ORPSoC User Guide

Julius Baxter
OpenCores
Issue 4 for ORPSoC

This file documents the OpenRISC Reference Platform SoC, ORPSoC.

Published by OpenCores

Table of Contents

1 Introduction

ORPSoC is intended to be a reference implementation of processors in the OpenRISC family. It provides a smallest-possible reference system, primarily for testing of the processors. It also provides systems intended to be synthesized and programmed on physical hardware.

The reference system is the least complex implementation and consists of just enough to test the processor's functionality. The board-targeted builds typically include many additional peripherals.

The next section in this document outlines the organisation and structure of the project. The section "*Getting Started*" goes through getting the project source and setting up any necessary tools. Each following section outlines a particular implementation of an OpenRISC-based system, beginning with the reference system. Each implementation section has an overview of the structure of the project (which probably won't vary much between the implementations), a section on setting up the required tools, running simulation, and if applicable, backend and debugging steps. There may be additional sections on modifying or customising each implementation system.

2 Project Organisation

2.1 Organisation Overview

The ORPSoC project is intended to serve dual purposes. One is to act as a development platform for OpenRISC processors, and as a development platform of OpenRISC-based SoCs targeted at specific hardware.

Organising a single project to satisfy these requirements can lead to some overlap and redundancy. This section is intended to make the organisation of the project clear.

The reference implementation based in the root (base directory) of the project contains enough components to create a simple OpenRISC-based SoC. Each board build is intended to implement as fully-featured a system as possible, depending on the targeted hardware.

The project is organised in such a way that each board build can use both the reference implementation's RTL modules and software, as well as its own set of RTL and software. The reference implementation is limited to what is available in the RTL and software directories in the root of the project, and is not technology dependent.

The following sections outline the organisation of the software, RTL, and board designs.

2.2 Software

The `sw` path contains primarily target software (code intended for cross-compilation and execution on an OpenRISC processor.) There is also a path, `sw/utils` containing custom tools, intended to be run on the host, for manipulation of binary software images.

Driver software, implementing access functions for hardware modules, are found under `sw/drivers`.

There is a minimal support library under the `sw/lib` path. Both drivers and support library are compiled together to create a library called `liborpsoc` which is placed in `sw/lib`.

All CPU-related functions are made available through the file `cpu-utils.h` which is located in `sw/lib/include` and depending on the CPU being used, can be used to switch between different CPU driver functions. All CPU drivers are under the `sw/drivers` path.

Note: It is expected in the future that the OpenRISC toolchain based on newlib will provide all of the necessary support software provided in this CPU-specific driver path. When the first release of the newlib-based toolchain occurs it is expected the software in ORPSoC will be changed to use this toolchain instead.

Test software is found under `sw/tests`. Typically, each is for a specific module, or for a particular capability (eg. tests for the UART capability are under `sw/tests/uart`, rather than `sw/tests/uart16550` which.) There are no specific rules here.

Under each test directory are two directories, `board` and `sim`, containing appropriate test software. Code for simulation will produce less printfs and aim to execute within realistic timeframes for RTL simulation. Board targeted test software is obviously written with the opposite considerations in mind and be more verbose and perhaps run orders of magnitudes more tests.

2.2.1 Software Test Naming

The rules for naming software tests are important to adhere to, so the automation scripts can locate them. The test directory name must be a single word (potentially with underscores), and then the tests must be in files of the format *testdirname-testname*.extension, eg. `uart-simple.c` or `or1200-fp.S`.

See Chapter 10 [Software], page 39, for further details.

2.3 RTL

The HDL code layout conforms to those outlined in the OpenCores.org coding guidelines. http://cdn.opencores.org/downloads/opencores_coding_guidelines.pdf

There are, however, some naming restrictions for this project.

The directory name (presumably the name of the module, something like `uart16550`) should also be the name of the top level file, eg. `uart16550.v`, and the top level module should also be simply this name, eg. `module uart16550 (...);`.

This will avoid confusion and help the scripts locate modules.

2.3.1 Verilog HDL

All RTL included in the project at this point is Verilog HDL, although it would be fine to add VHDL to a board build.

2.4 Testbench

For each design in ORPSoC there will be a testbench instantiating the top level, and any required peripherals.

Despite this being far from a thorough verification platform, it is considered useful to be able to perform enough simulation to ensure that the fundamental system is correctly assembled and can communicate with the peripherals.

It is expected that by running the command `make rtl-test` in each board's simulation run path, a basic simulation of the system initialising should be run.

2.5 Reference And Board Designs

The goal of the reference design is to provide an environment to develop and test OpenRISC processors (also, potentially, basic components.) The goal of the various board-targeted designs is to provide ready-to-go implementations for hardware.

2.5.1 Module Selection

Typically, a board-targeted design will wish to reuse common components (processor, debug interface, Wishbone arbiters, UART etc.)

The project has been configured so a board build will use modules in the "common" RTL path (`rtl/verilog/`) *unless* there is a copy in the board's "local" RTL path (`boards/vendor/boardname/rtl/verilog`) or the board includes an external module in `boards/vendor/boardname/modules`.

For example, in the event that modification to a module in the common RTL set is required for it to function correctly in a board build, it's advisable to copy that module to the board's

local RTL path and modify it there. Simulation and backend scripts should then use this board-specific version instead of the one in the common RTL path.

3 Getting Started

3.1 Supported Host Platforms

At present the majority of ORPSoC's development occurs with tools that run under the GNU/Linux operating system. All of the tools required to run the basic implementation are free, open source, and easily installable in any modern GNU/Linux distribution.

Unless indicated otherwise, support for the project under Cygwin on Microsoft Windows platforms cannot be assumed.

3.2 Obtaining Project Source

The source for ORPSoC can be obtained from the OpenCores subversion (SVN) server.

```
svn export http://opencores.org/ocsvn/openrisc/openrisc/trunk/orpsocv2
```

3.3 Required Tools

Performing the installation of tools required to design, simulate, verify, compile and debug a SoC is not for the faint hearted. The various sets of tools must be first installed, and the user's environment configured to allow them to run correctly.

First the host system must be capable of building and running development tools, next the various compilers, simulators and utilities must be installed, and finally, if required, additional tools to interact with the built design are installed. Once complete, the set up rarely needs to be touched, and results in greatly improved productivity.

The required tools can be divided into four groups.

- Host system tools - things like gcc, make, texinfo.
- Target system toolchain and software - the OpenRISC GNU toolchain, with gcc cross-compiler, support libraries, the GNU debugger (gdb), OpenRISC port of various OSes and RTOS, etc.
- Electronic design automation (EDA) tools - preprocessors, simulators, FPGA tool suites, etc.
- Debug tools - tools providing control over the system on target

The first two items are likely to be the same for most of the designs included in ORPSoC, however the final two can vary greatly depending on the FPGA vendor, part and configuration, and the application of the SoC design.

There will be a section on the tools for each design in ORPSoC. This section is intended to provide a list of tools required for each particular build. Any lengthy instructions on installing a particular tool will be attached as an appendix, which can be references by several build prerequisite lists in order to save repetition of information.

Anyone implementing their own board port is encouraged to document, as best they can, any problematic tool installations and contribute them to this document.

4 Reference Design

4.1 Overview

The reference design included in ORPSoC is intended to be the minimal implementation (or thereabouts) of a SoC required to exercise an OpenRISC processor. Very little apart from the processor, memory, debug interface and interconnect modules are instantiated.

The primary role for this design is to implement a system that an OpenRISC processor can be instantiated in for for development purposes. The automated testing mechanism, capable of compiling, executing and checking software on the design, is used as a method of regression testing for the processor as it is developed. After features are added or modified in the processor, new software tests can be added to the existing suite, checking for the expected functionality and ensuring legacy behavior is also unchanged.

The design can be simulated two ways. The first uses the standard event-driven simulators such as Icarus Verilog and Mentor Graphics' Modelsim. The second method involves creating a cycle accurate (C or SystemC) model from the Verilog HDL description using the Verilator tool.

The simulations begin with the desired software image preloaded in memory. For debugging the design, the models provide an interface to the system allowing the GNU debugger to control the target processor in a manner similar to that of physical hardware.

The design is not intended for implementation on an FPGA or ASIC, rather purely for development and testing in simulation environments. The board targeted builds in the ORPSoC project, however, are intended for implementation on hardware.

4.2 Structure

The reference design's paths are all based in the root directory of ORPSoC. This is different from the board-targeted builds, which are based in their specific board paths.

As synthesis and physical implementation is not intended for the reference design there are no `syn` or `backend` paths in the root directory of ORPSoC.

4.2.1 RTL

At present only Verilog HDL is included in the reference implementation of ORPSoC, as the open source tools intended to simulate the design do not support VHDL.

The directory structure consists of an `rtl` directory in the root, and a `verilog` path under that. Within the `rtl/verilog` path, each module has its own directory.

