Or1ksim User Guide

Jeremy Bennett
Embecosm Limited
Issue 1 for Or1ksim 2012-04-27

This file documents the OpenRISC Architectural Simulator, Or1ksim.

Published by Embecosm Limited

Table of Contents

Scope of this Document

This document is the user guide for Or1ksim, the OpenRISC 1000 Architectural Simulator.

1 Installation

Installation follows standard GNU protocols.

1.1 Preparation

Unpack the software and create a *separate* directory in which to build it:

```
tar jxf or1ksim-2012-04-27.tar.bz2
mkdir builddir_or1ksim
cd builddir_or1ksim
```

1.2 Configuring the Build

Configure the software using the `configure` script in the main directory.

The most significant argument is `--target`, which should specify the OpenRISC 1000 32-bit architecture. If this argument is omitted, it will default to OpenRISC 1000 32-bit with a warning

```
../or1ksim-2012-04-27/configure --target=or32-elf ...
```

There are several other options available, many of which are standard to GNU `configure` scripts. Use *`configure --help`* to see all the options. The most useful is `--prefix` to specify a directory for installation of the tools.

For testing (using `make check`), the `--target` parameter may be specified, to allow the target tool chain to be selected. If not specified, it will default to `or32-elf`, which is the same prefix used with the standard OpenRISC toolchain installation script.

A number of Or1ksim specific features in the simulator do require enabling at configuration. These include

`--enable-profiling`
`--disable-profiling`
: If enabled, Or1ksim is compiled for profiling with `gprof`. This is disabled by default. Only really of value for developers of Or1ksim.

`--enable-execution=simple`
`--enable-execution=complex`
: Or1ksim has developed to improve functionality and performance. This feature allows three versions of Or1ksim to be built

  `--enable-execution=simple`
  : Build the original simple interpreting simulator

  `--enable-execution=complex`
  : Build a more complex interpreting simulator. Experiments suggest this is 50% faster than the simple simulator. This is the default.

  The default is `--enable-execution=complex`.

`--enable-ethphy`
`--disable-ethphy`
: If enabled, this option allows the Ethernet to be simulated by connecting via a socket (the alternative reads and writes, from and to files). This must then be configured using the relevant fields in the `ethernet` section of the configuration file. See Section 3.4.4 [Ethernet Configuration], page 29.

  The default is for this to be disabled.

`--enable-unsigned-xori`
`--disable-unsigned-xori`

Historically, `l.xori`, has sign extended its operand. This is inconsistent with the other logical opcodes (`l.andi`, `l.ori`), but in the absence of `l.not`, it allows a register to be inverted in a single instruction using:

```
l.xori  rD,rA,-1
```

This flag causes Or1ksim to treat the immediate operand as unsigned (i.e to zero-extend rather than sign-extend).

The default is to sign-extend, so that existing code will continue to work.

> **Caution:** The GNU compiler tool chain makes heavy use of this instruction. Using unsigned behavior will require the compiler to be modified accordingly.
>
> This option is provided for experimentation. A future version of OpenRISC may adopt this more consistent behavior and also provide a `l.not` opcode.

`--enable-range-stats`
`--disable-range-stats`

If enabled, this option allows statistics to be collected to analyse register access over time. The default is for this to be disabled.

`--enable-debug`
`--disable-debug`

This is a feature of the Argtable2 package used to process arguments. If enabled, some debugging features are turned on in Argtable2. It is provided for completeness, but there is no reason why this feature should ever be needed by any Or1ksim user.

`--enable-all-tests`
`--disable-all-tests`

Some of the tests (at the time of writing just one) will not compile without error. If enabled with this flag, all test programs will be compiled with `make check`.

This flag is intended for those working on the test package, who wish to get the missing test(s) working.

A number of configuration flags have been removed since version 0.3.0, because they led to invalid behavior of Or1ksim. Those removed are:

`--enable-arith-flag`
`--disable-arith-flag`

If enabled, this option caused certain instructions to set the flag (`F` bit) in the supervision register if the result were zero. The instructions affected by this were `l.add`, `l.addc`, `l.addi`, `l.and` and `l.andi`.

If set, this caused incorrect behavior. Whether or not flags are set is part of the OpenRISC 1000 architectural specification. The only flags which should set this are the "set flag" instructions: `l.sfeq`, `l.sfeqi`, `l.sfges`, `l.sfgesi`, `l.sfgeu`, `l.sfgeui`, `l.sfgts`, `l.sfgtsi`, `l.sfgtu`, `l.sfgtui`, `l.sfles`, `l.sflesi`, `l.sfleu`, `l.sfleui`, `l.sflts`, `l.sfltsi`, `l.sfltu`, `l.sfltui`, `l.sfne` and `l.sfnei`.

`--enable-ov-flag`
`--disable-ov-flag`

This flag caused certain instructions to set the overflow flag. If not, those instructions would not set the overflow flat. The instructions affected by this were `l.add`, `l.addc`, `l.addi`, `l.and`, `l.andi`, `l.div`, `l.divu`, `l.mul`, `l.muli`, `l.or`, `l.ori`, `l.sll`, `l.slli`, `l.srl`, `l.srli`, `l.sra`, `l.srai`, `l.sub`, `l.xor` and `l.xori`.

This guaranteed incorrect behavior. The OpenRISC 1000 architecture specification defines which flags are set by which instructions.

Within the above list, the arithmetic instructions (`l.add`, `l.addc`, `l.addi`, `l.div`, `l.divu`, `l.mul`, `l.muli` and `l.sub`), together with `l.addic` which is missed out, set the overflow flag. All the others (`l.and`, `l.andi`, `l.or`, `l.ori`, `l.sll`, `l.slli`, `l.srl`, `l.srli`, `l.sra`, `l.srai`, `l.xor` and `l.xori`) do not.

1.3 Building and Installing

Build the tool with:

```
make all
```

If you have the OpenRISC tool chain and DejaGNU installed, you can verify the tool as follows (otherwise omit this step):

```
make check
```

Install the tool with:

```
make install
```

This will install the three variations of the Or1ksim tool, `or32-elf-sim`, `or32-elf-psim` and `or32-elf-mpsim`, the Or1ksim library, '`libsim`', the header file, '`or1ksim.h`' and this documentation in `info` format.

The documentation may be created and installed in alternative formats (PDF, Postscript, DVI, HTML) with for example:

```
make pdf
make install-pdf
```

1.4 Known Problems and Issues

Full details of outstanding issues may be found in the '`NEWS`' file in the main directory of the distribution. The OpenRISC tracker may be used to see the current state of these issues and to raise new problems and feature requests. It may be found at bugtracker.

The following issues are long standing and unlikely to be fixed in Or1ksim in the near future.

- The Supervision Register Little Endian Enable (LEE) bit is ignored. Or1ksim can be built for either little endian or big endian use, but that behavior cannot be changed dynamically.
- Or1ksim is not reentrant, so a program cannot instantiate multiple instances using the library. This is clearly a problem when considering multi-core applications. However it stems from the original design, and can only be fixed by a complete rewrite. The entire source code uses static global constants liberally!

2 Usage

2.1 Standalone Simulator

The general form the standalone command is:

```
or32-elf-sim [-vhiqVt] [-f file] [--nosrv] [--srv=[n]]
                 [-m <n>][-d str]
                 [--enable-profile] [--enable-mprofile] [file]
```

Many of the options have both a short and a long form. For example `-h` or `--help`.

`-v`
`--version`
: Print out the version and copyright notice for Or1ksim and exit.

`-h`
`--help`
: Print out help about the command line options and what they mean.

`-i`
`--interactive`
: After starting, drop into the Or1ksim interactive command shell.

`-q`
`--quiet`
: Do not generate any information messages, only error messages.

`-V`
`--verbose`
: Generate extra output messages (equivalent of specifying the "verbose" option in the simulator configuration section (see see Section 3.2.1 [Simulator Behavior], page 16).

`-t`
`--trace`
: Dump instruction just executed and any register/memory location chaged after each instruction (one line per instruction).

`--trace-physical`
`--trace-virtual`
: When tracing instructions, show the physical address (`--trace-physical`) and/or the virtual address (`--trace-virtual`) of the instruction being executed. Both flags may be specified, in which case both physical and virtual addresses are shown, physical first.

> **Note:** Either or both flags may be specified without `--trace`, to indicate how addresses should be shown if subsequently enabled by a `SIGUSER1` signal or `l.nop 8` opcode (see Section 2.4 [Trace Generation], page 8).

`-f` *`file`*
`--file=`*`file`*
: Read configuration commands from the specified file, looking first in the current directory, and otherwise in the '`$HOME/.or1k`' directory. If this argument is not specified, the file '`sim.cfg`' in those two locations is used. Failure to find the file is a fatal error. See Chapter 3 [Configuration], page 15, for detailed information on configuring Or1ksim.

`--nosrv`
: Do not start up the *Remote Serial Protocol* debug server. This overrides any setting specified in the configuration file. This option may not be specified with `--srv`. If it is, a rude message is printed and the `--nosrv` option is ignored.

`--srv`

`--srv=n` Start up the *Remote Serial Protocol* debug server. This overrides any setting specified in the configuration file. If the parameter, *n*, is specified, use that as the TCP/IP port for the server, otherwise a random value from the private port range (41920-65535) will be used. This option may not be specified with `--nosrv`. If it is, a rude message is printed and the `--nosrv` option is ignored.

`-m size`
`--memory=size`
: Configure a memory block of *size* bytes, starting at address zero. The size may be followed by '`k`', '`K`', '`m`', '`M`', '`g`', '`G`', to indicate kilobytes (2^{10} bytes), megabytes (2^{20} bytes) and gigabytes (2^{30} bytes).

  This is mainly intended for use when Or1ksim is used without a configuration file, to allow just the processor and memory to be set up. This is the equivalent of specifying a configuration memory section with `baseaddr = 0` and `size = size` and all other parameters taking their default value.

  If a configuration file is also used, it should be sure not to specify an overlapping memory block.

`-d config_string`
`--debug-config=config_string`
: Enable selected debug messages in Or1ksim. This parameter is for use by developers only, and is not covered further here. See the source code for more details.

`--report-memory-errors`
: By default all exceptions are now handled silently. If this option is specified, bus exceptions will be reported with a message to standard error indicating the address at which the exception occurred.

  This was the default behaviour up to Or1ksim 0.4.0. This flag is provided for those who wish to keep that behavior.

`--strict-npc`
: In real hardware, setting the next program counter (NPC, SPR 16), flushes the processor pipeline. The consequence of this is that until the pipeline refills, reading the NPC will return zero. This is typically the case when debugging, since the processor is stalled.

  Historically, Or1ksim has always returned the value of the NPC, irrespective of when it is changed. If the `--strict-npc` option is used, then Or1ksim will mirror real hardware more accurately. If the NPC is changed while the processor is stalled, subsequent reads of its value will return 0 until the processor is unstalled.

  This is not currently the default behavior, since tools such as GDB have been implemented assuming the historic Or1ksim behavior. However at some time in the future it will become the default.

`--enable-profile`
: Enable instruction profiling.

`--enable-mprofile`
: Enable memory profiling.

2.2 Profiling Utility

This utility analyses instruction profile data generated by Or1ksim. It may be invoked as a standalone command, or from the Or1ksim CLI. The general form the standalone command is:

```
or32-elf-profile [-vhcq] [-g=file]
```

Many of the options have both a short and a long form. For example `-h` or `--help`.

`-v`
`--version`
Print out the version and copyright notice for the Or1ksim profiling utility and exit.

`-h`
`--help` Print out help about the command line options and what they mean.

`-c`
`--cumulative`
Show cumulative sum of cycles in functions

`-q`
`--quiet` Suppress messages

`-g=`*`file`*
`--generate=`*`file`*
The data file to analyse. If omitted, the default file, '`sim.profile`' is used.

2.3 Memory Profiling Utility

This utility analyses memory profile data generated by Or1ksim. It may be invoked as a standalone command, or from the Or1ksim CLI. The general form the standalone command is:

```
or32-elf-mprofile  [-vh] [-m=m] [-g=n] [-f=file] from to
```

Many of the options have both a short and a long form. For example `-h` or `--help`.

`-v`
`--version`
Print out the version and copyright notice for the Or1ksim memory profiling utility and exit.

`-h`
`--help` Print out help about the command line options and what they mean.

