gsutils

Generated by Doxygen 1.9.2

Chapter 1

gsutils

[//]: # DONT EDIT THIS FILE

1.0.1 LIBGSUTILS

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

1.1 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

```
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

```
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""</tt>
</blockquote> 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 <tt>path.to.module</tt> 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).  @subsection autotoc_md9 Using yaml for
configuration If you would prefer to use yaml as a configuration
language, <tt>lyaml</tt> provides a link.  This can be downloaded
from https://github.com/gvvaughan/lyaml The following lua code will
load "conf.yaml".  @code 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 @endcode
@subsection autotoc_md10 Using the ConfigurableBroker The broker
will self register in the SystemC CCI hierarchy.  All brokers have
a parameter <tt>lua_file</tt> which will be read and used to configure
parameters held within the broker.  This file is read at the <em>local</em>
level, and paths are <em>relative</em> to the location where the
ConfigurableBroker is instanced.  These brokers can be used as global
brokers.  The <tt>gs::ConfigurableBroker</tt> can be instanced
in 3 ways:  1.  <tt>ConfigurableBroker()</tt> This will instance
a 'Private broker' and will hide <strong>ALL</strong> parameters
held within this broker.  A local <tt>lua_file</tt> can be read
and will set parameters in the private broker.  This can be prevented
by passing 'false' as a construction parameter (<tt>Configurable↩
Broker(false)</tt>).  2.  <tt>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 <tt>ConfigurationBroker({})</tt>.
This is useful to provide a <em>local</em> configuration broker
than can, for instance, read a local configuration file.  A local
<tt>lua_file</tt> 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 (<tt>ConfigurableBroker(false)</tt>).
The <tt>lua_file</tt> will be read <strong>AFTER</strong> the
```

construction key-value list and hence can be used to over-right default values in the code. 3. <tt>ConfigurableBroker(argc, argv)</tt> 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: > <tt>-p, --param path.to.param=\<value\></tt> this option will allow individual parameters to be set. > <tt>-l, --gs_luafile \<FILE.lua\></tt> 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.

1.2.5 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.

1.2.6 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

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 cci::cci_value_converter< gs::ConfigurableBroker * > Struct Reference

Public Types

- typedef gs::ConfigurableBroker * **type**

Static Public Member Functions

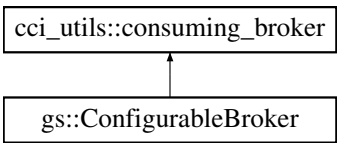
- static bool **pack** (cci_value::reference dst, type const &src)
- static bool **unpack** (type &dst, cci_value::const_reference src)

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

- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/cciutils.h

5.2 gs::ConfigurableBroker Class Reference

Inheritance diagram for gs::ConfigurableBroker:



Public Member Functions

- std::vector< cci_name_value_pair > **get_consumed_preset_values** () const
- void **print_help** (bool top=true)
- **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, std::initializer_list< std::pair< std::string, std::string > > alias_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
- void **ignore_unconsumed_preset_values** (const cci_preset_value_predicate &pred)
- 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**

Friends

- class **cci_value_converter**< **ConfigurableBroker** >

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

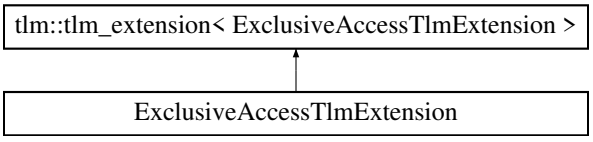
- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/cciutils.h

5.3 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** , **EXCLUSIVE_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

5.3.1 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:

- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/tlm-extensions/exclusive-access.h

5.4 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:

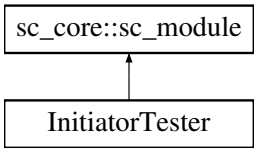
- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/tlm-extensions/exclusive-access.h

5.5 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)

5.5.1 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.

5.5.2 Member Function Documentation

5.5.2.1 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

5.5.2.2 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

5.5.2.3 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

5.5.2.4 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

5.5.2.5 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

5.5.2.6 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

5.5.2.7 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

5.5.2.8 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

5.5.2.9 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

5.5.2.10 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

5.5.2.11 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

5.5.2.12 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

5.5.2.13 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

5.5.2.14 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

5.5.2.15 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

5.5.2.16 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

5.5.2.17 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:

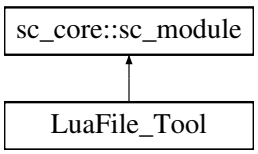
- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/tests/initiator-tester.h

5.6 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.

5.6.1 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)

5.6.2 Member Function Documentation

5.6.2.1 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");
}
```

5.6.2.2 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(...).

5.6.2.3 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

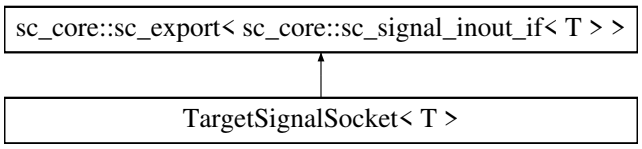
argc	The argc of main(...).
argv	The argv of main(...).

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

- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/luafile_tool.h

5.7 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

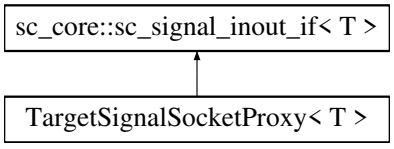
- TargetSignalSocketProxy< T > **m_proxy**

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

- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/ports/target-signal-socket.h

5.8 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

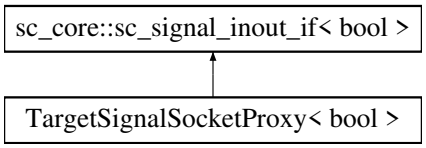
- TargetSignalSocket< T > & **m_parent**
- T **m_val**
- ValueChangedCallback **m_cb**
- sc_core::sc_event **m_ev**

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

- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/ports/target-signal-socket.h

5.9 TargetSignalSocketProxy< bool > Class 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:

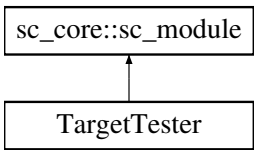
- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/ports/target-signal-socket.h

5.10 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)

5.10.1 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.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 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

5.10.3 Member Function Documentation

5.10.3.1 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

5.10.3.2 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

5.10.3.3 get_last_txn()

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

Return a copy of the last transaction payload.

@detail 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.

5.10.3.4 get_last_txn_delay()

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

Return a copy of the last transaction delay value.

@detail 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.

5.10.3.5 last_txn_is_valid()

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

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

@detail 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

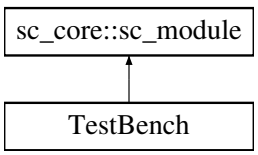
true if the copy of the last transaction is valid

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

- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/tests/target-tester.h

5.11 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:

- /Users/mark/Desktop/desktop/work/workspace-mac/libgsutils/include/greensocs/gsutils/tests/test-bench.h

