gssync

Contents

Chapter 1

Main Page

[//]: # DONT EDIT THIS FILE

The GreenSocs Synchronization library provides a number of different policies for synchronizing between an external simulator (typically QEMU) and SystemC.

These are based on a proposed standard means to handle the SystemC simulator. This library provides a backwards compatibility layer, but the patched version of SystemC will perform better.

1.1   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>_SOUR↩ CE_DIR`. Hence you may wish to use your own copy of SystemC CCI "`bash cmake -B build -DCPM_↩ SystemCCCI_SOURCE=/path/to/your/cci/source`

```
It may also be convenient to have all the source files downloaded, you may do this by running
```bash
cmake -B build -DCPM_SOURCE_CACHE=`pwd`/Packages
```

This will populate the directory `Packages` Note that the cmake file system will automatically use the directory called `Packages` as source, if it exists.

NB, CMake holds a cache of compiled modules in ∼/.cmake/ Sometimes this can confuse builds. If you seem to be picking up the wrong version of a module, then it may be in this cache. It is perfectly safe to delete it.

1.2.2.1 Common CMake options

`CMAKE_INSTALL_PREFIX` : Install directory for the package and binaries. `CMAKE_BUILD_TYPE` : DEBUG or RELEASE

The library assumes the use of C++14, and is compatible with SystemC versions from SystemC 2.3.1a.

For a reference docker please use the following script from the top level of the Virtual Platform:

```
curl --header 'PRIVATE-TOKEN: W1Z9U8S_5BUEX1_Y29iS'
     'https://git.greensocs.com/api/v4/projects/65/repository/files/docker_vp.sh/raw?ref=master' -o docker_vp.sh
chmod +x ./docker_vp.sh
./docker_vp.sh
> cmake -B build;cd build; make -j
```

1.2.2.2 passwords for git.greensocs.com

To avoid using passwords for git.greensocs.com please add a ssh key to your git account. You may also use a key-chain manager. As a last resort, the following script will populate ∼/.git-credentials with your username and password (in plain text)

```
git config --global credential.helper store
```

1.2.3 More documentation

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

1.2.4 The GreenSocs SystemC simple components library.

This includes simple models such as routers, memories and exclusive monitor. The components are "Loosely timed" only. They support DMI where appropriate, and make use of CCI for configuration.

It also has several unit tests for memory, router and exclusive monitor.

1.2.5 LIBGSUTILS

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

1.2.6 Information about building and using the libgssync library

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

1.2.7 Information about building and using the base-components library

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

1.2.8 Information about building and using the libgsutils library

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

The GreenSocs CCI libraries allows two options for setting configuration parameters

`--gs_luafile <FILE.lua>` this option will read the lua file to set parameters.

`--param path.to.param=<value>` this option will allow individual parameters to be set.

NOTE, order is important, the last option on the command line to set a parameter will take preference.

This library includes a Configurable Broker (gs::ConfigurableBroker) which provides additional functionality. Each broker can be configured separately, and has a parameter itself for the configuration file to read. This is `lua_file`. Hence

`--param path.to.module.lua_file="\"/host/path/to/lua/file""`

Note that a string parameter must be quoted.

The lua file read by the ConfigurableBroker has relative paths - this means that in the example above the `path.↩ to.module` portion of the absolute path should not appear in the (local) configuration file. (Hence changes in the hierarchy will not need changes to the configuration file).

1.2.9 Using yaml for configuration

If you would prefer to use yaml as a configuration language, `lyaml` provides a link. This can be downloaded from
https://github.com/gvvaughan/lyaml

The following lua code will load "conf.yaml".

```
local lyaml  = require "lyaml"

function readAll(file)
    local f = assert(io.open(file, "rb"))
    local content = f:read("*all")
    f:close()
    return content
end

print "Loading conf.yaml"
yamldata=readAll("conf.yaml")
ytab=lyaml.load(yamldata)
for k,v in pairs(ytab) do
    _G[k]=v
end
yamldata=nil
ytab=nil
```

1.2.10 Functionality of the synchronization library

In addition the library contains utilities such as an thread safe event (async_event) and a real time speed limited for SystemC.

1.2.10.1 Suspend/Unsuspend interface

This patch adds four new basic functions to SystemC:

