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.

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`

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.

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

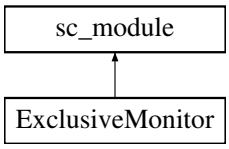
Class Documentation

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:

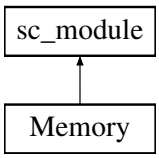
- /home/thomas/Documents/GreenSocs/build-lib/base-components/include/greensocs/base-components/misc/exclusive-monitor.h

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:

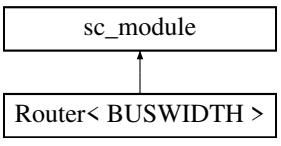
- /home/thomas/Documents/GreenSocs/build-lib/base-components/include/greensocs/base-components/memory.↩ h

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 GreenSocs/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