gsutils

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

[//]: # DONT EDIT THIS FILE

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

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 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).

1.2.5 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
```

1.2.6 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.

1.2.7 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.

1.2.8 The GreenSocs utils Tests

Tests are available for you to check that the library is working properly.

To do this, you must run the following command in the build directoty `build/`:

```
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

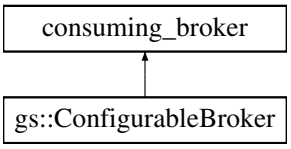
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 gs::ConfigurableBroker Class Reference

Inheritance diagram for gs::ConfigurableBroker:



Public Member Functions

- void **print_help** ()
- **ConfigurableBroker** (const std::string &name=BROKERNAME, bool load_conf_file=true)
- **ConfigurableBroker** (bool load_conf_file)
- **ConfigurableBroker** (std::initializer_list< cci_name_value_pair > list, bool load_conf_file=true)
- **ConfigurableBroker** (const int argc, char *const argv[ ], std::initializer_list< cci_name_value_pair > list={}, bool load_conf_file=true)
- std::string **relname** (const std::string &n) const
- cci_originator **get_value_origin** (const std::string &parname) const
- bool **has_preset_value** (const std::string &parname) const
- cci_value **get_preset_cci_value** (const std::string &parname) const
- void **lock_preset_value** (const std::string &parname)
- cci_value **get_cci_value** (const std::string &parname) const
- void **add_param** (cci_param_if *par)
- void **remove_param** (cci_param_if *par)
- std::vector< cci_name_value_pair > **get_unconsumed_preset_values** () const
- cci_preset_value_range **get_unconsumed_preset_values** (const cci_preset_value_predicate &pred) const
- void **set_preset_cci_value** (const std::string &parname, const cci_value &cci_value, const cci_originator &originator)
- cci_param_untyped_handle **get_param_handle** (const std::string &parname, const cci_originator &originator) const
- std::vector< cci_param_untyped_handle > **get_param_handles** (const cci_originator &originator) const
- bool **is_global_broker** () const

Public Attributes

- std::set< std::string > **expose**
- cci_param< std::string > **conf_file**

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

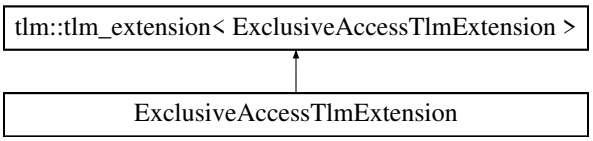
- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/cciutils.h

4.2 ExclusiveAccessTlmExtension Class Reference

Exclusive load/store TLM extension.

`#include <exclusive-access.h>`

Inheritance diagram for ExclusiveAccessTlmExtension:



Classes

- class InitiatorId

Public Types

- enum **ExclusiveStoreStatus** { **EXCLUSIVE_STORE_NA** = 0, **EXCLUSIVE_STORE_SUCCESS**, **EXCL↩ USIVE_STORE_FAILURE** }

Public Member Functions

- **ExclusiveAccessTlmExtension** (const ExclusiveAccessTlmExtension &)=default
- virtual tlm_extension_base * **clone** () const override
- virtual void **copy_from** (const tlm_extension_base &ext) override
- void **set_exclusive_store_success** ()
- void **set_exclusive_store_failure** ()
- ExclusiveStoreStatus **get_exclusive_store_status** () const
- void **add_hop** (int id)
- const InitiatorId & **get_initiator_id** () const

4.2.1 Detailed Description

Exclusive load/store TLM extension.

Exclusive load/store TLM extension. It embeds an initiator ID (InitiatorId) and a store status (ExclusiveStoreStatus).

The initiator ID is meant to be composed by all the routers on the path that support this extension. Each router can call add_hop on the extension with an unique ID correponding to the initiator the request is comming from (typically the index of the initiator on the router). The first initiator is not required call add_hop since an empty InitiatorId is a perfectly valid ID (in the case the initiator would be directly connected to a target, without routers in between). It can still do it if it needs to emit exclusive transactions with different exclusive IDs.

The store status is valid after a TLM_WRITE_COMMAND transaction and indicate whether the exclusive store succeeded or not.