`-m=`*`m`*
`--mode=`*`m`* Specify the mode out output. Permitted options are

`detailed`
`d` Detailed output. This is the default if no mode is specified.

`pretty`
`p` Pretty printed output.

`access`
`a` Memory accesses only.

`width`
`w` Access width only.

`-g=`*`n`*
`--group=`*`n`*
Group 2^n bits of successive addresses together.

`-f=`*`file`*
`--filename=`*`file`*
The data file to analyse. If not specified, the default, '`sim.profile`' is used.

`from`
`to` *from* and *to* are respectively the start and end address of the region of memory to be analysed.

2.4 Trace Generation

An execution trace can be generated at run time with options passed by the command line, or via the operating system's signal passing mechanism, or by `l.nop` opcodes in an application program.

The following flag can be used to create an execution dump.

`-t`
`--trace` Dump instruction just executed and any register/memory location changed after each instruction (one line per instruction). Each line starts with either "S" or "U" to indicate whether the processor was in supervisor or user mode *when the instruction completed*. It is worth bearing in mind that tracing happens at completion of instruction execution and shows the state at that time.

Passing a signal `SIGUSR1` while the simulator is running toggles trace generation. This can be done with the following command, assuming Or1ksim's executable name is `or32-elf-sim`:

```
pkill -SIGUSR1 or32-elf-sim
```

This is useful in the case where trace output is desired after a significant amount of simulation time, where it would be inconvenient to generate trace up to that point.

If the `pkill` utility is not available, the `kill` utility can be used if Or1ksim's process number is known. Use the following to determine the process ID of the `or32-elf-sim` and then send the `SIGUSR1` command to toggle execution trace generation:

```
ps a | grep or32-elf-sim
kill -SIGUSR1 process-number
```

Tracing can also be enabled and disabled from within a target program using the `l.nop 8` and `l.nop 9` opcodes to enable and disable tracing respectively.

By default tracing will show the virtual address of each instruction traced. This may be controlled by two options, `--trace-physical` to show the physical address and/or `--trace-virtual` to show the virtual address. If neither is specified, the virtual address is shown.

> **Note:** Either or both flags may be specified without `--trace`, to indicate how addresses should be shown if subsequently enabled by a `SIGUSER1` signal or `l.nop 8` opcode.

2.5 Simulator Library

Or1ksim may be used as a static of dynamic library, '`libsim.a`' or '`libsim.so`'. When compiling with the static library, the flag, `-lsim` should be added to the link command.

The header file '`or1ksim.h`' contains appropriate declarations of the functions exported by the Or1ksim library. These are:

`int or1ksim_init` (*int* **`argc`**, *char* ***`argv`**, *void* ***`class_ptr`**, *int* (***`upr`**)(*void* ***`class_ptr`**, *unsigned long int* **`addr`**, *unsigned char* **`mask`**[], *unsigned char* **`rdata`**[], *int* **`data_len`**), *int* (***`upw`**)(*void* ***`class_ptr`**, *unsigned long int* **`addr`**, *unsigned char* **`mask`**[], *unsigned char* **`wdata`**[], *int* **`data_len`**)) ['`or1ksim.h`']

The initialization function is supplied with a vector of arguments, which are interpreted as arguments to the standalone version (see see Section 2.1 [Standalone Simulator], page 5), a pointer to the calling class, *class_ptr* (since the library may be used from C++) and two up-call functions, one for reads, *upr*, and one for writes, *upw*.

upw is called for any write to an address external to the model (determined by a `generic` section in the configuration file). *upr* is called for any reads to an external address. The *class_ptr* is passed back with these upcalls, allowing the function to associate the call with the class which originally initialized the library. Both *upw* and *upr* should return zero on success and non-zero otherwise. At the present time the meaning of non-zero values is not defined but this may change in the future.

mask indicates which bytes in the data are to be written or read. Bytes to be read/written should have 0xff set in *mask*. Otherwise the byte should be zero. The adddress, *addr*, is the *full* address, since the upcall function must handle all generic devices, using the full address for decoding.

Endianness is not a concern, since Or1ksim is transferring byte vectors, not multi-byte values.

The result indicates whether the initialization was successful. The integer values are available as an `enum or1ksim`, with possible values `OR1KSIM_RC_OK` and `OR1KSIM_RC_BADINIT`.

> **Caution:** This is a change from versions 0.3.0 and 0.4.0. It further simplifies the interface, and makes Or1ksim more consistent with payload representation in SystemC TLM 2.0.
>
> **Note:** The current implementation of Or1ksim always transfers single words (4 bytes), using masks if smaller values are required. In this it mimcs the behavior of the WishBone bus.

`int or1ksim_run` (*double* ***duration***) [‘`or1ksim.h`’]

Run the simulator for the simulated duration specified (in seconds). A duration of -1 indicates ‘run forever’

The result indicates how the run terminated. The integer values are available as an `enum or1ksim`, with possible values `OR1KSIM_RC_OK` (ran for the full duration), `OR1KSIM_RC_BRKPT` (terminated early due to hitting a breakpoint) and `OR1KSIM_RC_HALTED` (terminated early due to hitting `l.nop 1`).

`void or1ksim_reset_duration` (*double* ***duration***) [‘`or1ksim.h`’]

Change the duration of a run specified in an earlier call to `or1ksim_run`. Typically this is called from an upcall, which realizes it needs to change the duration of the run specified in the call to `or1ksim_run` that has been interrupted by the upcall.

The time specified is the amount of time that the run must continue for (i.e the duration from *now*, not the duration from the original call to `or1ksim_run`).

`void or1ksim_set_time_point` () [‘`or1ksim.h`’]

Set a timing point. For use with `or1ksim_get_time_period`.

`double or1ksim_get_time_period` () [‘`or1ksim.h`’]

Return the simulated time (in seconds) that has elapsed since the last call to `or1ksim_set_time_point`.

`int or1ksim_is_le` () [‘`or1ksim.h`’]

Return 1 (logical true) if the Or1ksim simulation is little-endian, 0 otherwise.

`unsigned long` *int or1ksim_clock_rate* () [‘`or1ksim.h`’]

Return the Or1ksim clock rate (in Hz). This is the value specified in the configuration file.

`void or1ksim_interrupt` (*int* ***i***) [‘`or1ksim.h`’]

Generate an edge-triggered interrupt on interrupt line *i*. The interrupt must be cleared separately by clearing the corresponding bit in the PICSR SPR. Until the interrupt is cleared, any further interrupts on the same line will be ignored with a warning. A warning will be generated and the interrupt request ignored if level sensitive interrupts have been configured with the programmable interrupt controller (see Section 3.3.5 [Interrupt Configuration], page 24).

`void or1ksim_interrupt_set` (*int* **i**) [‘or1ksim.h’]

Assert a level-triggered interrupt on interrupt line *i*. The interrupt must be cleared separately by an explicit call to `or1ksim_interrupt_clear`. Until the interrupt is cleared, any further setting of interrupts on the same line will be ignored with a warning. A warning will be generated, and the interrupt request ignored if edge sensitive interrupts have been configured with the programmable interrupt controller (see Section 3.3.5 [Interrupt Configuration], page 24).

`void or1ksim_interrupt_clear` (*int* **i**) [‘or1ksim.h’]

Clear a level-triggered interrupt on interrupt line *i*, which was previously asserted by a call to `or1ksim_interrupt_set`. A warning will be generated, and the interrupt request ignored if edge sensitive interrupts have been configured with the programmable interrupt controller (see Section 3.3.5 [Interrupt Configuration], page 24).

`double or1ksim_jtag_reset ()` [‘or1ksim.h’]

Drive a reset sequence through the JTAG interface. Return the (model) time taken for this action. Remember that the JTAG has its own clock, which can be an order of magnitude slower than the main clock, so even a reset (5 JTAG cycles) could take 50 processor clock cycles to complete.

`double or1ksim_jtag_shift_ir` (*unsigned char* ***jreg**, *int* **num_bits**) [‘or1ksim.h’]

Shift the supplied register through the JTAG instruction register. Return the (model) time taken for this action. The register is supplied as a byte vector, with the least significant bits in the least significant byte. If the total number of bits is not an exact number of bytes, then the odd bits are found in the least significant end of the highest numbered byte.

For example a 12-bit register would have bits 0-7 in byte 0 and bits 11-8 in the least significant 4 bits of byte 1.

`double or1ksim_jtag_shift_dr` (*unsigned char* ***jreg**, *int* **num_bits**) [‘or1ksim.h’]

Shift the supplied register through the JTAG data register. Return the (model) time taken for this action. The register is supplied as a byte vector, with the least significant bits in the least significant byte. If the total number of bits is not an exact number of bytes, then the odd bits are found in the least significant end of the highest numbered byte.

For example a 12-bit register would have bits 0-7 in byte 0 and bits 11-8 in the least significant 4 bits of byte 1.

`int or1ksim_read_mem` (*unsigned long int* **addr**, *unsigned char* ***buf**, *int* **len**) [‘or1ksim.h’]

Read *len* bytes from *addr*, placing the result in *buf*. Return *len* on success and 0 on failure.

> **Note:** This function was added in Or1ksim 0.5.0.

`int or1ksim_write_mem` (*unsigned long int* **addr**, *const unsigned char* ***buf**, *int* **len**) [‘or1ksim.h’]

Write *len* bytes to *addr*, taking the data from *buf*. Return *len* on success and 0 on failure.

> **Note:** This function was added in Or1ksim 0.5.0.

`int or1ksim_read_spr` (*int* **sprnum**, *unsigned long int* ***sprval_ptr**) [‘or1ksim.h’]

Read the SPR specified by *sprnum*, placing the result in *sprval_ptr*. Return non-zero on success and 0 on failure.

> **Note:** This function was added in Or1ksim 0.5.0.

`int or1ksim_write_spr` (*int* **sprnum**, *unsigned long int* **sprva**) [‘or1ksim.h’]

Write *sprval* to the SPR specified by *sprnum*. Return non-zero on success and 0 on failure.

> **Note:** This function was added in Or1ksim 0.5.0.

`int or1ksim_read_reg` (*int* **`regnum`**, *unsigned long int* *`*regval_ptr`*) [‘`or1ksim.h`’]

Read the general purpose register specified by *regnum*, placing the result in *regval_ptr*. Return non-zero on success and 0 on failure.

> **Note:** This function was added in Or1ksim 0.5.0.

`int or1ksim_write_reg` (*int* **`regnum`**, *unsigned long int* **`regva`**) [‘`or1ksim.h`’]

Write *regval* to the general purpose register specified by *regnum*. Return non-zero on success and 0 on failure.

> **Note:** This function was added in Or1ksim 0.5.0.

`void or1ksim_set_stall_state` (*int* **`state`**) [‘`or1ksim.h`’]

Set the processor’s state according to *state* (1 = stalled, 0 = not stalled).

> **Note:** This function was added in Or1ksim 0.5.0.

The libraries will be installed in the ‘`lib`’ sub-directory of the main installation directory (as specified with the ‘`--prefix`’ option to the `configure` script).

For example if the main installation directory is ‘`/opt/or1ksim`’, the library will be found in the ‘`/opt/or1ksim/lib`’ directory. It is available as both a static library (‘`libsim.a`’) and a shared object (‘`libsim.so`’).

To link against the library add the ‘`-lsim`’ flag when linking and do one of the following:

- Add the library directory to the `LD_LIBRARY_PATH` environment variable during execution. For example:

  ```
  export LD_LIBRARY_PATH=/opt/or1ksim/lib:$LD_LIBRARY_PATH
  ```

- Add the library directory to the `LD_RUN_PATH` environment variable during linking. For example:

  ```
  export LD_RUN_PATH=/opt/or1ksim/lib:$LD_RUN_PATH
  ```

- Use the linker ‘`--rpath`’ option and specify the library directory when linking your program. For example

  ```
  gcc ...  -Wl,--rpath -Wl,/opt/or1ksim/lib ...
  ```

- Add the library directory to ‘`/etc/ld.so.conf`’

2.6 Ethernet TUN/TAP Interface

When an Ethernet peripheral is configured (see Section 3.4.4 [Ethernet Configuration], page 29), one option is to tunnel traffic through a TUN/TAP interface. The low level TAP interface is used to tunnel raw Ethernet datagrams.