External modules using the OpenCores structure can be put under each boards `modules` directory. The scripts will the look for verilog files under `modules/<module_name>/rtl/verilog`.

A common Verilog include path, `rtl/verilog/include` directory is used. The Verilog HDL include files for each module should be moved here. This allows each ORPSoC implementation (board design) to maintain their own include path, and thus configure the modules for their specific implementation.

4.2.2 Software

The software run on the reference design is found in the ORPSoC root directory, under the `sw` path.

The test software for the or1200 processor is found under `sw/tests/or1200/sim`.

A minimal set of drivers is built into `liborpsoc`, and they are found under `sw/tests/drivers`.

In addition to these drivers, a set of support C functions is build into `liborpsoc`, which are found in the `sw/lib` path.

4.2.3 Simulation

The simulation of the reference design is run from the `sim/run` path.

The script controlling simulation is the file `sim/bin/Makefile`.

The generated output is kept in the `sim/out` path, and is cleared away when *`make clean`* is run.

When the Verilator-processed cycle accurate model is built, it is done in the `sim/vlt` path, which is also cleaned away when *`make clean`* is run.

4.3 Tools

4.3.1 Host Tools

Standard development suite of tools: gcc, make, etc.

4.3.2 Target System Tools

OpenRISC GNU toolchain. For installation, see OpenRISC GNU toolchain page on OpenCores.org.

4.3.3 EDA Tools

RTL simulation: Icarus Verilog (also compatible with Mentor Graphics' Modelsim) Cycle Accurate Simulation: Verilator, Verilog-Perl, System-Perl, SystemC

4.3.4 Debug Tools

None. The target is purely simulation, no extra utilities are required to debug.

4.4 Simulation

4.4.1 RTL

All simulations of the reference design are run from the `sim/run` path.

Running RTL Regression Test

The simplest way of starting a run through the regression test, using the default RTL simulator, Icarus Verilog, can be done with:

```
make rtl-tests
```

This will compile the software and RTL, and run a new simulation for each software test. Defining which tests are run is the variable `TESTS`, set by default in the `sw/bin/Makefile`

script. Other default options are that a processor execution log is generated (in `sim/out/testname-executed.log`), but VCDs are not.

Running An Individual Test

An individual test can be run, by specifying the test name through the `TEST` environment variable (which must correspond to a file in `sw/tests/module/sim/` where *module* is the name of the module to be tested. In the following example the test *or1200-basic* is run.

```
make rtl-test TEST=or1200-basic
```

Running A Set Of Specific Tests

A specific set of tests can be run in the same fashion as the regression tests but with the actual tests to run set in the `TESTS` environment variable.

```
make rtl-tests TESTS="sdram-rows uart-simple or1200-mmu or1200-fp"
```

Providing A Precompiled ELF Executable

It's possible to specify the path to an OR32 ELF executable to be run in simulation instead of test software. Use the `USER_ELF` environment variable to specify the path to the ELF to run.

```
make rtl-test USER_ELF=/path/to/myapp.elf
```

The ELF will be converted to binary format, and then converted to a VMEM to be loaded into the model for execution at runtime.

Providing A Custom VMEM Image

It's possible to specify the path to a pre-existing VMEM image to be run in simulation instead of test software. Use the `USER_VMEM` environment variable to specify the path to the VMEM image to be run.

```
make rtl-test USER_VMEM=/path/to/myapp.vmem
```

This VMEM file will be copied into the working directory, and renamed according to what the simulated memory expects.

Options For RTL Tests

There are some options, which can be specified through shell environment variables when running the test.

`VCD` Set to '1' to enable *value change dump* (VCD) creation in `sim/out/testname.vcd`

`VCD_DELAY`
Delay VCD creation start time by this number of timesteps (used as a Verilog `#delay` in the testbench.)

`VCD_DELAY_INSNS`
Delay VCD creation start time until this number of instructions has been executed by the processor. Useful for creating a dump just before a bug exhibited and correlated to an instruction number (from execution trace file.)

`END_TIME` Force simulation end (`$finish`) at this time.

`DISABLE_PROCESSOR_LOGS`
: Turn off processor monitor's execution trace generation. This helps speed up the simulation (less time writing to files) and avoids creating very large execution logs (in the GBs) for very long simulations.

`SIMULATOR`
: Specify simulator to use. Default is Icarus Verilog, can be set to `modelsim` to use Mentor Graphics' Modelsim. No others are supported right now.

`MGC_NO_VOPT`
: When using Modelsim (specifying `SIMULATOR=modelsim`), if the version does not include the individual `vopt` executable, specify `MGC_NO_VOPT=1` when compiling.

`VPI`
: Pass `VPI=1` to have the an external JTAG debug module stall the processor just after bootup, and then provide a GDB stub (interacting with the Verilog sim via the VPI) to allow control of the system in a similar fashion to that of a physical target controlled over JTAG via a debug proxy application. The port for GDB is hard-coded to 50002. See the code in `bench/verilog/vpi/c` for more details. If running with Modelsim, ensure the path `MGC_PATH` is set and points to a directory containing a path named `modeltech`, which should be the Modelsim install.

4.4.2 Cycle Accurate

Running Cycle Accurate Regression Test

The simplest way of starting a run through the regression test using the cycle accurate model can be done with:

```
make vlt-tests
```

This will build the cycle accurate model and run a new simulation for each software test. Defining which tests are run is the variable `TESTS`, set by default in the `sw/bin/Makefile` script.

Running An Individual Test

An individual test can be run, by specifying the test name through the `TEST` environment variable (which must correspond to a file in `sw/tests/`*`module`*`/sim/` where *`module`* is the name of the module to be tested. In the following example the test *or1200-basic* is run.

```
make vlt-test TEST=or1200-basic
```

Generating Cycle Accurate Simulator Executable

The cycle accurate model is somewhat similar to the OpenRISC architectural simulator, in that it can be run as a standalone application, although it is not as configurable at runtime. The standalone application can be built with the following command from the `sim/run` path.

```
make prepare-vlt
```

The resulting executable will be *Vorpsoc_top* in the `sim/vlt` path. Run it with the *-h* option for usage instructions.

Generating Automatically Profiled Cycle Accurate Simulator Executable

An automatic profiling and compilation set of commands in the script can be used to create a higher performance cycle accurate model. The following make target will first compile the cycle accurate design to generate profiling outputs, run some software, and recompile using the profiling information.

```
make prepare-vlt-profiled
```

Cycle Accurate Model Executable Usage

The executable generated by running any of the above commands is in the `sim/vlt` path. The usage options can be listed by running it with the `--help` switch.

```
Vorpsoc_top --help
```

A short list of options is given here.

`-f` *file*
`--program` *file*
: Load software from OR32 ELF image *file*
: If unspecified, model attempts to load VMEM file `sram.vmem`

`-v`
`--vcd`
: Dump VCD file

`-e` *value*
`--endtime` *value*
: End simulation after *value* simulated ns

`-l` *file*
`--log` *file*
: Log processor execution trace to *file*

4.4.3 Results

The following files are generated from the event driven simulation. For output options of the cycle accurate model, see the section on Cycle Accurate Model Executable Usage.

Processor Execution Trace

A trace of the processor after each executed instruction is generated by both the event and cycle accurate models.

In the event driven simulations, the log is created by default, and is stored in `sim/out` and will be named *test-name*`-executed.log`.

Processor SPR Access Log

A list of processor special purpose registers (SPR) accesses is created when processor logging is enabled.

These values are logged to a file in `sim/out` named *test-name*`-sprs.log`.

Processor Instruction Excecution Time Log

A list of when each instruction was executed is generated when processor execution logging is enabled.

This is useful when debugging with VCD, and detecting at what time the problematic instructions were executed.

These values are logged to a file in `sim/out` named *`test-name`*`-lookup.log`.

Processor Report Mechanism Log

The use of the processor's report mechanism is commonplace in the test software, as it allows for the checking of intermediate values after simulation.

These values are logged to a file in `sim/out` named *`test-name`*`-general.log`. This is not optional.

Value Change Dump (VCD)

When VCD files are generated they are placed in the `sim/out` path, and are named *`test-name`*`.vcd`. They should be viewable with programs like *GTKWave*.

4.5 Synthesis

The reference design is not intended to be synthesised, and thus no backend scripts are provided. See the sections on the board-specific builds.

5 ORDB1A3PE1500

5.1 Overview

The ORDB1 (ORSoC development board 1) with Actel A3PE1500 FPGA is supported by this build.

As the ORDB1 is intended to be a daughter board for a variety of I/O boards its options for configuration are extensive.

This board port of ORPSoC implements an example of a configurable system, with many cores that can be enabled or disabled as required by the expansion board's capabilities.

The port was mainly developed with the ORSoC Ethernet expansion board (OREEB1), and was used with the OpenRISC port of the Linux kernel and BusyBox suite running network applications.

This guide will overview how to simulation, synthesize and customise the system.