Chapter 6

File Documentation

6.1 cciutils.h

```
1 /*
2  *  Copyright (C) 2020  GreenSocs
3  *
4  */
5
6 #ifndef CCIUTILS_H
7 #define CCIUTILS_H
8
9 #include <iostream>
10 #include <list>
11
12 #include "luafile_tool.h"
13 #include <cci_configuration>
14 #define SC_INCLUDE_DYNAMIC_PROCESSES
15 #include <systemc>
16 #include <tlm>
17
18 /*********************
19  * CCI Convenience
20  *********************/
21
22 namespace gs {
23 using namespace cci;
24
25 /* return the leaf name of the given module name */
26 static std::string sc_cci_leaf_name(std::string name)
27 {
28     return name.substr(name.find_last_of(".") + 1);
29 }
30
31 /* return a list of children from the given module name, can be used inside
32  * or outside the heirarchy */
33 static std::list<std::string> sc_cci_children(sc_core::sc_module_name name)
34 {
35     cci_broker_handle m_broker = (sc_core::sc_get_current_object())
36         ? cci_get_broker()
37         : cci_get_global_broker(cci_originator("gs__sc_cci_children"));
38     std::list<std::string> children;
39     int l = strlen(name) + 1;
40     auto uncon = m_broker.get_unconsumed_preset_values([&name](const std::pair<std::string, cci_value>&
      iv) { return iv.first.find(std::string(name) + ".") == 0; });
41     for (auto p : uncon) {
42         children.push_back(p.first.substr(l, p.first.find(".", l) - l));
43     }
44     children.sort();
45     children.unique();
46     return children;
47 }
48
49 template <typename T>
50 T cci_get(std::string name)
51 {
52     auto m_broker = cci::cci_get_broker();
53     m_broker.ignore_unconsumed_preset_values(
54         [name](const std::pair<std::string, cci::cci_value>& iv) -> bool { return iv.first == name; });
55     m_broker.lock_preset_value(name);
56     T ret;
57     if (!m_broker.get_preset_cci_value(name).template try_get<T>(ret)) {
```

```
58          SC_REPORT_ERROR("Loader", ("Unable to get parameter " + name).c_str());
59      };
60      return ret;
61  }
62
63  class ConfigurableBroker : public cci_utils::consuming_broker {
64
65  public:
66      // a set of perameters that should be exposed up the broker stack
67      std::set<std::string> expose;
68      cci_param<std::string> conf_file;
69
70  private:
71      bool has_parent;
72      cci_broker_if& m_parent;
73      std::string m_orig_name;
74      cci_originator m_originator;
75      cci_param<ConfigurableBroker*>* m_child_ref;
76
77      friend class cci_value_converter<ConfigurableBroker>;
78
79      // convenience function for constructor
80      cci_broker_if& get_parent_broker()
81      {
82          if (sc_core::sc_get_current_object()) {
83              has_parent = true;
84              return unwrap_broker(cci_get_broker());
85          } else {
86              // We ARE the global broker
87              has_parent = false;
88              return *this;
89          }
90      }
91      std::string hierarchical_name()
92      {
93          if (!sc_core::sc_get_current_object()) {
94              return std::string("");
95          } else {
96              return cci_originator().name();
97          }
98      }
99      cci_originator get_cci_originator(const char* n)
100      {
101          if (!sc_core::sc_get_current_object()) {
102              return cci_originator(n);
103          } else {
104              return cci_originator();
105          }
106      }
107      /*
108     * private function to determine if we send to the parent broker or not
109     */
110      bool sendToParent(const std::string& parname) const
111      {
112          return ((expose.find(parname) != expose.end()) && (!is_global_broker()));
113      }
114
115      /*
116     * Expose all params that have not been configured
117     */
118
119      void
120      initialize_params(const std::initializer_list<cci_name_value_pair>& list)
121      {
122          using namespace cci;
123
124          // Expose everything
125          for (auto p : m_parent.get_unconsumed_preset_values()) {
126              expose.insert(p.first);
127          }
128          // Hide configured things
129          for (auto& p : list) {
130              expose.erase(relname(p.first));
131              set_preset_cci_value(relname(p.first), p.second, m_originator);
132          }
133      }
134
135    void alias_params(const std::initializer_list<std::pair<std::string, std::string>> &list) {
136          // handle aliases
137      for (auto &p : list) {
138          std::string aliasname = relname(p.second);
139          set_preset_cci_value(relname(p.first), get_preset_cci_value(aliasname), m_originator);
140          alias_param(relname(p.first), aliasname);
141      }
142    }
148    void untyped_post_write_callback(const cci::cci_param_write_event<> & ev ,
149                                     cci::cci_param_handle synced_handle) {
```

```
150       synced_handle.set_cci_value(ev.new_value);
151     }
160     void sync_values(cci::cci_param_handle _param_handle_1,
161                      cci::cci_param_handle _param_handle_2) {
162       // In order to synchronize even the default values of the owner modules,
163       // use cci_base_param of one parameter as reference, write the same value
164       // to the other pararmeter's cci_base_param using JSON
165       _param_handle_1.set_cci_value(_param_handle_2.get_cci_value());
166
167       post_write_cb_vec.push_back(_param_handle_1.register_post_write_callback(
168               sc_bind(&ConfigurableBroker::untyped_post_write_callback,
169                       this, sc_unnamed::_1, _param_handle_2)));
170
171       post_write_cb_vec.push_back(_param_handle_2.register_post_write_callback(
172               sc_bind(&ConfigurableBroker::untyped_post_write_callback,
173                       this, sc_unnamed::_1, _param_handle_1)));
174     }
175
176     std::vector<cci::cci_callback_untyped_handle> post_write_cb_vec;
177
178     cci_param_create_callback_handle register_cb;
179     std::vector<std::pair<std::string, std::string» alias_list;
180
181     void alias_param(std::string a, std::string b) {
182         alias_list.push_back(std::make_pair(a, b));
183         if (register_cb==cci_param_create_callback_handle()) {
184            register_cb = register_create_callback(sc_bind(&ConfigurableBroker::alias_param_callback, this
      , sc_unnamed::_1), m_originator);
185         }
186         alias_param_callback(cci::cci_param_handle());
187     }
188
189     void alias_param_callback(const cci_param_untyped_handle &ph)
190     {
191       alias_list.erase(
192       std::remove_if(alias_list.begin(), alias_list.end(),
193         [&](auto a) {
194
195               cci_param_untyped_handle h_a =
196                   get_param_handle(a.first, m_originator);
197               if (h_a.is_valid() && h_a.get_mutable_type()==cci::CCI_IMMUTABLE_PARAM) {
198                 return true;
199               }
200               cci_param_untyped_handle h_b =
201                       get_param_handle(a.second, m_originator);
202               if (h_b.is_valid() && !h_a.is_valid()) {
203                   set_preset_cci_value(a.first, h_b.get_cci_value(), m_originator);
204                   return false; // maybe it'll be mutable when it arrives?
205               }
206               if (h_a.is_valid() && h_b.is_valid()) {
207                   sync_values(h_a, h_b);
208                   return true;
209               }
210               return false;
211         } ),
212       alias_list.end());
213
214     if (alias_list.size()==0) {
215         unregister_create_callback(register_cb, m_originator);
216     }
217 }
218
219 class help_helper : public sc_core::sc_module {
220 public:
221     help_helper(sc_core::sc_module_name name) { }
222     std::function<void()> help_cb;
223     void register_cb (std::function<void()> cb) {help_cb=cb;}
224     void end_of_elaboration()
225     {
226         if (help_cb) help_cb();
227     }
228
229     void start_of_simulation()
230     {
231         auto m_broker = cci::cci_get_broker();
232         // remove lua builtins
233         m_broker.ignore_unconsumed_preset_values(
234             [](const std::pair<std::string, cci::cci_value>& iv) -> bool { return ((iv.first)[0]=='_' ||
      iv.first == "math.maxinteger") || (iv.first == "math.mininteger") || (iv.first ==
      "utf8.charpattern"); });
235
236         auto uncon = m_broker.get_unconsumed_preset_values();
237         for (auto p : uncon) {
238             SC_REPORT_INFO("Params", ("WARNING: Unconsumed parameter : " + p.first + " = " +
      p.second.to_json()).c_str());
239         }
240
```

```
241         if (help_cb) exit(0);
242     }
243 };
244 help_helper m_help_helper;
245 
246 public:
247     /*
248  * public interface functions
249  */
250 
251      std::vector<cci_name_value_pair> get_consumed_preset_values() const
252     {
253         std::vector<cci_name_value_pair> consumed_preset_cci_values;
254         std::map<std::string, cci_value>::const_iterator iter;
255         std::vector<cci_preset_value_predicate>::const_iterator pred;
256 
257         for (iter = m_unused_value_registry.begin(); iter != m_unused_value_registry.end(); ++iter) {
258             for (pred = m_ignored_unconsumed_predicates.begin(); pred !=
    m_ignored_unconsumed_predicates.end(); ++pred) {
259                 const cci_preset_value_predicate& p = *pred; // get the actual predicate
260                 if (p(std::make_pair(iter->first, iter->second))) {
261                     break;
262                 }
263             }
264             if (pred != m_ignored_unconsumed_predicates.end()) {
265                 consumed_preset_cci_values.push_back(std::make_pair(iter->first, iter->second));
266             }
267         }
268         return consumed_preset_cci_values;
269     }
270 
271     void print_help(bool top = true)
272     {
273         /* NB there is a race condition between this and the QEMU uart which
274          * has a tendency to wipe the buffer. We will use cerr here */
275 
276         if (top) {
277             std::cerr « std::endl
278                      « "Available Configuration Parameters:" « std::endl
279                      « "====================================" « std::endl;
280         }
281 
282         for (auto p : get_consumed_preset_values())
283         {
284             std::cerr « "Consumed parameter : "« p.first « " (with value 0x"«std::hex «
    p.second«")"«std::endl;
285         }
286 
287         std::string ending = "childbroker";
288         for (auto p : get_param_handles(get_cci_originator("Command line help"))) {
289             if (!std::equal(ending.rbegin(), ending.rend(), std::string(p.name()).rbegin())) {