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

- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/tlm-extensions/exclusive-access.h

4.3 ExclusiveAccessTlmExtension::InitiatorId Class Reference

Public Member Functions

- void **add_hop** (int id)
- bool **operator<** (const InitiatorId &o) const
- bool **operator==** (const InitiatorId &o) const
- bool **operator!=** (const InitiatorId &o) const

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

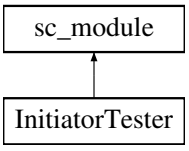
- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/tlm-extensions/exclusive-access.h

4.4 InitiatorTester Class Reference

A TLM initiator to do testing on a target.

`#include <initiator-tester.h>`

Inheritance diagram for InitiatorTester:



Public Types

- using **TlmGenericPayload** = tlm::tlm_generic_payload
- using **TlmResponseStatus** = tlm::tlm_response_status
- using **TlmDmi** = tlm::tlm_dmi
- using **InvalidateDirectMemPtrFn** = std::function< void(uint64_t, uint64_t)>

Public Member Functions

- **InitiatorTester** (const sc_core::sc_module_name &n)
- TlmResponseStatus do_b_transport (TlmGenericPayload &txn)

  Perform a b_transport TLM transaction using the `txn` TLM payload.
- TlmResponseStatus do_transport_dbg (TlmGenericPayload &txn)

  Perform a transport_dbg TLM transaction using the `txn` TLM payload.
- TlmResponseStatus do_transaction (TlmGenericPayload &txn, bool debug=false)

  Perform a TLM transaction using the `txn` TLM payload.
- TlmResponseStatus do_read_with_txn_and_ptr (TlmGenericPayload &txn, uint64_t addr, uint8_t *data, size_t len, bool debug=false)

  Perform a simple read into the buffer pointed by `data` with a pre-set payload.
- TlmResponseStatus do_write_with_txn_and_ptr (TlmGenericPayload &txn, uint64_t addr, const uint8_t *data, size_t len, bool debug=false)

  Perform a simple write with data pointed by `data` with a pre-set payload.
- TlmResponseStatus do_read_with_ptr (uint64_t addr, uint8_t *data, size_t len, bool debug=false)

  Perform a simple read into the buffer pointed by `data`
- TlmResponseStatus do_write_with_ptr (uint64_t addr, const uint8_t *data, size_t len, bool debug=false)

  Perform a simple write with data pointed by `data`
- template<class T >
  TlmResponseStatus do_read_with_txn (TlmGenericPayload &txn, uint64_t addr, T &data, bool debug=false)

  Perform a simple read with a pre-set payload.
- template<class T >
  TlmResponseStatus do_write_with_txn (TlmGenericPayload &txn, uint64_t addr, const T &data, bool debug=false)

  Perform a simple write with a pre-set payload.
- template<class T >
  TlmResponseStatus do_read (uint64_t addr, T &data, bool debug=false)

  Perform a simple read into `data`
- template<class T >
  TlmResponseStatus do_write (uint64_t addr, const T &data, bool debug=false)

  Perform a simple write.
- void set_next_txn_delay (const sc_core::sc_time &delay)

  Set the delay value to use for the next b_transport call.
- const sc_core::sc_time & get_last_txn_delay () const

  Get the delay value resulting of the last b_transport call.
- unsigned int get_last_transport_debug_ret () const

  Get the return value of the last transport_dbg call.
- bool get_last_dmi_hint () const

  Get the DMI hint value of the last transaction (the is_dmi_allowed() flag in the payload)
- bool do_dmi_request (uint64_t addr)

  Perform a get_direct_mem_ptr call by specifying an address.
- const TlmDmi & get_last_dmi_data () const

  Get the DMI data returned by the last get_direct_mem_ptr call.
- void register_invalidate_direct_mem_ptr (InvalidateDirectMemPtrFn cb)

  Register a callback on invalidate_direct_mem_ptr event.

Public Attributes

- tlm_utils::simple_initiator_socket< InitiatorTester > **socket**

Protected Member Functions

- virtual void **prepare_txn** (TlmGenericPayload &txn, bool is_read, uint64_t addr, uint8_t *data, size_t len)

4.4.1 Detailed Description

A TLM initiator to do testing on a target.

This class allows to test a target by providing helpers to standard TLM operations. Those helpers rangs from the most generic to the most simplified one. The idea is to provide simple helpers for the most common cases, while still allowing full flexibility if needed.