5.2 Structure

Note that in this chapter the term *board path* refers to the path in the project for this board port; `boards/actel/ordb1a3pe1500`.

The board port's structure is similar to that of a standalone project which accords with the OpenCores coding guidelines. However, all software and RTL that is available in the reference design is also available to the board port, with any local (ie. in the board's `rtl` or `sw` paths) versions taking precedence over the versions available in the reference design.

The Verilog RTL specific to this board is under `rtl/verilog` in the board path. The `include` path in there is the place where all required definitions files, configuring the RTL, are found.

Backend files, things such as PLLs and buffers generated by Actel's *smartgen* tool, are found in the board's `backend/rtl/verilog` path.

5.3 Tools

5.3.1 Host Tools

Standard development suite of tools: gcc, make, etc.

5.3.2 Target System Tools

OpenRISC GNU toolchain. For installation, see OpenRISC GNU toolchain page on OpenCores.org.

5.3.3 EDA Tools

RTL, gatelevel simulation: Mentor Graphics' Modelsim Synthesis: Synopsys Synplify (included in Actel Libero Suite) Backend: Actel Designer (included in Actel Libero Suite) Programming: Actel FlashPRO (included in Actel Libero Suite)

This has been tested with with Libero versions 8.6, 9.0sp1 and 9.1 under Ubuntu Linux. It is recommended the very latest version available be used.

5.3.4 Debug Tools

or_debug_proxy, ORPmon

5.4 Simulating

Run RTL Regression Test

To run the default set of regression tests for the build, run the following command in the board's `sw/run` path.

```
make rtl-tests
```

The same set of options for RTL tests available in the reference design should be available in this build. See [Running A Set Of Specific Reference Design RTL Tests], page 8.

Options specific to the ORDB1A3PE1500 build.

`PRELOAD_RAM`
: Set to '1' to enable loading of the software image into RAM at the beginning of simulation.

  If the chosen bootROM program (set via a define in software header file in the board's `sw/board/include` path) will jump straight to RAM to begin execution, then the software image will need to be in RAM for the simulation to work. This define *must* be used in that case for the simulation to do anything.

5.5 Synthesis

Synthesis of the board port for the Actel technology with the Synplify tool can be run in the board's `syn/synplify/run` path with the following command.

```
make all
```

This will create a EDIF netlist in `syn/synplify/out`.

Hopefully it's all automated enough so that, as long as the design is simulating as desired, the correct set of RTL will be picked up and synthesized without any need for customising scripts for the tool.

5.5.1 Options

The following can be passed as environment variables when running `make all`.

`RTL_TOP` Default's to the designs top level module, *orpsoc_top*, but if wishing to synthesize a particular module, its name (not instantiated name) should be set here.

`FPGA_PART`
: Defaults to A3PE1500 but if targeting any other set it with this.

`FPGA_FAMILY`
: Defaults to the A3PE1500's *ProASIC3E* but if targeting any other set it with this.

`FPGA_PACKAGE`
: Defaults to PQFP208 but if targeting any other set it with this.

`FPGA_SPEED_GRADE`
: Defaults to Std but if targeting any other set it with this.

`FREQ`
: Target frequency for synthesis.

`MAXFAN`
: Maximum net fanout.

`MAXFAN_HARD`
: Hard limit on maximum net fanout.

`GLOBALTHRESH`
: Threshold of fanout before promoting signal to a global routing net.

`RETIMING`
: Defaults to '1' (on) but set to '0' to disable.

`RESOURCE_SHARING`
: Defaults to '1' (on) but set to '0' to disable.

`DISABLE_IO_INSERTION`
: Defaults to '0' (off) but set to '1' to enable. Useful when synthesizing individual modules not intended as a top level.

5.5.2 Checks

The following is a list of some considerations before synthesis.

- bootrom.v

  If the bootROM module is being used to provide the processor with a program at startup, check that board software include file, in the board's `sw/board/include` path, is selecting the correct bootROM program.

  Do a *`make distclean`* from the synthesis run directory to be sure that the previous bootROM file is cleared away and regenerated when synthesis is run.

- Clean away old leftovers

  If the unwanted files from an old synthesis run are still there before the next run, it's best to clean them away with *`make clean`* from the synthesis run directory.

- Check Command Line Options

  If using any command line settings, they can be checked by passing them to the following make target: *`make print-config`*

5.6 Place and Route

Place and route is run from the board's `backend/par/run` path with the following command.

```
make all
```

This will create a `.adb` file in the same path.

All steps, up to and including programming file generation are done here. FPGA device programming must be done using the programming FlashPro tool under Windows if using a free license.

5.6.1 Options

Most of the design's parameters are determined by processing the `orpsoc-defines.v` file and extracting, for example, the frequency of the clocks entering the design.

The following can be passed as environment variables when running *`make all`*.

`FPGA_PART`
: Defaults to A3PE1500 but if targeting any other set it with this.

`FPGA_FAMILY`
: Defaults to the A3PE1500's *ProASIC3E* but if targeting any other set it with this.

`FPGA_PACKAGE`
: Defaults to "208 PQFP" but if targeting any other set it with this.

`FPGA_SPEED_GRADE`
: Defaults to Std but if targeting any other set it with this.

`FPGA_VOLTAGE`
: Defaults to 1.5 but if targeting any other set it with this.

`FPGA_TEMP_RANGE`
: Defaults to COM but if targeting any other set it with this.

`FPGA_VOLT_RANGE`
: Defaults to COM but if targeting any other set it with this.

`PLACE_INCREMENTAL`
: Defaults to off.

`ROUTE_INCREMENTAL`
: Defaults to off.

`PLACER_HIGH_EFFORT`
: Defaults to off.

`BOARD_CONFIG`
: Defaults to `orsoccpuexpio.mkpinassigns`

5.6.2 Constraints

A *synposys design constraints* (SDC) file, and *physical design constraints* (PDC) file are generated automatically by the scripts. The main Verilog defines file is parsed to detect which modules are included in the design, and generates the appropriate constraints which are embedded in the Makefile.

The PDC relies on the `BOARD_CONFIG` environment variable to indicate which pin assignment file to pull into the Makefile (they live in `backend/par/bin`). The PDC also depends on the actual contents of the main place and route Makefile, `backend/par/bin/Makefile`.

By default these should have support for the peripherals included with ORPSoC. Double check, however, that the correct constraints are set, by running the following command before starting place and route.

```
make pdc-file sdc-file
```

These can be generated and edited and then used when running place and route, they will not get replaced.

5.7 Programming File Generation

The `.adb` file resulting from place and route can be opened in the Actel *Designer* tool and a programming file generated there.

Once this programming file is created, Actel's *FlashPro* can used to program the ORDB1A3PE1500 board.

5.8 Customising

The versatile nature of the ORDB1A3PE1500 means the design that goes on it must be equally versatile, if not more so.

The following sections have information on how to configure the design.

5.8.1 Enabling Existing RTL Modules

The design relies on the Verilog HDL *define* function to indicate which modules are included.

There are only a few modules included by default.

- Processor - *or1200*
- Clock and reset generation - *clkgen*
- Bus arbiters - *arbiter_ibus*, *arbiter_dbus*, *arbiter_bytebus*

The rest are optional, depending on what is defined in the board's `rtl/verilog/include/orpsoc-defines.v` file.

Inspect that file to see which modules are able to be included. At present the list includes USB 1.1 host controller and/or slave interface, I2C master/slave core, and SPI master cores.

These cores should be supported and ready to go by just defining them (uncomment in the `orpsoc-defines.v` file.)

5.8.2 Adding RTL Modules

There are a number of steps to take when adding a new module to the design.

- RTL Files

  Create a directory under the board's `rtl/verilog` directory, and name it the same as the top level of the module.

  Ensure the module's top level file and actual name of the module when it will be instantiated are *all the same*.

  Place any include files into the board's `rtl/verilog/include` path.
- Instantiate in ORPSoC Top Level File

  Instantiate the module in the ORPSoC top level file, `rtl/verilog/orpsoc_top/orpsoc_top.v`, and be sure to take care of the following.
  - Create appropriate *'define* in `orpsoc-defines.v` and surround module instantiation with it.
  - Add required I/Os (surrounded by appropriate *'ifdef*)
  - Attach to appropriate bus arbiter, declaring any signals required. Be sure to tie them off if modules is not included.
  - Update appropriate bus arbiter (in board's `rtl/verilog/arbiters` path) adding (uncommenting) additional ports as needed.