290                 std::cerr « p.name() « " : " « p.get_description() « " (configured value " «
    p.get_cci_value() « ") " « std::endl;
291             } else {
292                 cci_param_typed_handle<ConfigurableBroker*> c(p);
293                 ConfigurableBroker* cc = c.get_value();
294                 if (cc != this) {
295                     cc->print_help(false);
296                 }
297             }
298         }
299         if (top) {
300             std::cerr « "---" « std::endl;
301 //            exit(0);
302         }
303     }
304 /*
305  * default constructor:
306  * When constructed with no initialised parameters, it is assumed that ALL
307  * parameters are to be treated as private
308  */
309 #define BROKERNAME "gs::ConfigurableBroker"
310     ConfigurableBroker(const std::string& name = BROKERNAME,
311         bool load_conf_file = true)
312         : m_orig_name(hierarchical_name())
313         , m_originator(get_cci_originator(name.c_str()))
314         , consuming_broker(hierarchical_name() + "." + name)
315         , conf_file("lua_conf", "",
316               cci_broker_handle(get_parent_broker(),
317                   get_cci_originator(name.c_str())),
318               "Local lua configuration file", CCI_RELATIVE_NAME,
319               get_cci_originator(name.c_str()))
320         , m_parent(get_parent_broker()) // local convenience function
321         , m_child_ref(NULL)
322         , m_help_helper("help_helper")
323     {
324         if (has_parent) {
```

```
325             m_child_ref = new cci_param<ConfigurableBroker*>(
326                 (hierarchical_name() + "." + name + ".childbroker").c_str(),
327                 (this), "");
328         }
329 
330         cci_register_broker(this);
331 
332         if (load_conf_file && !(std::string(conf_file).empty())) {
333             LuaFile_Tool lua("lua", m_orig_name);
334             lua.config(std::string(conf_file).c_str());
335         }
336     }
337 
338     /*
339     * Constructor with just boolean, for convenience
340     */
341     ConfigurableBroker(bool load_conf_file)
342         : ConfigurableBroker(BROKERNAME, load_conf_file)
343     {
344     }
345 
346     /*
347     * initialised list constructor:
348     * When constructed with a list of initialised parameters, all other params
349     * will be exported to the parent broker
350     */
351     ConfigurableBroker(std::initializer_list<cci_name_value_pair> list,
352         std::initializer_list<std::pair<std::string, std::string» alias_list = {},
353         bool load_conf_file = true)
354         : ConfigurableBroker(false)
355     {
356         initialize_params(list);
357         alias_params(alias_list);
358         if (load_conf_file && !(std::string(conf_file).empty())) {
359             LuaFile_Tool lua("lua", m_orig_name);
360             lua.config(std::string(conf_file).c_str());
361         }
362     }
363     /*
364     * in this case, the expectation is that this is being used at (or near) the
365     * top level of the design, and this broker will act as a global broker. A
366     * list of values will be used to set default values, but will be overwritten
367     * by configuration passed on the command line
368     */
369     ConfigurableBroker(const int argc, char* const argv[],
370         std::initializer_list<cci_name_value_pair> list = {},
371         bool load_conf_file = true)
372         : ConfigurableBroker(false)
373     {
374 
375         for (auto& p : list) {
376             set_preset_cci_value(relname(p.first), p.second, m_originator);
377         }
378 
379         LuaFile_Tool lua("lua", m_orig_name);
380         lua.parseCommandLine(argc, argv);
381 
382         static const char* optstring = "h";
383         static struct option long_options[] = {
384             { "help", 0, 0, 'h' }, // '--help' = '-h'
385             { 0, 0, 0, 0 }
386         };
387         optind = 1;
388         while (1) {
389             int c = getopt_long(argc, argv, optstring, long_options, 0);
390             if (c == EOF)
391                 break;
392             switch (c) {
393             case 'h': // -h and --help
394                 sc_core::sc_spawn_options opts;
395                 m_help_helper.register_cb([&]() -> void { print_help();});
396                 break;
397             }
398         }
399         optind = 1;
400 
401         /* check to see if the conf_file was set ! */
402         if (load_conf_file && !(std::string(conf_file).empty())) {
403             LuaFile_Tool lua("lua", m_orig_name);
404             lua.config(std::string(conf_file).c_str());
405         }
406     }
407 
408     std::string relname(const std::string& n) const
409     {
410         return m_orig_name + std::string(".") + n;
411     }
```

```
412
413     ~ConfigurableBroker()
414     {
415         if (m_child_ref) {
416             delete m_child_ref;
417         }
418     }
419
420     cci_originator get_value_origin(const std::string& parname) const
421     {
422         if (sendToParent(parname)) {
423             return m_parent.get_value_origin(parname);
424         } else {
425             return consuming_broker::get_value_origin(parname);
426         }
427     }
428
429     /* NB  missing from upstream CCI 'broker' see
430    * https://github.com/OSCI-WG/cci/issues/258 */
431     bool has_preset_value(const std::string& parname) const
432     {
433         if (sendToParent(parname)) {
434             return m_parent.has_preset_value(parname);
435         } else {
436             return consuming_broker::has_preset_value(parname);
437         }
438     }
439
440     cci_value get_preset_cci_value(const std::string& parname) const
441     {
442         if (sendToParent(parname)) {
443             return m_parent.get_preset_cci_value(parname);
444         } else {
445             return consuming_broker::get_preset_cci_value(parname);
446         }
447     }
448
449     void lock_preset_value(const std::string& parname)
450     {
451         if (sendToParent(parname)) {
452             return m_parent.lock_preset_value(parname);
453         } else {
454             return consuming_broker::lock_preset_value(parname);
455         }
456     }
457
458     cci_value get_cci_value(const std::string& parname) const
459     {
460         if (sendToParent(parname)) {
461             return m_parent.get_cci_value(parname);
462         } else {
463             return consuming_broker::get_cci_value(parname);
464         }
465     }
466
467     void add_param(cci_param_if* par)
468     {
469         if (sendToParent(par->name())) {
470             return m_parent.add_param(par);
471         } else {
472             return consuming_broker::add_param(par);
473         }
474     }
475
476     void remove_param(cci_param_if* par)
477     {
478         if (sendToParent(par->name())) {
479             return m_parent.remove_param(par);
480         } else {
481             return consuming_broker::remove_param(par);
482         }
483     }
484
485     // Upstream consuming broker fails to pass get_unconsumed_preset_value to parent
486     std::vector<cci_name_value_pair> get_unconsumed_preset_values() const
487     {
488         std::vector<cci_name_value_pair> r = consuming_broker::get_unconsumed_preset_values();
489         if (has_parent) {
490             std::vector<cci_name_value_pair> p = m_parent.get_unconsumed_preset_values();
491             r.insert(r.end(), p.begin(), p.end());
492         }
493         return r;
494     }
495
496     // Upstream consuming broker fails to pass ignore_unconsumed_preset_values
497     void ignore_unconsumed_preset_values(const cci_preset_value_predicate& pred)
498     {
```

```
499         if (has_parent) {
500             m_parent.ignore_unconsumed_preset_values(pred);
501         }
502         consuming_broker::ignore_unconsumed_preset_values(pred);
503     }
504
505     cci_preset_value_range get_unconsumed_preset_values(const cci_preset_value_predicate& pred) const
506     {
507         return cci_preset_value_range(pred, ConfigurableBroker::get_unconsumed_preset_values());
508     }
509
510     // Functions below here require an orriginator to be passed to the local
511     // method variant.
512
513     void set_preset_cci_value(const std::string& parname,
514         const cci_value& cci_value,
515         const cci_originator& originator)
516     {
517         if (sendToParent(parname)) {
518             return m_parent.set_preset_cci_value(parname, cci_value, originator);
519         } else {
520             return consuming_broker::set_preset_cci_value(parname, cci_value,
521                 originator);
522         }
523     }
524     cci_param_untyped_handle
525     get_param_handle(const std::string& parname,
526         const cci_originator& originator) const
527     {
528         if (sendToParent(parname)) {
529             return m_parent.get_param_handle(parname, originator);
530         }
531         cci_param_if* orig_param = get_orig_param(parname);
532         if (orig_param) {
533             return cci_param_untyped_handle(*orig_param, originator);
534         }
535         if (has_parent) {
536             return m_parent.get_param_handle(parname, originator);
537         }
538         return cci_param_untyped_handle(originator);
539     }
540
541     std::vector<cci_param_untyped_handle>
542     get_param_handles(const cci_originator& originator) const
543     {
544         if (has_parent) {
545             std::vector<cci_param_untyped_handle> p_param_handles = m_parent.get_param_handles();
546             std::vector<cci_param_untyped_handle> param_handles =
    consuming_broker::get_param_handles(originator);
547             // this is likely to be more efficient the other way round, but it keeps
548             // things consistent and means the local (more useful) params will be at
549             // the head of the list.
550             param_handles.insert(param_handles.end(), p_param_handles.begin(),
551                 p_param_handles.end());
552             return param_handles;
553         } else {
554             return consuming_broker::get_param_handles(originator);
555         }
556     }
557
558     bool is_global_broker() const { return (!has_parent); }
559 };
560 } // namespace gs
561
562 template <>
563 struct cci::cci_value_converter<gs::ConfigurableBroker*> {
564     typedef gs::ConfigurableBroker* type;
565     static bool pack(cci_value::reference dst, type const& src)
566     {
567         dst.set_string(src->name());
568         return true;
569     }
570     static bool unpack(type& dst, cci_value::const_reference src)
571     {
572         return false;
573     }
574 };
575
576 #endif // CCIUTILS_H
```