The `prepare_txn` method can be overriden if needed when inheriting this class, to customize the way payloads are filled before a transaction. One can also use the *_with_txn helpers and provide an already filled payload with e.g. an extension. Please note however that `prepare_txn` is still called on the payload to fill compulsory fields (namely the address, data pointer, data length and TLM command).

Read/write helpers return the `tlm::tlm_response_status` value of the resulting transaction. The DMI hint value of the last transaction (the `is_dmi_allowed()` flag) can be retrieved using the `get_last_dmi_hint` method.

When using standard read/write helpers, one can specify the value of the b_transport `delay` parameter, using the `set_next_txn_delay` method. This delay value can then be retrieved after the transaction using the `get_last_txn_delay` method (to check the value written back by the target).

Some helpers have a `debug` argument defaulting to `false`, when set to true, transport_dbg is called instead of b_transport on the socket. The transport_dbg return value is accessible through the `get_last_transport_debug_ret` method.

Regarding DMI requests, one can use the `do_dmi_request` helper to do a simple get_direct_mem_ptr call with only an address. The resulting tlm::tlm_dmi data can be retrieved using the `get_last_dmi_data` method.

One can also register a callback to catch DMI invalidations on the backward path of the socket, using the `register_invalidate_direct_mem_ptr` method.

4.4.2 Member Function Documentation

4.4.2.1 do_b_transport()

```
TlmResponseStatus InitiatorTester::do_b_transport (
            TlmGenericPayload & txn )  [inline]
```

Perform a b_transport TLM transaction using the `txn` TLM payload.

This method performs a b_transport transaction using the `txn` pre-filled payload. The transaction is not altered by this method so it should be completely filled prior to calling this method.

Parameters

in,out	*txn*	The payload to use for the transaction

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.2 do_dmi_request()

```
bool InitiatorTester::do_dmi_request (
            uint64_t addr )  [inline]
```

Perform a get_direct_mem_ptr call by specifying an address.

Perform a DMI request by specifying an address for the request. The DMI data can be retrieved using the `get_↩ last_dmi_data` method.

Returns

the value returned by the get_direct_mem_ptr call

4.4.2.3 do_read()

```
template<class T >
TlmResponseStatus InitiatorTester::do_read (
            uint64_t addr,
            T & data,
            bool debug = false )  [inline]
```

Perform a simple read into `data`

Parameters

in	*addr*	Address of the read
out	*data*	Where to retrieve the read value
in	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.4 do_read_with_ptr()

```
TlmResponseStatus InitiatorTester::do_read_with_ptr (
            uint64_t addr,
            uint8_t * data,
            size_t len,
            bool debug = false )  [inline]
```

Perform a simple read into the buffer pointed by `data`

Parameters

in	*addr*	Address of the read
out	*data*	Pointer to the buffer where to store the read data
in	*len*	Length of the read
in	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.5 do_read_with_txn()

```
template<class T >
TlmResponseStatus InitiatorTester::do_read_with_txn (
            TlmGenericPayload & txn,
            uint64_t addr,
            T & data,
            bool debug = false )  [inline]
```

Perform a simple read with a pre-set payload.

This method reads data into `data`. It uses the `txn` payload for the transaction, by overwriting the address, data pointer, data lenght and command fields of it. Other field are left untouched by this initiator (they could be altered by the target).

Parameters

in,out	*txn*	The payload to use for the transaction
in	*addr*	Address of the read
out	*data*	Where to retrieve the read value
in	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.6 do_read_with_txn_and_ptr()

```
TlmResponseStatus InitiatorTester::do_read_with_txn_and_ptr (
            TlmGenericPayload & txn,
            uint64_t addr,
            uint8_t * data,
            size_t len,
            bool debug = false )  [inline]
```

Perform a simple read into the buffer pointed by `data` with a pre-set payload.

This method performs a read into the buffer pointed by `data`. It uses the `txn` payload for the transaction, by overwriting the address, data pointer, data lenght and command fields of it. Other field are left untouched by this initiator (they could be altered by the target).