  - Update board's `rtl/verilog/include/orpsoc-params.v` file with appropriate set of parameters for new module, as well as arbiter memory mapping assignment.
  - Attach appropriate clocks and resets, modify the board's `rtl/verilog/clkgen/clkgen.v` file generating appropriate clocks if required.
  - Attach any interrupts to the processor's PIC vector in, assigned as the last thing in the file.
- Update ORPSoC Testbench

  Update the board's `bench/verilog/orpsoc_testbench.v` file with appropriate ports (surrounded by appropriate *'ifdef.*)

  Add any desired models to help test the module to the board's `bench/verilog` path and instantiate it correctly in the testbench.
- Add Software Drivers and Tests

  In a similar fashion to what is already in the board's `sw/drivers` and `sw/tests` path, create desired driver and test software to be used during simulation (and potentially on target.)
- Update Backend Scripts

  If any I/O is added, or special timing specified, the board's backend main Makefile, `backend/par/bin/Makefile` and pinout files (in `backend/par/bin` will need to be updated.

  The section in `backend/par/bin/Makefile` mapping signals to Makefile variables will need to have these new signals added to them. The section in the file begins with `$(PDC_FILE):` and is actually a set of long bash lines.

  Continuing the format already there should be easy enough. Remember that the `orpsoc-defines.v` file is parsed and it's possible to tell if the module is included by testing if the variable is defined.

  For example, to add I/Os for a module called `foo`, and in `orpsoc-defines.v` a value `FOO1` is defined, we can add I/Os `foo1_srx_i` and `foo1_tx_o[3:0]` with the following.

```
$(Q)if [ ! -z $$FOO1 ]; then \
        echo "set_io foo1_srx_i " $(FOO_SRX_BUS_SETTINGS) " \
        -pinname "$(FOO1_SRX_PIN) >> $@; \
echo "set_io foo1_tx_o\\[0\\] " $(FOO_TX_BUS_SETTINGS) " \
         -pinname "$(FOO1_TX0_PIN) >> $@; \
echo "set_io foo1_tx_o\\[1\\] " $(FOO_TX_BUS_SETTINGS) "  \
        -pinname "$(FOO1_TX1_PIN) >> $@; \
echo "set_io foo1_tx_o\\[2\\] " $(FOO_TX_BUS_SETTINGS) "  \
        -pinname "$(FOO1_TX2_PIN) >> $@; \
echo "set_io foo1_tx_o\\[3\\] " $(FOO_TX_BUS_SETTINGS) "  \
        -pinname "$(FOO1_TX3_PIN) >> $@; \
fi
```

  (ensure there is no whitespace after the trailing backslashes.)

  It's a little hard to follow, but it's essentially one `set_io` line for each I/O line.

  First the line checks if the module's define was exported (which happens automatically if it's defined in `orpsoc-defines.v`.

Note that what is defined can be checked by running *make print-defines* in the board's `backend/par/run` path.

The values of the bus settings variables depend on the desired I/O standards and other examples in the Makefile can be referenced.

The pin numbers need to be set in the `.mkpinassigns` which is included into the Makefile (according to the `BOARD_CONFIG` variable set when running the `make` command.)

These files are simple assignments of values to variables (and potentially then to other variables) which correspond to the variables finally used in the main Makefile.

The physical constraints file can be generated manually with the *make pdc-file* command.

6 ML501

6.1 Overview

The Xilinx ML501 board contains a Virtex LX50 part, varied memories and peripherals. See Xilinx's site for specific details:

http://www.xilinx.com/products/devkits/HW-V5-ML501-UNI-G.htm

The OR1200 core and Wishbone bus is set to run at 66MHz. The OR1200 core, as defined here, is almost fully featured, with every hardware arithmetic option enabled, and the largest possible cache configuration, of 1024 sets with 8 words per line which is 32 kilobytes of cache each for instruction and data buses.

Not all peripherals are supported, and adding support for each is a goal.

At present the build contains a memory controller for the DDR2 SDRAM (based around a Xilinx MIG derived controller), SSRAM (remains to be verified), and CFI flash. None of the other peripherals (VGA/AC97/PS2/USB/LCD) have controllers in the design yet.

The OpenCores 10/100 Ethernet MAC can be used for Ethernet, but only with the PHY in 10/100 mode using the MII interface to the Marvel Alaska Ethernet PHY IC. There still may be bugs in the FIFO buffer configuration in the ML501's `ethmac_defines.v` file should not be changed.

The ML501 build's scripts are capable of generating either a programming bitstream, `.bit`, file for direct programming of the FPGA via JTAG, or a flash image , `.mcs`, file which can be programmed into the on-board SPI flash which the FPGA can configure itself from on power-on. This flash image file may also contain a software image the processor can load at reset to, for example, present the user with the prompt for a boot monitor at power-on.

This guide is a work in progress, and provides the basics on simulating, building and modifying the design.

6.2 Structure

Note that in this chapter the term *board path* refers to the path in the project for this board port; `boards/xilinx/ml501`.

The board port's structure is similar to that of a standalone project which accords with the OpenCores coding guidelines. However, all software and RTL that is available in the reference design is also available to the board port, with any local (ie. in the board's `rtl` or `sw` paths) versions taking precedence over the versions available in the reference design.

The Verilog RTL specific to this board is under `rtl/verilog` in the board path. The `include` path in there is the place where all required definitions files, configuring the RTL, are found.

The default configuration will result in a design which executes from the RAM at reset. For simulation, the `PRELOAD_RAM` option will need to be used to ensure the processor starts up correctly. To configure and simulate the build to boot from flash, see below.

Backend files, mainly binary NGC files for mapping, are found in the board's `backend/bin` path.

6.2.1 ML501 Xilinx Environment Setup

Ensure the Xilinx environment has been setup before running all scripts for this board build. See Section 11.1 [Xilinx Environment Setup], page 42.

6.3 Tools

6.3.1 Host Tools

Standard development suite of tools: gcc, make, etc.

6.3.2 Target System Tools

OpenRISC GNU toolchain. For installation, see OpenRISC GNU toolchain page on OpenCores.org.

6.3.3 EDA Tools

RTL, gatelevel simulation: Mentor Graphics' Modelsim Synthesis: XST (from Xilinx ISE) Backend: ngdbuild/map/par/bitgen/promgen, etc. (from Xilinx ISE) Programming: iMPACT (from Xilinx ISE)

This has been tested with Xilinx ISE 11.1 under Ubuntu Linux.

6.3.4 Debug Tools

or_debug_proxy, ORPmon

6.4 Simulating

Ensure the Xilinx environment has been setup before running all simulations for this board build. See Section 11.1 [Xilinx Environment Setup], page 42.

Run RTL Regression Test

To run the default set of regression tests for the build, run the following command in the board's `sw/run` path.

```
make rtl-tests
```

The same set of options for RTL tests available in the reference design should be available in this build. See [Running A Set Of Specific Reference Design RTL Tests], page 8.

Options specific to the ML501 build.

`PRELOAD_RAM`

Set to '1' to enable loading of the software image into RAM at the beginning of simulation.

If the chosen bootROM program (set via a define in software header file in the board's `sw/board/include` path) will jump straight to RAM to begin execution, then the software image will need to be in RAM for the simulation to work. This define *must* be used in that case for the simulation to do anything.

6.4.1 Simulating Boot From Flash

By default, the build will boot from RAM, but it is often useful to compile the design for FPGA so that it boots from a large software image in the 32MB CFI flash part (u-boot bootloader or a Linux kernel image, etc.)

It is, then, useful to be able to simulate this.

6.4.1.1 Configure the design

First, ensure the build is configured to boot from the flash memory. See Section 6.10.1 [ML501 boot from flash configuration], page 26.

6.4.1.2 Prepare the image

Next, generate an image to preload into the flash. *Note*: this is not done automatically at present and must be done manually.

Create the executable for the program you wish to boot out of flash. Generate a binary of it using `or32-elf-objcopy` like so:

```
or32-elf-objcopy -O binary yourexe.elf youroutputfile.bin
```

Note: how to create software to execute out of flash on OpenRISC is not covered here, but it's likely an executable intended to execute from RAM will require subtle modifications to the reset vector code and linker script to ensure it executes from flash correctly. Consulting the OpenRISC project wiki is recommended.

Next, turn this into a VMEM image that the flash model can load. To do this use the tool found in the root *`sw/utils`* path called *`bin2vmem`*. Save this resulting VMEM it into the board's *`sim/run`* path and name it *`cfi-flash.vmem`*.

```
.../orpsocv2/sw/utils/bin2vmem youroutputfile.bin -bpw=2 > \
.../orpsocv2/boards/xilinx/ml501/sim/run/cfi-flash.vmem
```

The resulting VMEM file should consist of the program in 2-byte words (the point of the *`-bpw=2`* switch.)

6.4.1.3 Run the simulation

As the software to be run is potentially not from ORPSoC's testsuite, and there's no way to override the name of the test being run in a way that allows things to compile correctly, simply run the following, and all results will be stored in the board's *`sim/out`* path prefixed with the name of the default test, *`or1200-simple`*.

```
make rtl-test
```

The processor will now boot from *`0xf0000100`* and begin executing the image from the flash.

6.5 Synthesis

Synthesis of the board port for the Actel technology with the Synplify tool can be run in the board's `syn/xst/run` path with the following command.

```
make all
```

This will create an NGC file in `syn/xst/run` named `orpsoc.ngc`.

