riscv64-base-vp

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

[//]: # DONT EDIT THIS FILE

1.1 Virtual Platform of RISCV64 for GreenSocs

1.1.1 1. Overview

This release contains an example of a virtual platform based on an RISCV64.

1.1.2 2. Requirements

You can build this release natively on Ubuntu 18.04.

Install dependencies:

```
apt update && apt upgrade -y
apt install -y make cmake g++ wget flex bison unzip python pkg-config libpixman-1-dev libglib2.0-dev
```

1.1.3 3. Fetch the qbox sources

If you have an SSH key:

```
cd $HOME
git@git.greensocs.com:platforms/greensocs-riscv64.git
```

otherwise:

```
cd $HOME
https://git.greensocs.com/platforms/greensocs-riscv64.git
```

this will extract the platform in $HOME/greensocs-riscv64

1.1.4 4. Build the platform

```
cd $HOME/riscv64
mkdir build && cd build
cmake .. [OPTIONS]
```

It is possible that during the recovery of the sources of the libraries the branch of the repo of one of the libraries is not good in these cases it is necessary to add the option `-DGIT_BRANCH=next`. As mentioned above, it is also possible to get the sources of a library locally with the option `-DCPM_<package>_SOURCE=/path/to/your/library`.

```
make -j
```

This will take some time.

1.1.5 5. Run

Run the Virtual Platform

```
cd ../
./build/vp --gs_luafile conf.lua
```

You should see the following output:

```
OpenSBI v0.9
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : freechips,rocketchip-unknown
Platform Features         : timer,mfdeleg
Platform HART Count       : 2
Firmware Base             : 0x70000000
Firmware Size             : 152 KB
Runtime SBI Version       : 0.2
[...]
Welcome to Buildroot
buildroot login: root
# ls /
bin      init     linuxrc  opt      run      tmp
dev      lib      media    proc     sbin     usr
etc      lib64    mnt      root     sys      var
```

Once the kernel has booted, you can log in with the 'root' account (no password required).

For configuration parameters, see conf.lua (and see below)

NB: Some parameters allow to load raw binary files into the memories of the VP (the `platform.*_blob_file` parameters). Please note that once at least one of those parameters is in use, the simulated bootloader of the VP is disabled and the related bootloader options (`opensbi_file`, `kernel_file`, `dtb_file`) have no effect.

1.1.5.1 6. Explore Sources

The sc_main(), where the virtual platform is created is in `src/main.cc`.

You can find all the recovered sources in the folder `build/_deps/<package>-src/`.

1.2 GreenSocs Build and make system

1.3 How to build

This project may be built using cmake

```
cmake -B build;pushd build; make -j; popd
```

cmake may ask for your git.greensocs.com credentials (see below for advice about passwords)

1.3.1 cmake version

cmake version 3.14 or newer is required. This can be downloaded and used as follows

```
curl -L https://github.com/Kitware/CMake/releases/download/v3.20.0-rc4/cmake-3.20.0-rc4-linux-x86_64.tar.gz
       | tar -zxf -
./cmake-3.20.0-rc4-linux-x86_64/bin/cmake
```

1.3.2 details

This project uses CPM https://github.com/cpm-cmake/CPM.cmake in order to find, and/or download missing components. In order to find locally installed SystemC, you may use the standards SystemC environment variables: SYSTEMC_HOME and CCI_HOME. CPM will use the standard CMAKE find_package mechanism to find installed packages https://cmake.org/cmake/help/latest/command/find_package.↩ html To specify a specific package location use <package>_ROOT CPM will also search along the CMAKE↩ _MODULE_PATH

Sometimes it is convenient to have your own sources used, in this case, use the CPM_<package>_SOUR↩ CE_DIR. Hence you may wish to use your own copy of SystemC CCI "bash cmake -B build -DCPM_↩ SystemCCCI_SOURCE=/path/to/your/cci/source

```
It may also be convenient to have all the source files downloaded, you may do this by running
```bash
cmake -B build -DCPM_SOURCE_CACHE=`pwd`/Packages
```

This will populate the directory Packages Note that the cmake file system will automatically use the directory called Packages as source, if it exists.

NB, CMake holds a cache of compiled modules in ∼/.cmake/ Sometimes this can confuse builds. If you seem to be picking up the wrong version of a module, then it may be in this cache. It is perfectly safe to delete it.

1.3.2.1 Common CMake options

CMAKE_INSTALL_PREFIX : Install directory for the package and binaries. CMAKE_BUILD_TYPE : DEBUG or RELEASE

The library assumes the use of C++14, and is compatible with SystemC versions from SystemC 2.3.1a.

For a reference docker please use the following script from the top level of the Virtual Platform:

```
curl --header 'PRIVATE-TOKEN: W1Z9U8S_5BUEX1_Y29iS'
       'https://git.greensocs.com/api/v4/projects/65/repository/files/docker_vp.sh/raw?ref=master' -o docker_vp.sh
chmod +x ./docker_vp.sh
./docker_vp.sh
> cmake -B build;cd build; make -j
```
```

1.3.2.2 passwords for git.greensocs.com

To avoid using passwords for git.greensocs.com please add a ssh key to your git account. You may also use a key-chain manager. As a last resort, the following script will populate ∼/.git-credentials with your username and password (in plain text)

```
git config --global credential.helper store
```

1.3.3 More documentation

More documentation, including doxygen generated API documentation can be found in the `/docs` directory.

1.3.4 The GreenSocs SystemC simple components library.

This includes simple models such as routers, memories and exclusive monitor. The components are "Loosely timed" only. They support DMI where appropriate, and make use of CCI for configuration.

It also has several unit tests for memory, router and exclusive monitor.

1.3.5 LIBGSSYNC

The GreenSocs Synchronization library provides a number of different policies for synchronizing between an external simulator (typically QEMU) and SystemC.

These are based on a proposed standard means to handle the SystemC simulator. This library provides a backwards compatibility layer, but the patched version of SystemC will perform better.

1.3.6 LIBGSUTILS

The GreenSocs basic utilities library contains utility functions for CCI, simple logging and test functions. It also includes some basic tlm port types

1.3.7 LIBQEMU-CXX

Libqemu-cxx encapsulates QEMU as a C++ object, such that it can be instanced (for instance) within a SystemC simulation framework.

1.3.8 LIBQBOX

Libqbox encapsulates QEMU in SystemC such that it can be instanced as a SystemC TLM-2.0 model.

1.3.9 Information about building and using the base-components library

The base-components library depends on the libraries : Libgsutls, SystemC, RapidJSON, SystemCCI, Lua and GoogleTest.

1.3.10 Information about building and using the libgssync library

The libgssync library depends on the libraries : base-components, libgsutils, SystemC, RapidJSON, SystemCCI, Lua and GoogleTest.

1.3.11 Information about building and using the libgsutils library

The libgsutils library depends on the libraries : SystemC, RapidJSON, SystemCCI, Lua and GoogleTest.

The GreenSocs CCI libraries allows two options for setting configuration parameters

`--gs_luafile <FILE.lua>` this option will read the lua file to set parameters.

`--param path.to.param=<value>` this option will allow individual parameters to be set.

NOTE, order is important, the last option on the command line to set a parameter will take preference.

This library includes a Configurable Broker (gs::ConfigurableBroker) which provides additional functionality. Each broker can be configured separately, and has a parameter itself for the configuration file to read. This is `lua_file`. Hence

```
--param path.to.module.lua_file="\"/host/path/to/lua/file""
```

Note that a string parameter must be quoted.

The lua file read by the ConfigurableBroker has relative paths - this means that in the example above the `path.to.module` portion of the absolute path should not appear in the (local) configuration file. (Hence changes in the hierarchy will not need changes to the configuration file).

1.3.12 Using yaml for configuration

If you would prefer to use yaml as a configuration language, `lyaml` provides a link. This can be downloaded from https://github.com/gvvaughan/lyaml

The following lua code will load "conf.yaml".

```
local lyaml   = require "lyaml"

function readAll(file)
    local f = assert(io.open(file, "rb"))
    local content = f:read("*all")
    f:close()
    return content
end

print "Loading conf.yaml"
yamldata=readAll("conf.yaml")
ytab=lyaml.load(yamldata)
for k,v in pairs(ytab) do
    _G[k]=v
end
yamldata=nil
ytab=nil
```

1.3.13 Information about building and using the libqemu-cxx library

The libgsutils library does not depend on any library.