```
void sc_suspend_all(sc_simcontext* csc= sc_get_curr_simcontext())
void sc_unsuspend_all(sc_simcontext* csc= sc_get_curr_simcontext())
void sc_unsuspendable()
void sc_suspendable()
```

suspend_all/unsuspend_all : This pair of functions requests the kernel to 'atomically suspend' all processes (using the same semantics as the thread suspend() call). This is atomic in that the kernel will only suspend all the processes together, such that they can be suspended and unsuspended without any side effects. Calling suspend_all(), and subsiquently calling unsuspend_all() will have no effect on the suspended status of an individual process. A process may call suspend_all() followed by unsuspend_all, the calls should be 'paired', (multiple calls to either suspend_all() or unsuspend_all() will be ignored). Outside of the context of a process, it is the programmers responsibility to ensure that the calls are paired. As a consequence, multiple calls to suspend_all() may be made (within separate process, or from within sc_main). So long as there have been more calls to suspend_all() than to unsuspend_all(), the kernel will suspend all processes.

[note, this patch set does not add convenience functions, including those to find out if suspension has happened, these are expected to be layered ontop]

unsusbendable()/suspendable(): This pair of functions provides an 'opt-out' for specific process to the suspend←_all(). The consequence is that if there is a process that has opted out, the kernel will not be able to suspend_all (as it would no longer be atomic). These functions can only be called from within a process. A process should only call suspendable/unsuspendable in pairs (multiple calls to either will be ignored). *Note that the default is that a process is marked as suspendable.*

Use cases: *1 : Save and Restore* For Save and Restore, the expectation is that when a save is requested, 'suspend_all' will be called. If there are models that are in an unsuspendable state, the entire simulation will be allowed to continue until such a time that there are no unsuspendable processes.

2 : External sync When an external model injects events into a SystemC model (for instance, using an 'async_↩ request_update()'), time can drift between the two simulators. In order to maintain time, SystemC can be prevented from advancing by calling suspend_all(). If there are process in an unsuspendable state (for instance, processing on behalf of the external model), then the simulation will be allowed to continue. NOTE, an event injected into the kernel by an async_request_update will cause the kernel to execute the associated update() function (leaving the suspended state). The update function should arrange to mark any processes that it requires as unsuspendable before the end of the current delta cycle, to ensure that they are scheduled.

1.2.11 List of options of sync policy parameter

The libgssync library allows you to set several values to the `p_sync_policy` parameter :

- `tlm2`

- `multithread`

- `multithread-quantum`

- `multithread-rolling`

- `multithread-unconstrained`

By default the parameter is set to `multithread-quantum`.

1.2.12 The GreenSocs component library memory

The memory component allows you to add memory when creating an object of type `Memory("name",size)`.

The memory component consists of a simple target socket :`tlm_utils::simple_target_socket<Memory> socket`

1.2.13 The GreenSocs component library router

The router offers `add_target(socket, base_address, size)` as an API to add components into the address map for routing. (It is recommended that the addresses and size are CCI parameters).

It also allows to bind multiple initiators with `add_initiator(socket)` to send multiple transactions. So there is no need for the bind() method offered by sockets because the add_initiator method already takes care of that.

1.2.14 Using the ConfigurableBroker

The broker will self register in the SystemC CCI hierarchy. All brokers have a parameter `lua_file` which will be read and used to configure parameters held within the broker. This file is read at the *local* level, and paths are *relative* to the location where the ConfigurableBroker is instanced.

These brokers can be used as global brokers.

The `gs::ConfigurableBroker` can be instanced in 3 ways:

1. `ConfigurableBroker()` This will instance a 'Private broker' and will hide **ALL** parameters held within this broker.

   A local `lua_file` can be read and will set parameters in the private broker. This can be prevented by passing 'false' as a construction parameter (`ConfigurableBroker(false)`).

2. `ConfigurableBroker({{"key1","value1"},{"key2","value2")...})` This will instance a broker that sets and hides the listed keys. All other keys are passed through (exported). Hence the broker is 'invisible' for parameters that are not listed. This is specifically useful for structural parameters.

   It is also possible to instance a 'pass through' broker using `ConfigurationBroker({})`. This is useful to provide a *local* configuration broker than can, for instance, read a local configuration file.

   A local `lua_file` can be read and will set parameters in the private broker (exported or not). This can be prevented by passing 'false' as a construction parameter (`ConfigurableBroker(false)`). The `lua↩_file` will be read **AFTER** the construction key-value list and hence can be used to over-right default values in the code.

3. `ConfigurableBroker(argc, argv)` This will instance a broker that is typically a global broker. The argc/argv values should come from the command line. The command line will be parsed to find:

   > `-p, --param path.to.param=<value>` this option will allow individual parameters to be set.

   > `-l, --gs_luafile <FILE.lua>` this option will read the lua file to set parameters. Similar functionality can be achieved using –param lua_file="<FILE.lua>".

   A `{{key,value}}` list can also be provided, otherwise it is assumed to be empty. Such a list will set parameter values within this broker. These values will be read and used **BEFORE** the command line is read.

   Finally **AFTER** the command line is read, if the `lua_file` parameter has been set, the configuration file that it indicates will also be read. This can be prevented by passing 'false' as a construction parameter (`ConfigurableBroker(argc, argv, false)`). The `lua_file` will be read **AFTER** the construction key-value list, and after the command like, so it can be used to over-right default values in either.

1.2.15 The GreenSocs Synchronization Tests

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

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

In this test folder you will find several executable which each correspond to a test. You can run the executable with :

