Implementation (BERI) is a platform for performing research into the hardware-software interface that has been developed as part of the CTSRD project at SRI International and the University of Cambridge. It consists of a 64-bit FPGA soft-core processor implemented in Bluespec System Verilog and a complete software stack based on FreeBSD, Clang/LLVM, and a range of popular open-source software products. BERI implements a roughly 1994-vintage version of the 64-bit MIPS ISA with FPU and system coprocessor sufficient to support a full operating-system implementation. It also implements research extensions such as the CHERI ISA, which supports fine-grained memory protection and scalable compartmentalization within conventional address spaces. Wherever possible, BERI makes use of BSD- and Apache-licensed software to maximize opportunities for technology transition.

1.2 FreeBSD

FreeBSD is an open-source UNIX operating system originating from the Berkeley Software Distribution in the 1980s. Released under the liberal BSD open-source license, FreeBSD is widely used in service provider environments (e.g., Yahoo!, Verio, ISC, Netflix) and as a foundation for commercial appliance and embedded products (e.g., NetApp, Juniper, Cisco, EMC, Apple). We have adapted FreeBSD to run on BERI, which includes a platform support package and a set of device drivers for common Altera and Terasic peripherals. As part of the BERI

effort, we have invested significant effort in improving upstream components such as LLVM and LLDB to work well with the 64-bit MIPS ISA.

FreeBSD can be cross-compiled from 32-bit and 64-bit x86 workstations and servers running FreeBSD (or from a VM running FreeBSD). We have also adapted the FreeBSD third-party package build system ('ports') to support cross-compilation, making tens of thousands of open-source applications (e.g., Apache) available on BERI. FreeBSD is of particular interest to teaching and research in the hardware-software interface due to tight integration with the Clang/LLVM compiler suite.

1.3 Getting BERI

We distribute the BERI prototype and software stack as open source via the BERI website:

`http://www.beri-cpu.org/`

1.4 Licensing

The BERI hardware design, simulated peripherals, and software tools are available under the BERI Hardware-Software License, a lightly modified version of the Apache Software License that takes into account hardware requirements.

We have released our extensions to the FreeBSD operating system to support BERI under a BSD license; initial support for BERI was included in FreeBSD 10.0, but further features will appear in FreeBSD 10.1. We have also released versions of FreeBSD and Clang/LLVM that support the CHERI ISA under a BSD license; these are distributed via GitHub.

We welcome contributions to the BERI project; however, we are only able to accept non-trivial changes when an individual or corporate contribution agreement has been signed. The BERI hardware-software license and contribution agreement may be found at:

`http://www.beri-open-systems.org/`

1.5 Version History

Portions of this document were previously made available as part of the *CHERI User's Guide*.

1.0 An initial version of the *CHERI User's Guide* documented the implementation status of the CHERI prototype, including the CHERI ISA and processor implementation, as well as user information on how to build, simulate, debug, test, and synthesize the prototype.

1.1 Minor refinements were made to the text and presentation of the document, with incremental updates to its descriptions of the SRI/Cambridge development and testing environments.

1.2 This version of the *CHERI User's Guide* followed an initial demonstration of CHERI synthesized for the Terasic tPad FPGA platform. The Guide contained significant updates on the usability of CHERI features, the build process, and debugging features such as CHERI's debug unit. A chapter was added on Deimos, a demonstration microkernel for the CHERI architecture.

1.3 The document was restructured into hardware prototype and software reference material. Information on the status of MIPS ISA implementation was updated and expanded, especially with

respect to the MMU. Build dependencies were updated, as was information on the CHERI simulation environment. The distinction between BERI and CHERI was discussed in detail. The Altera development environment was described in its own chapter. A new chapter was added that detailed bus and device configuration and use of the Terasic tPad and DE4 boards, including the Terasic/Cambridge MTL touch screen display. New chapters were added on building and using CheriBSD, as well as a chapter on FreeBSD device drivers on BERI/CHERI. A new chapter was added on cross-building and using the CHERI-modified Clang/LLVM suite, including C-language extensions for capabilities.

1.4 This version introduced improved Altera build and Bluespec simulation instructions. A number of additional C-language extensions that can be mapped into capability protections were introduced. FreeBSD build instructions were updated for changes to the FreeBSD cross-build system. Information on the CHERI2 prototype was added.

1.5 In this version of the *CHERI User's Guide*, several chapters describe the CHERI hardware prototype have been moved into a separate document, the *CHERI Platform Reference Manual*, leaving the User's Guide focused on software-facing activities.

1.6 This version updated the *CHERI User's Guide* for changes in the CheriBSD build including support for the CFI driver, incorporation of Subversion into the FreeBSD base tree, and non-root cross builds. It also added information on the `quartus_pgm` command, and made a number of minor clarifications and corrections throughout the document.

