Z4800 Platform Documentation

Will Simoneau

March 31, 2012

The Z4800 platform is an open-source (mostly GPLv2) project consisting of a synthesizable soft-core CPU plus supporting hardware and software. The included reference designs allow construction of a complete FPGA computer complemented by a full-featured operating system capable of running many real-world programs. This platform can be used for many purposes, particularly benchmark studies; the processor core can be modified and has an extendable performance counter system.

The Z4800 CPU itself is a clean-room design capable of executing some binaries compiled for a variant of the MIPS® R4000™ processor. The main differences are that it does not implement the 64-bit or floating-point instructions, and that its native endianness is little-endian. Aside from these differences, most existing user-mode and kernel-mode software will work.

A fully functional Linux kernel is supported; source code for the modified kernel is included. These modifications are mostly to support the peripherals on the FPGA boards – the Z4800 CPU functions enough like an R4K that the existing Linux/mips R4K code can be used directly. SMP (multiprocessor) support is included.

To download the complete Z4800 source code, run the following command. You will need at least Git 1.5.3 because this project uses submodules.

```
git clone --recursive git://trogdor.ele.uri.edu/z4800.git
```

A paper about this project, *An FPGA-based Multi-Core Platform for Testing and Analysis of Architectural Techniques*, was written for and accepted to the *2012 IEEE International Symposium on Performance Analysis of Systems and Software*.

0.1 To-Do List

In no particular order:

- Publish public documentation on the Z4800 CPU internals
- Add screenshots to the documentation to show example output
- Create and publish virtual machine images providing a canned development environment
- Create and publish tarball providing a canned root-FS for target board (DE2-115, etc.)
- Publish public version of benchmarking infrastructure, with accompanying documentation

0.2 Legal Notice

MIPS and R4000 are trademarks or registered trademarks of MIPS Technologies, Inc. in the United States and other countries. The Z4800 project is neither licensed by nor endorsed in any way by MIPS Technologies, Inc. We only claim to have limited binary compatibility; we claim neither full compatibility nor full compliance with any MIPS® products and/or specifications.

0.3 Contacts

The author of the Z4800 source code is Will Simoneau (simoneau@ele.uri.edu). He may be contacted for any technical questions. The ISPASS 2012 paper was co-authored with Resit Sendag (sendag@ele.uri.edu).

1 Building & Booting

The most complicated part of using the Z4800 design is setting up a working development environment for it. Many pieces of software need to work together for the developer (*you*) to accomplish anything useful. Setting up such an environment is no small task; doing so from scratch is extremely difficult for a new developer. The goal of this section is to guide you through setting everything up, to the point where you should be able to compile the reference board designs and software from scratch and demonstrate them functioning on real FPGA boards.

The environment was designed for Linux hosts. It should be possible to develop on Windows using Cygwin, but we haven't tried constructing such an environment. If your host absolutely must run Windows, you will need to install Linux (Debian, etc.) inside of VirtualBox and develop on the VM.

1.1 Altera Software

You'll need Altera's Quartus II software (§2, p.5). Version 11.0sp1 should work. Run their Linux installers and get it set up.

1.2 GNU mipsel Toolchain

The Z4800 CPU can run code compiled for a 32-bit only, little-endian variant of the R4K. GNU binutils and GCC can provide the minimal cross-toolchain you'll need to compile the bootloaders and Linux kernel. You'll need to compile these yourself; this section covers how to do that. We will be targetting `mipsel-unknown-linux-gnu`.