```
./build/tests/integration_tests/<name_of_component>_test
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Class Documentation

4.1 gs::async_event Class Reference

Inheritance diagram for gs::async_event:
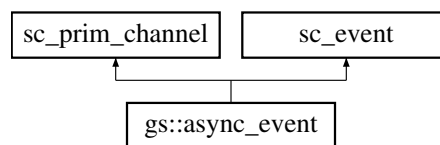
Public Member Functions

- **async_event** (bool start_attached=true)
- void **async_notify** ()
- void **notify** (sc_core::sc_time delay=sc_core::sc_time(sc_core::SC_ZERO_TIME))
- void **async_attach_suspending** ()
- void **async_detach_suspending** ()
- void **enable_attach_suspending** (bool e)

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/async_event.h

4.2 gs::RunOnSysC::AsyncJob Class Reference

Public Types

- using **Ptr** = std::shared_ptr< AsyncJob >

Public Member Functions

- **AsyncJob** (std::function< void()> &&job)
- **AsyncJob** (std::function< void()> &job)
- **AsyncJob** (const AsyncJob &)=delete
- void **operator()** ()
- void cancel ()

    Cancel a job.
- void **wait** ()
- bool **is_cancelled** () const

4.2.1 Member Function Documentation

4.2.1.1 cancel()

```
void gs::RunOnSysC::AsyncJob::cancel ( )  [inline]
```

Cancel a job.

Cancel a job by setting m_cancelled to true and by resetting the task. Any waiter will then be unblocked immediately.

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/runonsysc.h

4.3 gs::global_pause Class Reference

Public Member Functions

- **global_pause** (const global_pause &)=delete
- void **suspendable** ()
- void **unsuspendable** ()
- void **unsuspend_all** ()
- void **suspend_all** ()
- bool **attach_suspending** (sc_core::sc_prim_channel ∗p)
- bool **detach_suspending** (sc_core::sc_prim_channel ∗p)
- void **async_wakeup** ()

Static Public Member Functions

- static global_pause & **get** ()

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/pre_suspending_sc↩
  _support.h
- /home/thomas/Documents/GreenSocs/build-lib/libgssync/src/pre_suspending_sc_support.cc

4.4   gs::InLineSync Class Reference

Inheritance diagram for gs::InLineSync:

```
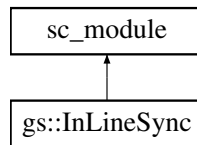┌─────────────┐
│  sc_module  │
└─────────────┘
       ▲
       │
┌─────────────┐
│ gs::InLineSync │
└─────────────┘
```

Public Member Functions

- **SC_HAS_PROCESS** (InLineSync)
- **InLineSync** (const sc_core::sc_module_name &name)
- void **b_transport** (tlm::tlm_generic_payload &trans, sc_core::sc_time &delay)
- void **get_direct_mem_ptr** (tlm::tlm_generic_payload &trans, tlm::tlm_dmi &dmi_data)
- void **transport_dgb** (tlm::tlm_generic_payload &trans)
- void **invalidate_direct_mem_ptr** (sc_dt::uint64 start_range, sc_dt::uint64 end_range)

Public Attributes

- tlm_utils::simple_target_socket< InLineSync > **target_socket**
- tlm_utils::simple_initiator_socket< InLineSync > **initiator_socket**