1.7 In this revision, the *BERI Software Reference* became an independent document from the *CHERI User's Guide*. This version was updated for the complete merge of FreeBSD/BERI to the FreeBSD Subversion repository, and migration of CheriBSD to GitHub. It reflects the change from `cherictl` to `berictl` and a number of enhancements to `berictl` that avoid the need to manually invoke Altera's underlying tools for FPGA programming or console access. The `isf` driver has been replaced with use of the stock FreeBSD `cfi` driver. We no longer recommend explicitly building the cross-toolchain; instead, rely on `world`.

1.8 - UCAM-CL-TR-853 This version of the *BERI Software Reference* was made available as a University of Cambridge Technical Report. Information was updated to reflect open sourcing of BERI/CHERI and its software stack.

1.9 - UCAM-CL-TR-869 This version of the *BERI Software Reference* was made available as a University of Cambridge Technical Report. It has been timed to coincide with the second open-source release of the BERI processor implementations. FreeBSD device-driver documentation has now been upstreamed to the FreeBSD source tree via a series of manual pages, and so has been removed from this reference. The creation of memory root filesystems is now documented in greater detail.

1.6 Document Structure

This document is an introduction and user manual for Version 1 of the Bluespec Extensible RISC Implementation (BERI) CPU prototype:

Chapter 2 describes how to build the FreeBSD/BERI port from source. You do not need to do this if using prebuilt kernels.

Chapter 3 describes how to use FreeBSD/BERI.

Chapter **??** provides additional reference material for device driver configuration and use under FreeBSD/BERI.

Chapter 2

Building FreeBSD/BERI

FreeBSD/BERI is an adaptation of the open-source FreeBSD operating system to run on the Bluespec Extensible RISC Implementation (BERI). This support has been 'upstreamed' to the mainstream FreeBSD distribution, and appears from version FreeBSD 10.0 onwards. As we are actively continuing development of FreeBSD/BERI, we have extended it to support evolving features in BERI itself, including device drivers for new hardware peripherals, and kernel support for new CPU features. FreeBSD/BERI can be cross-compiled from 32-bit or 64-bit FreeBSD/x86 running FreeBSD 10.0. CheriBSD, a set of extensions to FreeBSD/BERI to support the CHERI capability coprocessor, are described in a separate document, the *CHERI Programmer's Guide*.

2.1 Obtaining FreeBSD/BERI Source Code

FreeBSD/BERI has been merged to the FreeBSD source tree as released in FreeBSD 10.0. However, due to post-FreeBSD 10.0 enhancements, we recommend using FreeBSD 10-STABLE, which can be tracked in the following Subversion branch:

`http://svn.freebsd.org/base/stable/10`

For closer (and likely faster) access to the repository see the list of mirrors found at:

`http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/svn-mirrors.html`

With some caution, the FreeBSD development head (11-CURRENT) might also be used in order to get the very latest BERI features. (For example, at the time of writing, BERI boot-loader support is only present in that branch.) However, as 11-CURRENT includes many other experimental in-development OS features, this is not recommended unless you are able to closely track FreeBSD development mailing lists to be aware of evolving sources of instability:

`http://svn.freebsd.org/base/head`

2.2 About FreeBSD/BERI

The FreeBSD/BERI port is adapted from the FreeBSD/MALTA 64-bit MIPS port, which offers the closest match in terms of ISA. BERI- related kernel files may be found in directories listed in Table 2.1. Wherever possible, FreeBSD/BERI reuses generic MIPS platform code, and is successful in almost all cases. BERI uses flattened device tree (FDT); currently, `DTS` files describing BERI hardware are stored

Filename	Description
sys/mips/beri/	BERI-specific processor/platform code.
sys/boot/fdt/dts/mips	Home of BERI flattened device tree (FDT) description files.
sys/dev/altera/atse	Altera Triple-Speed Ethernet MAC.
sys/dev/altera/avgen	Avalon "generic" driver to export I/O address ranges to userspace.
sys/dev/altera/jtag_uart	Altera JTAG UART device driver.
sys/dev/altera/sdcard	Altera University Program SD Card IP core device driver.
sys/dev/terasic/de4led	Terasic DE4 LED array device driver.
sys/dev/terasic/mtl	Terasic Multitouch LCD device driver.

Table 2.1: FreeBSD/BERI directories in src/sys/

in the FreeBSD source tree, but we hope to embed them in ROMs in BERI bitfiles in the future. Table 2.2 lists BERI-specific files in the common MIPS configuration directory.

2.3 Building FreeBSD/BERI

The following sections describe how to build a FreeBSD/BERI system. Examples assume the Cambridge zenith development environment, a FreeBSD 10.0 x86/64 server. With appropriate pathname and username substitutions, they should work on other FreeBSD 10 build hosts. The FreeBSD userspace build will automatically build suitable cross-compilers and tools. Once userspace has been built, it must be installed to a suitable directory tree from which disk images can be created. If you wish to build a kernel that includes a memory root filesystem, userspace must be built, installed to a temporary location, and a memory file system image created, before the kernel can be built.

Where appropriate, cheribsd may be substituted for freebsd, and kernel configuration file names changed, to build CheriBSD instead of FreeBSD/BERI. Please consult the *CHERI Programmer's Guide* for instructions on checking out CheriBSD source code.

