base-components

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

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.

1.1 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 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>_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 LIBGSUTILS

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

1.2.5 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).

1.2.6 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
```

1.2.7 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`

1.2.8 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.

1.2.9 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.

1.2.10 The GreenSocs component Tests

It is possible to test the correct functioning of the different components with the tests that are proposed.

Once you have compiled your library, you will have a tests folder in your construction directory.

In this test folder you will find several folders, each corresponding to a component and each containing an executable.

You can run the executable with :

```
./build/tests/<name_of_component>/<name_of_component>-tests
```

If you want a more general way to check the correct functioning of the components you can run all the tests at the same time with :

```
make test
```

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

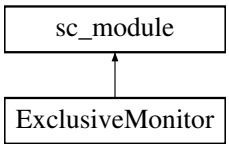
Class Documentation

4.1 ExclusiveMonitor Class Reference

ARM-like global exclusive monitor.

`#include <exclusive-monitor.h>`

Inheritance diagram for ExclusiveMonitor:



Public Member Functions

- **ExclusiveMonitor** (const sc_core::sc_module_name &name)
- **ExclusiveMonitor** (const ExclusiveMonitor &)=delete

Public Attributes

- tlm_utils::simple_target_socket< ExclusiveMonitor > **front_socket**
- tlm_utils::simple_initiator_socket< ExclusiveMonitor > **back_socket**

4.1.1 Detailed Description

ARM-like global exclusive monitor.

This component models an ARM-like global exclusive monitor. It connects in front of an target and monitors accesses to it. It behaves as follows:

- On an exclusive load, it internally marks the corresponding region as locked. The load is forwarded to the target.

- On an exclusive store to the same region, the region is unlocked, and the store is forwarded to the target.
- When an initiator perform an exclusive load while already owning a region, the region gets unlocked before the new one is locked.
- A regular store will unlock all intersecting regions
- If an exclusive store fails, that it, corresponds to a region which is not locked, or is locked by another initiator, or does not exactly match the store boundaries, the failure is reported into the TLM exclusive extension and the store is *not* forwarded to the target.
- DMI invalidation is performed when a region is locked.
- DMI requests are intercepted and modified accordingly to match the current locking state.
- DMI hints (the is_dmi_allowed() flag in transactions) is also intercepted and modified if necessary.

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

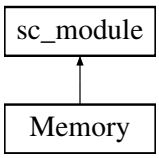
- /home/thomas/Documents/GreenSocs/build-lib/base-components/include/greensocs/base-components/misc/exclusive-monitor.h

4.2 Memory Class Reference

A memory component that can add memory to a virtual platform project.

`#include <memory.h>`

Inheritance diagram for Memory:



Public Member Functions

- **Memory** (sc_core::sc_module_name name, uint64_t size)
- **Memory** (const Memory &)=delete
- uint64_t size ()

  this function returns the size of the memory
- void map (const char *filename)

  This function maps a host file system file into the memory, such that the results of the memory will be maintained between runs. This can be useful for emulating a flash ram for instance.
- size_t load (std::string filename, uint64_t addr)

  This function reads a file into memory and can be used to load an image.
- void load (const uint8_t *ptr, uint64_t len, uint64_t addr)

  copy an existing image in host memory into the modelled memory

Public Attributes

- tlm_utils::simple_target_socket< Memory > **socket**

4.2.1 Detailed Description

A memory component that can add memory to a virtual platform project.

This component models a memory. It has a simple target socket so any other component with an initiator socket can connect to this component. It behaves as follows:

- The memory does not manage time in any way
- It is only an LT model, and does not handle AT transactions
- It does not manage exclusive accesses
- You can manage the size of the memory during the initialization of the component
- Memory does not allocate individual "pages" but a single large block
- It supports DMI requests with the method `get_direct_mem_ptr`
- DMI invalidates are not issued.

4.2.2 Member Function Documentation

4.2.2.1 load() [1/2]

```
size_t Memory::load (
            std::string filename,
            uint64_t addr )  [inline]
```

This function reads a file into memory and can be used to load an image.

Parameters

filename	Name of the file
addr	the address where the memory file is to be read

Returns

size_t

4.2.2.2 load() [2/2]

```
void Memory::load (
            const uint8_t * ptr,
            uint64_t len,
            uint64_t addr )  [inline]
```

copy an existing image in host memory into the modelled memory

Parameters

ptr	Pointer to the memory
len	Length of the read
addr	Address of the read

4.2.2.3 map()

```
void Memory::map (
            const char * filename )  [inline]
```

This function maps a host file system file into the memory, such that the results of the memory will be maintained between runs. This can be useful for emulating a flash ram for instance.

Parameters

filename	Name of the file

4.2.2.4 size()

```
uint64_t Memory::size ( )  [inline]
```

this function returns the size of the memory

Returns

the size of the memory of type uint64_t

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

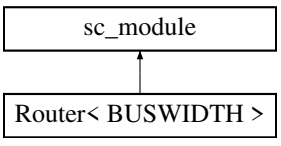
- /home/thomas/Documents/GreenSocs/build-lib/base-components/include/greensocs/base-components/memory.↩ h

4.3 Router< BUSWIDTH > Class Template Reference

A Router component that can add router to a virtual platform project to manage the various transactions.

`#include <router.h>`

Inheritance diagram for Router< BUSWIDTH >:



Public Member Functions

- **Router** (const sc_core::sc_module_name &nm)
- **Router** (const Router &)=delete
- void add_target (TargetSocket &t, uint64_t address, uint64_t size)

  This method will bind a target to the router. The router will register its address and size according to the parameters we have given it.
- void add_initiator (InitiatorSocket &i)

  This method will bind a Initiator to the router.

Protected Member Functions

- virtual void **before_end_of_elaboration** ()

4.3.1 Detailed Description

template<unsigned int BUSWIDTH = 32>
class Router< BUSWIDTH >

A Router component that can add router to a virtual platform project to manage the various transactions.

This component models a router. It has a single multi-target socket so any other component with an initiator socket can connect to this component. It behaves as follows:

- Manages exclusive accesses, adding this router as a 'hop' in the exclusive access extension (see Green↩ Socs/libgsutils).
- Manages connections to multiple initiators and targets with the method `add_initiator` and `add_↩ target`.
- Allows to manage read and write transactions with `b_transport` and `transport_dbg` methods.
- Supports passing through DMI requests with the method `get_direct_mem_ptr`.
- Handles invalidation of multiple DMI pointers with the method `invalidate_direct_mem_ptr` which passes the invalidate back to *all* initiators.
- It checks for each transaction if the address is valid or not and returns an error if the address is invalid with the method `decode_address`.

4.3.2 Member Function Documentation

4.3.2.1 add_initiator()

```
template<unsigned int BUSWIDTH = 32>
void Router< BUSWIDTH >::add_initiator (
             InitiatorSocket & i )  [inline]
```

This method will bind a Initiator to the router.

Parameters

i	initiator socket which will allow to bind the router with the initiator

4.3.2.2 add_target()

```
template<unsigned int BUSWIDTH = 32>
void Router< BUSWIDTH >::add_target (
             TargetSocket & t,
             uint64_t address,
             uint64_t size )  [inline]
```

This method will bind a target to the router. The router will register its address and size according to the parameters we have given it.

Parameters

t	target socket which will allow to bind the router with the target (ex: memory)
address	Address of the target
size	Size of the target

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

- /home/thomas/Documents/GreenSocs/build-lib/base-components/include/greensocs/base-components/router.↩
h

Index