1.3.14 Information about building and using the greensocs Qbox library

The greensocs Qbox library depends on the libraries : base-components, libgssync, libqemu-cxx, libgsutils, SystemC, RapidJSON, SystemCCI, Lua and GoogleTest.

1.3.15 The GreenSocs component library memory

The memory component allows you to add memory when creating an object of type `Memory("name",size)`.

The memory component consists of a simple target socket :`tlm_utils::simple_target_socket<Memory> socket`

1.3.16 The GreenSocs component library router

The router offers `add_target(socket, base_address, size)` as an API to add components into the address map for routing. (It is recommended that the addresses and size are CCI parameters).

It also allows to bind multiple initiators with `add_initiator(socket)` to send multiple transactions. So there is no need for the bind() method offered by sockets because the add_initiator method already takes care of that.

1.3.17 Functionality of the synchronization library

In addition the library contains utilities such as an thread safe event (async_event) and a real time speed limited for SystemC.

1.3.17.1 Suspend/Unsuspend interface

This patch adds four new basic functions to SystemC:

```
void sc_suspend_all(sc_simcontext* csc= sc_get_curr_simcontext())
void sc_unsuspend_all(sc_simcontext* csc= sc_get_curr_simcontext())
void sc_unsuspendable()
void sc_suspendable()
```

suspend_all/unsuspend_all : This pair of functions requests the kernel to 'atomically suspend' all processes (using the same semantics as the thread suspend() call). This is atomic in that the kernel will only suspend all the processes together, such that they can be suspended and unsuspended without any side effects. Calling suspend_all(), and subsiquently calling unsuspend_all() will have no effect on the suspended status of an individual process. A process may call suspend_all() followed by unsuspend_all, the calls should be 'paired', (multiple calls to either suspend_all() or unsuspend_all() will be ignored). Outside of the context of a process, it is the programmers responsibility to ensure that the calls are paired. As a consequence, multiple calls to suspend_all() may be made (within separate process, or from within sc_main). So long as there have been more calls to suspend_all() than to unsuspend_all(), the kernel will suspend all processes.

[note, this patch set does not add convenience functions, including those to find out if suspension has happened, these are expected to be layered ontop]

unsusbendable()/suspendable(): This pair of functions provides an 'opt-out' for specific process to the suspend_all(). The consequence is that if there is a process that has opted out, the kernel will not be able to suspend_all (as it would no longer be atomic). These functions can only be called from within a process. A process should only call suspendable/unsuspendable in pairs (multiple calls to either will be ignored). *Note that the default is that a process is marked as suspendable.*

Use cases: *1 : Save and Restore* For Save and Restore, the expectation is that when a save is requested, 'suspend_all' will be called. If there are models that are in an unsuspendable state, the entire simulation will be allowed to continue until such a time that there are no unsuspendable processes.

2 : External sync When an external model injects events into a SystemC model (for instance, using an 'async_request_update()'), time can drift between the two simulators. In order to maintain time, SystemC can be prevented from advancing by calling suspend_all(). If there are process in an unsuspendable state (for instance, processing on behalf of the external model), then the simulation will be allowed to continue. NOTE, an event injected into the kernel by an async_request_update will cause the kernel to execute the associated update() function (leaving the suspended state). The update function should arrange to mark any processes that it requires as unsuspendable before the end of the current delta cycle, to ensure that they are scheduled.

1.3.18 Using the ConfigurableBroker

The broker will self register in the SystemC CCI hierarchy. All brokers have a parameter `lua_file` which will be read and used to configure parameters held within the broker. This file is read at the *local* level, and paths are *relative* to the location where the ConfigurableBroker is instanced.

These brokers can be used as global brokers.

The `gs::ConfigurableBroker` can be instanced in 3 ways:

1. `ConfigurableBroker()` This will instance a 'Private broker' and will hide **ALL** parameters held within this broker.

   A local `lua_file` can be read and will set parameters in the private broker. This can be prevented by passing 'false' as a construction parameter (`ConfigurableBroker(false)`).