Parameters

`in,out`	*txn*	The payload to use for the transaction
`in`	*addr*	Address of the read
`out`	*data*	Pointer to the buffer where to store the read data
`in`	*len*	Length of the read
`in`	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.7 do_transaction()

```
TlmResponseStatus InitiatorTester::do_transaction (
            TlmGenericPayload & txn,
            bool debug = false )  [inline]
```

Perform a TLM transaction using the `txn` TLM payload.

This method performs a transaction using the `txn` pre-filled payload. The transaction is not altered by this method so it should be completely filled prior to calling this method.

Parameters

`in,out`	*txn*	The payload to use for the transaction
`in`	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.8 do_transport_dbg()

```
TlmResponseStatus InitiatorTester::do_transport_dbg (
            TlmGenericPayload & txn )  [inline]
```

Perform a transport_dbg TLM transaction using the `txn` TLM payload.

This method performs a transport_dbg transaction using the `txn` pre-filled payload. The transaction is not altered by this method so it should be completely filled prior to calling this method.

Parameters

`in,out`	*txn*	The payload to use for the transaction

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.9 do_write()

```
template<class T >
TlmResponseStatus InitiatorTester::do_write (
            uint64_t addr,
            const T & data,
            bool debug = false )  [inline]
```

Perform a simple write.

Parameters

`in`	*addr*	Address of the write
`in`	*data*	Data to write (note that this method does not guarantee `data` won't be modified by the target. It does not perform a prior copy to enforce this)
`in`	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.10 do_write_with_ptr()

```
TlmResponseStatus InitiatorTester::do_write_with_ptr (
            uint64_t addr,
            const uint8_t * data,
            size_t len,
            bool debug = false )  [inline]
```

Perform a simple write with data pointed by `data`

Parameters

in	*addr*	Address of the write
in	*data*	Pointer to the data to write (note that this method does not guarantee `data` won't be modified by the target. It does not perform a prior copy to enforce this)
in	*len*	Length of the write
in	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.11 do_write_with_txn()

```
template<class T >
TlmResponseStatus InitiatorTester::do_write_with_txn (
            TlmGenericPayload & txn,
            uint64_t addr,
            const T & data,
            bool debug = false )  [inline]
```

Perform a simple write with a pre-set payload.

This method performs a write from `data`. It uses the `txn` payload for the transaction, by overwriting the address, data pointer, data lenght and command fields of it. Other field are left untouched by this initiator (they could be altered by the target).

Parameters

in,out	*txn*	The payload to use for the transaction
in	*addr*	Address of the write
in	*data*	The value to write (note that this method does not guarantee `data` won't be modified by the target. It does not perform a prior copy to enforce this)
in	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.12 do_write_with_txn_and_ptr()

```
TlmResponseStatus InitiatorTester::do_write_with_txn_and_ptr (
            TlmGenericPayload & txn,
            uint64_t addr,
            const uint8_t * data,
```

```
size_t len,
bool debug = false )  [inline]
```

Perform a simple write with data pointed by `data` with a pre-set payload.

This method performs a write from the buffer pointed by `data`. It uses the `txn` payload for the transaction, by overwriting the address, data pointer, data lenght and command fields of it. Other field are left untouched by this initiator (they could be altered by the target).

Parameters

in	*addr*	Address of the write
in	*data*	Pointer to the data to write (note that this method does not guarantee `data` won't be modified by the target. It does not perform a prior copy to enforce this)
in	*len*	Length of the write
in	*debug*	Perform a transport_dbg instead of a b_transport

Returns

the tlm::tlm_response_status value of the transaction

4.4.2.13 get_last_dmi_data()

```
const TlmDmi& InitiatorTester::get_last_dmi_data ( ) const  [inline]
```

Get the DMI data returned by the last get_direct_mem_ptr call.

Returns

the DMI data returned by the last get_direct_mem_ptr call

4.4.2.14 get_last_dmi_hint()

```
bool InitiatorTester::get_last_dmi_hint ( ) const  [inline]
```

Get the DMI hint value of the last transaction (the is_dmi_allowed() flag in the payload)

Returns

the DMI hint value of the last transaction

4.4.2.15 get_last_transport_debug_ret()

```
unsigned int InitiatorTester::get_last_transport_debug_ret ( ) const  [inline]
```

Get the return value of the last transport_dbg call.