6.2 luafile_tool.h

```
1 /*
```

```
2  *  Copyright (C) 2020  GreenSocs
3  *
4  */
5
6 #ifndef __LUAFILE_TOOL_H__
7 #define __LUAFILE_TOOL_H__
8
9 // Set to true (or use -DGC_LUA_VERBOSE=true argument) to show the parameters
10 // set
11 #ifndef GC_LUA_VERBOSE
12 #define GC_LUA_VERBOSE false
13 #endif
14
15 // Set to true (or use -DGC_LUA_DEBUG=true argument) to show what was not set as
16 // a parameter
17 #ifndef GC_LUA_DEBUG
18 #define GC_LUA_DEBUG false
19 #endif
20
21 #define DEBUG(name, msg)                                                    \
22   std::cout « "@" « sc_core::sc_time_stamp() « " /"                        \
23             « (unsigned)sc_core::sc_delta_count() « " (" « name            \
24             « "): " « msg « std::endl
25
26 #define ENABLE_SHORT_COMMAND_LINE_OPTIONS // enables the short synonyms for the
27                                           // gs_ options
28
29 // if this should use the unix getopt function for the parsing of the command
30 // line options or (if not) the boost program_options should be used (don't
31 // forget to link the lib boost_program_options!) default: do NOT define this!
32 // #define USE_GETOPT
33
34 #include <cci_configuration>
35 #include <iostream>
36 #include <string>
37
38 #define USE_GETOPT
39
40 #ifdef USE_GETOPT
41 #include <getopt.h>
42 #else
43 #include <boost/program_options.hpp>
44 #endif
45
46 #ifdef HAS_LUA
47 extern "C" {
48 #include <lauxlib.h>
49 #include <lua.h>
50 #include <lualib.h>
51 }
52 #endif
53
54 #ifndef USE_GETOPT
55 namespace po = boost::program_options;
56 #endif
57
59
70 class LuaFile_Tool : public sc_core::sc_module {
71
72   cci::cci_broker_handle m_broker;
73   std::string m_orig_name;
74
75   std::string rel(std::string &n) const {
76     if (m_orig_name.empty()) return n;
77     else return (m_orig_name + n);
78   }
79
80 public:
82   LuaFile_Tool(sc_core::sc_module_name name, std::string _orig_name = "")
83       : sc_core::sc_module(name), m_broker(cci::cci_get_broker()) {
84     if (_orig_name.empty()) m_orig_name = "";
85     else m_orig_name = _orig_name + ".";
86   }
87
89
103   int config(const char *config_file) {
104 #ifndef HAS_LUA
105     std::cerr « "Lua file specified, but no LUA support compiled in\n";
106     exit(-1);
107 #else
108     DEBUG(name(), "Read lua file '" « config_file « "'");
109
110     // start Lua
111     lua_State *L = luaL_newstate();
112     luaL_openlibs(L);
113
114     // load a script as the function "config_chunk"
```

```
115     int error = luaL_loadfile(L, config_file);
116     switch (error) {
117     case 0:
118       break;
119     case LUA_ERRSYNTAX:
120       fprintf(stderr, "Syntax error reading config file: %s\n", config_file);
121       return 1;
122     case LUA_ERRMEM:
123       fprintf(stderr, "Error allocating memory to read config file: %s\n",
124               config_file);
125       return 1;
126     case LUA_ERRFILE:
127       fprintf(stderr, "Error opening/reading the config file: %s\n",
128               config_file);
129       return 1;
130     default:
131       fprintf(stderr, "Unknown error loading config file: %s\n", config_file);
132       return 1;
133     }
134     lua_setglobal(L, "config_chunk");
135 
136     // little script to run the file
137     const char *config_loader =
138         "-- put some commands here to run before the user script\n"
139         "config_chunk()";
140 
141     // run
142     if (luaL_dostring(L, config_loader)) {
143       fprintf(stderr, "%s\n", lua_tostring(L, -1));
144       lua_pop(L, 1); /* pop error message from the stack */
145     }
146 
147     // traverse the environment table setting global variables as parameters
148     //      lua_getfield(L, LUA_GLOBALSINDEX, "_G");
149     // getglobal should work for both lua 5.1 and 5.2
150     lua_getglobal(L, "_G");
151     error = setParamsFromLuaTable(L, lua_gettop(L));
152     if (error < 0) {
153       fprintf(stderr, "Error loading lua config file: %s\n", config_file);
154       return error;
155     }
156     return 0;
157 #endif
158   }
159 
160 
167   void
168   parseCommandLine(const int argc,
169                    const char *const *argv) /* throw(CommandLineException) */ {
170 #ifdef USE_GETOPT
171     parseCommandLineWithGetOpt(argc, argv);
172 #else
173     parseCommandLineWithBoost(argc, argv);
174 #endif
175   }
176 
177 protected:
178 #ifndef USE_GETOPT
179 
187   void parseCommandLineWithBoost(
188       const int argc, const char *const *argv) /*throw(CommandLineException)*/ {
189     DEBUG(name(), "Parse command line for --gs_luafile option ("
190                       « argc « " arguments)");
191 
192     po::options_description desc("Allowed options");
193 #ifdef ENABLE_SHORT_COMMAND_LINE_OPTIONS
194     desc.add_options()("help,h",
195                        "  Command line usage for command line Config parser")(
196         "gs_luafile,l", po::value<std::vector<std::string»(),
197         "<filename>   execute a Lua script and loads all the globals as "
198         "parameters");
199 #else
200     desc.add_options()("help",
201                        "  Command line usage for command line Config parser")(
202         "gs_luafile", po::value<std::vector<std::string»(),
203         "<filename>   execute a Lua script and loads all the globals as "
204         "parameters");
205 #endif
206 
207     po::variables_map vm;
208     // po::store(po::parse_command_line(argc, argv, desc), vm); // without
209     // allowing unknown options
210     po::store(po::command_line_parser(argc, const_cast<char **>(argv))
211                   .options(desc)
212                   .allow_unregistered()
213                   .run(),
214               vm); // allows unknown options
```

```
215
216     if (vm.count("help")) {
217       std::cout « "Command line usage for lua file command line parser:"
218                 « std::endl;
219       std::cout « "  Usage: options_description [options]" « std::endl;
220       std::cout « desc;
221       return;
222     }
223
224     if (vm.count("gs_luafile")) {
225       const std::vector<std::string> *vec =
226           &vm["gs_luafile"].as<std::vector<std::string»();
227       for (unsigned int i = 0; i < vec->size(); i++) {
228         DEBUG(name(), "Option gs_luafile with value " « vec->at(i).c_str());
229         std::cout « "Lua file command line parser: parse option --gs_luafile "
230                   « vec->at(i).c_str() « std::endl;
231         config(vec->at(i).c_str());
232       }
233     }
234   }
235
236 #else
237
239
245   void parseCommandLineWithGetOpt(
246       const int argc,
247       const char *const *argv) /* throw(CommandLineException) */ {
248     DEBUG(name(), "Parse command line for --gs_luafile option ("
249                       « argc « " arguments)");
250
251     // getopt permutes argv array, so copy it to argv_cp
252     const int BUFFER_SIZE = 8192;
253     char argv_buffer[BUFFER_SIZE];
254     char *buffer_p = argv_buffer;
255     char **argv_cp = new char *[argc];
256     for (int i = 0; i < argc; i++) {
257       size_t len = strlen(argv[i]) + 1; // count \0
258       strcpy(buffer_p, argv[i]);
259       argv_cp[i] = buffer_p;
260       buffer_p += len;
261     }
262
263     // Check the rare case that BUFFER_SIZE is not enough
264     if (buffer_p - argv_buffer > BUFFER_SIZE) {
265       throw std::overflow_error("buffer overflow");
266     }
267
268     // configure getopt
269     optind = 0; // reset of getopt
270     opterr = 0; // avoid error message for not recognized option
271 #ifdef ENABLE_SHORT_COMMAND_LINE_OPTIONS
272     static const char *optstring = "l:p:h";
273 #else
274     static const char *optstring = "";
275 #endif
276     static struct option long_options[] = {
277         {"gs_luafile", 1, 0, 'l'}, // '--luafile filename'
278         {"param", 1, 0, 'p'},      // --param foo.baa=10
279         {"help", 0, 0, 'h'},       // '--help' = '-h'
280         {0, 0, 0, 0}};
281
282     while (1) {
283
284       int c = getopt_long(argc, argv_cp, optstring, long_options, 0);
285       if (c == EOF)
286         break;
287
288       switch (c) {
289
290       case 'l': // -l and --gs_luafile
291         DEBUG(name(), "Option --gs_luafile with value " « optarg);
292         std::cout « "Lua file command line parser: parse option --gs_luafile "
293                   « optarg « std::endl;
294         config(optarg);
295         break;
296
297       case 'p': // -p and --param
298       {
299         std::stringstream ss(optarg);
300         std::string key, value;
301         if (!std::getline(ss, key, '=')) {
302           std::cout « "parameter name not found!" « std::endl;
303           exit(-1);
304         }
305         if (!std::getline(ss, value)) {
306           std::cout « "parameter value not found!" « std::endl;
307           exit(-1);
```

```
308         }
309         DEBUG(name(), "Setting param " « rel(key) « " to value " « value);
310         m_broker.set_preset_cci_value(
311             rel(key), cci::cci_value(cci::cci_value::from_json(value)));
312         break;
313       }
314       case 'h': // -h and --help
315         std::cout « "Lua file command line parser: parse option --help "
316                   « std::endl;
317         std::cout « "  Command line usage for lua file command line parser:"
318                   « std::endl;
319         std::cout « std::endl;
320         std::cout « "     Possible Options/Arguments:" « std::endl;
321         std::cout « std::endl;
322         std::cout « "      --gs_luafile <filename>" « std::endl;
323         std::cout « "        execute a Lua script and loads all the globals "
324                      "as parameters"
325                   « std::endl;
326         std::cout « std::endl;
327         std::cout « "      --param <param_name=value>" « std::endl;
328         std::cout « "        set param name (foo.baa) to value" « std::endl;
329         std::cout « std::endl;
330         std::cout « "      --help" « std::endl;
331         std::cout « "        this help" « std::endl;
332         std::cout « std::endl;
333         std::cout « std::endl;
334         break;
335
336       case '?':
337       case ':':
338         DEBUG(name(), "Option "
339                       « c
340                       « " not processed in lua file command line parser: "
341                       « optopt);
342         break;
343       }
344     }
345   }
346 #endif
347
348 #ifdef HAS_LUA
350
356   int setParamsFromLuaTable(lua_State *L, int t, char *level = NULL) {
357     /* start up */
358     const int MAX_NAME_SIZE = 1000;
359     static char static_key[MAX_NAME_SIZE];
360     static char *key;
361     static char value[100]; // used only to convert LUA_TNUMBER
362     int should_inc_integer_index_count;
363     int integer_index_count = 0;
364     char *next_level;
365     if (level == NULL) {
366       key = static_key;
367       level = key;
368     }
369
370     /* test for overflow (hopefully unlikelly, so test only after it happens,
371      * for sanity) */
372     if (level - static_key > MAX_NAME_SIZE) {
373       static_key[MAX_NAME_SIZE - 1] = 0;
374       fprintf(stderr,
375               "FATAL Error: parameter name too big (bigger then %d): %s",
376               MAX_NAME_SIZE, static_key);
377     }
378
379     /* is it really a table? */
380     if (lua_type(L, t) != LUA_TTABLE) {
381       fprintf(stderr, "Error: argument is not a table");
382       return -1;
383     }
384
385     /* traverse table */
386     lua_pushnil(L); /* first key */
387
388     /* adjust t if relative index */
389     if (t < 0)
390       --t;
391
392     while (lua_next(L, t) != 0) {
393
394       /* reset flag */
395       should_inc_integer_index_count = false;
396
397       /* set the key */
398       switch (lua_type(L, -2)) {
399
400       case LUA_TNUMBER:
```

```
401         // key must be integer values (ignore floating part)
402         // also convert from 1-based to 0-based indexes (decrement 1)
403         should_inc_integer_index_count = true;
404 #ifdef __MINGW32__
405         next_level =
406             level +
407             __mingw_sprintf(level, "%lld", (long long)lua_tonumber(L, -2) - 1);
408 #else
409         next_level =
410             level + sprintf(level, "%lld", (long long)lua_tonumber(L, -2) - 1);
411 #endif
412         break;
413
414       case LUA_TSTRING:
415         // avoid using stpcpy as it is not defined in MSVS, use strcpy+length
416         // instead
417         next_level = strcpy(level, lua_tostring(L, -2));
418         next_level += strlen(level);
419         break;
420
421       default:
422         fprintf(stderr, "Error loading lua file: invalid key");
423         return -1;
424       }
425
426       /* set key value in the database */
427       switch (lua_type(L, -1)) {
428
429       case LUA_TNUMBER:
430         // Avoid setting some Lua specific values as parameters
431         if (strcmp(key, "math.huge") == 0 || strcmp(key, "math.pi") == 0 || 0) {
432           if (GC_LUA_DEBUG)
433             fprintf(stderr, "(%s) %s   (ignored because it's Lua specific)\n",
434                     lua_typename(L, lua_type(L, -1)), key);
435         } else {
436           double num = lua_tonumber(L, -1);
437           if ((uint64_t)num == num) {
438             std::string keys = key;
439             m_broker.set_preset_cci_value(rel(keys),
440                                           cci::cci_value((uint64_t)num));
441           } else if ((int64_t)num == num) {
442             std::string keys = key;
443             m_broker.set_preset_cci_value(rel(keys),
444                                           cci::cci_value((int64_t)num));
445           } else {
446             std::string keys = key;
447             m_broker.set_preset_cci_value(rel(keys), cci::cci_value(num));
448           }
449
450           if (GC_LUA_VERBOSE) {
451             std::string keys = key;
452             fprintf(stderr, "(SET %s) %s = %s\n",
453                     lua_typename(L, lua_type(L, -1)), rel(keys).c_str(),
454                     cci::cci_value(num).to_json().c_str());
455           }
456           if (should_inc_integer_index_count)
457             ++integer_index_count;
458         }
459         break;
460
461       case LUA_TBOOLEAN: {
462         std::string keys = key;
463         m_broker.set_preset_cci_value(
464             rel(keys), cci::cci_value((bool)lua_toboolean(L, -1)));
465         if (GC_LUA_VERBOSE)
466           fprintf(stderr, "(SET %s) %s = %s\n",
467                   lua_typename(L, lua_type(L, -1)), rel(keys).c_str(),
468                   lua_toboolean(L, -1) ? "true" : "false");
469         if (should_inc_integer_index_count)
470           ++integer_index_count;
471         break;
472       }
473       case LUA_TSTRING:
474         // Avoid setting some Lua specific values as parameters
475         if (strcmp(key, "_VERSION") == 0 || strcmp(key, "package.cpath") == 0 ||
476             strcmp(key, "package.config") == 0 ||
477             strcmp(key, "package.path") == 0 || 0) {
478           if (GC_LUA_DEBUG)
479             fprintf(stderr, "(%s) %s   (ignored because it's Lua specific)\n",
480                     lua_typename(L, lua_type(L, -1)), key);
481         } else {
482           std::string keys = key;
483           m_broker.set_preset_cci_value(
484               rel(keys), cci::cci_value(std::string(lua_tostring(L, -1))));
485           if (GC_LUA_VERBOSE)
486             fprintf(stderr, "(SET %s) %s = %s\n",
487                     lua_typename(L, lua_type(L, -1)), rel(keys).c_str(),
```

```
488                     lua_tostring(L, -1));
489           if (should_inc_integer_index_count)
490             ++integer_index_count;
491         }
492         break;
493
494       case LUA_TTABLE:
495         // Avoid recursion on some tables
496         if (strcmp(key, "_G") == 0 || strcmp(key, "package.loaded") == 0 ||
497             strcmp(level, "__index") == 0) {
498           if (GC_LUA_DEBUG)
499             fprintf(stderr, "(%s) %s   (ignored to avoid recursion)\n",
500                     lua_typename(L, lua_type(L, -1)), key);
501         } else {
502           if (GC_LUA_DEBUG)
503             fprintf(stderr, "(table) %s\n", key);
504           *next_level++ = '.';
505           // CS
506           // int int_index_count =
507           setParamsFromLuaTable(L, -1, next_level);
508           // CS
509           // if (int_index_count > 0) {
510           // sprintf(value, "%d", int_index_count);
511           // mApi->setInitValue((std::string(key).substr(0, next_level-key) +
512           // "init_size").c_str(), value); if (GC_LUA_VERBOSE) fprintf(stderr,
513           // "(SET number) %s = %s\n", (std::string(key).substr(0,
514           // next_level-key) + "init_size").c_str(), value);
515           //}
516         }
517         break;
518
519       case LUA_TFUNCTION:
520       case LUA_TNIL:
521       case LUA_TUSERDATA:
522       case LUA_TTHREAD:
523       case LUA_TLIGHTUSERDATA:
524       default:
525         // Ignore other types
526         if (GC_LUA_DEBUG)
527           fprintf(stderr, "(%s) %s\n", lua_typename(L, lua_type(L, -1)), key);
528       }
529
530       /* removes 'value'; keeps 'key' for next iteration */
531       lua_pop(L, 1);
532     }
533
534     return integer_index_count;
535   }
536 #endif
537 };
538
539 #endif
```