Note well: The details of the build process are likely to change over time as we merge changes from the upstream FreeBSD tree due to the rapid evolution of MIPS support. Users should take care to ensure that they are using a *BERI Software reference* that is contemporary with their source tree.

2.3.1 Configuring the Build Environment

By default, the FreeBSD build system will use /usr/obj as its scratch area. Instead, create and configure your own per-user scratch space:

```
mkdir -p ${HOME}/obj
export MAKEOBJDIRPREFIX=${HOME}/obj
```

You may wish to modify your .cshrc or .bashrc to automatically configure the MAKEOBJDIRPREFIX variable every time you log in.

Filename	Description
BERI_DE4.hints	Terasic DE4 hardware configuration hints
BERI_SIM.hints	Bluespec simulation hardware configuration hints
BERI_TPAD.hints	Terasic tPad hardware configuration hints
BERI_TEMPLATE	FreeBSD/BERI template configuration entries, included by more specific kernel configuration files
BERI_DE4_BASE	FreeBSD/BERI template configuration entries for DE4 configurations, included by DE4 kernel configuration files
BERI_DE4_MDROOT	FreeBSD/BERI kernel configuration to use a memory root filesystem on the Terasic DE4
BERI_DE4_SDROOT	FreeBSD/BERI kernel configuration to use an SD Card root filesystem on the Terasic DE4
BERI_SIM_BASE	FreeBSD/BERI template configuration entries for DE4 configurations, included by DE4 kernel configuration files
BERI_SIM_MDROOT	FreeBSD/BERI kernel configuration to use a memory root filesystem while in simulation
BERI_SIM_SDROOT	FreeBSD/BERI kernel configuration to use a simulated SD Card root filesystem

Table 2.2: FreeBSD/BERI files in src/sys/mips/conf; note that hints files have been deprecated in favor of FDT DTS files for board configuration.

2.3.2 Cross-Building World

In FreeBSD parlance, "world" refers to all elements of the base operating system other than the kernel – i.e., userspace. This includes system libraries, the toolchain (including the compiler), userland utilities, daemons, and generated configuration files. It excludes third-party software such as Apache, X11, and Chrome. Cross-build a big-endian, 64-bit MIPS world with the following commands:

```
cd ${HOME}/freebsd
make -j 16 TARGET=mips TARGET_ARCH=mips64 DEBUG_FLAGS=-g \
  -DMALLOC_PRODUCTION buildworld
```

Note: the `DEBUG_FLAGS=-g` requests generation of debugging symbols for all userland components.

The `atsectl` utility allows the MAC address to be set in flash when FreeBSD/BERI is run on a Terasic DE4 FPGA board. It is part of the FreeBSD source tree, but not built by default as it is only useful on that platform and often run only once per board. It is stored in the `tools/tools/atsectl` directory and can be built and installed with `world` by adding the following to the `make` command lines:

```
LOCAL_DIRS="tools/tools/atsectl"
```

2.3.3 Cross-Building a Kernel

FreeBSD kernels are compiled in the context of a configuration file. For BERI, we have provided several reference configuration files as described earlier in this chapter. In this section we describe only how to build a kernel without a memory root filesystem. Information on memory root filesystems may be found in Section 2.4.3. The following commands cross-build a BERI kernel:

```
cd ${HOME}/freebsd
make -j 16 TARGET=mips TARGET_ARCH=mips64 buildkernel \
  KERNCONF=BERI_DE4_SDROOT
```

Notice that the kernel configuration used here is `BERI_DE4_SDROOT`; replace this with other configuration file names as required.

BERI uses FDT to describe most aspects of hardware configuration (including bus topology and peripheral attachments). The only significant exception is physical memory configuration, which is passed directly to the kernel by the boot loader; we anticipate that physical memory will also be configured using FDT in the future. For the time being, `DTS` files describing hardware are embedded in the FreeBSD source tree in a manner similar to `hints` files used in earlier iterations of BERI. In the future, these will be embedded in hardware and a pointer to the configuration will be passed to the FreeBSD kernel on boot by the loader.

2.4 Cross-Installing FreeBSD

The install phase of the FreeBSD build process takes generated userspace and kernel binaries from the `MAKEOBJDIRPREFIX` and installs them into a directory tree that can then be converted into a filesystem image. Most `make` targets in this phase make use of the `DESTDIR` variable to determine where files should be installed to. Typically, `DESTDIR` will be set to a dedicated scratch directory such as `${HOME}/freebsd-root`. We advise you to remove the directory between runs to ensure that no artifacts slip from one instance into a later one:

```
rm -Rf ${HOME}/freebsd-root
mkdir -p ${HOME}/freebsd-root
```