2. `ConfigurableBroker({{"key1","value1"},{"key2","value2")...})` This will instance a broker that sets and hides the listed keys. All other keys are passed through (exported). Hence the broker is 'invisible' for parameters that are not listed. This is specifically useful for structural parameters.

   It is also possible to instance a 'pass through' broker using `ConfigurationBroker({})`. This is useful to provide a *local* configuration broker than can, for instance, read a local configuration file.

   A local `lua_file` can be read and will set parameters in the private broker (exported or not). This can be prevented by passing 'false' as a construction parameter (`ConfigurableBroker(false)`). The `lua_file` will be read **AFTER** the construction key-value list and hence can be used to over-right default values in the code.

3. `ConfigurableBroker(argc, argv)` This will instance a broker that is typically a global broker. The argc/argv values should come from the command line. The command line will be parsed to find:

   > `-p, --param path.to.param=<value>` this option will allow individual parameters to be set.

   > `-l, --gs_luafile <FILE.lua>` this option will read the lua file to set parameters. Similar functionality can be achieved using –param lua_file="<FILE.lua>".

   A `{{key,value}}` list can also be provided, otherwise it is assumed to be empty. Such a list will set parameter values within this broker. These values will be read and used **BEFORE** the command line is read.

   Finally **AFTER** the command line is read, if the `lua_file` parameter has been set, the configuration file that it indicates will also be read. This can be prevented by passing 'false' as a construction parameter (`ConfigurableBroker(argc, argv, false)`). The `lua_file` will be read **AFTER** the construction key-value list, and after the command like, so it can be used to over-right default values in either.

1.3.19 Print out the available params

It is possible to display the list of available cci parameters with the `-h` option when launching the virtual platform.

CAUTION:

This will only print the parameters at the begining of simulation.

1.3.20 Instanciate Qemu

A QemuManager is required in order to instantiate a Qemu instance. A QemuManager will hold, and maintain the instance until the end of execution. The QemuInstance can contain one or many CPU's and other devices. To create a new instance you can do this:

```
{c++}
    QemuInstanceManager m_inst_mgr;
```

then you can initialize it by providing the QemuInstance object with the QemuInstanceManager object which will call the `new_instance` method to create a new instance.

```
{c++}
    QemuInstance m_qemu_inst(m_inst_mgr.new_instance(QemuInstance::Target::AARCH64))
```

In order to add a CPU device to an instance they can be constructed as follows:

```
{c++}
    sc_core::sc_vector<QemuCpuArmCortexA53> m_cpus

    m_cpus("cpu", 32, [this] (const char *n, size_t i) { return new QemuCpuArmCortexA53(n, m_qemu_inst); })
```

You can change the CPUs to those listed below in the "CPU" section

Interrupt Controllers and others devices also need a QEMU instance and can be set up as follows:

```
{c++}
    QemuArmGicv3 m_gic("gic", m_qemu_inst);
    QemuUartPl011 m_uart("uart", m_qemu_inst)
```

1.3.21 QEMU Arguments

QEMU arguments can be added to an entire instance using the configuration mechanism. The argument name should be in a form "`name.of.your.qemu.instance.args.-ARG" = "value`".

The QEMU instance provides the following default arguments :

```
"-M", "none", /* no machine */
"-m", "2048", /* used by QEMU to set some interal buffer sizes */
"-monitor", "null", /* no monitor */
"-serial", "null", /* no serial backend */
"-display", "none", /* no GUI */
```

Example : Using the lua file configuration mechanism to set `-monitor` to enable telnet communication with QE↩ MU, with the QEMU instance "platform.QemuInstance" the lua file should contain :

```
["platform.QemuInstance.args.-monitor"] = "tcp:127.0.0.1:55555,server,nowait",
```

To check that the QEMU argument has been added QEMU will report : `Added QEMU argument: "name of the argument" "value of the argument"`

In the example it's : `Added QEMU argument : -monitor tcp:127.0.0.1:55555,server,nowait`

Telnet can be used to connector to the monitor as follows:

```
$ telnet 127.0.0.1 55555
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
QEMU 5.1.0 monitor - type 'help' for more information
(qemu) quit
quit
Connection closed by foreign host.
```

NOTE :

This should not be used to enable GDB.