You will need to install some dependencies to be able to compile these. On Debian Stable (as of this writing, Debian "Squeeze"), you will need to do the following:

```
# apt-get install build-essential libgmp3-dev libmpfr-dev libmpc-dev
```

1.2.1 binutils

The first piece of the puzzle is a cross-binutils. Download the binutils source (§2, p.5) and get it installed. The final command requires root access if you use the default installation paths, since it will be installing to `/usr/local`.

```
$ tar xvjf binutils-2.20.1.tar.bz2
$ cd binutils-2.20.1
$ ./configure --target=mipsel-unknown-linux-gnu
$ make
$ sudo make install
```

1.2.2 GCC

Now we need a cross-compiler, so get the GCC source (§2, p.5). We don't need to cross-compile userland binaries so we don't need any external headers, and just a basic C compiler will do. GCC does not like being built in its own source directory; you need to create a separate build directory for it.

```
$ tar xvjf gcc-4.5.3.tar.bz2
$ cd gcc-4.5.3
$ mkdir build
$ cd build
$ ../configure --target=mipsel-unknown-linux-gnu --without-headers \
  --without-threads --enable-languages=c
$ make
$ sudo make install
```

1.3 CPU Boot Code

At this point you should have a functional cross-compilation environment, and should be able to successfully issue `make` in `core/boot` in the Z4800 git tree. This will build the bootloader(s) which will be embedded into the FPGA's on-chip ROMs. If this does not work, something is not set up correctly. Usually the problem is that your new binutils, GCC, or the Altera NIOS tools (particularly `elf2hex`) are not in your $PATH.

1.4 Generate SOPC Files

Next, you need to open the Quartus project (example: `core/projects/de2_115_smp/prj_de2_115_smp.qpf`) in Quartus. Go to the "Tools" menu and run "SOPC Builder". (Or run `sopc_builder` from the project's directory.)

The first time you use SOPC Builder with any Z4800 project, you will need to add the right directory to the IP search path. In SOPC Builder, go to "Tools", "Options", select "IP Search Path", and add the `core/hdl` directory (which is a subdirectory of wherever you left the checked-out git repository). This should result in `core/hdl/**/*` being searched, which should pick up all the custom SOPC Builder modules.

Now you can open the SOPC Builder system file for the project (example: `sys_de2_115_smp.sopc`). If everything goes well, no errors should be listed. If that's the case, hit Generate and wait until it finishes successfully. This can take a couple minutes depending on your workstation's hardware.

1.5 Compile Project in Quartus

Now that all the relevant files are generated, you can compile the Quartus project itself. Click "Processing", "Start Complation". If there are no errors, this will take from several minutes to a few hours, depending on your workstation's hardware. (You could also do this without using the Quartus GUI: `quartus_sh --flow compile project_name` from inside the project's directory.)

When compilation is complete, you will have a .sof file in the project directory (example: `core/projects/de2_115_smp/prj_de2_115_smp_time_limited.sof`). This is the bitstream image of the completed design, ready to be used to program the FPGA.

1.6 Kernel, OS, and NFS-root Hints

If you got this far successfully, the next part you need to build is the target's Linux kernel. An example kernel .config is provided in the board's directory (`kernel.config`). You will probably need to adjust `CONFIG_CMDLINE`, as it hardcodes the network configuration for NFS-root.

You can build the kernel from the board directory using a command such as the following, which will put object files (and the final `vmlinux` ELF binary) into the `kobj` subdirectory.

```
$ mkdir kobj
$ make -C ../../kernel KCONFIG_CONFIG=../kernel.config ARCH=mips \
  CROSS_COMPILE=mipsel-unknown-linux-gnu- O=../projects/de2_115_smp/kobj \
  vmlinux
```

1.6.1 Target OS

You will need to have a local NFS server prepared; I suggest serving a Debian install. To set up this directory with a usable root filesystem, you can use the `debootstrap` or `cdebootstrap` tools. The key to using these is the `--foreign` option, which permits bootstrapping on architectures other than that of the host machine. You will need to refer to the tools' documentation and think about your network setup to finish this step. Make sure your NFS server is properly exporting the directory which will become the target's root filesystem.

It can be helpful to use QEMU to emulate a somewhat generic MIPS machine and bootstrap the OS before attempting to boot an FPGA; how you accomplish this is up to you. Either way, by doing it yourself you will become familiar with how things work.