The TAP interface can then be connected to a physical Ethernet through a bridge, allowing the Or1ksim model to connect to a physical network. This is particularly when Or1ksim is running the OpenRISC Linux kernel image.

This section explains how to set up a bridge for use by Or1ksim. It does require superuser access to the host machine (or at least the relevant network capabilities). A system administrator can modify these guidelines so they are executed on reboot if appropriate.

2.6.1 Setting Up a Persistent TAP device

TUN/TAP devices can be created dynamically, but this requires superuser privileges (or at least `CAP_NET_ADMIN` capability). The solution is to create a persistent TAP device. This can be done using either `openvpn` or `tunctl`. In either case the package must be installed on the host system.

Using `openvpn`, the following would set up a TAP interface for a specified user and group.

```
openvpn --mktun --dev tapn --user username --group groupname
```

2.6.2 Establishing a Bridge

A bridge is a "virtual" local area network interfaces, subsuming two or more existing network interfaces. In this case we will bridge the physical Ethernet interface of the host with the TAP interface that will be used by Or1ksim.

The Ethernet and TAP must lose their own individual IP addresses (by setting them to 0.0.0.0) and are replaced by the IP address of the bridge interface. To do this we use the `bridge-utils` package, which must be installed on the host system. These commands are require superuser privileges or `CAP_NET_ADMIN` capability. To create a new interface `brn` the following commands are appropriate.

```
brctl addbr brn
brctl addif brn ethx
brctl addif brn tapy

ifconfig ethx 0.0.0.0 promisc up
ifconfig tapy 0.0.0.0 promisc up

dhclient brn
```

The last command instructs the bridge to obtain its IP address, netmask, broadcast address, gateway and nameserver information using DHCP. In a network without DHCP it should be replaced by `ifconfig` to set a static IP address, netmask and broadcast address.

> **Note:** This will leave a spare dhclient process running in the background, which should be killed for tidiness. There is a technique to avoid this using `omshell`, but that is beyond the scope of this guide.
>
> **Note:** It is not clear to the author why the existing interfaces need to be brought up in promiscuous mode, but it seems to cure various problems.

2.6.3 Opening the Firewall

Firewall rules should be added to ensure traffic flows freely through the TAP and bridge interfaces. As superuser the following commands are appropriate.

```
iptables -A INPUT -i tapy -j ACCEPT
iptables -A INPUT -i brn -j ACCEPT
iptables -A FORWARD -i brn -j ACCEPT
```

2.6.4 Disabling Ethernet Filtering

Some systems may have ethernet filtering enabled (`ebtables`, `bridge-nf`, `arptables`) which will stop traffic flowing through the bridge.

The easiest way to disable this is by writing zero to all '`bridge-nf-*`' entries in '`/proc/sys/net/bridge`'. As superuser the following commands will achieve this.

```
cd /proc/sys/net/bridge
for f in bridge-nf-*; do echo 0 > $f; done
```

2.6.5 Networking from OpenRISC Linux and BusyBox

The main use of this style of Ethernet interface to Or1ksim is when running the OpenRISC Linux kernel with BusyBox. The following commands in the BusyBox console window will configure the Ethernet interface (assumed to be `eth0`) and bring it up with a DHCP assigned address.

```
ifconfig eth0
ifup eth0
```

At this stage interface to IP addresses will work correctly.

For DNS to work the BusyBox system needs to know where to find a nameserver. Under BusyBox, `udhcp` does not configure '`/etc/resolv.conf`' automatically.

The solution is to duplicate the nameserver entry from the '`/etc/resolv.conf`' file of the host on the BusyBox system. A typical file might be as follows:

```
nameserver 192.168.0.1
```

It is convenient to make this permanent within the Linux initramfs. Add the file as `arch/openrisc/support/initramfs/etc/resolv.conf` within the Linux source tree and rebuild `vmlinux`. It will then be present automatically.

One of the most useful functions that is possible is to mount the host file system through NFS. For example, from the BusyBox console:

```
mount -t nfs -o nolock 192.168.0.60:/home /mnt
```

Another useful technique is to telnet into the BusyBox system from the host. This is particularly valuable when a console process locks up, since the `xterm` console will not recognize ctrl-C. Instead the rogue process can be killed from a telnet connection.

2.6.6 Tearing Down a Bridge

There is little reason why a bridge should ever need to be torn down, but if desired, the following commands will achieve the effect.

```
ifconfig brn down
brctl delbr brn

dhclient ethx
```

As before this will leave a spare `dhclient` process in the background which should be killed.

If desired the TAP interface can be deleted using

```
openvpn --rmtun -dev tapy
```

Caution: The TAP interface should not be in use when running this command. For example any OpenRISC Linux/BusyBox sessions should be closed first.

2.7 l.nop Opcode Support

The OpenRISC `l.nop` opcode can take a parameter. This has no effect on the semantics of the opcode, but can be used to trigger side effect behavior in a simulator. Within Or1ksim, the following parameters are supported.

`l.nop 0`

The equivalent to `l.nop` with no parameter. Has no side effects.

`l.nop 1`

Execution of Or1ksim is terminated. This is used to implement the library `exit` functions.

`l.nop 2`

Report the value in `r3` on the console as a 32-bit hex value.

`l.nop 3`

In earlier versions of Or1ksim this treated `r3` as a pointer to a `printf` style format string, and registers `r4` through `r8` as parameters for that format string.

This opcode is no longer supported, and has no effect if used.

`l.nop 4`

The value in `r3` is printed to standard output as an ASCII character. All library output routines are implemented using this opcode.

`l.nop 5`
: The statistics counters are reset.

`l.nop 6`
: The number of clock ticks since start of execution (a 64-bit value) is returned in `r11` (low 32 bits) and `r12` (high 32 bits).

`l.nop 7`
: The number of picoseconds per clock cycle is returned in `r11`. This is used with `l.nop 6` to implement timing functions.

`l.nop 8`
: Instruction tracing is turned on.

`l.nop 9`
: Instruction tracing is turned off.

`l.nop 10`
: A 32-bit random number is returned in `r11`.

The random numbers are generated using `random`, which in turn is seeded through `srandom` using the host '`/dev/urandom`' if available, or else the process ID of the Or1ksim instance.

This opcode is particularly useful for situations where a target program running on Or1ksim needs to obtain genuine system entropy to generate random numbers.

`l.nop 11`
: Return a non-zero value in `r11`.

This opcode can be used to detect if a target is running under Or1ksim. Set `r11` to zero, issue this opcode, and look to see if `r11` is non-zero.

3 Configuration

Or1ksim is configured through a configuration file. This is specified through the `-f` parameter to the Or1ksim command, or passed as a string when initializing the Or1ksim library. If no file is specified, the default '`sim.cfg`' is used. The file is looked for first in the current directory, then in the '`$HOME/.or1ksim`' directory of the user.

3.1 Configuration File Format

The configuration file is a plain text file. A reference example, '`sim.cfg`', is included in the top level directory of the distribution.

3.1.1 Configuration File Preprocessing

The configuration file may include C style comments (i.e. delimited by `/*` and `*/`).

3.1.2 Configuration File Syntax

The configuration file is divided into a series of sections, with the general form:

```
section section_name

  <contents>...

end
```

Sections may also have sub-sections within them (currently only the ATA/ATAPI disc interface uses this).

Within a section, or sub-section are a series of parameter assignments, one per line, withe the general form

```
  parameter = value
```

Depending on the parameter, the value may be a named value (an enumeration), an integer (specified in any format acceptable in C) or a string in doubple quotes. For flag parameters, the value 1 is used to mean "true" or "on" and the value "0" to mean "false" or "off". An example from a memory section shows each of these

```
section memory
  type    = random
  pattern = 0x00
  name    = "FLASH"
  ...
end
```

Many parameters are optional and take reasonable default values if not specified. However there are some parameters (for example the `ce` parameter in `section memory`) *must* be specified.

Subsections are introduced by a keyword, with a parameter value (no `=` sign), and end with the same keyword prefixed by `end`. Thus the ATA/ATAPI inteface (`section ata`) has a `device` subsection, thus:

```
section ata
  ...
  device 0
    type    = 1
    file = "filename"
    ...
  enddevice
  ...
```

```
end
```

Some sections (for example `section sim`) should appear only once. Others (for example `section memory` may appear multiple times.

Sections may be omitted, *unless they contain parameters which are non-optional*. If the section describes a part of the simulator which is optional (for example whether it has a UART), then that functionality will not be provided. If the section describes a part of the simulator which is not optional (for example the CPU), then all the parameters of that section will take their default values.

All optional parts of the functionality are always described by sections including a `enabled` parameter, which can be set to 0 to ensure that functionality is explicitly omitted.

Even if a section is disabled, all its parameters will be read and stored. This is helpful if the section is subsequently enabled from the Or1ksim command line (see Chapter 4 [Interactive Command Line], page 37).

> **Tip:** It generally clearer to have sections describing *all* components, with omitted functionality explicitly indicated by setting the `enabled` parameter to 0

The following sections describe the various configuration sections and the parameters which may be set in each.

3.2 Simulator Configuration

3.2.1 Simulator Behavior

Simulator behavior is described in `section sim`. This section should appear only once. The following parameters may be specified.

`verbose = 0|1`
: If 1 (true), print extra messages. Default 0.

`debug = 0-9`
: 0 means no debug messages. 1-9 means produce debug messages. The higher the value the greater the number of messages. Default 0. Negative values will be treated as 0 (with a warning). Values that are too large will be treated as 9 (with a warning).

`profile = 0|1`
: If 1 (true) generate a profiling file using the file specified in the `prof_file` parameter or otherwise '`sim.profile`'. Default 0.

`prof_file = ``*filename*''`
: Specifies the file to be used with the `profile` parameter. Default '`sim.profile`'. For backwards compatibility, the alternative name `prof_fn` is supported for this parameter, but deprecated. Default '`sim.profile`'.

`mprofile = 0|1`
: If 1 (true) generate a memory profiling file using the file specified in the `mprof_file` parameter or otherwise '`sim.mprofile`'. Default 0.

`mprof_file = ``*filename*''`
: Specifies the file to be used with the `mprofile` parameter. Default '`sim.mprofile`'. For backwards compatibility, the alternative name `mprof_fn` is supported for this parameter, but deprecated. Default '`sim.mprofile`'.

`history = 0|1`
: If 1 (true) track execution flow. Default 0.

> **Note:** Setting this parameter seriously degrades performance.

> **Note:** If this execution flow tracking is enabled, then `dependstats` must be enabled in the CPU configuration section (see Section 3.3.1 [CPU Configuration], page 19).

`exe_log = 0|1`
: If 1 (true), generate an execution log. Log is written to the file specified in parameter `exe_log_file`. Default 0.

> **Note:** Setting this parameter seriously degrades performance.

`exe_log_type = default|hardware|simple|software`
: Type of execution log to produce.

  `default` Produce default output for the execution log. In the current implementation this is the equivalent of `hardware`.

  `hardware` After each instruction execution, log the number of instructions executed so far, the next instruction to execute (in hex), the general purpose registers (GPRs), status register, exception program counter, exception, effective address register and exception status register.

  `simple` After each instruction execution, log the number of instructions executed so far and the next instruction to execute, symbolically disassembled.

  `software` After each instruction execution, log the number of instructions executed so far and the next instruction to execute, symbolically disassembled. Also show the value of each operand to the instruction.

  Default value `hardware`. Any unrecognized keyword (case insensitive) will be treated as the default with a warning.

> **Note:** Execution logs can be *very* big.

`exe_log_start =` *`value`*
: Address of the first instruction to start logging. Default 0.

`exe_log_end =` *`value`*
: Address of the last instruction to log. Default no limit (i.e once started logging will continue until the simulator exits).

`exe_log_marker =` *`value`*
: Specifies the number of instructions between printing horizontal markers. Default is to produce no markers.

`exe_log_file =` *`filename`*
: Filename for the execution log filename if `exe_log` is enabled. Default '`executed.log`'. For backwards compatibility, the alternative name `exe_log_fn` is supported for this parameter, but deprecated.

`exe_bin_insn_log = 0|1`
: Enable logging of executed instructions to a file in binary format. This is helpful for off-line dynamic execution analysis.

> **Note:** Execution logs can be *very* big. For example, while booting the Linux kernel, version 2.6.34, a log file 1.2GB in size was generated.