Returns

the return value of the last transport_dbg call

4.4.2.16 get_last_txn_delay()

```
const sc_core::sc_time& InitiatorTester::get_last_txn_delay ( ) const  [inline]
```

Get the delay value resulting of the last b_transport call.

Returns

the delay value resulting of the last b_transport call

4.4.2.17 set_next_txn_delay()

```
void InitiatorTester::set_next_txn_delay (
            const sc_core::sc_time & delay )  [inline]
```

Set the delay value to use for the next b_transport call.

Parameters

in	*delay*	The delay value to use for the next b_transport call

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

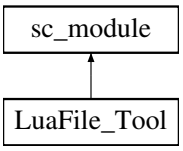
- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/tests/initiator-tester.h

4.5 LuaFile_Tool Class Reference

Tool which reads a Lua configuration file and sets parameters.

```
#include <luafile_tool.h>
```

Inheritance diagram for LuaFile_Tool:



Public Member Functions

- LuaFile_Tool (sc_core::sc_module_name name, std::string _orig_name="")
  Constructor.
- int config (const char *config_file)
  Makes the configuration.
- void parseCommandLine (const int argc, const char *const *argv)
  Parses the command line and extracts the luafile option.

Protected Member Functions

- void parseCommandLineWithGetOpt (const int argc, const char *const *argv)
  Parses the command line with getopt and extracts the luafile option.

4.5.1 Detailed Description

Tool which reads a Lua configuration file and sets parameters.

Lua Config File Tool which reads a configuration file and uses the Tool_GCnf_Api to set the parameters during intitialize-mode.

One instance can be used to read and configure several lua config files.

The usage of this Tool:

- instantiate one object
- call config(filename)

4.5.2 Member Function Documentation

4.5.2.1 config()

```
int LuaFile_Tool::config (
            const char * config_file )  [inline]
```

Makes the configuration.

Configure parameters from a lua file.

May be called several times with several configuration files

Example usage:

```
int sc_main(int argc, char *argv[]) {
  LuaFile_Tool luareader;
  luareader.config("file.lua");
  luareader.config("other_file.lua");
}
```

4.5.2.2 parseCommandLine()

```
void LuaFile_Tool::parseCommandLine (
            const int argc,
            const char *const * argv )  [inline]
```

Parses the command line and extracts the luafile option.

Throws a CommandLineException.

Parameters

argc	The argc of main(...).
argv	The argv of main(...).

4.5.2.3 parseCommandLineWithGetOpt()

```
void LuaFile_Tool::parseCommandLineWithGetOpt (
            const int argc,
            const char *const * argv )  [inline], [protected]
```

Parses the command line with getopt and extracts the luafile option.

Throws a CommandLineException.

Parameters

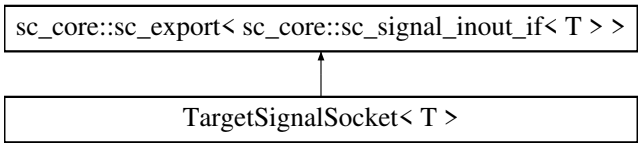
argc	The argc of main(...).
argv	The argv of main(...).

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

- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/luafile_tool.h

4.6 TargetSignalSocket< T > Class Template Reference

Inheritance diagram for TargetSignalSocket< T >:



Public Types

- using **Iface** = typename TargetSignalSocketProxy< T >::Iface
- using **Parent** = sc_core::sc_export< Iface >
- using **ValueChangedCallback** = typename TargetSignalSocketProxy< T >::ValueChangedCallback

Public Member Functions

- **TargetSignalSocket** (const char *name)
- void **register_value_changed_cb** (const ValueChangedCallback &cb)
- const T & **read** () const

Protected Attributes

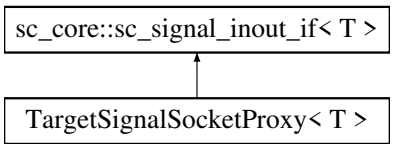
- TargetSignalSocketProxy< T > **m_proxy**

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

- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/ports/target-signal-socket.h

4.7 TargetSignalSocketProxy< T > Class Template Reference

Inheritance diagram for TargetSignalSocketProxy< T >:



Public Types

- using **Iface** = sc_core::sc_signal_inout_if< T >
- using **ValueChangedCallback** = std::function< void(const T &)>