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/inlinesync.h

4.5   gs::RunOnSysC Class Reference

Inheritance diagram for gs::RunOnSysC:

```
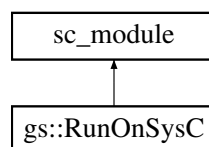┌─────────────┐
│  sc_module  │
└─────────────┘
       ▲
       │
┌─────────────┐
│ gs::RunOnSysC │
└─────────────┘
```

Classes

- class AsyncJob

Public Member Functions

- **RunOnSysC** (const sc_core::sc_module_name &n=sc_core::sc_module_name("run-on-sysc"))
- void cancel_pendings ()

  Cancel all pending jobs.
- void cancel_all ()

  Cancel all pending and running jobs.
- void **end_of_simulation** ()
- void **fork_on_systemc** (std::function< void()> job_entry)
- bool run_on_sysc (std::function< void()> job_entry, bool wait=true)

  Run a job on the SystemC kernel thread.

Protected Member Functions

- void **jobs_handler** ()
- void **cancel_pendings_locked** ()

Protected Attributes

- std::thread::id **m_thread_id**
- std::queue< AsyncJob::Ptr > **m_async_jobs**
- AsyncJob::Ptr **m_running_job**
- std::mutex **m_async_jobs_mutex**
- async_event **m_jobs_handler_event**

4.5.1   Member Function Documentation

4.5.1.1   cancel_all()

```
void gs::RunOnSysC::cancel_all ( )  [inline]
```

Cancel all pending and running jobs.

Cancel all the pending jobs and the currently running job. The callers will be unblocked if they are waiting for the job. Note that if the currently running job is resumed, the behaviour is undefined. This method is meant to be called after simulation has ended.

4.5.1.2   cancel_pendings()

```
void gs::RunOnSysC::cancel_pendings ( )  [inline]
```

Cancel all pending jobs.

Cancel all the pending jobs. The callers will be unblocked if they are waiting for the job.

4.5.1.3   run_on_sysc()

```
bool gs::RunOnSysC::run_on_sysc (
            std::function< void()> job_entry,
            bool wait = true )  [inline]
```

Run a job on the SystemC kernel thread.

Parameters

in	*job_entry*	The job to run
in	*wait*	If true, wait for job completion

Returns

true if the job has been succesfully executed or if `wait` was false, false if it has been cancelled (see Run↵
OnSysC::cancel_all).

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/runonsysc.h

4.6 gs::semaphore Class Reference

Public Member Functions

- void **notify** ()
- void **wait** ()
- bool **try_wait** ()

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/semaphore.h

4.7 gs::tlm_quantumkeeper_extended Class Reference

Inheritance diagram for gs::tlm_quantumkeeper_extended:
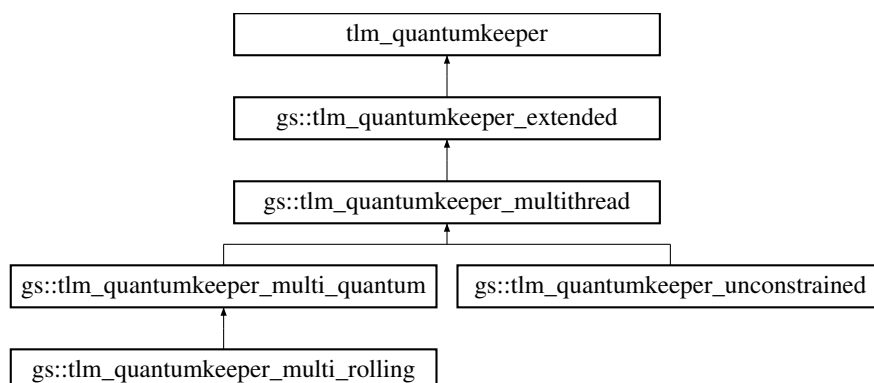
Public Member Functions

- virtual sc_core::sc_time **time_to_sync** ()
- virtual void **stop** ()
- virtual void **start** (std::function< void()> job=nullptr)
- virtual SyncPolicy::Type **get_thread_type** () const
- virtual bool **need_sync** ()
- virtual bool **need_sync** () const override
- virtual void **run_on_systemc** (std::function< void()> job)

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/qk_extendedif.h

4.8   gs::tlm_quantumkeeper_multi_quantum Class Reference

Inheritance diagram for gs::tlm_quantumkeeper_multi_quantum:

Additional Inherited Members

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/qkmulti-quantum.h

4.9   gs::tlm_quantumkeeper_multi_rolling Class Reference

Inheritance diagram for gs::tlm_quantumkeeper_multi_rolling:

Additional Inherited Members

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/qkmulti-rolling.h

4.10 gs::tlm_quantumkeeper_multithread Class Reference

Inheritance diagram for gs::tlm_quantumkeeper_multithread:
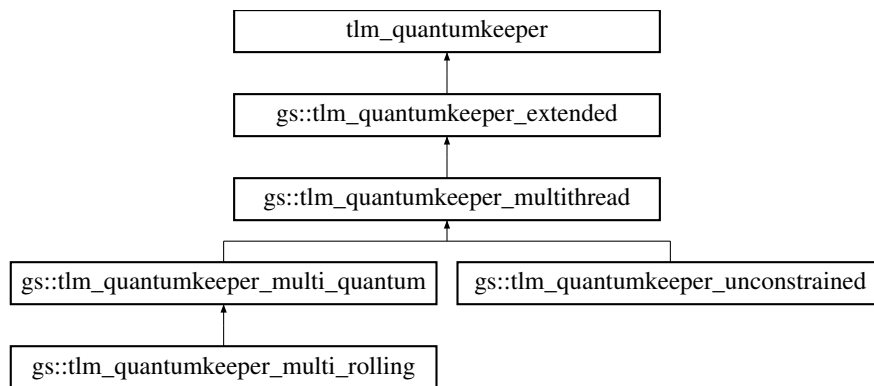
Public Member Functions

- virtual SyncPolicy::Type **get_thread_type** () const override
- virtual void **start** (std::function< void()> job=nullptr) override
- virtual void **stop** () override
- virtual sc_core::sc_time **time_to_sync** () override
- void **inc** (const sc_core::sc_time &t) override
- void **set** (const sc_core::sc_time &t) override
- virtual void **sync** () override
- void **reset** () override
- sc_core::sc_time **get_current_time** () const override
- sc_core::sc_time **get_local_time** () const override
- virtual bool **need_sync** () override

Protected Types

- enum **jobstates** { **NONE**, **RUNNING**, **STOPPED** }

Protected Member Functions

- virtual bool **is_sysc_thread** () const

Protected Attributes

- enum gs::tlm_quantumkeeper_multithread::jobstates **status**
- async_event **m_tick**

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/qkmultithread.h
- /home/thomas/Documents/GreenSocs/build-lib/libgssync/src/qkmultithread.cc

4.11 gs::tlm_quantumkeeper_unconstrained Class Reference

Inheritance diagram for gs::tlm_quantumkeeper_unconstrained:

```
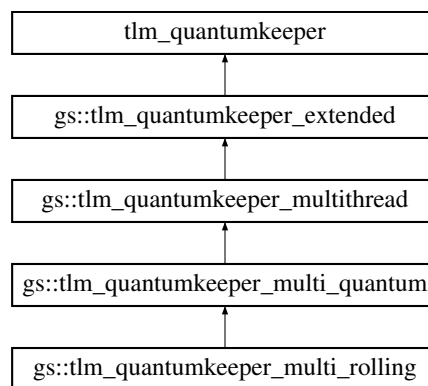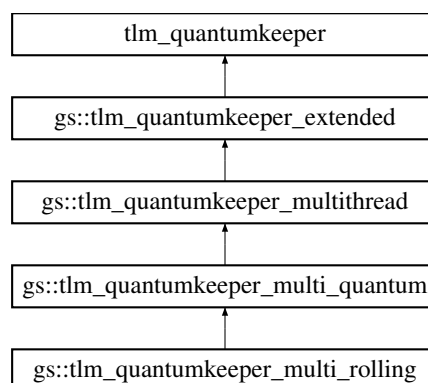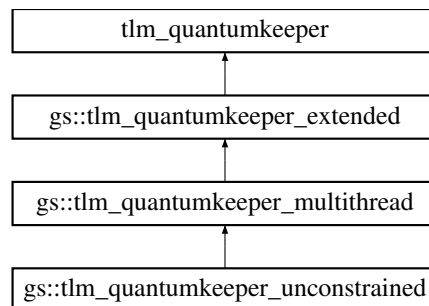          tlm_quantumkeeper
                 ↑
     gs::tlm_quantumkeeper_extended
                 ↑
    gs::tlm_quantumkeeper_multithread
                 ↑
   gs::tlm_quantumkeeper_unconstrained
```

Additional Inherited Members

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

- /home/thomas/Documents/GreenSocs/build-lib/libgssync/include/greensocs/libgssync/qkmulti-unconstrained.↩
  h

Index