6.3 initiator-signal-socket.h

```
1  /*
2   *  This file is part of libgsutils
3   *  Copyright (c) 2021 Luc Michel
4   *
5   *  This program is free software; you can redistribute it and/or
6   *  modify it under the terms of the GNU General Public License
7   *  as published by the Free Software Foundation; either version 2
8   *  of the License, or (at your option) any later version.
9   *
10  *  This program is distributed in the hope that it will be useful,
11  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  *  GNU General Public License for more details.
14  *
15  *  You should have received a copy of the GNU General Public License
16  *  along with this program; if not, write to the Free Software
17  *  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
18  */
19
20 #ifndef _LIBGSUTILS_PORTS_INITIATOR_SIGNAL_SOCKET_H
21 #define _LIBGSUTILS_PORTS_INITIATOR_SIGNAL_SOCKET_H
22
23 #include <systemc>
24
25 template <class T>
26 using InitiatorSignalSocket = sc_core::sc_port<sc_core::sc_signal_inout_if<T>,
27                                                1, sc_core::SC_ZERO_OR_MORE_BOUND>;
28
```

```
29 #endif
30
```

6.4 target-signal-socket.h

```
1 /*
2  *  This file is part of libgsutils
3  *  Copyright (c) 2021 Luc Michel
4  *
5  *  This program is free software; you can redistribute it and/or
6  *  modify it under the terms of the GNU General Public License
7  *  as published by the Free Software Foundation; either version 2
8  *  of the License, or (at your option) any later version.
9  *
10  *  This program is distributed in the hope that it will be useful,
11  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  *  GNU General Public License for more details.
14  *
15  *  You should have received a copy of the GNU General Public License
16  *  along with this program; if not, write to the Free Software
17  *  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
18  */
19
20 #ifndef _LIBGSUTILS_PORTS_TARGET_SIGNAL_SOCKET_H
21 #define _LIBGSUTILS_PORTS_TARGET_SIGNAL_SOCKET_H
22
23 #include <functional>
24 #include <cassert>
25
26 #include <systemc>
27
28 template <class T>
29 class TargetSignalSocket;
30
31 template <class T>
32 class TargetSignalSocketProxy : public sc_core::sc_signal_inout_if<T> {
33 public:
34     using Iface = sc_core::sc_signal_inout_if<T>;
35     using ValueChangedCallback = std::function<void (const T&)>;
36
37 protected:
38     TargetSignalSocket<T> &m_parent;
39
40     T m_val;
41     ValueChangedCallback m_cb;
42     sc_core::sc_event m_ev;
43
44 public:
45     TargetSignalSocketProxy(TargetSignalSocket<T> &parent)
46         : m_parent(parent)
47     {}
48
49     void register_value_changed_cb(const ValueChangedCallback &cb)
50     {
51         m_cb = cb;
52     }
53
54     TargetSignalSocket<T> & get_parent()
55     {
56         return m_parent;
57     }
58
59     void notify()
60     {
61         m_ev.notify();
62     }
63
64     /* SystemC interfaces */
65
66     /* sc_core::sc_signal_in_if<T> */
67     virtual const sc_core::sc_event& default_event() const
68     {
69         return m_ev;
70     }
71
72     virtual const sc_core::sc_event& value_changed_event() const
73     {
74         return m_ev;
75     }
76
77     virtual const T& read() const
78     {
```

```
79          return m_val;
80      }
81
82      virtual const T& get_data_ref() const
83      {
84          return m_val;
85      }
86
87      virtual bool event() const
88      {
89          /* Not implemented */
90          assert(false);
91          return false;
92      }
93
94      /* sc_core::sc_signal_write_if<T> */
95      virtual void write(const T& val)
96      {
97          bool changed = (val != m_val);
98
99          m_val = val;
100
101          if (m_cb) {
102              m_cb(val);
103          }
104
105          if (changed) {
106              m_ev.notify();
107          }
108      }
109 };
110
111 template <>
112 class TargetSignalSocketProxy<bool> : public sc_core::sc_signal_inout_if<bool> {
113 public:
114     using Iface = sc_core::sc_signal_inout_if<bool>;
115     using ValueChangedCallback = std::function<void (const bool&)>;
116
117 protected:
118     TargetSignalSocket<bool> &m_parent;
119
120     bool m_val;
121     ValueChangedCallback m_cb;
122     sc_core::sc_event m_ev;
123     sc_core::sc_event m_posedge_ev;
124     sc_core::sc_event m_negedge_ev;
125
126 public:
127     TargetSignalSocketProxy<bool>(TargetSignalSocket<bool> &parent)
128         : m_parent(parent)
129     {}
130
131     void register_value_changed_cb(const ValueChangedCallback &cb)
132     {
133         m_cb = cb;
134     }
135
136     TargetSignalSocket<bool> & get_parent()
137     {
138         return m_parent;
139     }
140
141     void notify()
142     {
143         m_ev.notify();
144     }
145
146     /* SystemC interfaces */
147
148     /* sc_core::sc_signal_in_if<bool> */
149     virtual const sc_core::sc_event& default_event() const
150     {
151         return m_ev;
152     }
153
154     virtual const sc_core::sc_event& value_changed_event() const
155     {
156         return m_ev;
157     }
158
159     virtual const sc_core::sc_event& posedge_event() const
160     {
161         return m_posedge_ev;
162     }
163
164     virtual const sc_core::sc_event& negedge_event() const
165     {
```

```
166         return m_negedge_ev;
167     }
168
169     virtual const bool& read() const
170     {
171         return m_val;
172     }
173
174     virtual const bool& get_data_ref() const
175     {
176         return m_val;
177     }
178
179     virtual bool event() const
180     {
181         /* Not implemented */
182         assert(false);
183         return false;
184     }
185
186     virtual bool posedge() const
187     {
188         /* Not implemented */
189         assert(false);
190         return false;
191     }
192
193     virtual bool negedge() const
194     {
195         /* Not implemented */
196         assert(false);
197         return false;
198     }
199
200     /* sc_core::sc_signal_write_if<bool> */
201     virtual void write(const bool& val)
202     {
203         bool changed = (val != m_val);
204         m_val = val;
205
206         if (m_cb) {
207             m_cb(val);
208         }
209
210         if (changed) {
211             m_ev.notify();
212             val ? m_posedge_ev.notify() : m_negedge_ev.notify();
213         }
214     }
215 };
216
217 /* TODO: TargetSignalSocketProxy specialization for sc_dt::sc_logic type */
218
219 template <class T>
220 class TargetSignalSocket
221     : public sc_core::sc_export< sc_core::sc_signal_inout_if<T> > {
222 public:
223     using Iface = typename TargetSignalSocketProxy<T>::Iface;
224     using Parent = sc_core::sc_export<Iface>;
225     using ValueChangedCallback = typename TargetSignalSocketProxy<T>::ValueChangedCallback;
226
227 protected:
228     TargetSignalSocketProxy<T> m_proxy;
229
230 public:
231     TargetSignalSocket(const char *name)
232         : Parent(name)
233         , m_proxy(*this)
234     {
235         Parent::bind(m_proxy);
236     }
237
238     void register_value_changed_cb(const ValueChangedCallback &cb)
239     {
240         m_proxy.register_value_changed_cb(cb);
241     }
242
243     const T& read() const
244     {
245         return m_proxy.read();
246     }
247 };
248
249 #endif
```