Hopefully it's all automated enough so that, as long as the design is simulating as desired, the correct set of RTL will be picked up and synthesized without any need for customising scripts for the tool.

6.5.1 Options

Use the following command int the `syn/xst/run` path to get a list of the variables used during synthesis. Any can be set on the command line when running `make all`.

```
make print-config
```

6.5.2 Checks

The following is a list of some considerations before synthesis.

- bootrom.v

  If the bootROM module is being used to provide the processor with a program at startup (reset address in processor's define file is set to `0xf0000100` or similar), check that board software include file, in the board's `sw/board/include` path, is selecting the correct bootROM program.

  Do a *`make distclean`* from the synthesis run directory to be sure that the previous bootROM file is cleared away and regenerated when synthesis is run.

- Clean away old leftovers

  If the unwanted files from an old synthesis run are still there before the next run, it's best to clean them away with *`make clean`* from the synthesis run directory.

6.5.3 Netlist generation

To create a Verilog HDL netlist of the post-synthesis design, run the following in the board's `syn/xst/run` path.

```
make orpsoc.v
```

6.6 Place and Route

Place and route of the design can be run from the board's `backend/par/run` path with the following command.

```
make orpsoc.ncd
```

6.7 Post-PAR STA Report

The `trce` tool can be used to generate a timing report of the post-place and route design.

```
make timingreport
```

6.8 Back-annotated Netlist

A post-PAR back-annotated netlist can be generated with the following command.

```
make netlist
```

This will make a new directory under the board's `backend/par/run` path named `netlist` and will contain a Verilog netlist and SDF file with timing information.

6.8.1 Options

To get a list of options that can be set when running the backend flow, run the following in the board's `backend/par/run` path.

```
make print-config
```

6.8.2 Constraints

A Xilinx User Constraints File (UCF) file is in the board's `backend/par/bin` path. It is named `ml501.ucf`. It should be edited if any extra I/O or constraints are required.

Eventually it would be good to dynamically generated this, based on what is included in the design, but for now this must be hand modified if modules are added ore removed from the design.

6.9 Programming File Generation

Programming file generation is run from the board's `backend/par/run` path with the following command.

```
make orpsoc.bit
```

This file can then be loaded into the Xilinx iMPACT program and programmed onto the Virtex 5 part on the ML501.

6.9.1 SPI programming file generation

To generate a file, which can be programmed into the SPI flash on the board (and thus allowing the FPGA to be configured without using iMPACT each time) run the following command in the board's `backend/par/run` path.

```
make orpsoc.mcs
```

6.9.2 SPI programming file generation with software

To generate a file, which can be programmed into the SPI flash on the board (and thus allowing the FPGA to be configured without using iMPACT each time) and also has a bootloader the processor can run (such as ORPmon) run the following command in the board's `backend/par/run` path.

```
make orpsoc.mcs BOOTLOADER_BIN=/path/to/bootloader-binary-file.bin
```

The image is allowed to be up to 256KBytes in size.

As the SPI flash on the ML501 is only 2MBytes in size, and the FPGA configuration image takes up almost 1.5MBytes, the final 256KBytes are reserved for a software image to be loaded at reset by the processor.

This mark (the last 256KBytes of memory) is at hex address `0x1c0000`. This value is passed to the `promgen` tool when creating the `.mcs` file, and is set in the board's `board.h` file so the embedded bootloader in the design knows which address to load from.

If changing the address of the bootloader, to accommodate a larger image for example, be sure to update the address in the `board.h` file and set the environment variable `SPI_BOOTLOADER_SW_OFFSET_HEX` to the hex address to embed the binary image at (hexadecimal value without the leading "`0x`".) Note that changing the address to load from in `board.h` will require the entire design is re synthesized.

The file pointed to by `BOOTLOADER_BIN` should be a binary image with the size of the image embedded in the first word.

The tool `bin2binsizeword` in ORPSoC's `sw/utils` path can add the sizeword to a binary image. A binary image is something created with the processor toolchains `objcopy -O binary` option. A tool like ORPmon is a good candidate for being embedded in the SPI flash as bootloader software - a binary image is automatically created when it's compiled, usually named `orpmon.or32.bin`. To embed that, it would simply need to be passed to the `bin2binsizeword` like the following:

bin2binsizeword /path/to/orpmon/orpmon.or32.bin orpmon-sizeword.bin

This `orpmon-sizeword.bin` file should then be passed via the BOOTLOADER_BIN option when creating the `.mcs` file to embed it.

If once the FPGA configuration image, and a bootloader image is embedded in the SPI flash, the FPGA can be configured with ORPSoC and then the processor can load the bootloader (like ORPmon) with a single press of the board's `PROG` button. This makes developing on the board very easy.

6.9.3 SPI flash programming

There are two ways to program the M25P16 2MByte SPI flash from the Xilinx iMPACT tool - *direct* and *indirect*. Direct programming means the Xilinx programmer has a direct connection from its pins to the SPI bus. It then performs SPI accesses on the bus to erase and program the part. Indirect programming involves the FPGA and sets up connections to the SPI via it. Indirect programming may be slower, but it is the only supported method as of ISE 12 onwards.

There may be a way of programming directly using the open source *xc3sprog* tool, http://sourceforge.net/projects/xc3sprog/ , but the author is yet to figure out how, and would greatly appreciate anyone who can provide a quick rundown on how this could be achieved.

Once programmed, booting from the SPI flash to ORPmon prompt is about 3 to 4 seconds.

6.9.3.1 Direct SPI flash programming

Note: As of ISE 12, direct SPI flash programming is no longer supported. ISE 11 must be used if this method is to be used. Indirect SPI flash programming is the recommended method by Xilinx now. How annoying.

For a guide on how to actually set up the ML501 board to program the SPI flash, see the section under "*My Own SPI Flash Image Demonstration*" on page 26 of the Xilinx UG228 document, http://www.xilinx.com/support/documentation/boards_and_kits/ug228.pdf . Follow steps 1 to 4, and then 9 to 16, and supply the `.mcs` file generated here.

A more general explanation of direct SPI flash programming can be found in XAPP951- http://www.xilinx.com/support/documentation/application_notes/xapp951.pdf

Be sure to set the *CONFIG* switches to `00010101` (left-to-right when Xilinx logo in North-West of board) before attempting to program the SPI flash. The be sure to switch them back to `00000101` before attempting to boot the image.

Note: Direct SPI flash programming will require fly-leads from the Xilinx programming cable to the the board. See page 6 of XAPP1053 for a picture of this for a *different* board,

but to get the idea: http://www.xilinx.com/support/documentation/application_notes/xapp1053.pdf.

Note: If leaving the SPI programming fly leads in place and attempting to boot the image, be sure to remove the `Vref` (`VCC3V3` on JP2) connection before attempting to boot. Be sure the configuration DIP SW15 is set back to the `00000101` position!

Note: The other cable from the programmer (going to the JP1 header) *must* be unplugged from the board before attempting to program the SPI flash.

6.9.3.2 Indirect SPI flash programming

The indirect method of programming the SPI flash has the memory show up as an extrnal module off the FPGA when performing an automatic JTAG boundary scan.

The following page has more information about the steps required. http://www.xilinx.com/support/documer manuals/xilinx11/pim_p_configure_spi_bpi_through_fpga.htm The `.mcs` file required is the one generated in previous steps in this guide.

Note: As we generate the `.mcs` file with bit/byte swapping disabled (with the use of the `-spi` option when running the promgen tool) we must disable iMPACT's automatic bit/byte swapping when programming the SPI flash. In ISE 12 this option is found by going to the *Edit menu -> Preferences*, and in the *Configuration Preferences* category, set the *SPI Byte Swap* option to *Ignore Setting*.

Note: iMPACT from ISE 12 introduced errors in the software image when being programmed. It is advisable that versions of iMPACT from ISEs other than 12 are used until this bug is fixed.

6.9.4 platform flash programming file generation

There are options in the backend makefile to generate programming files for the platform flash found on the ML501.

The following command will make the `orpsoc_platformflash.mcs` file which can be programmed into the flash via the iMPACT tool.

```
boards/xilinx/ml501/backend/par/run$ make orpsoc_platformflash.mcs
```

This flash is designed to be loaded in the high-speed selectMAP mode, as it is formatted in such a way as it should be read out over a 16-bit data bus.

To program the platform flash, launch iMPACT, and initialise the scanchain.

Assign the orpsoc_platformflash.mcs file to the xcf32p part. *Note:* I got a warning about the .mcs file not appearing to be a configuration file and that "This file will not configure a Xilinx device in Serial or SelectMap modes and may not be intended for configuration."

The option where the PROM is slave during programming was left enabled.

6.10 Customising

The large amount of peripherals on the ML501 means that things will want to be added or removed to suit the design.

The following sections have information on how to configure the design.

6.10.1 Enabling Flash Boot Configuration

The following will outline how to make the OR1200 boot from the CFI flash part at reset.