`exe_bin_insn_log_file =` *`filename`*
: Filename for the binary execution log filename if `exe_bin_insn_log` is enabled. Default '`exe-insn.bin`'.

`clkcycle =` *`value`*`[ps|ns|us|ms]`
: Specify the time taken by one clock cycle. If no units are specified, `ps` is assumed. Default 4000ps (250MHz).

3.2.2 Verification API (VAPI) Configuration

The Verification API (VAPI) provides a TCP/IP interface to allow components of the simulation to be controlled externally. See Chapter 5 [Verification API], page 40, for more details.

Verification API configuration is described in `section vapi`. This section may appear at most once. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true), verification API is enabled and its server started. If 0 (the default), it is disabled.

`server_port = *value*`
: When VAPI is enabled, communication will be via TCP/IP on the port specified by *value*. The value must lie in the range 1 to 65535. The default value is 50000.

> **Tip:** There is no registered port for Or1ksim VAPI. Good practice suggests users should adopt port values in the *Dynamic* or *Private* port range, i.e. 49152-65535.

`log_enabled = 0|1`
: If 1 (true), all VAPI requests and sent commands will be logged. If 0 (the default), logging is diabled. Logs are written to the file specified by the `vapi_log_file` field (see below).

> **Caution:** This can generate a substantial amount of file I/O and seriously degrade simulator performance.

`hide_device_id = 0|1`
: If 1 (true) don't log the device ID. If 0 (the default), log the device ID. This feature (when set to 1) is provided for backwards compatibility with an old version of VAPI.

`vapi_log_file = "*filename*"`
: Use '`filename`' as the file for logged data is logging is enabled (see `log_enabled` above). The default is `"vapi.log"`. For backwards compatibility, the alternative name `vapi_log_fn` is supported for this parameter, but deprecated.

3.2.3 Custom Unit Compiler (CUC) Configuration

The Custom Unit Compiler (CUC) was a project by Marko Mlinar to generate Verilog from ANSI C functions. The project seems to not have progressed beyond the initial prototype phase. The configuration parameters are described here for the record.

CUC configuration is described in `section cuc`. This section may appear at most once. The following parameters may be specified.

`memory_order = none|weak|strong|exact`
: This parameter specifies the memory ordering required:

  `memory_order=none`
  : Different memory ordering, even if there are dependencies. Bursts can be made, width can change.

  `memory_order=weak`
  : Different memory ordering, even if there are dependencies. If dependencies cannot occur, then bursts can be made, width can change.

  `memory_order=strong`
  : Same memory ordering. Bursts can be made, width can change.

  `memory_order=exact`
  : Exactly the same memory ordering and widths.

The default value is `memory_order=exact`. Invalid memory orderings are ignored with a warning.

`calling_convention = 0|1`
: If 1 (true), programs follow OpenRISC calling conventions. If 0 (the default), they may use other convenitions.

`enable_bursts = 0 | 1`
: If 1 (true), bursts are detected. If 0 (the default), bursts are not detected.

`no_multicycle = 0 | 1`
: If 1 (true), no multicycle logic paths will be generated. If 0 (the default), multicycle logic paths will be generated.

`timings_file = "`*`filename`*`"`
: *filename* specifies a file containing timing information. The default value is `"virtex.tim"`. For backwards compatibility, the alternative name `timings_fn` is supported for this parameter, but deprecated.

3.3 Configuring the OpenRISC Architectural Components

3.3.1 CPU Configuration

CPU configuration is described in `section cpu`. This section should appear only once. At present Or1ksim does not model multi-CPU systems. The following parameters may be specified.

`ver = `*`value`*
`cfg = `*`value`*
`rev = `*`value`*
: The values are used to form the corresponding fields in the `VR` Special Purpose Register (SPR 0). Default values 0. A warning is given and the value truncated if it is too large (8 bits for `ver` and `cfg`, 6 bits for `rev`).

`upr = `*`value`*
: Used as the value of the Unit Present Register (UPR) Special Purpose Register (SPR 1) to *value*. Default value is 0x0000075f, i.e.

- UPR present (0x00000001)
- Data cache present (0x00000002)
- Instruction cache present (0x00000004)
- Data MMU present (0x00000008)
- Instruction MMU present (0x00000010)
- Debug unit present (0x00000040)
- Power management unit present (0x00000100)
- Programmable interrupt controller present (0x00000200)
- Tick timer present (0x00000400)

However, with the exection of the UPR present (0x00000001) and tick timer present, the various fields will be modified with the values specified in their corresponding configuration sections.

`cfgr = `*`value`*
: Sets the CPU configuration register (Special Purpose Register 2) to *value*. Default value is 0x00000020, i.e. support for the ORBIS32 instruction set. Attempts to set any other value are accepted, but issue a warning that there is no support for the instruction set.

`sr = value`
: Sets the supervision register Special Purpose Register (SPR 0x11) to *value*. Default value is 0x00008001, i.e. start in supervision mode (0x00000001) and set the “Fixed One” bit (0x00008000).

> **Note:** This is particularly useful when an image is held in Flash at high memory (0xf0000000). The EPH bit can be set, so that interrupt vectors are basedf at 0xf0000000, rather than 0x0.

`superscalar = 0|1`
: If 1, the processor operates in superscalar mode. Default value is 0.

In the current simulator, the only functional effect of superscalar mode is to affect the calculation of the number of cycles taken to execute an instruction.

> **Caution:** The code for this does not appear to be complete or well tested, so users are advised not to use this option.

`hazards = 0|1`
: If 1, data hazards are tracked in a superscalar CPU. Default value is 0.

In the current simulator, the only functional effect is to cause logging of hazard waiting information if the CPU is superscalar. However nowhere in the simulator is this data actually computed, so the net result is probably to have no effect.

if harzards are tracked, current hazards can be displayed using the simulator’s `r` command.

> **Caution:** The code for this does not appear to be complete or well tested, so users are advised not to use this option.

`dependstats = 0|1`
: If 1, inter-instruction dependencies are calculated. Default value 0.

If these values are calculated, the depencies can be displayed using the simulator’s `stat` command.

> **Note:** This field must be enabled, if execution execution flow tracking (field `history`) has been requested in the simulator configuration section (see Section 3.2.1 [Simulator Behavior], page 16).

`sbuf_len = value`
: The length of the store buffer is set to *value*, which must be no greater than 256. Larger values will be truncated to 256 with a warning. Negative values will be treated as 0 with a warning. Use 0 to disable the store buffer.

When the store buffer is active, stores are accumulated and committed when I/O is idle.

`hardfloat = 0|1`
: If 1, hardfloat instructions are enabled. Default value 0.

3.3.2 Memory Configuration

Memory configuration is described in `section memory`. This section may appear multiple times, specifying multiple blocks of memory.

> **Caution:** The user may choose whether or not to enable a memory controller. If a memory controller is enabled, then appropriate initalization code must be provided. The section describing memory controller configuration describes the steps necessary for using smaller or larger memory sections (see Section 3.4.1 [Memory Controller Configuration], page 26).
>
> The *uClibc* startup code initalizes a memory controller, assumed to be mapped at 0x93000000. If a memory controller is *not* enabled, then the standard C library

code will generate memory access errors. The solution is to declare an additional writable memory block, mimicing the memory controller's register bank as follows.

```
section memory
  pattern = 0x00
  type = unknown
  name = "MC shadow"
  baseaddr = 0x93000000
  size     = 0x00000080
  delayr = 2
  delayw = 4
end
```

The following parameters may be specified.

`type=random|pattern|unknown|zero|exitnops`
: Specifies the values to which memory should be initialized. The default value is `unknown`.

  `random` Set the memory values to be a random value. A seed for the random generator may be set using the `random_seed` field in this section (see below), thus ensuring the same "random" values are used each time.

  `pattern` Set the memory values to be a pattern value, which is set using the `pattern` field in this section (see below).

  `unknown` The memory values are not initialized (i.e. left "unknown"). This option will yield faster initialization of the simulator. This is the default.

  `zero` Set the memory values to be 0. This is the equivalent of `type=pattern` and a `pattern` value of 0, and implemented as such.

  > **Note:** As a consequence, if the `pattern` field is *subsequently* specified in this section, the value in that field will be used instead of zero to initialize the memory.

  `exitnops` Set the memory values to be an instruction used to signal end of simulation. This is useful for causing immediate end of simulation when PC corruption occurs.

`random_seed = `*`value`*
: Set the seed for the random number generator to *value*. This only has any effect for memory type `random`.

  The default value is -1, which means the seed will be set from a call to the `time` function, thus ensuring different random values are used on each run. The simulator prints out the seed used in this case, allowing repeat runs to regenerate the same random values used in any particular run.

`pattern = `*`value`*
: Set the pattern to be used when initializing memory to *value*. The default value is 0. This only has any effect for memory type `pattern`. The least significant 8 bits of this value is used to initialize each byte. More than 8 bits can be specified, but will ignored with a warning.

  > **Tip:** The default value, is equivalent to setting the memory `type` to be `zero`. If that is what is intended, then using `type=zero` explicitly is better than using `type=pattern` and not specifying a value for `pattern`.

`baseaddr = `*`value`*
: Set the base address of the memory to *value*. It should be aligned to a multiple of the memory size rounded up to the nearest 2^n. The default value is 0.

`size = value`
Set the size of the memory block to be *value* bytes. This should be a multiple of 4 (i.e. word aligned). The default value is 1024.

Note: When allocating memory, the simulator will allocate the nearest 2^n bytes greater than or equal to *value*, and will not notice memory misses in any part of the memory between *value* and the amount allocated.

As a consequence users are strongly recommended to specify memory sizes that are an exact power of 2. If some other amount of memory is required, it should be specified as separate, contiguous blocks, each of which is a power of 2 in size.

`name = "text"`
Name the block. Typically these describe the type of memory being modeled (thus `"SRAM"` or `"Flash"`. The default is `"anonymous memory block"`.

Note: It is not clear that this information is currently ever used in normal operation of the simulator. Even the `info` command of the simulator ignores it.

`ce = value`
Set the chip enable index of the memory instance. Each memory instance should have a unique chip enable index, which should be greater than or equal to zero. This is used by the memory controller when identifying different memory instances.

There is no requirement to set `ce` if a memory controller is not enabled. The default value is -1 (invalid).

`mc = value`
Specifies the memory controller this memory is connected to. It should correspond to the `index` field specified in a `section mc` for a memory controller (see Section 3.4.1 [Memory Controller Configuration], page 26).

There is no requirement to set `mc` if a memory controller is not enabled. Default value is 0, which is also the default value of a memory controller `index` field. This is suitable therefore for designs with just one memory controller.

`delayr = value`
The number of cycles required for a read access. Set to -1 if the memory does not support reading. Default value 1. The simulator will add this number of cycles to the total instruction cycle count when reading from main memory.

`delayw = value`
The number of cycles required for a write access. Set to -1 if the memory does not support writing. Default value 1. The simulator will add this number of cycles to the total instruction cycle count when writing to main memory.

`log = "file"`
If specified, '`file`' names a file for all memory accesses to be logged. If not specified, the default value, NULL is used, meaning that the memory is not logged.

3.3.3 Memory Management Configuration

Memory Management Unit (MMU) configuration is described in `section dmmu` (for the data MMU) and `section immu` (for the instruction MMU). Each section should appear at most once. The following parameters may be specified.

`enabled = 0|1`
If 1 (true), the data or instruction (as appropriate) MMU is enabled. If 0 (the default), it is disabled.

`nsets = value`
Sets the number of data or instruction (as appropriate) TLB sets to *value*, which must be a power of two, not exceeding 128. Values which do not fit these criteria are ignored with a warning. The default value is 1.

`nways = value`
Sets the number of data or instruction (as appropriate) TLB ways to *value*. The value must be in the range 1 to 4. Values outside this range are ignored with a warning. The default value is 1.

`pagesize = value`
The data or instruction (as appropriate) MMU page size is set to *value*, which must be a power of 2. Values which are not a power of 2 are ignored with a warning. The default is 8192 (0x2000).

`entrysize = value`
The data or instruction (as appropriate) MMU entry size is set to *value*, which must be a power of 2. Values which are not a power of 2 are ignored with a warning. The default value is 1.