2.4.1 Cross-Installing World

This phase consists of two steps: `installworld`, which installs libraries, daemons, command line utilities, and so on; and `distribution`, which creates additional files and directories used in the installed configuration, such as `/etc` and `/var`. The following commands cross-install to `${HOME}/freebsd-root`

```
cd ${HOME}/freebsd
make DESTDIR=${HOME}/freebsd-root \
  TARGET=mips TARGET_ARCH=mips64 -DDB_FROM_SRC -DNO_ROOT \
  installworld distribution
```

To install the software in the `tools/tools/atsectl` directory, the following should be added to the `make` command line:

```
LOCAL_DIRS="tools/tools/atsectl"
```

2.4.2 Cross-Installing Kernels

As with world, kernels can be installed to a target directory tree along with any associated modules. The following commands install the `BERI_DE4_SDROOT` kernel into `${HOME}/freebsd-root`:

```
cd ${HOME}/freebsd
make DESTDIR=${HOME}/freebsd-root \
  TARGET=mips TARGET_ARCH=mips64 -DDB_FROM_SRC -DNO_ROOT \
  KERNCONF=BERI_DE4_SDROOT installkernel
```

2.4.3 Preparing a Memory Root Filesystem

To build the `BERI_DE4_MDROOT` kernel, a memory file system image must first be made available. A demonstration script, `makeroot.sh`, is available via the CTSRD Subversion repository or CHERI distribution; most users will wish to customize the script arguments based on their specific environment. The following command generates a 26-megabyte root filesystem image in `${HOME}/mdroot.img`, and requires that previous steps to install `world` have been completed and assumes access to a checkout of the CTSRD repository in `${HOME}/ctsrd`:

```
cd ${HOME}/ctsrd/cheribsd/trunk/bsdtools
sh makeroot.sh -B big -e extras/sdroot.mtree -s 26112k -f net.files \
   ${HOME}/mdroot.img ${HOME}/freebsd-root
```

You must also customize `BERI_DE4_MDROOT` in order to notify it of the memory location of the root filesystem image. Modify the following section of its configuration file to reflect the size of the generated filesystem:

```
options  MD_ROOT # MD is a potential root device
options  MD_ROOT_SIZE="26112"
options  ROOTDEVNAME=\"ufs:md0\"
```

You may optionally add the following line to include the filesystem when the kernel is built:

```
makeoptions MFS_IMAGE=${HOME}/mdroot.img
```

Alternatively, you can embed the image by running the following command on a kernel that was previouly built without the MFS_IMAGE line:

```
sh ${HOME}/freebsd/sys/tools/embed_mfs.sh kernel.debug \
    ${HOME}/mdroot.img
```

Once the root filesystem image is generated, and the kernel configuration file updated, you can build the kernel:

```
cd ${HOME}/freebsd
make -j 16 TARGET=mips TARGET_ARCH=mips64 buildkernel \
  KERNCONF=BERI_DE4_MDROOT
```

The resulting kernel can be found in a file named kernel in your MAKEOBJDIRPREFIX tree. With suitable substitutions for ${SRCDIR} and ${KERNELNAME}, the following path should point at the generated kernel:

```
${MAKEOBJDIRPREFIX}/mips.mips64/${SRCDIR}/sys/${KERNELNAME}/kernel
```

If you did not add the MFS_IMAGE variable you must then run embed_mfs.sh.

2.5 Preparing a FreeBSD SD Card Image

Build and install world and distribution as described in earlier sections. Then build and install a kernel using the configuration file BERI_DE4_SDROOT. Apply a few tweaks to configuration files, then use the makefs command to generate a UFS image file:

```
sudo su -
echo "/dev/altera_sdcard0 / ufs rw 1 1" > \
  ${HOME}/freebsd-root/etc/fstab
RCCONF=${HOME}/freebsd-root/etc/rc.conf
cat > ${RCCONF} <<EOF
hostname="beri1"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
cron_enable="NO"
tmpmfs="YES"
EOF
METALOG=${HOME}/freebsd-root/METALOG
echo "./etc/fstab type=file uname=root gname=wheel mode=0644" >> \
  ${METALOG}
echo "./etc/rc.conf type=file uname=root gname=wheel mode=0644" >> \
  ${METALOG}
cd ${HOME}/freebsd-root && makefs -B big -s 1886m -D \
  -N ${HOME}/freebsd-root/etc \
  ${HOME}/beribsd-sdcard.img METALOG
```

This command creates a big-endian filesystem of size $1,977,614,336$ bytes – the size of the Terasic 2 GB SD Cards shipped with the tPad and DE4 boards. The resulting beribsd-rootfs.img can then be installed on an SD Card using dd as described in later sections.

If a smaller filesystem is desired (e.g., one that can be more quickly prepared and written to the SD card), then size $1,977,614,336$ bytes (2GiB) can be replaced by $512m$ or the -s argument can be omitted entirely to create a minimally sized root image.