6.5 report.h

```
1 #ifndef BASIC_PLATFORM_REPORT_H
2 #define BASIC_PLATFORM_REPORT_H
3
4 #include <systemc>
5 #include <cinttypes>
6
7 namespace gs {
8
9     static const char *log_enabled = std::getenv("GS_LOG");
10     static const char *log_enabled_stdout = std::getenv("GS_LOG_STDOUT");
11     static char __gs_log_buffer[100];
12     #define GS_LOG(...)                                                    \
13         do {                                                               \
14             if (gs::log_enabled) {                                         \
15                 if (gs::log_enabled_stdout) {                              \
16                     fprintf(stdout, "%s:%d ", __FILE__, __LINE__);         \
17                     fprintf(stdout, __VA_ARGS__);                          \
18                     fprintf(stdout, "\n");                                 \
19                 } else {                                                   \
20                     snprintf(gs::__gs_log_buffer,sizeof(gs::__gs_log_buffer),__VA_ARGS__); \
21                     auto p=sc_core::sc_get_current_process_b();            \
22                     if (p) sc_core::sc_report_handler::report(sc_core::SC_INFO,
      p->get_parent_object()->basename(), gs::__gs_log_buffer, __FILE__, __LINE__); \
23                     else sc_core::sc_report_handler::report(sc_core::SC_INFO, "non_module",
      gs::__gs_log_buffer, __FILE__, __LINE__); \
24                 }                                                          \
25             }                                                              \
26         } while (0)
27
28 }
29 #endif //BASIC_PLATFORM_REPORT_H
```

6.6 initiator-tester.h

```
1 /*
2  *  This file is part of libgsutils
3  *  Copyright (c) 2021 GreenSocs SAS
4  *
5  *  This program is free software; you can redistribute it and/or
6  *  modify it under the terms of the GNU General Public License
7  *  as published by the Free Software Foundation; either version 2
8  *  of the License, or (at your option) any later version.
9  *
10  *  This program is distributed in the hope that it will be useful,
11  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  *  GNU General Public License for more details.
14  *
15  *  You should have received a copy of the GNU General Public License
16  *  along with this program; if not, write to the Free Software
17  *  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
18  */
19
20 #ifndef _GREENSOCS_GSUTILS_TESTS_INITIATOR_TESTER_H
21 #define _GREENSOCS_GSUTILS_TESTS_INITIATOR_TESTER_H
22
23 #include <functional>
24
25 #include <systemc>
26 #include <tlm>
27 #include <tlm_utils/simple_initiator_socket.h>
28
68 class InitiatorTester : public sc_core::sc_module {
69 public:
70     using TlmGenericPayload = tlm::tlm_generic_payload;
71     using TlmResponseStatus = tlm::tlm_response_status;
72     using TlmDmi = tlm::tlm_dmi;
73
74     using InvalidateDirectMemPtrFn = std::function<void (uint64_t, uint64_t)>;
75
76 private:
77     sc_core::sc_time m_last_txn_delay;
78     unsigned int m_last_transport_debug_ret;
79     bool m_last_dmi_hint = false;
80
81     TlmDmi m_last_dmi_data;
82
83     InvalidateDirectMemPtrFn m_dmi_inval_cb;
84
85     void invalidate_direct_mem_ptr(sc_dt::uint64 start, sc_dt::uint64 end)
86     {
```

```
87          if (m_dmi_inval_cb) {
88              m_dmi_inval_cb(start, end);
89          }
90      }
91
92  protected:
93      virtual void prepare_txn(TlmGenericPayload &txn, bool is_read,
94                               uint64_t addr, uint8_t *data, size_t len)
95      {
96          using namespace tlm;
97
98          tlm_command cmd = is_read ? TLM_READ_COMMAND : TLM_WRITE_COMMAND;
99
100         txn.set_address(addr);
101         txn.set_data_length(len);
102         txn.set_data_ptr(data);
103         txn.set_command(cmd);
104         txn.set_streaming_width(sizeof(data));
105         txn.set_byte_enable_ptr(nullptr);
106         txn.set_response_status(TLM_INCOMPLETE_RESPONSE);
107     }
108
109 public:
110     tlm_utils::simple_initiator_socket<InitiatorTester> socket;
111
112     InitiatorTester(const sc_core::sc_module_name &n)
113         : sc_core::sc_module(n)
114     {
115         socket.register_invalidate_direct_mem_ptr(this,
116                                                   &InitiatorTester::invalidate_direct_mem_ptr);
117     }
118
119     virtual ~InitiatorTester() {}
120
121     /*
122      * b_transport / transport_dbg helpers
123      * -----------------------------------
124      */
125
137     TlmResponseStatus do_b_transport(TlmGenericPayload &txn)
138     {
139         socket->b_transport(txn, m_last_txn_delay);
140         m_last_dmi_hint = txn.is_dmi_allowed();
141
142         return txn.get_response_status();
143     }
144
156     TlmResponseStatus do_transport_dbg(TlmGenericPayload &txn)
157     {
158         m_last_transport_debug_ret = socket->transport_dbg(txn);
159
160         return txn.get_response_status();
161     }
162
175     TlmResponseStatus do_transaction(TlmGenericPayload &txn, bool debug = false)
176     {
177         if (debug) {
178             return do_transport_dbg(txn);
179         } else {
180             return do_b_transport(txn);
181         }
182     }
183
202     TlmResponseStatus do_read_with_txn_and_ptr(TlmGenericPayload &txn,
203                                                uint64_t addr, uint8_t *data,
204                                                size_t len, bool debug = false)
205     {
206         prepare_txn(txn, true, addr, data, len);
207         return do_transaction(txn, debug);
208     }
209
229     TlmResponseStatus do_write_with_txn_and_ptr(TlmGenericPayload &txn,
230                                                 uint64_t addr, const uint8_t *data,
231                                                 size_t len, bool debug = false)
232     {
233         prepare_txn(txn, false, addr, const_cast<uint8_t *>(data), len);
234         return do_transaction(txn, debug);
235     }
236
237
248     TlmResponseStatus do_read_with_ptr(uint64_t addr, uint8_t *data,
249                                        size_t len, bool debug = false)
250     {
251         TlmGenericPayload txn;
252
253         return do_read_with_txn_and_ptr(txn, addr, data, len, debug);
254     }
```

```
255
268     TlmResponseStatus do_write_with_ptr(uint64_t addr, const uint8_t *data,
269                                         size_t len, bool debug = false)
270     {
271         TlmGenericPayload txn;
272
273         return do_write_with_txn_and_ptr(txn, addr, data, len, debug);
274     }
275
276
292     template <class T>
293     TlmResponseStatus do_read_with_txn(TlmGenericPayload &txn, uint64_t addr,
294                                        T& data, bool debug = false)
295     {
296         uint8_t *ptr = reinterpret_cast<uint8_t*>(&data);
297
298         return do_read_with_txn_and_ptr(txn, addr, ptr, sizeof(data), debug);
299     }
300
318     template <class T>
319     TlmResponseStatus do_write_with_txn(TlmGenericPayload &txn, uint64_t addr,
320                                         const T& data, bool debug = false)
321     {
322         const uint8_t *ptr = reinterpret_cast<const uint8_t*>(&data);
323
324         return do_write_with_txn_and_ptr(txn, addr, ptr, sizeof(data), debug);
325     }
326
327
337     template <class T>
338     TlmResponseStatus do_read(uint64_t addr, T& data, bool debug = false)
339     {
340         uint8_t *ptr = reinterpret_cast<uint8_t*>(&data);
341
342         return do_read_with_ptr(addr, ptr, sizeof(data), debug);
343     }
344
356     template <class T>
357     TlmResponseStatus do_write(uint64_t addr, const T& data, bool debug = false)
358     {
359         const uint8_t *ptr = reinterpret_cast<const uint8_t*>(&data);
360
361         return do_write_with_ptr(addr, ptr, sizeof(data), debug);
362     }
363
364
370     void set_next_txn_delay(const sc_core::sc_time &delay)
371     {
372         m_last_txn_delay = delay;
373     }
374
380     const sc_core::sc_time &get_last_txn_delay() const
381     {
382         return m_last_txn_delay;
383     }
384
390     unsigned int get_last_transport_debug_ret() const
391     {
392         return m_last_transport_debug_ret;
393     }
394
401     bool get_last_dmi_hint() const
402     {
403         return m_last_dmi_hint;
404     }
405
406     /*
407      * get_direct_mem_ptr helpers
408      * --------------------------
409      */
410
419     bool do_dmi_request(uint64_t addr)
420     {
421         TlmGenericPayload txn;
422
423         prepare_txn(txn, true, addr, nullptr, 0);
424         return socket->get_direct_mem_ptr(txn, m_last_dmi_data);
425     }
426
432     const TlmDmi &get_last_dmi_data() const
433     {
434         return m_last_dmi_data;
435     }
436
437     /*
438      * Backward interface helpers
439      * --------------------------
```

```
440      */
441
445     void register_invalidate_direct_mem_ptr(InvalidateDirectMemPtrFn cb)
446     {
447         m_dmi_inval_cb = cb;
448     }
449 };
450
451 #endif
```