In the board's `rtl/verilog/include/orpsoc-defines.v` ensure the following is uncommented:

```
`define CFI_FLASH
```

In the board's `rtl/verilog/include/or1200_defines.v` ensure the following is uncommented:

```
`define OR1200_BOOT_PCREG_DEFAULT 30'h3c00003f
`define OR1200_BOOT_ADR 32'hf0000100
```

All other `OR1200_BOOT_*` defines nearby should be commented out.

This will ensure that (1) the CFI flash controller is enabled, and (2) the OR1200 will go straight to its reset vector in the flash memory at reset. Note that this will completely bypass the bootrom (at `0xe0000000`) so its configuration now is meaningless. This is on purpose, as any init code can and should be handled by software in the flash (such as u-boot.)

6.10.2 Enabling Existing RTL Modules

The design relies on the Verilog HDL *define* function to indicate which modules are included. See the board's `rtl/verilog/include/orpsoc-defines.v` file to determine which options are enabled by uncommented `` `define `` values.

These `` `defines `` will correspond to defines in the board's top level RTL file `boardpath/rtl/verilog/orpsoc_top/orpsoc_top.v`.

There are only a few modules included by default.

- Processor - *or1200*
- Clock and reset generation - *clkgen*
- Bus arbiters - *arbiter_ibus*, *arbiter_dbus*, *arbiter_bytebus*

The rest are optional, depending on what is defined in the board's `rtl/verilog/include/orpsoc-defines.v` file.

6.10.3 Adding RTL Modules

There are a number of steps to take when adding a new module to the design.

- RTL Files
  - Create a directory under the board's `rtl/verilog` directory, and name it the same as the top level of the module.

    or

    Create a directory under the board's `modules` directory, containing a `rtl/verilog` directory, and name it the same as the top level of the module
  - Ensure the module's top level file and actual name of the module when it will be instantiated are *all the same*.
  - Place any include files into the board's `rtl/verilog/include` path.
- Instantiate in ORPSoC Top Level File

  Instantiate the module in the ORPSoC top level file, `rtl/verilog/orpsoc_top/orpsoc_top.v`, and be sure to take care of the following.

  - Create appropriate *'define* in `orpsoc-defines.v` and surround module instantiation with it.
  - Add required I/Os (surrounded by appropriate *'ifdef*)
  - Attach to appropriate bus arbiter, declaring any signals required. Be sure to tie them off if modules is not included.
  - Update appropriate bus arbiter (in board's `rtl/verilog/arbiters` path) adding (uncommenting) additional ports as needed.
  - Update board's `rtl/verilog/include/orpsoc-params.v` file with appropriate set of parameters for new module, as well as arbiter memory mapping assignment.
  - Attach appropriate clocks and resets, modify the board's `rtl/verilog/clkgen/clkgen.v` file generating appropriate clocks if required.
  - Attach any interrupts to the processor's PIC vector in, assigned as the last thing in the file.
- Update ORPSoC Testbench

  Update the board's `bench/verilog/orpsoc_testbench.v` file with appropriate ports (surrounded by appropriate *'ifdef*.)

  Add any desired models to help test the module to the board's `bench/verilog` path and instantiate it correctly in the testbench.
- Add Software Drivers and Tests

  In a similar fashion to what is already in the board's `sw/drivers` and `sw/tests` path, create desired driver and test software to be used during simulation (and potentially on target.)
- Update Backend Scripts

  If any I/O is added, or special timing specified, the board's UCF file will need updating - see `boardpath/backend/par/bin/ml501.ucf`.

6.11 Running And Debugging Software

6.11.1 Debug Interface

The debug interface uses a separate JTAG tap and some fly-leads must be connected from an *ORSoC USB debugger* (3) to the ML501.

From the USB debugger, a fly lead must take the following signals to the following pins on header J4 on the ML501.

- tdo - HDR2_6
- tdi - HDR2_8
- tms - HDR2_10
- tck - HDR2_12

This corresponds to right-most column of pins on the J4 header, starting on the third row going down.

Supply and ground pins must also be hooked up for the USB debugger.

The left column of pins on J4 are all tied to ground. All pins on J7 (expansion header located adjacent to J4) are all tied to VCC2V5, 2.5V DC, and this is OK for supplying

the buffers on the USB debug cable, and can be used. So essentially put the supply leads anywhere on J7 and ground leads anywhere on the left column of J4.

Once the debug interface is connected, the `or_debug_proxy` application can be used to provide a stub for GDB to connect to. See debugging_physical for more information.

6.11.2 UART

There are 2 ways of connecting to the UART in the design.

One is via the usual serial port connector, P3, on the ML501. This will obviously require a PC with a serial input and appropriate terminal application.

There is also a connection available via the USB debugger mentioned in the previous subsection.

The following pins are used for RX/TX to/from the design on header J4.

- UART RX - HDR2_2
- UART TX - HDR2_4

Again, supply and ground leads for the UART drivers on the USB debugger can be sourced from J7/left-column J4 as per the debug interface subsection.

If both UART and debug interface are connected via the ORSoC USB debugger, this ultimately ends up with the first 2 pins on the right column of J4 as RX/TX for the UART then the JTAG TDO, TDI, TMS and TCK in succession down the right column of J4.

See the ML501 schematic (`http://www.xilinx.com/support/documentation/boards_and_kits/ml501_20061010_bw.pdf`) for more details on these headers, and refer to the pinouts in the ML501 UCF, in the board's `backend/par/bin/ml501.ucf` file.

7 S3ADSP1800

7.1 Overview

The Xilinx XtremeDSP Starter Platform - Spartan-3A DSP 1800A Edition is known as the s3adsp1800 board in ORPSoC.

All information and resources relating to the board were sourced from Xilinx's board portal page:

http://www.xilinx.com/products/boards-and-kits/HW-SD1800A-DSP-SB-UNI-G.htm

The build currently contains the bare essentials to get the board up and running some software on the OpenRISC processor.

The system, at present contains the OpenRISC SoC system running at 25MHz with OR1200, Wishbone B3 arbiters, a Xilinx DDR2 memory controller (125MHz) with Wishbone wrapper and the OpenCores 10/100 Ethernet MAC. Debug capabilities are available via a ORSoC USB debug cable connected to pins on header J8.

The S3ADSP1800 build's scripts are capable of generating a programming bitstream, `.bit`, file for direct programming of the FPGA via JTAG.

7.2 Structure

Note that in this chapter the term *board path* refers to the path in the project for this board port; `boards/xilinx/s3adsp1800`.

The board port's structure is similar to that of a standalone project which accords with the OpenCores coding guidelines. However, all software and RTL that is available in the reference design is also available to the board port, with any local (ie. in the board's `rtl` or `sw` paths) versions taking precedence over the versions available in the reference design.

The Verilog RTL specific to this board is under `rtl/verilog` in the board path. The `include` path in there is the place where all required definitions files, configuring the RTL, are found.

Backend files, mainly binary NGC files for mapping, are found in the board's `backend/bin` path.

7.2.1 S3ADSP1800 Xilinx Environment Setup

Ensure the Xilinx environment has been setup before running all scripts for this board build. See Section 11.1 [Xilinx Environment Setup], page 42.

7.3 Tools

7.3.1 Host Tools

Standard development suite of tools: gcc, make, etc.

7.3.2 Target System Tools

OpenRISC GNU toolchain. For installation, see OpenRISC GNU toolchain page on OpenCores.org.

7.3.3 EDA Tools

RTL, gatelevel simulation: Mentor Graphics' Modelsim Synthesis: XST (from Xilinx ISE) Backend: ngdbuild/map/par/bitgen/promgen, etc. (from Xilinx ISE) Programming: iMPACT (from Xilinx ISE)

This has been tested with Xilinx ISE 13.1 under Ubuntu 11.04.

7.3.4 Debug Tools

or_debug_proxy

7.4 Simulating

Note that if this path is not set, simulations will not compile.

Run RTL Regression Test

To run the default set of regression tests for the build, run the following command in the board's `sw/run` path.

```
make rtl-tests
```

The same set of options for RTL tests available in the reference design should be available in this build. See [Running A Set Of Specific Reference Design RTL Tests], page 8.

Note that no OpenCores 10/100 Ethernet MAC ("ethmac") tests will function correctly for the time being.

Options specific to the S3ADSP1800 build.

`PRELOAD_RAM`

Set to '1' to enable loading of the software image into RAM at the beginning of simulation.

If the chosen bootROM program (set via a define in software header file in the board's `sw/board/include` path) will jump straight to RAM to begin execution, then the software image will need to be in RAM for the simulation to work. This define *must* be used in that case for the simulation to do anything.

7.5 Synthesis

Synthesis of the board port for the Actel technology with the Synplify tool can be run in the board's `syn/xst/run` path with the following command.

```
make all
```

This will create an NGC file in `syn/xst/run` named `orpsoc.ngc`.

