

qbox-extra

Generated by Doxygen 1.8.13

Contents

1	Main Page	1
1.1	GreenSocs Build and make system	1
1.2	How to build	1
1.2.1	cmake version	1
1.2.2	details	1
1.2.2.1	Common CMake options	2
1.2.2.2	passwords for git.greensocs.com	2
1.2.3	More documentation	2
1.2.4	The GreenSocs SystemC simple components library.	2
1.2.5	LIBGSSYNC	2
1.2.6	LIBGSUTILS	2
1.2.7	LIBQEMU-CXX	3
1.2.8	LIBQBOX	3
1.2.9	Information about building and using the extra-components library	3
1.2.10	Information about building and using the base-components library	3
1.2.11	Information about building and using the libgssync library	3
1.2.12	Information about building and using the libgsutils library	3
1.2.13	Using yaml for configuration	4
1.2.14	The GreenSocs extra-component library	4
1.2.15	The GreenSocs component library memory	5
1.2.16	The GreenSocs component library router	5
1.2.17	Functionality of the synchronization library	5
1.2.17.1	Suspend/Unsuspend interface	5
1.2.18	Using the ConfigurableBroker	6

2	Hierarchical Index	7
2.1	Class Hierarchy	7
3	Class Index	9
3.1	Class List	9
4	Class Documentation	11
4.1	QemuGPEX::Device Class Reference	11
4.2	QemuGPEX Class Reference	11
4.3	QemuNvmeDisk Class Reference	12
	Index	15

Chapter 1

Main Page

This library support PCI devices - specifically NVME

1.1 GreenSocs Build and make system

1.2 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)

1.2.1 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
```

1.2.2 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 standard 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>_SOURCE_↵_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.

1.2.2.1 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
```

1.2.2.2 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
```

1.2.3 More documentation

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

1.2.4 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.

1.2.5 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.

1.2.6 LIBGSUTILS

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

1.2.7 LIBQEMU-CXX

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

1.2.8 LIBQBOX

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

1.2.9 Information about building and using the extra-components library

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

1.2.10 Information about building and using the base-components library

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

1.2.11 Information about building and using the libgssync library

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

1.2.12 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).

1.2.13 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
```

1.2.14 The GreenSocs extra-component library

The extra-component library contains complex components.

Components available:

- **GPEX** : The extra-component `Gpex` allow you to add devices with the method `add_device(Device &dev)`, you can add for example an NVME disk:

```
{c++}
    m_gpex.add_device(m_disk1);
```

- **NVME** : This allows an NVME disk to be added. Initialise `QemuNvmeDisk` with a `qemu` instance from the `libqbox` library (see the `libqbox` section to initiate a `qemu` instance):

```
{c++}
    QemuNvmeDisk m_disk1("disk1", m_qemu_inst)
```

The class `QemuNvmeDisk` has 3 parameters: The constructor : `QemuNvmeDisk(const sc_core::sc_<_module_name &name, QemuInstance &inst)`

- "name" : name of the disk
- "inst" : instance Qemu

The parameters :

- "serial" : Serial name of the nvme disk
- "bloc-file" : Blob file to load as data storage
- "drive-id"

1.2.15 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`

1.2.16 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.

1.2.17 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.

1.2.17.1 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 subsequently 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]

unsuspendable()/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.

1.2.18 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 instantiated.

These brokers can be used as global brokers.

The `gs::ConfigurableBroker` can be instantiated 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 line, so it can be used to over-right default values in either.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

QemuDevice	
QemuGPEX	11
QemuGPEX::Device	11
QemuNvmeDisk	12
QemuInitiatorIface	
QemuGPEX	11

Chapter 3

Class Index

3.1 Class List

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

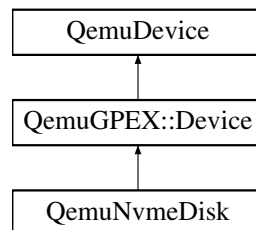
QemuGPEX::Device	11
QemuGPEX	11
QemuNvmeDisk	12

Chapter 4

Class Documentation

4.1 QemuGPEX::Device Class Reference

Inheritance diagram for QemuGPEX::Device:



Public Member Functions

- **Device** (const sc_core::sc_module_name &name, QemuInstance &inst, const char *qom_type)
- void **end_of_elaboration** () override

Protected Member Functions

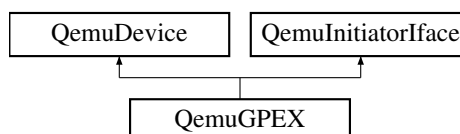
- virtual void **gpex_realize** (qemu::Bus &pcie_bus)

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

- /home/thomas/Documents/GreenSocs/build-lib/extra-components/include/libqbox-extra/components/pci/gpex.↔h

4.2 QemuGPEX Class Reference

Inheritance diagram for QemuGPEX:



Classes

- class [Device](#)

Public Member Functions

- **QemuGPEX** (const sc_core::sc_module_name &name, QemuInstance &inst, uint64_t mmio_addr, uint64_t mmio_size, uint64_t mmio_high_addr, uint64_t mmio_high_size)
- void **add_device** ([Device](#) &dev)
- void **before_end_of_elaboration** () override
- void **end_of_elaboration** () override
- virtual sc_core::sc_time **initiator_get_local_time** () override
- virtual void **initiator_set_local_time** (const sc_core::sc_time &t) override
- virtual void **initiator_customize_tlm_payload** (TlmPayload &payload) override
- virtual void **initiator_tidy_tlm_payload** (TlmPayload &payload) override

Public Attributes

- QemuInitiatorSocket **bus_master**
- QemuTargetSocket **ecam_iface**
- QemuTargetSocket **mmio_iface**
- QemuTargetSocket **mmio_iface_high**
- QemuTargetSocket **pio_iface**
- sc_core::sc_vector< QemuInitiatorSignalSocket > **irq_out**
- int **irq_num** [4]

Protected Attributes

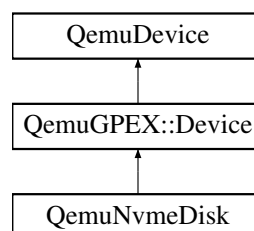
- uint64_t **m_mmio_addr**
- uint64_t **m_mmio_size**
- uint64_t **m_mmio_high_addr**
- uint64_t **m_mmio_high_size**
- qemu::MemoryRegion **m_mmio_alias**
- qemu::MemoryRegion **m_mmio_high_alias**
- std::vector< [Device](#) * > **devices**

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

- /home/thomas/Documents/GreenSocs/build-lib/extra-components/include/libqbox-extra/components/pci/gpex.h↔

4.3 QemuNvmeDisk Class Reference

Inheritance diagram for QemuNvmeDisk:



Public Member Functions

- **QemuNvmeDisk** (const sc_core::sc_module_name &name, QemuInstance &inst)
- void **before_end_of_elaboration** () override
- void **gpex_realize** (qemu::Bus &bus) override

Protected Attributes

- cci::cci_param< std::string > **p_serial**
- cci::cci_param< std::string > **p_blob_file**
- std::string **m_drive_id**

Additional Inherited Members

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

- /home/thomas/Documents/GreenSocs/build-lib/extra-components/include/libqbox-extra/components/pci/nvme.h↔

Index

QemuGPEX::Device, [11](#)

QemuGPEX, [11](#)

QemuNvmeDisk, [12](#)