2.6 Automated Builds

The files described in Chapter 3 can be built with the help of the `Makefile` in:

`cheribsd/trunk/bsdtools/`

The template config file in `Makefile.conf.template` can be copied to `Makefile.conf` and customized to your environment. The `worlds` target runs the `buildworld`, `installworld`, and `distribution` for each source tree. The `images` target builds filesystem images from the installed root directories. The `kernels` target builds kernels, including MDROOT kernels with images built by the `images` target. The `sdcard` target builds an image suitable for writing to the SD card. The `flash` target builds flash preparation images for each kernel. Finally, the `dated` target makes dated stamped files of each compressed kernel and image. All the above steps except for running the `dated` target may be accomplished with the default `all` target.

All these targets support the `make` flag `-j`. We strongly encourage passing an appropriate value to `-j`. In addition to standard FreeBSD tools, the Makefile requires installation of the `archives/pxz` port or package.

Chapter 3

Using FreeBSD/BERI

This chapter describes the installation and use of FreeBSD/BERI on the Terasic DE4 FPGA board. We have structured our modifications to FreeBSD into two development branches:

- FreeBSD/BERI is a version of FreeBSD that can run on the BERI hardware-software research platform as a general-purpose OS.
- CheriBSD is a version of FreeBSD/BERI that has been enhanced to make use of CHERI's experimental capability coprocessor features.

At the time of writing, FreeBSD has been modified to support a number of BERI features, such as peripheral devices present on the Terasic DE4 board. CheriBSD extends FreeBSD/BERI to initialize and maintain CHERI coprocessor registers; more information on CheriBSD can be found in the *CHERI User's Guide*.

3.1 Getting Started with FreeBSD

To get started with FreeBSD/BERI, you need the following:

- Ubuntu development PC or VM (version 14.04 recommended)
- Pre-built or custom-compiled FreeBSD kernel
- Pre-built or custom-compiled FreeBSD root filesystem image

For FPGA targets you also need:

- Terasic DE4 board with supplied 1GB DRAM
- Altera Quartus tools (version 13.1 recommended)
- 2GB SD card[1]
- BERI bitfile targeted for the Terasic DE4

[1]Note: Altera's University Program SD Card IP Core does not support SD cards larger than 2GB.

Synchronized versions of BERI FPGA bitfiles and FreeBSD must be used together: as the prototype evolves, hardware-software interfaces change, as do board configurations; mismatched combinations will almost certainly function incorrectly. The installed Quartus toolchain should also match the one used to generate the bitfile being programmed, in order to avoid documented incompatibility.

The remainder of the chapter describes how to obtain FreeBSD kernels and root file-system images, boot FreeBSD in simulation, write the FreeBSD root file-system image onto the SD card, program the Terasic DE4 FPGA with a BERI bitfile, set up the JTAG debugging tunnel so that the `berictl` tool can manipulate BERI via its debug unit, connect to the BERI console using `nios2-terminal` over JTAG, and optionally re-flash the DE4's on-board CFI flash with a bitfile and kernel to avoid the need to program them via JTAG on every boot.

3.1.1 Obtaining FreeBSD/BERI

Pre-generated images of FreeBSD/BERI and CheriBSD may be downloaded from the BERI website:

`http://www.beri-cpu.org/`

Additionally, FreeBSD/BERI and CheriBSD may be built from source code using the instructions found in Chapter 2. Finally, pre-compiled snapshots of key files and images may be found in `/usr/groups/ctsrd/cheri` in the Cambridge environment. Each file is named based on the date is was generated, consisting of `YYYYMMDDv` where `v` is an optional letter indicating further builds that occurred on the same day.

Table 3.1 describes file types that may be found on the early adopters page and in the above mentioned directory; all bitfiles and kernels are compressed using `bzip2` and all filesystem images are compressed with `xz`. Filesystem images must be decompressed before they can be used. Files passed to `berictl` may be uncompressed; alternatively, the `-z` flag may be used to decompress them on the fly.

3.1.2 Building berictl

`berictl` is a front-end to a variety of development and debugging features associated with the BERI processor, both in simulation and when synthesized to FPGA. `berictl` will generally communicate with the BERI debug unit over JTAG or a socket.

For building you will require the `libbz2` library (`libbz2-dev` package in Ubuntu 14.04). To compile:

```
cd cherilibs/trunk/tools/debug
make
```

3.1.3 Writing Out the SD Card Disk Image (FPGA only)

To use FreeBSD/BERI on FPGA with an SD card root filesystem, write out the file system image on an existing Mac, FreeBSD, Linux, or Windows workstation. The following command is typical for a UNIX system; ensure that the disk device name here is actually your SD card and not another drive!

```
$ dd if=arcina-beribsd-sdcard.img of=/dev/disk1 bs=512
```

On Mac OS X, `diskutil list` may be used to list possible devices to write to. You may need to use `diskutil unmountDisk` (or `DiskUtility.app`) to first unmount an auto-mounted FAT filesystem if one existed when the card was inserted. Note that SD cards should not be initialized with FAT or other filesystems, and such filesystems may need to be unmounted before the first image is written to an SD card; disk images include a complete UFS filesystem intended to be written directly to the SD card starting at the first block.