Hopefully it's all automated enough so that, as long as the design is simulating as desired, the correct set of RTL will be picked up and synthesized without any need for customising scripts for the tool.

7.5.1 Options

Use the following command int the `syn/xst/run` path to get a list of the variables used during synthesis. Any can be set on the command line when running `make all`.

```
make print-config
```

7.5.2 Checks

The following is a list of some considerations before synthesis.

- bootrom.v

  If the bootROM module is being used to provide the processor with a program at startup (reset address in processor's define file is set to `0xf0000100` or similar), check that board software include file, in the board's `sw/board/include` path, is selecting the correct bootROM program.

  Do a *`make distclean`* from the synthesis run directory to be sure that the previous bootROM file is cleared away and regenerated when synthesis is run.

- Clean away old leftovers

  If the unwanted files from an old synthesis run are still there before the next run, it's best to clean them away with *`make clean`* from the synthesis run directory.

7.5.3 Netlist generation

To create a Verilog HDL netlist of the post-synthesis design, run the following in the board's `syn/xst/run` path.

```
make orpsoc.v
```

7.6 Place and Route

Place and route of the design can be run from the board's `backend/par/run` path with the following command.

```
make orpsoc.ncd
```

7.7 Post-PAR STA Report

The `trce` tool can be used to generate a timing report of the post-place and route design.

```
make timingreport
```

7.8 Back-annotated Netlist

A post-PAR back-annotated netlist can be generated with the following command.

```
make netlist
```

This will make a new directory under the board's `backend/par/run` path named `netlist` and will contain a Verilog netlist and SDF file with timing information.

7.8.1 Options

To get a list of options that can be set when running the backend flow, run the following in the board's `backend/par/run` path.

```
make print-config
```

7.8.2 Constraints

A Xilinx User Constraints File (UCF) file is in the board's `backend/par/bin` path. It is named `s3adsp1800.ucf`. It should be edited if any extra I/O or constraints are required.

Note that if modules are enabled or disabled via their `‘define` line in `orpsoc-defines.v` and they have I/O declared, then this I/O will need to be manually commented out of the UCF to avoid errors during mapping.

7.9 Programming File Generation

Programming file generation is run from the board's `backend/par/run` path with the following command.

```
make orpsoc.bit
```

This file can then be loaded into the Xilinx iMPACT program and programmed onto the Spartan-3A part on the S3ADSP1800.

7.9.1 SPI flash programming

Unfortunately, it is particularly inconvient to program the flash memory that the FPGA can configure itself with at power-on.

The iMPACT tool does not contain the appropriate facility to enable this, and a Windows-only tool provided from Avnet.

See Avnet's page on this board. http://www.em.avnet.com/spartan3a-dsp and follow the links to "Support Files & Downloads" and the file named "Programming the Intel S33 Flash" is the guide. The following link may directly work to download the files: http://tinyurl.com/63k8r5c

Another way is to load on a Microblaze design and somehow program the flash via the Xilinx GUI tool. This method will not be covered here. The XAPP number of the user guide to do this escapes me right now, but I'm pretty sure you don't want to have to do that.

Potentially this flash could be programmed via the SPI core in this board build. This would result in having to flash the FPGA with the s3adsp18000 build image via JTAG first, before downloading SPI programming software with the FPGA configuration embedded in it, to be programmed into the SPI flash. This would probably be easy enough to do but the author did not have enough time to experiment it at the time of writing. Patches to this file's texinfo source describing how this could be done would be greatly appreciated.

7.10 Customising

The large amount of peripherals on the S3ADSP1800 means that things will want to be added or removed to suit the design.

The following sections have information on how to configure the design.

7.10.1 Enabling Existing RTL Modules

The design relies on the Verilog HDL *define* function to indicate which modules are included. See the board's `rtl/verilog/include/orpsoc-defines.v` file to determine which options are enabled by uncommented `‘define` values.

These `‘defines` will correspond to defines in the board's top level RTL file `boardpath/rtl/verilog/orpsoc_top/orpsoc_top.v`.

There are only a few modules included by default.

- Processor - *or1200*
- Clock and reset generation - *clkgen*
- Bus arbiters - *arbiter_ibus*, *arbiter_dbus*, *arbiter_bytebus*

The rest are optional, depending on what is defined in the board's `rtl/verilog/include/orpsoc-defines.v` file.

7.10.2 Adding RTL Modules

There are a number of steps to take when adding a new module to the design.

- RTL Files
  - Create a directory under the board's `rtl/verilog` directory, and name it the same as the top level of the module.

    or

    Create a directory under the board's `modules` directory, containing a `rtl/verilog` directory, and name it the same as the top level of the module
  - Ensure the module's top level file and actual name of the module when it will be instantiated are *all the same*.
  - Place any include files into the board's `rtl/verilog/include` path.
- Instantiate in ORPSoC Top Level File

  Instantiate the module in the ORPSoC top level file, `rtl/verilog/orpsoc_top/orpsoc_top.v`, and be sure to take care of the following.
  - Create appropriate *'define* in `orpsoc-defines.v` and surround module instantiation with it.
  - Add required I/Os (surrounded by appropriate *'ifdef*)
  - Attach to appropriate bus arbiter, declaring any signals required. Be sure to tie them off if modules is not included.
  - Update appropriate bus arbiter (in board's `rtl/verilog/arbiters` path) adding (uncommenting) additional ports as needed.
  - Update board's `rtl/verilog/include/orpsoc-params.v` file with appropriate set of parameters for new module, as well as arbiter memory mapping assignment.
  - Attach appropriate clocks and resets, modify the board's `rtl/verilog/clkgen/clkgen.v` file generating appropriate clocks if required.
  - Attach any interrupts to the processor's PIC vector in, assigned as the last thing in the file.
- Update ORPSoC Testbench

  Update the board's `bench/verilog/orpsoc_testbench.v` file with appropriate ports (surrounded by appropriate *'ifdef.*)

  Add any desired models to help test the module to the board's `bench/verilog` path and instantiate it correctly in the testbench.
- Add Software Drivers and Tests

  In a similar fashion to what is already in the board's `sw/drivers` and `sw/tests` path, create desired driver and test software to be used during simulation (and potentially on target.)

- Update Backend Scripts

  If any I/O is added, or special timing specified, the board's UCF file will need updating - see `backend/par/bin/s3adsp1800.ucf`.

7.11 Running And Debugging Software

This section indicates how to connect a USB JTAG debugger to the board to control the system. At present this setup has only been tested with the ft2232-based ORSoC USB debugger (3).

See the USB debugger documentation and schematics on orsoc.se: http://orsoc.se/usb-jtag-debugger/

Find more information about the Spartan 3 board (schematics, user guide, etc.) on the Xilinx site: http://www.xilinx.com/support/documentation/spartan-3a_dsp_board_and_kit_documentation.htm

7.11.1 Debug Interface

The debug interface uses a separate JTAG tap and some fly-leads must be connected from an *ORSoC USB debugger* to J8 on the S3ADSP1800.

From the USB debugger, a fly lead must take the following signals to the following pins on header J8 on the S3ADSP1800.

- tdo - D0
- tdi - D1
- tms - D2
- tck - D3

Supply and ground pins must also be hooked up for the USB debugger. They can also be found on the J8 header (either V2.5 or V3.3 should work for VCC.)

Once the debug interface is connected, the `or_debug_proxy` application can be used to provide a stub for GDB to connect to. See debugging_physical for more information.

7.11.2 UART

There are 2 ways of connecting to the UART in the design.

One is via the DB89 connector, P2. This will obviously require a PC with a serial input and appropriate terminal application.

There is also a connection available via the USB debugger mentioned in the previous subsection.

The following pins on the J8 are connected to the same UART core as goes to the P2 connector. The two UART RX lines are logically "AND"ed internally.

- UART RX - D4
- UART TX - D5

Again, supply and ground leads for the UART drivers on the USB debugger can be sourced from J8.

8 Atlys

8.1 Overview

The Atlys board is from Digilent and contains a Spartan 6 device.

More informatino can be found on the manufacturer's website: `http://www.digilentinc.com/atlys/`

Note that this board port is based on the ML501 and structure and use are very similar.

8.2 Structure

Note that in this chapter the term *board path* refers to the path in the project for this board port; `boards/xilinx/atlys`.

The board port's structure is similar to that of a standalone project which accords with the OpenCores coding guidelines. However, all software and RTL that is available in the reference design is also available to the board port, with any local (ie. in the board's `rtl` or `sw` paths) versions taking precedence over the versions available in the reference design.

The Verilog RTL specific to this board is under `rtl/verilog` in the board path. The `include` path in there is the place where all required definitions files, configuring the RTL, are found.

Backend files, mainly binary NGC files for mapping, are found in the board's `backend/bin` path.

8.2.1 Atlys Xilinx Environment Setup

Ensure the Xilinx environment has been setup before running all scripts for this board build. See Section 11.1 [Xilinx Environment Setup], page 42.