> **Note:** Or1ksim does not appear to use the `entrysize` parameter in its simulation of the MMUs. Thus setting this value does not seem to matter.

`ustates = value`
The number of instruction usage states for the data or instruction (as appropriate) MMU is set to *value*, which must be 2, 3 or 4. Values outside this range are ignored with a warning. The default value is 2.

> **Note:** Or1ksim does not appear to use the `ustates` parameter in its simulation of the MMUs. Thus setting this value does not seem to matter.

`hitdelay = value`
Set the number of cycles a data or instruction (as appropriate) MMU hit costs. Default value 1.

`missdelay = value`
Set the number of cycles a data or instruction (as appropriate) MMU miss costs. Default value 1.

3.3.4 Cache Configuration

Cache configuration is described in `section dc` (for the data cache) and `seciton ic` (for the instruction cache). Each section should appear at most once. The following parameters may be specified.

`enabled = 0|1`
If 1 (true), the data or instruction (as appropriate) cache is enabled. If 0 (the default), it is disabled.

`nsets = value`
Sets the number of data or instruction (as appropriate) cache sets to *value*, which must be a power of two, not exceeding `MAX_DC_SETS` (for the data cache) or `MAX_IC_SETS` (for the instruction cache). At the time of writing, these constants are both defined in the code to be 1024). The default value is 1.

`nways = value`
Sets the number of data or instruction (as appropriate) cache ways to *value*, which must be a power of two, not exceeding `MAX_DC_WAYS` (for the data cache) or `MAX_IC_`

WAYS (for the instruction cache). At the time of writing, these constants are both defined in the code to be 32). The default value is 1.

`blocksize = value`
The data or instruction (as appropriate) cache block size is set to *value* bytes, which must be either 16 or 32. The default is 16.

`ustates = value`
The number of instruction usage states for the data or instruction (as appropriate) cache is set to *value*, which must be 2, 3 or 4. The default value is 2.

`hitdelay = value`
Instruction cache only. Set the number of cycles an instruction cache hit costs. Default value 1.

`missdelay = value`
Instruction cache only. Set the number of cycles an instruction cache miss costs. Default value 1.

`load_hitdelay = value`
Data cache only. Set the number of cycles a data load cache hit costs. Default value 2.

`load_missdelay = value`
Data cache only. Set the number of cycles a data load cache miss costs. Default value 2.

`store_hitdelay = value`
Data cache only. Set the number of cycles a data store cache hit costs. Default value 0.

`store_missdelay = value`
Data cache only. Set the number of cycles a data store cache miss costs. Default value 0.

3.3.5 Interrupt Configuration

Programmable Interrupt Controller (PIC) configuration is described in `section pic`. This section may appear at most once—Or1ksim has no mechanism for handling multiple interrupt controllers. The following parameters may be specified.

`enabled = 0|1`
If 1 (true), the programmable interrupt controller is enabled. If 0 (the default), it is disabled.

`edge_trigger = 0|1`
If 1 (true, the default), the programmable interrupt controller is edge triggered. If 0 (false), it is level triggered.

The library interface (see Section 2.5 [Simulator Library], page 8) provides different functions for setting the different types of interrupt, and a function to clear level sensitive interrupts. Edge sensitive interrupts must be cleared by clearing the corresponding bit in the PICSR SPR.

Internal functions to set and clear interrupts are also provided for peripherals implemented within Or1ksim. See [Interrupts Internal], page 44 for more details.

`use_nmi = 0|1`
If 1 (true, the default), interrupt lines 0 and 1 are non-maskable. In other words the least significant 2 bits of the PICMR SPR are hard-wired to 1. If 0 (false), all interrupt lines are treated as equivalent.

> **Note:** These are not non-maskable in the true sense that they will pre-empt other interrupts. Rather they can never be masked out using the PICMR register. It is up the interrupt exception handler to give these interrupt lines priority, and indeed to decide on the priority order in general.

3.3.6 Power Management Configuration

Power management implementation is incomplete. At present the effect (which only happens when the power management unit is enabled) of setting the different bits in the power management Special Purpose Register (PMR, SPR 0x4000) is

`SDF (bit mask 0x0000000f)`
: No effect - these bits are ignored

`DME (bit mask 0x00000010)`
`SME (bit mask 0x00000020)`
: Both these bits cause the processor to stop executing instructions. However all other functions (debug interaction, CLI, VAPI etc) carry on as normal.

`DCGE (bit mask 0x00000004)`
: No effect - this bit is ignored

`SUME (bit mask 0x00000008)`
: Enabling this bit causes a message to be printed, advising that the processor is suspending and the simulator exits.

On reset all bits are cleared.

Power management configuration is described in `section pm`. This section may appear at most once. The following parameter may be specified.

`enabled = 0|1`
: If 1 (true), power management is enabled. If 0 (the default), it is disabled.

3.3.7 Branch Prediction Configuration

From examining the code base, it seems the branch prediction function is not fully implemented. At present the functionality seems restricted to collection of statistics.

Branch prediction configuration is described in `section bpb`. This section may appear at most once. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true), branch prediction is enabled. If 0 (the default), it is disabled.

`btic = 0|1`
: If 1 (true), the branch target instruction cache model is enabled. If 0 (the default), it is disabled.

`sbp_bf_fwd = 0|1`
: If 1 (true), use forward prediction for the `l.bf` instruction. If 0 (the default), do not use forward prediction for this instruction.

`sbp_bnf_fwd = 0|1`
: If 1 (true), use forward prediction for the `l.bnf` instruction. If 0 (the default), do not use forward prediction for this instruction.

`hitdelay = *value*`
: Set the number of cycles a branch prediction hit costs. Default value 0.

`missdelay = *value*`
: Set the number of cycles a branch prediction miss costs. Default value 0.

3.3.8 Debug Interface Configuration

The debug unit and debug interface configuration is described in `section debug`. This section may appear at most once. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true), the debug unit is enabled. If 0 (the default), it is disabled.

> **Note:** This enables the functionality of the debug unit (its registers etc) within the mode. It does not provide any external interface to the debug unit. For that, see `rsp_enabled` below.

`rsp_enabled = 0|1`
: If 1 (true), the GDB *Remote Serial Protocol* server is started, provding an interface to an external GNU debugger, using the port specified in the `rsp_port` field (see below), or the `or1ksim-rsp` TCP/IP service. If 0 (the default), the server is not started, and no external interface is provided.

For more detailed information on the interface to the GNU Debugger see Embecosm Application Note 2, *Howto: Porting the GNU Debugger Practical Experience with the OpenRISC 1000 Architecture*, by Jeremy Bennett, published by Embecosm Limited (`www.embecosm.com`).

`rsp_port = value`
: *value* specifies the port to be used for the GDB *Remote Serial Protocol* interface to the GNU Debugger (GDB). Default value 51000. If the value 0 is specified, Or1ksim will instead look for a TCP/IP service named `or1ksim-rsp`.

> **Tip:** There is no registered port for Or1ksim *Remote Serial Protocol* service `or1ksim-rsp`. Good practice suggests users should adopt port values in the *Dynamic* or *Private* port range, i.e. 49152-65535.

`vapi_id = value`
: *value* specifies the value of the Verification API (VAPI) base address to be used with the debug unit. See Chapter 5 [Verification API], page 40, for more details.

If this is specified and *value* is non-zero, all OpenRISC Remote JTAG protocol transactions will be logged to the VAPI log file, if enabled. This is the only functionality associated with VAPI for the debug unit. No VAPI commands are sent, nor requests handled.

3.3.9 Performance Counters Configuration

The performance counters unit is described in `section pcu`. This section may appear at most once. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true), the performance counters unit is enabled. If 0 (the default), it is disabled.

3.4 Configuring Memory Mapped Peripherals

All peripheral components are optional. If they are specified, then (unlike other components) by default they are enabled.

3.4.1 Memory Controller Configuration

The memory controller used in Or1ksim is the component implemented at OpenCores, and found in the top level SVN directory, '`mem_ctrl`'. It is described in the document *Memory Controller IP Core* by Rudolf Usselmann, which can be found in the '`doc`' subdirectory. It is a memory mapped component, which resides on the main OpenRISC Wishbone data bus.

The memory controller configuration is described in `section mc`. This section may appear multiple times, specifying multiple memory controllers.

> **Warning:** There are known to be problems with the current memory controller, which currently is not included in the regression test suite. Users are advised not to use the memory controller in the current release.
>
> **Caution:** There is no initialization code in the standard *newlib* library.
>
> The standard *uClibc* library assumes a memory controller mapped at 0x93000000 and will initialize the memory controller to expect 64MB memory blocks, and any memory declarations *must* reflect this.
>
> If smaller memory blocks are declared with a memory controller, then sufficient memory will not be allocated by Or1ksim, but out of range memory accesses will not be trapped. For example declaring a memory section from 0-4MB with a memory controller enabled would mean that accesses between 4MB and 64MB would be permitted, but having no allocated memory would likely cause a segmentation fault.
>
> If the user is determined to use smaller memories with the memory controller, then custom initialization code must be provided, to ensure the memory controller traps out-of-memory accesses.

The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this memory controller is enabled. If 0, it is disabled.

> **Note:** The memory controller can effectively also be disabled by setting an appropriate power on control register value (see below). However this should only be used if it is desired to specifically model this behavior of the memory controller, not as a way of disabling the memory controller in general.

`baseaddr = `*`value`*
: Set the base address of the memory controller's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

The memory controller has a 7 bit address bus, with a total of 19 32-bit registers, at addresses 0x00 through 0x4c (address 0x0c and addresses 0x50 through 0x7c are not used).

`poc = `*`value`*
: Specifies the value of the power on control register, The least signficant two bits specify the bus width (use 0 for an 8-bit bus, 1 for a 16-bit bus and 2 for a 32-bit bus) and the next two bits the type of memory connected (use 0 for a disabled interface, 1 for SSRAM, 2 for asyncrhonous devices and 3 for synchronous devices).

If other bits are specified, they are ignored with a warning.

> **Caution:** The default value, 0, corresponds to a disabled 8-bit bus, and is likely not the most suitable value

`index = `*`value`*
: Specify the index of this memory controller amongst all the memory controllers. This value should be unique for each memory controller, and is used to associate specific memories with the controller, through the `mc` field in the `section memory` configuration (see Section 3.3.2 [Memory Configuration], page 20).

The default value, 0, is suitable when there is only one memory controller.

3.4.2 UART Configuration

The UART implemented in Or1ksim follows the specification of the National Semiconductor 16450 and 16550 parts. It is a memory mapped component, which resides on the main OpenRISC Wishbone data bus.

The component provides a number of interfaces to emulate the behavior of an external terminal connected to the UART.

UART configuration is described in `section uart`. This section may appear multiple times, specifying multiple UARTs. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this UART is enabled. If 0, it is disabled.

`baseaddr =` *`value`*
: Set the base address of the UART's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

  The UART has a 3 bit address bus, with a total of 8 8-bit registers, at addresses 0x0 through 0x7.

`channel = "`*`type`*`:`*`args`*`"`
: Specify the channel representing the terminal connected to the UART Rx & Tx pins.

  `channel="file:‘rxfile’,‘txfile’"`
  : Read input characters from the file ‘`rxfile`’ and write output characters to the file ‘`txfile`’ (which will be created if required).

  `channel="xterm:`*`args`*`"`
  : Create an xterm on startup, write UART Tx traffic to the xterm and take Rx traffic from the keyboard when the xterm window is selected. Additional arguments to the xterm command (for example specifying window size may be specified in *args*, or this may be left blank.

  `channel="tcp:`*`value`*`"`
  : Open the TCP/IP port specified by *value* and read and write UART traffic from and to it.

    Typically a telnet session is connected to the other end of this port.

    > **Tip:** There is no registered port for Or1ksim telnet UART connection. Priviledged access is required to read traffic on the registered “well-known” telnet port (23). Instead users should use port values in the *Dynamic* or *Private* port range, i.e. 49152-65535.

  `channel="fd:rxfd,txfd"`
  : Read and write characters from and to the existing open numerical file descriptors, file `rxfd` and `txfd`.

  `channel="tty:device=/dev/ttyS0,baud=9600"`
  : Read and write characters from and to a physical serial port. The precise device (shown here as `/dev/ttyS0`) may vary from machine to machine.

  The default value for this field is `"xterm:"`.