Filename	Description
beribsd-de4-kernel-net-mdroot	DE4 kernel with built-in memory root filesystem with basic network tools
beribsd-de4-kernel-singleuser-mdroot	DE4 kernel with built-in memory root filesystem that drops to single user mode with limited tools
beribsd-de4-kernel-sdroot	DE4 kernel using an SD card as a root filesystem
beribsd-sim-kernel-mdroot	Simulation kernel with built-in memory root filesystem
beribsd-sim-kernel-sdroot	Simulation kernel with a simulated SD card as a root filesystem
beribsd-flashboot	FreeBSD boot2 compiled for direct execution and self relocation from flash. Not yet used
beribsd-jtagboot	FreeBSD boot2 compiled for execution at 0x100000, may be installed in flash in place of a kernel
beribsd-sdcard.img	SD card root filesystem image
cheribsd-*	Same as similarly named beribsd-* files
cheri-bitfile.bin	Altera bitfile for CHERI processor in binary format (suitable for writing to flash)
cheri-bitfile.sof	Altera bitfile for CHERI processor in SOF format (for use with Altera tools)
cfi0-de4-terasic	Vanilla CFI flash cfi0 image for the DE4

Table 3.1: Binary files available for FreeBSD/BERI. -dump files will sometimes also be present, which contain objdump -dS output for kernels and other binaries. Releases will have the release name (e.g. ambrunes-) prepended and snapshots a date string.

3.1.4 Setting Up the DE4 Development Environment (FPGA only)

Many commands in the chapter depend on Altera Quartus 12 tools. Specifically, the `nios2-terminal` must be in the user's `PATH`, and the `system-console` command must be available.

In the Cambridge environment, setup can be accomplished by configuring the CHERI build environment:

```
$ source cheri/trunk/setup.sh
```

A default user install of the Quartus 13 toolkit will also accomplish setup, so long as the `${HOME}/bin` directory is in the user's path.

The `berictl` command controls various aspects of CPU and board behavior. For example, it can be used to inspect register state, modify control flow, and load data into memory. `berictl` works with BERI in both simulation and in FPGA. Build `berictl` using the following commands:

```
$ cd cherilibs/trunk/tools/debug
$ make
```

For users without access to the Subversion repository, statically linked versions of `berictl` are distributed along with each BERI release.

3.1.5 JTAG (FPGA only)

Many hardware debugging functions rely on JTAG, which allows a host Linux workstation to program the FPGA board, read and write DRAM on the board, and also interact with the CHERI debug unit for the purposes of low-level system software debugging. Use of JTAG requires that a USB cable be connected from your Linux workstation to the Terasic DE4 board. In the remaining sections, JTAG will be used to access four different debugging funtions:

- programming the FPGA (via `berictl loadsof`);
- BERI's JTAG UART console (via `berictl console`);
- direct DRAM manipulation (via `berictl loadbin` or `berictl loaddram`);
- and to use BERI's debug unit (most other `berictl` commands).

3.2 Booting FreeBSD in Simulation

To run BERI in simulation, first download or build yourself a suitable kernel (the filename should contain '`sim`'). Make sure the build options match your environment (e.g. use a 'beri' target if capabilities are not enabled). Decompress this kernel:

```
bunzip2 20140616-cheribsd-beri-sim-mdroot-singleuser-kernel.bz2
```

`berictl` uses UNIX-domain sockets to communicate with BERI in the Bluespec simulator. These need to be provided as environment variables to the simulator binary. To simulate:

```
cd cheribsd/trunk/simboot
make
mkdir -p /tmp/beri
make run KERNEL=20140616-cheribsd-beri-sim-mdroot-singleuser-kernel \
  CHERI_CONSOLE_SOCKET=/tmp/beri/console-socket \
  BERI_DEBUG_SOCKET_0=/tmp/beri/debug-socket-0 \
  BERI_DEBUG_SOCKET_1=/tmp/beri/debug-socket-1
```

This will start the simulator running, but not attach a terminal to its console. You can do this with:

```
cd cherilibs/trunk/tools/debug
./berictl -s /tmp/beri/console-socket console
```

You should see some initial boot messages within a few seconds. Booting will take about an hour on a fast PC.

3.2.1 Using the berictl debugger

`berictl` also provides debugging facilities for BERI. In simulation mode this uses the first BERI debug socket, for example:

```
./berictl -s /tmp/beri/debug-socket-0 pc
```

will print the current program counter. berictl's other commands are listed in Table 3.2. This list is subject to change: the berictl man page gives full details and can be shown by:

```
./berictl man
```

3.3 Programming the DE4 FPGA

FPGA designs are encapsulated in a *bitfile*, which can be programmed dynamically using JTAG, or from the on-board CFI flash on the DE4 when the board is powered on. The former configuration will be used most frequently during development of processor or other hardware features; the latter will be used most frequently when developing software to run on BERI, as it effectively treats the board as a stand-alone computer whose firmware (and hence CPU!) may occasionally be upgraded. The DE4's FPGA may be programmed as follows:

```
$ cd cherilibs/trunk/tools/debug
$ ./berictl loadsof -z arcina-cheri-bitfile.sof.bz2
```

