General Overview

Prerequisites

A suitable compiler for the RISC-V ISA must be available. Since the RI5CY RISC-V core supports additional ISA extensions that are not supported by official toolchain, a special compiler must be used to take advantage of those.

For the basic RV32I instruction set also the official toolchain can be used.

Setup

The software compilation flow is based on CMake. A version of CMake >= 2.8.0 is required, but a version greater than 3.1.0 is recommended due to support for ninja.

CMake uses out-of-source builds which means you will need a separate build folder for the software, e.g. `build`

```
mkdir build
```

Then switch to the build folder and copy the cmake template configuration script there which resides in the `sw` folder. The name of template follows the following naming scheme: cmake-configure.{or1k/riscv}.{gcc/llvm}.sh Choose, copy, modify and then execute this script. It will setup the build environment for you.

Now you are ready to start compiling software!

Compiling

Switch to the build folder and compile the application you are interested in:

```
make applicationName
```

This command will compile the application and generate stimuli for RTL simulation using ModelSim.

To compile the RTL using ModelSim, use

```
make vcompile
```

Executing

To execute an application again CMake can be used. Switch to the build folder and execute

```
make applicationName.vsim
```

```
```

to start ModelSim in GUI mode.

To use console mode, use

```
make applicationName.vsimc
```

Tests

Automatic regression tests are supported using the ctest framework that comes with CMake.

use

```
ctest -L "riscv|sequential" --timeout 100
```

to launch the tests for the RISC-V core and some basic computation benchmarks

Applications

How to add a new application

CMake uses the concept of CMakeLists.txt files in each directory that is managed by the tool. Those files give instructions to the tool about which applications exist and which files belong to it.

An application is defined like this in a CMakeLists.txt file:

```
add_application(helloworld helloworld.c)
```

If an application consists of multiple source files it has be defined like this:

```
set(SOURCES main.c helper.c)
add_application(helloworld "${SOURCES}")
```

For ease-of-use we recommend that each application has its own source directory. Use the `add_subdirectory` macro of CMake to let the tool know about folder structures. Those macros are put in the parent folders until you hit a folder that is already managed by CMake. Each of the folders needs to have a CMakeLists.txt file.

All applications need to have their own build folders. This means that if you want to declare multiple applications in the same source folders, you have to make sure they do not share the same build folder. This can be done by the optional argument `SUBDIR` for `add_application`

```
add_application(helloworld helloworld.c SUBDIR "hello"))
```

The command above would put the application helloworld in a subdirectory called hello in the build folder structure.

CMake Targets

Each application supports the following targets:

- ${NAME}: Compile the application and generate all stimuli for simulation
- ${NAME}.vsim: Start modelsim in GUI mode
- ${NAME}.vsimc: Start modelsim in console
- ${NAME}.elf: Compile the application and generate the elf file
- ${NAME}.read: Perform an objdump of the binary and save it as ${NAME}.read