`irq =` *`value`*
: Use *value* as the IRQ number of this UART. Default value 0.

`16550 = 0|1`
: If 1 (true), the UART has the functionality of a 16550. If 0 (the default), it has the functionality of a 16450. The principal difference is that the 16550 can buffer multiple characters.

`jitter = *value*`
: Set the jitter, modeled as a time to block, to *value* milliseconds. Set to -1 to disable jitter modeling. Default value 0.

> **Note:** This functionality has yet to be implemented, so this parameter has no effect.

`vapi_id = *value*`
: *value* specifies the value of the Verification API (VAPI) base address to be used with the UART. See Chapter 5 [Verification API], page 40, for more details, which details the use of the VAPI with the UART.

3.4.3 DMA Configuration

The DMA controller used in Or1ksim is the component implemented at OpenCores, and found in the top level SVN directory, '`wb_dma`'. It is described in the document *Wishbone DMA/Bridge IP Core* by Rudolf Usselmann, which can be found in the '`doc`' subdirectory. It is a memory mapped component, which resides on the main OpenRISC Wishbone data bus. The present implementation is incomplete, intended only to support the Ethernet interface (see Section 3.4.4 [Ethernet Configuration], page 29), although the Ethernet interface is not yet completed.

DMA configuration is described in `section dma`. This section may appear multiple times, specifying multiple DMA controllers. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this DMA controller is enabled. If 0, it is disabled.

`baseaddr = *value*`
: Set the base address of the DMA's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

: The DMA controller has a 10 bit address bus, with a total of 253 32-bit registers. The first 5 registers at addresses 0x000 through 0x010 control the overall behavior of the DMA controller. There are then 31 blocks of 8 registers, controlling each of the 31 DMA channels available. Addresses 0x014 through 0x01c are not used.

`irq = *value*`
: Use *value* as the IRQ number of this DMA controller. Default value 0.

`vapi_id = *value*`
: *value* specifies the value of the Verification API (VAPI) base address to be used with the DMA controller. See Chapter 5 [Verification API], page 40, for more details, which details the use of the VAPI with the DMA controller.

3.4.4 Ethernet Configuration

Ethernet configuration is described in `section ethernet`. This section may appear multiple times, specifying multiple Ethernet interfaces. The following parameters may be specified.

The Ethernet MAC used in Or1ksim corresponds to the Verilog implementation in project *ethmac*. It's source code can be found in the top level SVN directory, '`ethmac`'. It also forms part of the OpenRISC reference SoC, ORPSoC. It is described in the document *Ethernet IP Core Specification* by Igor Mohor, which can be found in the '`doc`' subdirectory. It is a memory mapped component, which resides on the main OpenRISC Wishbone data bus.

`enabled = 0|1`
: If 1 (true, the default), this Ethernet MAC is enabled. If 0, it is disabled.

`baseaddr = value`
: Set the base address of the MAC's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

  The Ethernet MAC has a 7-bit address bus, with a total of 21 32-bit registers. Addresses 0x54 through 0x7c are not used.

  > **Note:** The Ethernet specification describes a Tx control register, `TXCTRL`, at address 0x50. However this register is not implemented in the Or1ksim model.

`dma = value`
: *value* specifies the DMA controller with which this Ethernet is associated. The default value is 0.

  > **Note:** Support for external DMA is not provided in the current implementation, and this value is ignored. In any case there is no equivalent field to which this can be matched in the current DMA component implementation (see Section 3.4.3 [DMA Configuration], page 29).

`irq = value`
: Use *value* as the IRQ number of this Ethernet MAC. Default value 0.

`rtx_type = "file"|"tap"`
: Specifies whether to use a TUN/TAP interface or file interface (the default) to model the external connection of the Ethernet.

  If a TUN/TAP interface is requested, Ethernet packets will be sent and received through the pesistent TAP interface specified in parameter `tap_dev` (see below).

  More details on configuring the TUN/TAP interface are given in the Usage section (see Section 2.6 [Ethernet TUN/TAP Interface], page 11).

  If a file interface (the default), is requested, the Ethernet will be modelled by reading and writing from and to the files specified in the `rxfile` and `txfile` parameters (see below).

  > **Caution:** If a file interface is specified, Or1ksim will terminate once the receive file specified by `rxfile` is exhausted.

`rx_channel = rxvalue`
`tx_channel = txvalue`
: *rxvalue* specifies the DMA channel to use for receive and *txvalue* the DMA channel to use for transmit. Both default to 0.

  > **Note:** As noted above, support for external DMA is not provided in the current implementation, and so these values are ignored.

`rxfile = "rxfile"`
`txfile = "txfile"`
: When `rtx_type` is 0 (see above), *rxfile* specifies the file to use as input and *txfile* specifies the fie to use as output.

  The file contains a sequence of packets. Each packet consists of a packet length (32 bits), followed by that many bytes of data. Once the input file is empty, the Ethernet MAC behaves as though there were no data on the Ethernet. The default values of these parameters are `"eth_rx"` and `"eth_tx"` respectively.

  The input file must exist and be readable. The output file must be writable and will be created if necessary. If either of these conditions is not met, a warning will be given.

> **Caution:** Or1ksim will terminate once the *rxfile* is exhausted.

`tap_dev = "tap"`
: When `rtx_type` is `"tap"` (see above), *tap_dev* specifies the TAP device to use for communication. This should be a persistent TAP device configured for the system (see Section 2.6 [Ethernet TUN/TAP Interface], page 11)

`phy_addr = value`
: *value* specifies the address for emulated ethernet PHY (default 0). If there are multiple Ethernet peripherals, they should each have a different PHY value.

`dummy_crc = 0|1`
: If 1 (true, the default), the length of the data transferred to the core will be increased by 4 bytes, as though the CRC were included.

> **Note:** This is for historical consistency with the OpenRISC Ethernet hardware MAC, which passes on the CRC in the data packet. This is unusual behavior for a MAC, but the OpenRISC Linux device drivers have been written to expect it.

`phy_addr = value`
: *value* specifies the address for emulated ethernet PHY (default 0). If there are multiple Ethernet peripherals, they should each have a different PHY value.

`vapi_id = value`
: *value* specifies the value of the Verification API (VAPI) base address to be used with the Ethernet PHY. See Chapter 5 [Verification API], page 40, for more details, which details the use of the VAPI with the DMA controller.

3.4.5 GPIO Configuration

The GPIO used in Or1ksim is the component implemented at OpenCores, and found in the top level SVN directory, '`gpio`'. It is described in the document *GPIO IP Core Specification* by Damjan Lampret and Goran Djakovic, which can be found in the '`doc`' subdirectory. It is a memory mapped component, which resides on the main OpenRISC Wishbone data bus.

GPIO configuration is described in `section gpio`. This section may appear multiple times, specifying multiple GPIO devices. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this GPIO is enabled. If 0, it is disabled.

`baseaddr = value`
: Set the base address of the GPIO's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

  The GPIO has a 6 bit address bus, with a total of 10 32-bit registers, although the number of bits that are actively used varies. Addresses 0x28 through 0x3c are not used.

`irq = value`
: Use *value* as the IRQ number of this GPIO. Default value 0.

`vapi_id = value`
: *value* specifies the value of the Verification API (VAPI) base address to be used with the GPIO. See Chapter 5 [Verification API], page 40, for more details, which details the use of the VAPI with the GPIO controller. For backwards compatibility, the alternative name `base_vapi_id` is supported for this parameter, but deprecated.

3.4.6 Display Interface Configuration

Or1ksim models a VGA interface to an external monitor. The VGA controller used in Or1ksim is the component implemented at OpenCores, and found in the top level SVN directory, '`vga_lcd`', with no support for the optional hardware cursors. It is described in the document *VGA/LCD Core v2.0 Specifications* by Richard Herveille, which can be found in the '`doc`' subdirectory. It is a memory mapped component, which resides on the main OpenRISC Wishbone data bus.

The current implementation provides only functionality to dump the screen to a file at intervals.

VGA controller configuration is described in `section vga`. This section may appear multiple times, specifying multiple VGA controllers. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this VGA is enabled. If 0, it is disabled.

`baseaddr =` *`value`*
: Set the base address of the VGA controller's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

  The VGA controller has a 12-bit address bus, with 7 32-bit registers, at addresses 0x000 through 0x018, and two color lookup tables at addresses 0x800 through 0xfff. The hardware cursor registers are not implemented, so addresses 0x01c through 0x7fc are not used.

`irq =` *`value`*
: Use *value* as the IRQ number of this VGA controller. Default value 0.

`refresh_rate =` *`value`*
: *value* specifies number of cycles between screen dumps. Default value is derived from the simulation clock cycle time (see Section 3.2.1 [Simulator Behavior], page 16), to correspond to dumping 50 times per simulated second.

`txfile = "`*`file`*`"`
: *file* specifies the base of the filename for screen dumps. Successive screen dumps will be in BMP format, in files with the name '*`filennnn`*`.bmp`', where *nnnn* is a sequential count of the screen dumps starting at zero. The default value is `"vga_out"`. For backwards compatibility, the alternative name `filename` is supported for this parameter, but deprecated.

3.4.7 Frame Buffer Configuration

> **Caution:** The frame buffer is only partially implemented. Its configuration fields are described here, but the component should not be used at this time. Like the VGA controller, it is designed to make screen dumps to file.

Frame buffer configuration is described in `section fb`. This section may appear multiple times, specifying multiple frame buffers. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this frame buffer is enabled. If 0, it is disabled.

`baseaddr =` *`value`*
: Set the base address of the frame buffer's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

  The frame buffer has an 121-bit address bus, with 4 32-bit registers, at addresses 0x000 through 0x00c, and a PAL lookup table at addresses 0x400 through 0x4ff. Addresses 0x010 through 0x3fc and addresses 0x500 through 0x7ff are not used.

`refresh_rate = value`
: *value* specifies number of cycles between screen dumps. Default value is derived from the simulation clock cycle time (see Section 3.2.1 [Simulator Behavior], page 16), to correspond to dumping 50 times per simulated second.

`txfile = "file"`
: *file* specifies the base of the filename for screen dumps. Successive screen dumps will be in BMP format, in files with the name '`filennnn.bmp`', where *nnnn* is a sequential count of the screen dumps starting at zero. The default value is `"fb_out"`. For backwards compatibility, the alternative name `filename` is supported for this parameter, but deprecated.

3.4.8 Keyboard Configuration (PS2)

The PS2 interface provided by Or1ksim is not documented. It may be based on the PS2 project at OpenCores, and found in the top level SVN directory, '`ps2`'. However this project lacks any documentation beyond its project webpage. Since most PS2 interfaces follow the Intel i8042 standard, this is presumably what is expected with this device.

The implementation only provides for keyboard support, which is modelled as a file of keystrokes. There is no mouse support.

> **Caution:** A standard i8042 device has two registers at addresses 0x60 (command) and 0x64 (status). Inspection of the code, suggests that the Or1ksim component places these registers at addresses 0x00 and 0x04.
>
> The port of Linux for the OpenRISC 1000, which runs on Or1ksim implements the i8042 device driver, anticipating these registers reside at their conventional address. It seems unlikel that this code will work.
>
> This component should be used with caution.

Keyboard configuration is described in `section kbd`. This section may appear multiple times, specifying multiple keyboard interfaces. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this keyboard is enabled. If 0, it is disabled.

`baseaddr = value`
: Set the base address of the keyboard's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

: The keyboard PS/2 interface has an 3-bit address bus, with 2 8-bit registers, at addresses 0x000 and 0x004.

> **Caution:** As noted above, a standard Intel 8042 interface would expect to find these registers at locations 0x60 and 0x64, thus requiring at least a 7-bit bus.

`irq = value`
: Use *value* as the IRQ number of this Keyboard interface. Default value 0.

`rxfile = "file"`
: '`file`' specifies a file containing raw key stroke data, which models the input from a physical keyboard. The default value is `"kbd_in"`.

3.4.9 Disc Interface Configuration

The ATA/ATAPI disc controller used in Or1ksim is the OCIDEC (OpenCores IDE Controller) component implemented at OpenCores, and found in the top level SVN directory, '`ata`'. It is described in the document *ATA/ATAPI-5 Core Specification* by Richard Herveille, which can be found in the 'doc' subdirectory. It is a memory mapped component, which resides on the main OpenRISC Wishbone data bus.