Note well: you must terminate all `berictl` *and* `nios2-terminal` *sessions connected to the DE4 before using* `berictl`*'s* `loadsof` *command. If you do not the board may not be reprogrammed or instances of* `system-console` *may crash.*

`berictl` **command**	**Description**
Hardware/Simulator access and control	
`boot`	tell miniboot to proceed to the next kernel/loader
`cleanup`	clean up external processes and files
`console`	connect to BERI PISM UART (via -s) or Altera UART
`drain`	drain the debug socket
`loadbin`	load binary file at address
`loaddram`	load binary file at address
`loadsof`	Program FPGA with SOF format file
Status	
`pc`	print program counter
`regs`	list general-purpose register contents
`c0regs`	list CP0 register contents
`c2regs`	list CP2 (capability) register contents (has side effects)
Execution control	
`breakpoint`	set breakpoint at address
`pause`	pause execution
`reset`	reset processor
`resume`	resume execution (optionally unpipelined)
`step`	single-step execution
`setpc`	set the program counter to address
`setreg`	register to value
`setthread`	set the thread to debug
Memory access	
`lbu`, `lhu`, `lwu`	load unsigned byte/half word/word from address
`ld`	load double word from address
`sb`, `sh`, `sw`, `sd`	store byte/half word/word/double word value at address
Tracing	
`settracefilter`	set a trace filter from stream_trace_filter.config
`streamtrace`	receive a stream of trace data
`printtrace`	print a binary trace file in human readable form
Device debugging	
`dumpatse`	dump all atse(4) MAC control registers
`dumpfifo`	dump status and metadata of a fifo
`dumppic`	dump PIC status
Help	
`help`	display help for command
`man`	display the berictl manpage

Table 3.2: Options to berictl

3.3.1 Writing an FPGA Bitfile to DE4 Flash from FreeBSD

When powered on, the Terasic DE4 board will attempt to automatically load a bitfile from the on-board CFI flash. New FPGA bitfiles in binary format may be written to the flash from FreeBSD; they take effect during the board's next power-cycle. This write operation can be done using `dd` (note the skipping of the first 128k):

```
# dd if=arcina-cheri-de4-bitfile.bin of=/dev/cfid0s.fpga0 iseek=256 \
    conv=oseek
```

Bitfiles in SOF format can be converted to binary format using the sof2flash.sh script found in the CTSRD Subversion repository at `cherilibs/trunk/tools/sof2bin.sh`.

To simplify the process and add reliability, a script called `/usr/sbin/flashit` performs these actions after verifying the MD5 checksum of the files and optionally decompressing bzip2 compressed images. Note that if `flashit` is writing a file `foo` a corresponding `foo.md5` file must exist. In addition to FPGA images, `flashit` can be used to write kernel images by replacing the `fpga` argument with `kernel`.

```
# flashit fpga Design.bin
```

Power to the board must not be lost during reflash, as this may corrupt the bitfile and prevent programming of the board on power-on. Therefore, battery-backed DE4 boards should be programmed only while fully charged.

3.3.2 Start a Console

Connect to the BERI JTAG UART using:

```
$ berictl console
```

The console many be terminated by typing `~.` (tilde followed by period) after a newline. If connecting to the console via `ssh` or other network terminal programs an appropriate number of ~ characters must be used as most use the same escape sequence. *Note:* `berictl` *starts an instance of* `nios2-terminal` *to connect to the console so that instance must be terminated before a kernel image or bitfile can be loaded.*

3.4 Models for Booting a FreeBSD/BERI Kernel

FreeBSD kernels may be booted via two different means: installation on the on-board CFI flash device on the Terasic DE4, or direct insertion of the kernel into DRAM using `berictl` via JTAG. The micro-boot loader embedded in ROM in CHERI on the DE4, `miniboot`, uses the USER_DIP1 switch to control whether a kernel is relocated from flash or started directly. Note that DIP switches are numbered 0-7, but the physical package has labels 1-8. The proper labels can be seen in Figure 3.1.

USER_DIP0 controls whether `miniboot` runs immediately or it spins while waiting for register 13 to be set to 0 before booting the kernel, leaving time for the kernel to be injected into DRAM following programming of the FPGA. Register 13 would normally be set to 0 using the debug unit.