6.7 target-tester.h

```
1 /*
2  *  This file is part of libgsutils
3  *  Copyright (c) 2021 GreenSocs SAS
4  *
5  *  This program is free software; you can redistribute it and/or
6  *  modify it under the terms of the GNU General Public License
7  *  as published by the Free Software Foundation; either version 2
8  *  of the License, or (at your option) any later version.
9  *
10  *  This program is distributed in the hope that it will be useful,
11  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  *  GNU General Public License for more details.
14  *
15  *  You should have received a copy of the GNU General Public License
16  *  along with this program; if not, write to the Free Software
17  *  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
18  */
19
20 #ifndef GREENSOCS_GSUTILS_TESTS_TARGET_TESTER_H
21 #define GREENSOCS_GSUTILS_TESTS_TARGET_TESTER_H
22
23 #include <cstring>
24 #include <functional>
25
26 #include <gtest/gtest.h>
27
28 #include <systemc>
29 #include <tlm>
30 #include <tlm_utils/simple_target_socket.h>
31
58 class TargetTester : public sc_core::sc_module {
59 public:
60     using TlmGenericPayload = tlm::tlm_generic_payload;
61     using TlmResponseStatus = tlm::tlm_response_status;
62     using TlmDmi = tlm::tlm_dmi;
63
64     using AccessCallbackFn = std::function<TlmResponseStatus (uint64_t addr, uint8_t *data, size_t len)>;
65     using DebugAccessCallbackFn = std::function<int (uint64_t addr, uint8_t *data, size_t len)>;
66     using GetDirectMemPtrCallbackFn = std::function<bool (uint64_t addr, TlmDmi &)>;
67
68 private:
69     size_t m_mmio_size;
70
71     /* Only valid within a transaction callback */
72     TlmGenericPayload *m_cur_txn = nullptr;
73     sc_core::sc_time *m_cur_txn_delay = nullptr;
74
75     bool m_last_txn_valid = false;
76     TlmGenericPayload m_last_txn;
77
78     bool m_last_txn_delay_valid = false;
79     sc_core::sc_time m_last_txn_delay;
80
81     AccessCallbackFn m_read_cb;
82     AccessCallbackFn m_write_cb;
83
84     DebugAccessCallbackFn m_debug_read_cb;
85     DebugAccessCallbackFn m_debug_write_cb;
86
87     GetDirectMemPtrCallbackFn m_dmi_cb;
88
89     /* Factorized version of b_transport and transport_dbg */
90     template <class RET, RET DEFAULT_RET, RET ADDRESS_ERROR_RET, RET DEFAULT_CB_RET, class CB_FN>
91     RET generic_access(TlmGenericPayload &txn, const CB_FN &read_cb, const CB_FN &write_cb)
92     {
93         using namespace tlm;
94
95         uint64_t addr;
96         uint8_t *ptr;
97         size_t len;
```

```
98          RET ret = DEFAULT_RET;
99
100         m_cur_txn = &txn;
101
102         if (txn.get_command() == TLM_IGNORE_COMMAND) {
103             goto out;
104         }
105
106         if (txn.get_address() + txn.get_data_length() >= m_mmio_size) {
107             ret = ADDRESS_ERROR_RET;
108             goto out;
109         }
110
111         addr = txn.get_address();
112         ptr = txn.get_data_ptr();
113         len = txn.get_data_length();
114
115         switch (txn.get_command()) {
116         case TLM_READ_COMMAND:
117             if (m_read_cb) {
118                 ret = read_cb(addr, ptr, len);
119             } else {
120                 ret = DEFAULT_CB_RET;
121             }
122             break;
123
124         case TLM_WRITE_COMMAND:
125             if (m_write_cb) {
126                 ret = write_cb(addr, ptr, len);
127             } else {
128                 std::memset(ptr, 0, len);
129                 ret = DEFAULT_CB_RET;
130             }
131             break;
132
133         default:
134             /* TLM_IGNORE_COMMAND already handled above */
135             ADD_FAILURE();
136         }
137
138 out:
139         m_cur_txn = nullptr;
140
141         m_last_txn.deep_copy_from(txn);
142         m_last_txn_valid = true;
143
144         return ret;
145     }
146
147 protected:
148     virtual void b_transport(TlmGenericPayload &txn, sc_core::sc_time &delay)
149     {
150         using namespace tlm;
151
152         TlmResponseStatus ret;
153
154         m_cur_txn_delay = &delay;
155         ret = generic_access<TlmResponseStatus, TLM_OK_RESPONSE, TLM_ADDRESS_ERROR_RESPONSE,
156                              TLM_OK_RESPONSE, AccessCallbackFn>(txn, m_read_cb, m_write_cb);
157         m_cur_txn_delay = nullptr;
158
159         txn.set_response_status(ret);
160
161         m_last_txn_delay = delay;
162         m_last_txn_delay_valid = true;
163     }
164
165     virtual unsigned int transport_dbg(TlmGenericPayload &txn)
166     {
167         int ret = 0;
168
169         ret = generic_access<int, 0, 0, -1,
170                              DebugAccessCallbackFn>(txn, m_debug_read_cb, m_debug_write_cb);
171
172         if (ret == -1) {
173             ret = txn.get_data_length();
174         }
175
176         return ret;
177     }
178
179     virtual bool get_direct_mem_ptr(TlmGenericPayload &txn, TlmDmi &dmi_data)
180     {
181         uint64_t addr;
182
183         m_last_txn.deep_copy_from(m_last_txn);
184
```

```
185         addr = txn.get_address();
186
187         if (m_dmi_cb) {
188             return m_dmi_cb(addr, dmi_data);
189         } else {
190             return false;
191         }
192     }
193
194 public:
195     tlm_utils::simple_target_socket<TargetTester> socket;
196
203     TargetTester(const sc_core::sc_module_name &n, size_t mmio_size)
204         : sc_core::sc_module(n)
205         , m_mmio_size(mmio_size)
206     {
207         socket.register_b_transport(this, &TargetTester::b_transport);
208         socket.register_transport_dbg(this, &TargetTester::transport_dbg);
209         socket.register_get_direct_mem_ptr(this, &TargetTester::get_direct_mem_ptr);
210     }
211
212     virtual ~TargetTester() {}
213
217     void register_read_cb(AccessCallbackFn cb)
218     {
219         m_read_cb = cb;
220     }
221
225     void register_write_cb(AccessCallbackFn cb)
226     {
227         m_write_cb = cb;
228     }
229
233     void register_debug_read_cb(DebugAccessCallbackFn cb)
234     {
235         m_debug_read_cb = cb;
236     }
237
241     void register_debug_write_cb(DebugAccessCallbackFn cb)
242     {
243         m_debug_write_cb = cb;
244     }
245
249     void register_get_direct_mem_ptr_cb(GetDirectMemPtrCallbackFn cb)
250     {
251         m_dmi_cb = cb;
252     }
253
264     bool last_txn_is_valid() const
265     {
266         return m_last_txn_valid;
267     }
268
278     const TlmGenericPayload &get_last_txn()
279     {
280         EXPECT_TRUE(m_last_txn_valid);
281         m_last_txn_valid = false;
282
283         return m_last_txn;
284     }
285
295     const sc_core::sc_time &get_last_txn_delay()
296     {
297         EXPECT_TRUE(m_last_txn_delay_valid);
298         m_last_txn_delay_valid = false;
299
300         return m_last_txn_delay;
301     }
302
312     TlmGenericPayload & get_cur_txn()
313     {
314         EXPECT_TRUE(m_cur_txn != nullptr);
315         return *m_cur_txn;
316     }
317
327     sc_core::sc_time & get_cur_txn_delay()
328     {
329         EXPECT_TRUE(m_cur_txn_delay != nullptr);
330         return *m_cur_txn_delay;
331     }
332 };
333
334 #endif
```