> **Warning:** In the current release of Or1ksim, parsing of the ATA section is broken. Users should not configure the disc interface in this release.

ATA/ATAPI configuration is described in `section ata`. This section may appear multiple times, specifying multiple disc controllers. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this ATA/ATAPI interface is enabled. If 0, it is disabled.

`baseaddr =` *`value`*
: Set the base address of the ATA/ATAPI interface's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

  The ATA/ATAPI PS/2 interface has an 5-bit address bus, with 8 32-bit registers. Depending on the version of the OCIDEC ATA/ATAPI interface selected (see `dev_id` below), not all registers will be available.

`irq =` *`value`*
: Use *value* as the IRQ number of this ATA/ATAPI interface. Default value 0.

`dev_id = 1|2|3`
: This parameter specifies which version of the OCIDEC ATA/ATAPI interface to model. The default value is 1.

  Version 1 supports only the `CTRL`, `STAT` and `PCTR` registers. Versions 2 & 3 add the `FCTR` registers, Version 3 adds the `DTR` registers and the `RXD/TXD` registers.

`rev =` *`value`*
: Set the *value* as the revision of the OCIDEC ATA/ATAPI interface. The default value is 1. The default value is 0. Its value should be in the range 0-15. Larger values are truncated with a warning. This only affects the reset value of the `STAT` register, where it forms bits 24-27.

`pio_mode0_t1 =` *`value`*
`pio_mode0_t2 =` *`value`*
`pio_mode0_t4 =` *`value`*
`pio_mode0_teoc =` *`value`*
: These parameters specify the timings for use with Programmed Input/Output (PIO) transfers. They are specified as the number of clock cycles - 2, rounded up to the next highest integer, or zero if that would be negative. The values should not exceed 255. If they do, they will be ignored with a warning.

  See the ATA/ATAPI-5 specification for explanations of each of these timing parameters. The default values are:

```
pio_mode0_t1   =  6
pio_mode0_t2   = 28
pio_mode0_t4   =  2
pio_mode0_teoc = 23
```

`dma_mode0_tm =` *`value`*
`dma_mode0_td =` *`value`*
`dma_mode0_teoc =` *`value`*
: These parameters specify the timings for use with DMA transfers. They are specified as the number of clock cycles - 2, rounded up to the next highest integer, or zero if that would be negative. The values should not exceed 255. If they do, they will be ignored with a warning.

See the ATA/ATAPI-5 specification for explanations of each of these timing parameters. The default values are:

```
dma_mode0_tm   =  4
dma_mode0_td   = 21
dma_mode0_teoc = 21
```

3.4.9.1 ATA/ATAPI Device Configuration

Within the `section ata`, each device is specified separately. The device subsection is introduced by

```
device value
```

value is the device number, which should be 0 or 1. The subsection ends with `enddevice`. Note that if the same device number is specified more than once, the previous values will be overwritten. Within the `device` subsection, the following parameters may appear:

`type = value`
: *value*specifies the type of device: 0 (the default) for “not connected”, 1 for hard disk simulated in a file and 2 for local system hard disk.

`file = "filename"`
: ‘`filename`’ specifies the file to be used for a simulated ATA device if the file type (see `type` above) is 1. Default value `"ata_filen"`, where *n* is the device number.

`size = value`
: *value* specifies the size of a simulated ATA device if the file type (see `type` above) is 1. The default value is zero.

`packet = 0|1`
: If 1 (true), implement the PACKET command feature set. If 0 (the default), do not implement the PACKET command feature set.

`firmware = "str"`
: Firmware to report in response to the “Identify Device” command. Default `"02207031"`.

`heads = value`
: Number of heads in the device. Default 7, use -1 to disable all heads.

`sectors = value`
: Number of sectors per track in the device. Default 32.

`mwdma = 0|1|2|-1`
: Highest multi-word DMA mode supported. Default 2, use -1 to disable.

`pio = 0|1|2|3|4`
: Highest PIO mode supported. Default 4.

3.4.10 Generic Peripheral Configuration

When used as a library (see Section 2.5 [Simulator Library], page 8), Or1ksim makes provision for any additional peripheral to be implemented externally. Any read or write access to this peripheral’s memory map generates *upcalls* to an external handler. This interface can support either C or C++, and was particularly designed to facilitate support for OSCI SystemC (see `http://www.systemc.org`).

Generic peripheral configuration is described in `section generic`. This section may appear multiple times, specifying multiple external peripherals. The following parameters may be specified.

`enabled = 0|1`
: If 1 (true, the default), this ATA/ATAPI interface is enabled. If 0, it is disabled.

`baseaddr = value`
: Set the base address of the generic peripheral's memory mapped registers to *value*. The default is 0, which is probably not a sensible value.

  The size of the memory mapped register space is controlled by the `size` paramter, described below.

`size = value`
: Set the size of the generic peripheral's memory mapped register space to *value* bytes. Any read or write accesses to addresses with offsets of 0 to *value*-1 bytes from the base address specified in parameter `baseaddr` (see above) will be directed to the external interface.

  value will be rounded up the nearest power of 2. It's default value is zero. If *value* is not an exact power of two, accesses to address offsets of *value* or above up to the next power of 2 will generate a warning, and have no effect (reads will return zero).

`name = "str"`
: This gives the peripheral the name `"str"`. This is used to identify the peripheral in error messages and warnings, and when reporting its status. The default value is `"anonymous external peripheral"`.

`byte_enabled = 0|1`
`hw_enabled = 0|1`
`word_enabled = 0|1`
: If 1 (true, the default), these parameters respectively enable the device for byte wide, half-word wide and word wide accesses. If 0, accesses of that width will fail.

4 Interactive Command Line

If started with the `-f` flag, or if interrupted with *`ctrl-C`*, Or1ksim provides the user with an interactive command line. The commands available, which may not be abbreviated, are:

`q` Exit the simulator

`r` Display all the General Purpose Registers (GPRs). Also shows the just executed and next to be executed instructions symbolically and the state of the flag in the Supervision Register.

`t` Execute the next instruction and then display register/instruction information as with the `r` command (see above).

`run` *`num`* `[ hush ]`
: Execute *num* instructions. The register/instruction information is displayed after each instruction, as with the `r` command (see above) *unless* `hush` is specified.

`pr` *`reg value`*
: Patch register *reg* with *value*.

`dm` *`fromaddr`* `[` *`toaddr`* `]`
: Display memory bytes between *fromaddr* and *toaddr*. If *toaddr* is not given, 64 bytes are displayed, starting at *fromaddr*.

> **Caution:** The output from this command is broken (a bug). Or1ksim attempts to print out 16 bytes per row. However, instead of printing out the address at the start of each row, it prints the address (of the first of the 16 bytes) before *each* byte.

`de` *`fromaddr`* `[` *`toaddr`* `]`
: Disassemble code between *fromaddr* and *toaddr*. If *toaddr* is not given, 16 instructions are disassembled.

The disassembly is entirely numerical, and gives no symbolic information.

`pm` *`addr value`*
: Patch the 4 bytes in memory starting at *addr* with the 32-bit *value*.

`pc` *`value`* Patch the program counter with *value*.

`cm` *`fromaddr toaddr size`*
: Copy *size* bytes in memory from *fromaddr* to *toaddr*.

`break` *`addr`*
: Toggle the breakpoint set at *addr*.

`breaks` List all set breakpoints

`reset` Reset the simulator. Includes modeling a reset of the processor, so execution will restart from the reset vector location, 0x100.

`hist` If saving the execution history has been configured (see Section 3.2.1 [Simulator Behavior], page 16), display the execution history.

`stall` Stall the processor, so that control is passed to the debug unit. When stalled, the processor can execute no instructions. This command is useful when debugging the JTAG interface, used by debuggers such as GDB.

`unstall` Unstall the processor, so that normal execution can continue. This command is useful when debugging the JTAG interface, used by debuggers such as GDB.

`stats` *category* `| clear`
: Print the statistics for the given *category*, if available, or clear if `clear` is specified. The categories are:

  1 Miscellaneous statistics: branch predictions (if branch predictions are enabled), branch target cache model (if enabled), cache (if enbaled), MMU (if enabled) and number of addtional load & store cycles.

    See Section 3.3 [Configuring the OpenRisc Achitectural Components], page 19, for details of how to enable these various features.

  2 Instruction usage statistics. Requires hazard analysis to be enabled (see Section 3.3.1 [CPU Configuration], page 19).

  3 Instruction dependency statistics. Requires hazard analysis to be enabled (see Section 3.3.1 [CPU Configuration], page 19).

  4 Functional unit dependency statistics. Requires hazard analysis to be enabled (see Section 3.3.1 [CPU Configuration], page 19).

  5 Raw register usage over time. Requires hazard analysis to be enabled (see Section 3.3.1 [CPU Configuration], page 19).

  6 Store buffer statistics. Requires the store buffer to be enabled (see Section 3.3.1 [CPU Configuration], page 19).

`info` Display detailed information about the simulator configuration. This is quite a lengthy about, because all MMU TLB information is displayed.

`dv` *fromaddr* `[` *toaddr* `] [` *module* `]`
: Dump the area of memory between *fromaddr* and *toaddr* as Verilog code for a synchronous, 23-bit wide SRAM module, named *module*. If *toaddr* is not specified, then 64 bytes are dumped (as 16 32-bit words). If *module* is not specified, `or1k_mem` is used.

  To save to a file, use the redirection function (described after this table, below).

`dh` *fromaddr* `[` *toaddr* `]`
: Dump the area of memory between *fromaddr* and *toaddr* as 32-bit hex numbers (no `0x`, or `32'h` prefix). If *toaddr* is not specified, then 64 bytes are dumped (as 16 32-bit words).

  To save to a file, use the redirection function (described after this table, below).

`setdbch` Toggle debug channels on/off. See Section 2.1 [Standalone Simulator], page 5, for a description of specifying debug channels on the command line.

`set` *section param* `=` *value*
: Set the configuration parameter *para* in section *section* to *value*. See Chapter 3 [Configuration], page 15, for details of configuration parameters and their settings.

`debug` Toggle the simulator debug mode. See Section 3.3.8 [Debug Interface Configuration], page 26, for information on this parameter.

> **Caution:** This is effectively enabling or disabling the debug unit. It does not effect the remote GDB debug interface. However using the remote debug interface while the debug unit is disabled will lead to undefined behavior and likely crash Or1ksim

`cuc` Enter the the Custom Unit Compiler command prompt (see Section 3.2.3 [CUC Configuration], page 18).

> **Caution:** The CUC must be properly configured, for this to succeed. In particular a timing file must be available and readable. Otherwise Or1ksim will crash.

`help` Print out brief information about each command available.

`mprofile [-vh] [-m` *`m`*`] [-g` *`n`*`] [-f` *`file`*`]` *`from to`*
: Run the memory profiling utility. This follows the same usage as the standalone command (see Section 2.3 [Memory Profiling Utility], page 7).

`profile [-vhcq] [-g` *`file`*`]`
: Run the instruction profiling utility. This follows the same usage as the standalone command (see Section 2.2 [Profiling Utility], page 6).

For all commands, it is possible to redirect the output to a file, by using the redirection operator, `>`.

```
command > filename
```

This is particularly useful for commands dumping a large amount of output, such as `dv`.

> **Caution:** Unfortunately there is a serious bug with the redirection operator. It does not return output to standard output after the command completes. Until this bug is fixed, file redirection should not be used.

5 Verification API (VAPI)

The Verification API (VAPI) provides a TCP/IP interface to allow components of the simulation to be controlled externally. The interface is polled for new requests on each simulated clock cycle. Components within the simulator may send responses to such requests.

The inteface is an asynchronous duplex protocol. On the request side it provides for simple commands, known as VAPI IDs (a 32 bit integer), with a single piece of data (also a 32 bit integer). On the send side, it provides for sending a single VAPI ID and data. However there is no explicit command-response structure. Some components just accept requests (e.g. to set values), some just generate sends (to report values), and some do both.

Each component has a base ID (32 bit) and its commands will start from that base ID. This provides a simple partitioning of the command space amongst components. Request commands will be directed to the component with the closest base ID lower than the VAPI ID of the command.

Thus if there are two components with base IDs of 0x200 and 0x300, and a request with VAPI ID of 0x203 is received, it will be directed to the first component as its command #3.