8.3 Tools

8.3.1 Host Tools

Standard development suite of tools: gcc, make, etc.

8.3.2 Target System Tools

OpenRISC GNU toolchain. For installation, see OpenRISC GNU toolchain page on OpenCores.org.

8.3.3 EDA Tools

RTL, gatelevel simulation: Mentor Graphics' Modelsim Synthesis: XST (from Xilinx ISE) Backend: ngdbuild/map/par/bitgen/promgen, etc. (from Xilinx ISE) Programming: iMPACT (from Xilinx ISE)

This has been tested with Xilinx ISE 11.1 under Ubuntu Linux.

8.3.4 Debug Tools

or_debug_proxy, ORPmon/u-boot

8.4 Simulating

Ensure the Xilinx environment has been setup before running all simulations for this board build. See Section 11.1 [Xilinx Environment Setup], page 42.

Run RTL Regression Test

To run the default set of regression tests for the build, run the following command in the board's `sw/run` path.

```
make rtl-tests
```

The same set of options for RTL tests available in the reference design should be available in this build. See [Running A Set Of Specific Reference Design RTL Tests], page 8.

Options specific to the Atlys build.

`PRELOAD_RAM`

Set to '1' to enable loading of the software image into RAM at the beginning of simulation.

If the chosen bootROM program (set via a define in software header file in the board's `sw/board/include` path) will jump straight to RAM to begin execution, then the software image will need to be in RAM for the simulation to work. This define *must* be used in that case for the simulation to do anything.

8.5 Synthesis

Synthesis of the board port for the Actel technology with the Synplify tool can be run in the board's `syn/xst/run` path with the following command.

```
make all
```

This will create an NGC file in `syn/xst/run` named `orpsoc.ngc`.

Hopefully it's all automated enough so that, as long as the design is simulating as desired, the correct set of RTL will be picked up and synthesized without any need for customising scripts for the tool.

8.5.1 Options

Use the following command int the `syn/xst/run` path to get a list of the variables used during synthesis. Any can be set on the command line when running `make all`.

```
make print-config
```

8.5.2 Checks

The following is a list of some considerations before synthesis.

- bootrom.v

  If the bootROM module is being used to provide the processor with a program at startup (reset address in processor's define file is set to `0xf0000100` or similar), check that board software include file, in the board's `sw/board/include` path, is selecting the correct bootROM program.

  Do a *`make distclean`* from the synthesis run directory to be sure that the previous bootROM file is cleared away and regenerated when synthesis is run.

- Clean away old leftovers

  If the unwanted files from an old synthesis run are still there before the next run, it's best to clean them away with *make clean* from the synthesis run directory.

8.5.3 Netlist generation

To create a Verilog HDL netlist of the post-synthesis design, run the following in the board's `syn/xst/run` path.

```
make orpsoc.v
```

8.6 Place and Route

Place and route of the design can be run from the board's `backend/par/run` path with the following command.

```
make orpsoc.ncd
```

8.7 Post-PAR STA Report

The `trce` tool can be used to generate a timing report of the post-place and route design.

```
make timingreport
```

8.8 Back-annotated Netlist

A post-PAR back-annotated netlist can be generated with the following command.

```
make netlist
```

This will make a new directory under the board's `backend/par/run` path named `netlist` and will contain a Verilog netlist and SDF file with timing information.

8.8.1 Options

To get a list of options that can be set when running the backend flow, run the following in the board's `backend/par/run` path.

```
make print-config
```

8.8.2 Constraints

A Xilinx User Constraints File (UCF) file is in the board's `backend/par/bin` path. It is named `atlys.ucf`. It should be edited if any extra I/O or constraints are required.

Eventually it would be good to dynamically generated this, based on what is included in the design, but for now this must be hand modified if modules are added ore removed from the design.

8.9 Programming File Generation

Programming file generation is run from the board's `backend/par/run` path with the following command.

```
make orpsoc.bit
```

This file can then be loaded into the Xilinx iMPACT program and programmed onto the Spartan 6 part on the Atlys.

9 Generic Designs

9.1 Overview

The paths under `boards/generic` contain designs similar to the reference design, in that they are not technology specific, and used for development of certain features of the processor, or peripherals.

These builds are a TODO, but should provide technology-independent builds, with any specialist modules required to debug, or assist in development or demonstration of a module.

10 Software

This section details the structure of the software library included in ORPSoC.

10.1 Overview

The software provided with ORPSoC is intended to be of sufficient functionality to develop and test the designs, with some additional utility programs for board bring up.

The bulk of the software library consists of drivers and tests for the included RTL modules, focusing on the processor. A basic C library, implementing basic support functions such as printf, is included. This alleviates the prerequisite of a compiler with supporting C library installed.

Each board port may contain additional software drivers and tests in its own software directory, the structure of which mimics that of the main software directory.

10.2 Components

This section outlines the different components of the software library in the `sw/` path in the root of ORPSoC.

10.2.1 Applications

There are some included applications, which are neither drivers or tests.

Typically these will contain a `README` file in their directories which contain information on the software and its use.

In general, these are to be run on hardware, and thus will need to be compiled for a specific board. Be sure to pass the `BOARD` environment variable when compiling to pick up the appropriate board configuration. See Section 10.3 [Software For Board Ports], page 40.

10.2.2 Drivers

Each RTL component may have a driver, which will be compiled into the liborpsoc library and be made available to applications and tests that use the library.

Each driver path should contain its source and an include path for driver headers.

10.2.3 CPU Drivers

An attempt has been made to make the interface to basic CPU functions as generic as possible. This can allow different CPUs to be implemented in ORPSoC.

The header file `cpu-utils.h` should be included to gain access to the CPU driver functions, such as timers, special purpose registers, memory access macros, etc. This header will, in turn, include the appropriate CPU driver header.

Note: What is included in the CPU driver, and how it should be interfaced is not documented yet, but in future every effort should be made to ensure a generic interface to CPU functions is used.

At present only the OR1200 has a driver, but it is intended that alternate OpenRISC processors can be implemented into ORPSoC and a driver for it to be easily used in the library.

The environment variable `CPU_DRIVER` is used to specify which driver is the CPU driver to be used at liborpsoc compile time.

10.2.4 Tests

Each test subdirectory contains directories for each target. Usually there's just `sim` and `board`, the difference between the two being longer run-time and use of UART for board-targeted tests.

Note: Test directory names should not contain hyphens or underscores. Test software files should be named with the single test directory name first, followed by a single word, eg. `or1200-simple.c`.

Test names are referenced using this `module-testname` pair. The automated testing mechanism implemented by the Makefile scripts will always search the `sim` paths for tests, rather than the `board` paths.

Note: There is no automated testing mechanism for the board-targeted software yet. It is anticipated that a testing harness for these will be developed, and we encourage users to help solve this problem.

10.2.5 Library

The `lib` path in the root software directory is where the code for the minimal C library is located, and is the location of the `liborpsoc` archive file after its compilation.

10.2.6 Board

The `board` path in the software directory may, in future, contain other board-specific code, but at present its `include` path houses just an important header, `board.h` used for configuring the software when compiling programs targeted at a specific board port.

This file contains mainly defines of things such as the CPU frequency and timer rate, peripheral base addresses, IRQ numbers, and other board-specific defines. Each board port should contain its own, and is one of the reasons for passing the `BOARD` environment variable when compiling software targeted at a specific board port - so its board-specific defines will be used instead of the reference design's.

10.2.7 Utilities

The `utils` path contains tools used to help manipulate binary software images for a variety of purposes. All tools are designed to be run on the host machine, and not on ORPSoC.

10.3 Software For Board Ports

Each board port will have its own software directory, if only to keep its `board.h` header file, specifying system parameters specific to the board.

It may also contain drivers and tests specific to peripherals for that board.

Note: For any tests or drivers named the same found in both a board's software path and the root software path, the *board's* software will be used instead.

Note: When compiling any software in the *root* software path (such as in the applications folder) intended to run on a particular board, make use of the `BOARD` variable to indicate which board's configuration (`board.h` file, and any board-specific drivers) to use. For example:

```
orpsoc/sw/apps/app1$ make app1.elf BOARD=xilinx/ml501
```

It's also advisable to do a `make distclean` prior to clear out any preexisting libraries that may not contain software appropriate for the targeted board port (it may have been built with the reference design's `board.h`, for example.)

11 EDA tool notes

EDA tool installation, setup and use notes.

11.1 Xilinx Environment Setup

Be sure to execute the Xilinx setup scripts prior to running the ORPSoC scripts of boards using that technology.

Find the script in your Xilinx tool suite install path (the installer tells you where this is at the end of installation) but it should be under the ISE_DS path in recent Xilinx ISE releases.

```
source /opt/Xilinx/13.2/ISE_DS/settings32.sh
```

Note that this affects the *LD_LIBRARY_PATH* environment variable and other programs may have issues caused by running this script.

12 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index