6.8 test-bench.h

```
1 /*
2  *  This file is part of libgsutils
3  *  Copyright (c) 2021 Luc Michel
4  *
5  *  This program is free software; you can redistribute it and/or
6  *  modify it under the terms of the GNU General Public License
7  *  as published by the Free Software Foundation; either version 2
8  *  of the License, or (at your option) any later version.
9  *
10  *  This program is distributed in the hope that it will be useful,
11  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  *  GNU General Public License for more details.
14  *
15  *  You should have received a copy of the GNU General Public License
16  *  along with this program; if not, write to the Free Software
17  *  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
18  */
19
20 /*
21  * GTest helper to workaround SystemC limitations regarding its non-resettable nature.
22  *
23  * You can use the TEST_BENCH macro as you would use the TEST_F GTest macro.
24  * Your fixture class must inherit from the TestBench class.
25  *
26  * Before each test, the test program is forked so that it starts from a fresh
27  * SystemC environment.
28  *
29  *
30  * Using GDB
31  * =========
32  *
33  * Note that debugging with GDB is a little more involved, because of the forks.
34  * You can do something like this:
35  *
36  * (gdb) set detach-on-fork off
37  * (gdb) set non-stop on
38  * (gdb) b somewhere
39  * (gdb) r
40  * Thread 2.1 "xxx" hit Breakpoint 1, somewhere() (...)
41     at ...
42
43  * (gdb) thread 2.1
44  * (gdb) ... you can now debug as normal in the child
45  *
46  * Note that in non-stop mode, other inferiors are still running. To stop one,
47  * select it with the 'thread' command and issue the 'interrupt' command.
48  *
49  * Example
50  * =======
51  *
52  * class MyTestBench : public TestBench {
53  * private:
54  *     SomeModule m_module_under_test;
55  *     AnotherModule m_mock_module;
56  *
57  * public:
58  *     MyTestBench(const sc_core::sc_module_name &n)
59  *         : TestBench(n)
60  *         , m_module_under_test("module-under-test")
61  *         , m_mock_module("mock-module")
62  *     {
63  *         m_mock_module.target_socket.bind(m_module_under_test.initiator_socket);
64  *     }
65  * };
66  *
67  * TEST_BENCH(MyTestBench, Test0)
68  * {
69  *     m_module_under_test.do_something();
70  *     ASSERT_TRUE(m_mock_module.expected_result());
71  * }
72  *
73  * int sc_main(int argc, char *argv[])
74  * {
75  *     ::testing::InitGoogleTest(&argc, argv);
76  *     return RUN_ALL_TESTS();
77  * }
78  *
79  */
80
81 #ifndef _GREENSOCS_GSUTILS_TESTS_TEST_BENCH_H
82 #define _GREENSOCS_GSUTILS_TESTS_TEST_BENCH_H
83
84 #include <systemc>
85
```

```
86 #include <gtest/gtest.h>
87
88 class TestBench : public sc_core::sc_module {
89 protected:
90     virtual void test_bench_body() = 0;
91
92 public:
93     SC_HAS_PROCESS(TestBench);
94     TestBench(const sc_core::sc_module_name &n)
95         : sc_core::sc_module(n)
96     {
97         SC_THREAD(test_bench_body);
98     }
99 };
100
101 #ifndef _WIN32
102 #include <sys/types.h>
103 #include <sys/wait.h>
104 #include <unistd.h>
105
106 static inline bool test_bench_succeeded(int ret)
107 {
108     return WIFEXITED(ret) && (WEXITSTATUS(ret) == 0);
109 }
110 template <class T>
111 static inline void run_test_bench()
112 {
113     pid_t pid;
114
115     pid = fork();
116
117     ASSERT_NE(pid, -1);
118
119     if (!pid) {
120         T test_bench("test-bench");
121         sc_core::sc_start();
122         exit(::testing::Test::HasFatalFailure());
123     } else {
124         int ret;
125         wait(&ret);
126         ASSERT_TRUE(test_bench_succeeded(ret));
127     }
128 }
129
130 #else /* _WIN32 */
131 #include <iostream>
132
133 static inline void run_test_bench()
134 {
135     std::cerr << "Running tests on Windows is not supported.\n";
136     ASSERT_TRUE(false);
137 }
138
139 #endif
140
141 #define TEST_BENCH_NAME(name) \
142     TestBench__ ## name
143
144 #define TEST_BENCH(test_bench, name)                                \
145     class TEST_BENCH_NAME(name) : public test_bench {               \
146     protected:                                                      \
147         void test_bench_body() override;                            \
148     public:                                                         \
149         TEST_BENCH_NAME(name)(const sc_core::sc_module_name &n)     \
150             : test_bench(n) {}                                      \
151     };                                                              \
152     TEST(test_bench, name)                                          \
153     {                                                               \
154         run_test_bench<TEST_BENCH_NAME(name)>();                    \
155     }                                                               \
156     void TEST_BENCH_NAME(name)::test_bench_body()
157
158 #endif
```

6.9 exclusive-access.h

```
1 /*
2  *  This file is part of libgsutils
3  *  Copyright (C) 2021  Luc Michel
4  *
5  *  This program is free software; you can redistribute it and/or
6  *  modify it under the terms of the GNU General Public License
7  *  as published by the Free Software Foundation; either version 2
```

```
8  *  of the License, or (at your option) any later version.
9  *
10 *  This program is distributed in the hope that it will be useful,
11 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13 *  GNU General Public License for more details.
14 *
15 *  You should have received a copy of the GNU General Public License
16 *  along with this program; if not, write to the Free Software
17 *  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
18 */
19
20 #ifndef _GREENSOCS_GSUTILS_TLM_EXTENSIONS_EXCLUSIVE_ACCESS_H
21 #define _GREENSOCS_GSUTILS_TLM_EXTENSIONS_EXCLUSIVE_ACCESS_H
22
23 #include <vector>
24
25 #include <tlm>
26
47 class ExclusiveAccessTlmExtension
48     : public tlm::tlm_extension<ExclusiveAccessTlmExtension>
49 {
50 public:
51     enum ExclusiveStoreStatus {
52         EXCLUSIVE_STORE_NA = 0,
53         EXCLUSIVE_STORE_SUCCESS,
54         EXCLUSIVE_STORE_FAILURE
55     };
56
57     class InitiatorId {
58     private:
59         std::vector<int> m_id;
60
61     public:
62         void add_hop(int id)
63         {
64             m_id.push_back(id);
65         }
66
67         bool operator<(const InitiatorId &o) const
68         {
69             if (m_id.size() != o.m_id.size()) {
70                 return m_id.size() < o.m_id.size();
71             }
72
73             auto it0 = m_id.begin();
74             auto it1 = o.m_id.begin();
75             for (; it0 != m_id.end(); it0++, it1++) {
76                 if (*it0 != *it1) {
77                     return *it0 < *it1;
78                 }
79             }
80
81             return false;
82         }
83
84         bool operator==(const InitiatorId &o) const
85         {
86             if (m_id.size() != o.m_id.size()) {
87                 return false;
88             }
89
90             auto it0 = m_id.begin();
91             auto it1 = o.m_id.begin();
92             for (; it0 != m_id.end(); it0++, it1++) {
93                 if (*it0 != *it1) {
94                     return false;
95                 }
96             }
97
98             return true;
99         }
100
101         bool operator!=(const InitiatorId &o) const
102         {
103             return !(*this == o);
104         }
105     };
106
107 private:
108     InitiatorId m_id;
109     ExclusiveStoreStatus m_store_sta = EXCLUSIVE_STORE_NA;
110
111 public:
112     ExclusiveAccessTlmExtension() = default;
113     ExclusiveAccessTlmExtension(const ExclusiveAccessTlmExtension &) = default;
114
```

```
115     virtual tlm_extension_base* clone() const override
116     {
117         return new ExclusiveAccessTlmExtension(*this);
118     }
119
120     virtual void copy_from(const tlm_extension_base &ext) override
121     {
122         const ExclusiveAccessTlmExtension &other =
123             static_cast<const ExclusiveAccessTlmExtension &>(ext);
124
125         m_id = other.m_id;
126         m_store_sta = other.m_store_sta;
127     }
128
129     void set_exclusive_store_success()
130     {
131         m_store_sta = EXCLUSIVE_STORE_SUCCESS;
132     }
133
134     void set_exclusive_store_failure()
135     {
136         m_store_sta = EXCLUSIVE_STORE_FAILURE;
137     }
138
139     ExclusiveStoreStatus get_exclusive_store_status() const
140     {
141         return m_store_sta;
142     }
143
144     void add_hop(int id)
145     {
146         m_id.add_hop(id);
147     }
148
149     const InitiatorId & get_initiator_id() const
150     {
151         return m_id;
152     }
153 };
154
155 #endif
156
```

6.10 libgsutils.h

```
1 /*
2  *  Copyright (C) 2021  GreenSocs
3  *
4  */
5
6 #include "greensocs/gsutils/report.h"
7 #include "greensocs/gsutils/cciutils.h"
8 #include "greensocs/gsutils/luafile_tool.h"
```

Index