Public Member Functions

- **TargetSignalSocketProxy** (TargetSignalSocket< T > &parent)
- void **register_value_changed_cb** (const ValueChangedCallback &cb)
- TargetSignalSocket< T > & **get_parent** ()
- void **notify** ()
- virtual const sc_core::sc_event & **default_event** () const
- virtual const sc_core::sc_event & **value_changed_event** () const
- virtual const T & **read** () const
- virtual const T & **get_data_ref** () const
- virtual bool **event** () const
- virtual void **write** (const T &val)

Protected Attributes

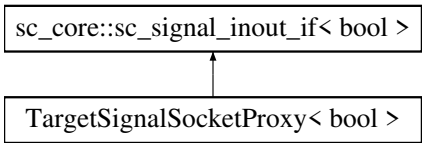
- TargetSignalSocket< T > & **m_parent**
- T **m_val**
- ValueChangedCallback **m_cb**
- sc_core::sc_event **m_ev**

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

- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/ports/target-signal-socket.h

4.8 TargetSignalSocketProxy< bool > Class Template Reference

Inheritance diagram for TargetSignalSocketProxy< bool >:



Public Types

- using **Iface** = sc_core::sc_signal_inout_if< bool >
- using **ValueChangedCallback** = std::function< void(const bool &)>

Public Member Functions

- **TargetSignalSocketProxy** (TargetSignalSocket< bool > &parent)
- void **register_value_changed_cb** (const ValueChangedCallback &cb)
- TargetSignalSocket< bool > & **get_parent** ()
- void **notify** ()
- virtual const sc_core::sc_event & **default_event** () const
- virtual const sc_core::sc_event & **value_changed_event** () const
- virtual const sc_core::sc_event & **posedge_event** () const
- virtual const sc_core::sc_event & **negedge_event** () const
- virtual const bool & **read** () const
- virtual const bool & **get_data_ref** () const
- virtual bool **event** () const
- virtual bool **posedge** () const
- virtual bool **negedge** () const
- virtual void **write** (const bool &val)

Protected Attributes

- TargetSignalSocket< bool > & **m_parent**
- bool **m_val**
- ValueChangedCallback **m_cb**
- sc_core::sc_event **m_ev**
- sc_core::sc_event **m_posedge_ev**
- sc_core::sc_event **m_negedge_ev**

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

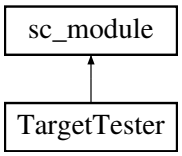
- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/ports/target-signal-socket.h

4.9 TargetTester Class Reference

A TLM target to do testing on an initiator.

`#include <target-tester.h>`

Inheritance diagram for TargetTester:



Public Types

- using **TlmGenericPayload** = tlm::tlm_generic_payload
- using **TlmResponseStatus** = tlm::tlm_response_status
- using **TlmDmi** = tlm::tlm_dmi
- using **AccessCallbackFn** = std::function< TlmResponseStatus(uint64_t addr, uint8_t *data, size_t len)>
- using **DebugAccessCallbackFn** = std::function< int(uint64_t addr, uint8_t *data, size_t len)>
- using **GetDirectMemPtrCallbackFn** = std::function< bool(uint64_t addr, TlmDmi &)>

Public Member Functions

- TargetTester (const sc_core::sc_module_name &n, size_t mmio_size)

  Construct a TargetTester object with a name and an MMIO size.
- void register_read_cb (AccessCallbackFn cb)

  Register callback called on b_tranport read transaction.
- void register_write_cb (AccessCallbackFn cb)

  Register callback called on b_tranport write transaction.
- void register_debug_read_cb (DebugAccessCallbackFn cb)

  Register callback called on transport_dbg read transaction.
- void register_debug_write_cb (DebugAccessCallbackFn cb)

  Register callback called on transport_dbg write transaction.

- void register_get_direct_mem_ptr_cb (GetDirectMemPtrCallbackFn cb)

  Register a callback called on a get_direct_mem_ptr call.
- bool last_txn_is_valid () const

  Return true if the copy of the last transaction is valid.
- const TlmGenericPayload & get_last_txn ()

  Return a copy of the last transaction payload.
- const sc_core::sc_time & get_last_txn_delay ()

  Return a copy of the last transaction delay value.
- TlmGenericPayload & get_cur_txn ()

  Get the current transaction payload.
- sc_core::sc_time & get_cur_txn_delay ()

  Get the current transaction delay.