The results of VAPI interactions are logged (by default in '`vapi.log`' unless an alternative is specified in `section vapi`).

Currently the following components support VAPI:

Debug Unit
: Although the Debug Unit can specify a base VAPI ID, it is not used to send commands or receive requests.

  Instead, if the base VAPI ID is set, all remote JTAG protocol exchanges are logged in the VAPI log file.

UART
: If a base VAPI ID is specified, the UART sends details of any chars or break characters sent, with dteails of the line control register etc encoded in the data packet sent.

  This supports a single VAPI command request, but encodes a sub-command in the top 8 bits of the associated data.

  `0x00` This stuffs the least significant 8 bits of the data into the serial register of the UART and the next 8 bits into the line control register, effectively providing control of the next character to be sent or received.

  `0x01` The divisor latch bytes are set from the least significant 16 bits of the data.

  `0x02` The line control register is set from bits 15-8 of the data.

  `0x03` The UART skew is set from the least significant 16 bits of the data

  `0x04` If the 16th most significant bit of the data is 1, start sending breaks, otherwise stop sending breaks. The breaks are sent or cleared after the number of UART clock divider ticks specified by the data (immediately if the data is zero).

DMA
: Although the DMA unit supports a base VAPI ID in its configuration (`section dma`), no VAPI data is sent, nor VAPI requests currently implemented.

Ethernet
: The following requests are handled by the Ethernet. Specified symbolically, these are the increments from the base VAPI ID of the Ethernet. At present no implementation is provided behind these VAPI requests.

ETH_VAPI_DATA (0)
ETH_VAPI_CTRL (0)

GPIO If a base VAPI ID is specified, the GPIO sends out on its base VAPI ID (symbolically, GPIO_VAPI_DATA (0) offset from the base VAPI ID) any changes in outputs.

The following requests are handled by the GPIO. Specified symbolically, these are the increments from the VAPI base ID of the GPIO.

GPIO_VAPI_DATA (0)
Set the next input to the commands data field

GPIO_VAPI_AUX (1)
Set the GPIO auxiliary inputs to the data field

GPIO_VAPI_CLOCK (2)
Add an external GPIO clock trigger of period specified in the data field.

GPIO_VAPI_RGPIO_OE (3)
Set the GPIO output enable to the data field

GPIO_VAPI_RGPIO_INTE (4)
Set the next interrupt to the data field

GPIO_VAPI_RGPIO_PTRIG (5)
Set the next trigger to the data field

GPIO_VAPI_RGPIO_AUX (6)
Set the next auxiliary input to the data field

GPIO_VAPI_RGPIO_CTRL (7)
Set th next control input to the data field

6 A Guide to Or1ksim Internals

These are notes to help those wanting to extend Or1ksim. This section assumes the use of a tag file, so file locations of entities' definitions are not in general provided. For more on tags, see the Linux manual page for `etags`. A tag file can be created with:

```
make tags
```

6.1 Coding Conventions for Or1ksim

This chapter provides some guidelines for coding, to facilitate extensions to Or1ksim

GNU Coding Standard

Code should follow the GNU coding standard for C (http://www.gnu.org/prep/standards/. If in doubt, put your code through the `indent` program.

`#include` *headers*

All C source code files should include '`config.h`' before any other file.

This should be followed by inclusion of any system headers (but see the comments about portability and '`port.h`' below) and then by any Or1ksim package headers.

If '`port.h`' is required, it should be the first package header to be included after the system headers.

All C source code and header files should directly include any system or package header they depend on, i.e. not rely on any other header having already included it. The two exceptions are

1. All header files may assume that '`config.h`' has already been included.
2. System headers which impose portability problems should be included by using the package header '`port.h`', rather than the system headers themselves. This is the case for code requiring
   - `strndup` (from '`string.h`')
   - Integer types (`intn_t`, `uintn_t`) (from '`inttypes.h`').
   - `isblank` (from '`ctype.h`')

`#include` *files once only*

All include files should be protected by `#ifndef` to ensure their definitions are only included once. For instance a header file '`x-y.h`' should surround its contents with:

```
#ifndef X_Y__H
#define X_Y__H

<body of the include file>

#endif  /* X_Y__H */
```

Avoid `typedef`

The GNU coding style for C does not have a clear way to distinguish between user type name and user variables. For this reason `typedef` should be avoided except for the most ubiquitous user defined types. This makes the code much easier to read.

There are some `typedef` declarations in the `argtable2` library and the ELF and COFF headers, because this code is taken from other places.

Within Or1ksim legacy uses of `typedef` have largely been purged, except in the Custom Unit Compiler (see Section 3.2.3 [Custom Unit Compiler (CUC) Configuration], page 18).

The remaining uses of `typedef` occur in two places:

- '`port/port.h`' defines types to replace those in header files that are not available (character functions, string duplication, integer types).

  '`cpu/or1k/arch.h`' defines types for the key Or1ksim entities: addresses (`oraddr_t`), unsigned register values (`uorreg_t`) and signed register (`orreg_t`) values.

Where new types are defined, they should appear in one of these two files as appropriate. Or1ksim specific types appearing in '`arch.h`' should always have the suffix '`_h`'.

Don't begin names with underscore
: Names beginning with `_` are intended to be part of the C infrastructure. They should not be used in the simulator code.

Keep Non-global top level entities static
: All top level entities (functions, variables), which are not explicitly part of a global interface should be declared static. This ensures that unwanted connections are not inadvertently built across the program.

Use of `inline`
: Code should not be declared `inline`. Modern compilers can work out for themselves what is best in this respect.

Initialization
: All data structures should be explicitly initialized. In particular code should not rely on static data structures being initialized to zero.

  The rationale is that in future static data structures may become dynamic. This has been a particular source of bugs in Or1ksim historically.

  A specific case is with new peripherals, which should always include a `start` function to pre-initialize all configuration parameters to sensible defaults

Configuration Validation
: All configuration values should be validated, preferably when encountered, if not when the `section` is closed, or otherwise at run time when the parameter is first used.

6.2 Global Data Structures

`config`
: The global variable `config` of type `struct config` holds the configuration data for some of the Or1ksim components which are always present. At present the components are:

  - The simulator defined in `section sim` (see Section 3.2 [Simulator Configuration], page 16).
  - The Verification API (VAPI) defined in `section vapi` (see Section 3.2.2 [Verification API (VAPI) Configuration], page 18).
  - The Custom Unit Compiler (CUC), defined in `section cuc` (see Section 3.2.3 [Custom Unit Compiler (CUC) Configuration], page 18).
  - The CPU, defined in `section cpu` (see Section 3.3.1 [CPU Configuration], page 19).
  - The data cache (but not the instruction cache), defined in `section dc` (see Section 3.3.4 [Cache Configuration], page 23).
  - The power management unit, defined in `section pm` (see Section 3.3.6 [Power Management Configuration], page 25).

- The programmable interrupt controller, defined in `section pic` (see Section 3.3.5 [Interrupt Configuration], page 24).
- Branch prediciton, defined in `section bpb` (see Section 3.3.7 [Branch Prediction Configuration], page 25).
- The debug unit, defined in `section debug` (see Section 3.3.8 [Debug Interface Configuration], page 26).

This struct is made of a collection of structs, one for each component. For example the simulator configuration is held in `config.sim`.

`config` This is a linked list of data structures holding configuration data for all sections which are not held in the main `config` data structure. In general these are components (such as peripherals and memory) which may occur multiple times. However it also handles some architectural components which may occur only once, such as the memory management units, the instruction cache, the interrupt controller and branch prediction.

`runtime` The global variable `runtime` of type `struct runtime` holds all the runtime information about the simulation. To access this variable, '`sim-config.h`' must be included.

This struct is itself made of 3 other structs, `cpu` (for CPU run time state), `vapi` (for Verification API state) and `cuc` (for Custom Unit Compiler state).

6.3 Concepts

Output Redirection

The current output stream is held in `runtime.cpu.fout`. Output should be explicitly written to this stream, or may use the `PRINTF` macro, which will write its arguments to this output stream.

Reset Hooks

Any peripheral may register a routine to be called when the the processor is reset by calling `reg_sim_reset`, providing a function and pointer to a data structure as arguments. On reset that function will be called with the data stucture pointer as argument.

Interrupts An internal peripheral can model the effect of an interrupt being asserted by calling `report_interrupt`. This is used for both edge and level sensitive interrupts.

The effect is to set the corresponding bit in the PICSR SPR and to queue an interrupt exception to take place after the current instruction completes execution.

Externally, the different interrupts require different mechanisms for clearing. Level sensitive interrupts should be cleared by deasserting the interrupt line, edge sensitive interrupts by clearing the corresponding bit in the PICSR SPR.

Internally this amounts to the same thing (clearing the PICSPR bit), so a single function is provided, `clear_interrupt`. Note however that when level sensitive interrupts are configured, PICSR is read only, and can only be cleared by calling `clear_interrupt`. Using the two functions provided will ensure the peripheral works correctly whichever type of interrupt is used.

> **Note:** Until an interrupt is cleared, all subsequent interrupts are ignored with a warning.

6.4 Internal Debugging

The function `debug` is like `printf`, but with an extra first argument, which is the debug level. If the debug level specified in the simulator configuration (see Section 3.2.1 [Simulator Behavior], page 16) is greater than or equal to this value, the remaining arguments are printed to the current output stream (see [Output Redirection], page 44).

6.5 Regression Testing

Or1ksim now includes a regression test suite for both standalone and library usage as described earlier (see Section 1.3 [Building and Installing], page 4). Running the tests requires that the OpenRISC toolchain and DejaGNU are both installed.

Tests are written using `expect`, a derivative of TCL. Documentation of DejaGnu, `expect` and TCL are freely available on the Web. The Embecosm Application Note 8, *Howto: Using DejaGnu for Testing: A Simple Introduction* (`http://www.embecosm.com/download/ean8.html`) provides a concise introduction.

All test code is found in the '`testsuite`' directory. The key files and directories used are as follows.

`global-conf.exp`
: This is the global DejaGNU configuration file used to set up parameters common to all tests. If the user has the environment varialbe `DEJAGNU` defined, it will be used instead, but this is not recommended.

`Makefile.am`
: This is the top level `automake` file for the testsuite. The only changes likely to be needed here is additional local cleanup of files created by new tests.

`README`
: This contains details of all the tests

`config`
: This contains DejaGnu board configurations. Since the tests are generally run on a Unix host, this should just contain '`Unix.exp`'.

`lib`
: This contains DejaGnu tool specific configurations. "Tool" has a specific meaning in DejaGNU, referring just to a grouping of tests. In this case there are two such "tools", "or1ksim" and "libsim" for tests of the standalone tool and tests of the library.

  Corresponding to this, there are two tool specific configuration files, '`or1ksim.exp`' and '`libsim.exp`'. These contain `expect`/TCL procedures for common use among the tests.

`libsim.tests`
`or1ksim.tests`
: These are the directories of tests of the Or1ksim library. They also include Or1ksim configuration files and each has a '`Makefile.am`' file. '`Makefile.am`' should be updated whenever files are added to this directory, to ensure they are included in the distribution.

`test-code`
: These are all the test programs to be compiled on the host (each in its own directory). In general these are programs to support testing of the library, and build various programs linking in the library.

`test-code`
: These are all the test programs to be compiled with the OpenRISC tool chain to run with either standalone Or1ksim or the library. This directory includes its own

'`configure.ac`', since it must set up a separate tool chain based on the target, not the host.

To add a new test needs the following steps.

- Put new host C code in its own directory within '`test-code`'. Add the directory to the existing '`Makefile.am`' in the '`test-code`' directory and create a '`Makefile.am`' in the new directory to drive building the test program(s). Don't forget to add the new '`Makefile`' to the top level '`configure.ac`' so it gets generated. Not all tests require code here.
- Put new target C code in its own directory within '`test-code-or1k`'. Once again modify & create '`Makefile.am`'. This time modify the '`configure.ac`' in the '`test-code-or1k`' so the '`Makefile`' gets generated. The existing programs provide examples to start from, including custom linker scripts where needed.
- Add one or more tests and configuration files to the relevant "tool" test directory. Use the existing tests as templates. They make heavy use of the `expect`/TCL procedures in the '`config`' directory to facilitate driving the tests.

7 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 noncommercially. 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

-

0

1

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W