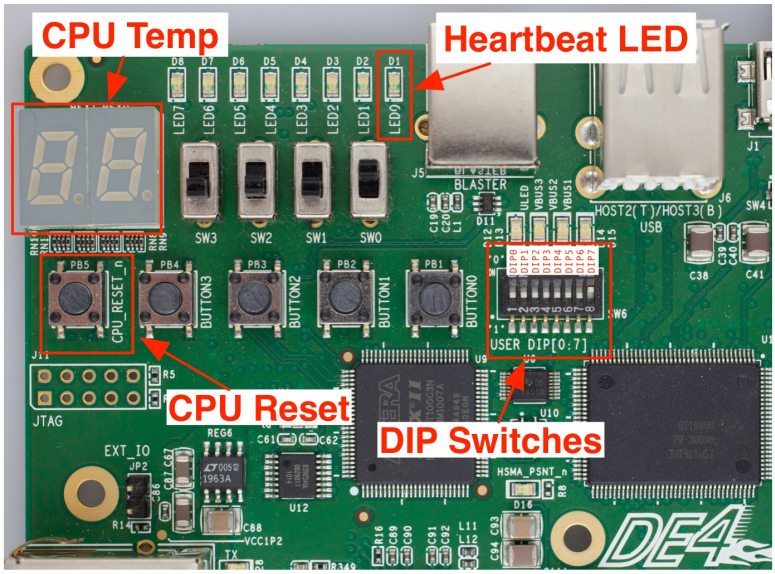

Figure 3.1: Buttons and switches on the DE4

3.4.1 Load a Kernel into DRAM over JTAG

To load the kernel into DDR2 memory, load it at the physical address `0x100000` where miniboot boot loader expects to find it. `miniboot` reads the ELF header in order to determine the kernel start address. You can then use `berictl loadbin`[2] to load the kernel to DDR2 memory starting at address `0x100000` (also see note below about USER_DIP0 and USER_DIP1):

```
$ cd cherilibs/trunk/tools/debug
$ ./berictl loadbin -z cheribsd-de4-kernel-sdroot.bz2 0x100000
```

To boot a kernel thus loaded, you must ensure that both USER_DIP0 and USER_DIP1 are on (toward the top of the board where the USB blaster is connected). USER_DIP0 will cause the processor to spin in very early boot, waiting for register 13 to be set to 0. This boot operation can be accomplished through the debug command:

```
$ ./berictl boot
```

USER_DIP1 will skip the relocation from flash routine that would overwrite your freshly inserted kernel.

3.4.2 Load a Kernel into Flash from FreeBSD

From FreeBSD you can use `dd` or the `flashit` script to load a kernel to flash:

```
# dd if=kernel of=/dev/map/kernel conv=osync
```

The safer option using `flashit` compresses the kernel with `bzip2` or `gzip`, and requires a `.md5` file to exist containing the `md5` output for the file:

[2]Some users may be able to use the faster `berictl loaddram` command, but it is broken for most configuration at this time – as it relies on undocumented Altera internals that have changed in recent releases.

```
# ls kernel.bz2*
kernel.bz2  kernel.bz2.md5
# flashit kernel kernel.bz2
```

At boot, a kernel written to flash will be relocated to DRAM and executed if USER_DIP1 is set to off. This relocation will occur at power on if USER_DIP0 is off, or when `berictl boot` is run if it is on.

3.5 Start Kernel Execution

If USER_DIP0 is set to on, then resume the processor after power on/reset:

```
$ ./berictl boot
```

If the DIP switch is unset, then boot will proceed as soon as the FPGA is programmed, either using JTAG or from flash. If all has gone well, you should see kernel boot messages in output from the console. If you are using the `BERI_DE4_MDROOT` or `CHERI_DE4_MDROOT` kernel configuration, a memory root filesystem will be used; single or multi-user mode should be reached, depending on the image. If you are using the `BERI_DE4_SDROOT` or `CHERI_DE4_SDROOT` kernel configuration, the SD card should be used for the root filesystem, and multi-user mode should be reached. Be warned that the SD-card IP core provided by Altera is extremely slow (100KB/s), and so multi-user boots can take several minutes.

3.6 Post-Boot Issues

After boot, FreeBSD/BERI is much like any FreeBSD system with a similar set of components. There are a few issues to keep in mind

- The MDROOT kernels are space limited and have minimal set of tools available.
- Since the root filesystems of MDROOT kernels are stored in memory, all configuration including ssh keys will be lost at reboot time.
- The Ethernet controllers have no default source of unique MAC addresses and thus default to a random address that changes on each boot.

3.6.1 Increasing the Size of an SD Card Root Filesystem

After boot, you can extend the filesystem to the size of the SD card using FreeBSD's `growfs` command:

```
$ growfs -y /dev/altera_sdcard0
```

Before running this command, make sure your filesystem is backed up or easily replacable.

3.6.2 Setting a MAC Address

The Altera Triple-Speed Ethernet (ATSE) devices obtain a unique MAC address from the configuration area at the beginning of the CFI flash. Unfortunately, all DE4 boards come from the factory with the same MAC address, so that address has been blacklisted by the driver; instead, a random address is generated at boot for each interface.

An address can be written to the DE4 using the `atsectl` command. An address derived from the factory PPR on the Intel StrataFlash on the DE4 can be written with the command:

```
$ atsectl -u
```

The default address has the locally administered bit set and uses the Altera prefix dedicated to this purpose.

In the Cambridge environment, the decision was made to leave the locally administered bit unset. This can be accomplished with the command:

```
$ atsectl -gu
```

If the board was configured following the *DE4 Factory Install Guide v1.0*, then an Altera prefixed MAC without the locally administered bit will have been installed on the DE4.