1.6.2 Booting the Target

The target board needs a way to load its kernel. It is possible to boot kernels out of Flash memory (`core/boot/boot_cfi.S` is a minimal bootloader for CFI Flash written in assembly), but that requires one to program the kernel onto the Flash chip, creating a chicken-and-egg problem. Instead, you can use a second board to control the target board and load its kernel. In this example we will pair a DE2-70 debugger board with a DE2-115 target containing the Z4800 CPU(s).

You will need to make a custom ribbon cable to safely connect the two FPGA boards. Start with a 40-pin ribbon with one connector at each end, about 8-10 inches long (a 40-pin single-disk IDE ribbon could work). *You MUST remove two wires from the ribbon: those connecting to pin 11 and pin 29.* These wires connect to the FPGA's power rails; it is inadvisable to short the two boards' power rails together. If you get this wrong, it may destroy the boards – you have been warned. Be very careful with the pinout, and refer to both the DE2-70 and DE2-115 manuals to be sure your connection is correct before powering anything on.

Use this ribbon cable to connect the DE2-70 to the DE2-115; either of the DE2-70's GPIO headers may be used. If you connect to GPIO0 (the left one), set SW17 down; if you connect to GPIO1 set SW17 up (there is a bus "switch" connecting the remote-DMA module to either header).

Now that the boards are connected, power on only the DE2-70, and program the `core/projects/de2_70_debugger/de` file onto it. Then, power on the DE2-115 and program it. Press and hold KEY0 (reset) on both boards simultaneously to ensure the link state is sane; LEDG0 through LEDG3 on both boards should be OFF if it is working correctly.

Attach a straight-through DB9 serial cable to the DE2-70's serial port. Run minicom to get a console for the DE2-70 (the serial parameters it uses are 115200 8N1). Then, boot the debugger's OS by running `nios2-download -C N -g core/projects/de2_70_debugger/zImage` where N is the JTAG cable number (see output from `jtagconfig`). If everything went right, it should boot a uClinux-based OS and drop you to a shell. Then, you need to run commands similar to the following (adjust as needed to fit your network setup):

```
ifconfig eth0 hw ether 00:07:ed:0a:03:29
ifconfig eth0 192.168.1.80 netmask 255.255.255.0
passwd
```

The DE2-70 runs an ssh daemon, so after you've set the root password you can log into it remotely. Use `scp` to copy the following files onto the DE2-70 board's ramdisk:

```
core/boot/boot_ram.elf
core/projects/de2_115_smp/kobj/vmlinux
```

You will want to connect a second straight-through DB9 serial cable to the DE2-115's serial port at this point so that you have access to the target's local console. The serial parameters are the same as the debugger (115200 8N1). You could just swap the cable over from the DE2-70 since you probably don't need its console anymore, but it's convenient if your host has 2 serial ports to just connect both.

Now you can use the `z48d` debugger program on the DE2-70 to connect to the DE2-115 to load and boot the kernel. The commands to do this are:

```
$ z48d -n x
l
boot_ram.elf
l
vmlinux
R
r
Q
```

(In order, these mean: start z48d, load, bootloader ELF image, load, kernel ELF image, reset, run, quit)

If everything went perfectly, the Z4800 target should boot Linux with NFS-root. Good luck!

2 Files

The following table lists the packages this documentation was written against. Newer versions of these packages may not completely match this documentation. Caveat user.

File name	md5sum	Description	License
11.0sp1_quartus_linux.sh	916ea16caafbc314cf385d139295e0fe	Quartus 11.0sp1 Base	Proprietary
11.0sp1_devices1_linux.sh	6da9b0dc1efdfa4af8f7415db96002ec	Quartus 11.0sp1 Devices 1	Proprietary
11.0sp1_devices2_linux.sh	5fc1730b6dc4a3a66963c815d1f1fbfe	Quartus 11.0sp1 Devices 2	Proprietary
binutils-2.20.1.tar.bz2	2b9dc8f2b7dbd5ec5992c6e29de0b764	GNU binutils	GNU GPL
gcc-4.5.3.tar.bz2	8e0b5c12212e185f3e4383106bfa9cc6	GNU GCC	GNU GPL