Public Attributes

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

Protected Member Functions

- virtual void **b_transport** (TlmGenericPayload &txn, sc_core::sc_time &delay)
- virtual unsigned int **transport_dbg** (TlmGenericPayload &txn)
- virtual bool **get_direct_mem_ptr** (TlmGenericPayload &txn, TlmDmi &dmi_data)

4.9.1 Detailed Description

A TLM target to do testing on an initiator.

This class allows to test an initiator by providing helpers to standard TLM operations. The class user can register various callbacks for classical TLM forward path calls. The goal of those callback is to provide an easy mean of accessing most often used data in the callback parameters directly. The complete payload of the current transaction is still accessible using the `get_cur_txn(_delay)` method.

When not registering any callbacks, this class behaves as a dummy target, responding correctly to transactions with the following behaviour:

- Standard b_transport behaviour with out-of-bound check (based on mmio_size given at construct time). On a write commmand, the data buffer if filled with zeros
- Standard transport_dbg behaviour with out-of-bound check, returning 0 on error, and the transaction data size on success. The transaction data buffer is filled with zeros on a write command.
- DMI request default behaviour is to always return false;

Regular TLM forward calls can be overriden when inheriting this class if one need fine control over the transaction. However, Be aware that by doing so, you'll loose the helpers functionnality of this class.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 TargetTester()

```
TargetTester::TargetTester (
            const sc_core::sc_module_name & n,
            size_t mmio_size )  [inline]
```

Construct a TargetTester object with a name and an MMIO size.

Parameters

in	*n*	The name of the SystemC module
in	*mmio_size*	The size of the memory mapped I/O region of this component

4.9.3 Member Function Documentation

4.9.3.1 get_cur_txn()

```
TlmGenericPayload& TargetTester::get_cur_txn ( )  [inline]
```

Get the current transaction payload.

This method returns the payload of the transaction in progress. It must be could from within a transaction callback only. The transaction payload can be altered if needed.

Returns

the current transaction payload

4.9.3.2 get_cur_txn_delay()

```
sc_core::sc_time& TargetTester::get_cur_txn_delay ( )  [inline]
```

Get the current transaction delay.

This method returns the delay value of the transaction in progress. It must be could from within a transaction callback only. The transaction delay can be altered if needed.

Returns

the current transaction delay

4.9.3.3 get_last_txn()

```
const TlmGenericPayload& TargetTester::get_last_txn ( )  [inline]
```

Return a copy of the last transaction payload.

This method returns an internal copy of the last transaction payload. An internal flag checks whether the payload is valid or not. Calling this method actually reset the flag so calling it two time in a row will trigger a test failure. This ensures that you actually got the transaction you expected to get.

4.9.3.4 get_last_txn_delay()

```
const sc_core::sc_time& TargetTester::get_last_txn_delay ( )  [inline]
```

Return a copy of the last transaction delay value.

This method returns an internal copy of the last transaction delay value. An internal flag checks whether the delay value is valid or not. Calling this method actually reset the flag so calling it two time in a row will trigger a test failure. This ensures that you actually got the transaction you expected to get.

4.9.3.5 last_txn_is_valid()

```
bool TargetTester::last_txn_is_valid ( ) const  [inline]
```

Return true if the copy of the last transaction is valid.

This method can be used to check whether this target effectively received a transaction or not. It will return true if the internal copy of the last transaction is valid. Note that this flag is reset when calling the `get_last_txn` method.

Returns

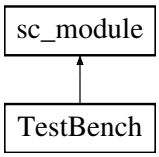
true if the copy of the last transaction is valid

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

- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/tests/target-tester.h

4.10 TestBench Class Reference

Inheritance diagram for TestBench:



Public Member Functions

- **SC_HAS_PROCESS** (TestBench)
- **TestBench** (const sc_core::sc_module_name &n)

Protected Member Functions

- virtual void **test_bench_body** ()=0

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

- /home/thomas/Documents/GreenSocs/build-lib/libgsutils/include/greensocs/gsutils/tests/test-bench.h

Index