1.3.22 Enabling GDB per CPU

In order to connect a GDB the CCI parameter `name.of.cpu.gdb-port` must be set a none zero value.

For instance

```
$ ./build/vp --gs_luafile conf.lua -p platform.cpu_1.gdb-port=1234
```

Will open a gdb server on port 1234, for `cpu_1`, and the virtual platform will wait for GDB to connect.

1.3.23 The components of libqbox

1.3.23.1 CPU

The libqbox library supports several CPU architectures such as ARM and RISCV.

- In ARM architectures the library supports the cortex-a53 and the Neoverse-N1 which is based on the cortex-a76 architecture which itself derives from the cortex-a75/73/72.
- In RISCV architecture, the library manages only the riscv64.

1.3.23.2 IRQ-CTRL

The library also manages interrupts by providing :

- ARM GICv2
- ARM GICv3 which are Arm Generic Interrupt Controller.

Then :

- SiFive CLINT
- SiFive PLIC which are also Interrupt controller but for SiFive.

1.3.23.3 UART

Finally, it has 2 uarts:

- pl011 for ARM
- 16550 for more general use

1.3.23.4 PORTS

The library also provides socket initiators and targets for Qemu

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

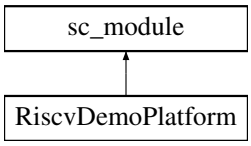
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 RiscvDemoPlatform Class Reference

Inheritance diagram for RiscvDemoPlatform:



Public Member Functions

- **RiscvDemoPlatform** (const sc_core::sc_module_name &n)

Static Public Attributes

- static constexpr size_t **ROM_SIZE** = 64 ∗ 1024
- static constexpr size_t **SRAM_SIZE** = 128 ∗ 1024
- static constexpr size_t **QSPI_SIZE** = 256 ∗ 1024 ∗ 1024
- static constexpr size_t **BOOT_GPIO_SIZE** = 8
- static constexpr size_t **UART_IRQ** = 273

Protected Member Functions

- void **setup_cpus** ()
- void **setup_memory_mapping** ()
- void **setup_irq_mapping** ()
- void **setup_boot_gpio** ()
- bool **load_blobs** ()
- void **do_openspi_linux_bootloader** ()
- void **do_bootloader** ()

Protected Attributes

- gs::ConfigurableBroker **m_broker**
- cci::cci_param< int > **m_quantum_ns**
- cci::cci_param< int > **m_gdb_port**
- cci::cci_param< int > **m_boot_gpio_val**
- cci::cci_param< int > **m_dram_size**
- cci::cci_param< std::string > **m_rom_blob_file**
- cci::cci_param< std::string > **m_sram_blob_file**
- cci::cci_param< std::string > **m_qspi_blob_file**
- cci::cci_param< std::string > **m_dram_blob_file**
- cci::cci_param< std::string > **m_opensbi_file**
- cci::cci_param< std::string > **m_kernel_file**
- cci::cci_param< std::string > **m_dtb_file**
- cci::cci_param< cci::uint64 > **m_addr_map_rom**
- cci::cci_param< cci::uint64 > **m_addr_map_sram**
- cci::cci_param< cci::uint64 > **m_addr_map_dram**
- cci::cci_param< cci::uint64 > **m_addr_map_uart**
- cci::cci_param< cci::uint64 > **m_addr_map_clint**
- cci::cci_param< cci::uint64 > **m_addr_map_plic**
- cci::cci_param< cci::uint64 > **m_addr_map_qspi_mmap**
- cci::cci_param< cci::uint64 > **m_addr_map_boot_gpio**
- QemuInstanceManager **m_inst_mgr**
- QemuInstance & **m_qemu_inst**
- sc_core::sc_vector< QemuCpuRiscv64Rv64 > **m_cpus**
- QemuRiscvSifivePlic **m_plic**
- QemuRiscvSifiveClint **m_clint**
- Router **m_router**
- Memory **m_resetvec_rom**
- Memory **m_rom**
- Memory **m_sram**
- Memory **m_dram**
- Memory **m_qspi**
- Memory **m_boot_gpio**
- QemuUart16550 **m_uart**

The documentation for this class was generated from the following file:

- /home/thomas/Documents/GreenSocs/build-platforme/greensocs-riscv64/src/main.cc

Index