SyoSil ApS UVM Scoreboard

1.0.2.3

Generated by Doxygen 1.6.1

Mon Sep 28 02:57:58 2015

Contents

Chapter 1

Main Page

User and implementation documentation for the UVM scoreboard This documentation provides the following additional documentation, besides the normal source code documentation:

1. Getting started: **Getting started** (p. 3)
2. How to integrate the UVM scoreboard: **How to integrate the UVM scoreboard** (p. 5)
3. Implementation notes: **Implementation notes** (p. 11)

It is assumed that the reader is familiar with the UVM scoreboard architecture described in the SyoSil paper on the subject: Versatile UVM Scoreboarding located in in the **docs** directory.

Chapter 2

Getting started

This software package also provides some simple examples beside the source code for the UVM scoreboard.

Before starting to integrate the UVM scoreboard into your own code then it might be beneficial to look at the provided examples. An example testbench is placed in the **tb** directory and the tests are in the **tb/test** directory.

To run the examples you need to select a Vendor since the examples can be run with all of the three major SystemVerilog simulator vendors: Mentor Graphics, Cadence and Synopsys. See **README.txt** for a description of how to select the vendor.

Once the vendor has been selected then the available Make targets for that vendor can be listed by typing: "make". Typically, you run the simulation with: **make sim**.

In general you can type: **make help** to get information about what Make options are available.

Chapter 3

How to integrate the UVM scoreboard

The UVM scoreboard is easily integrated into your existing testbench environment.

3.1 Compiling the UVM scoreboard

To get the UVM scoreboard compiled you need to add **src/pk_syoscb.sv (p. ??)** to your list of files that are complied when compiling your testbench. How this is done is highly dependent on the verification environment since some environments compile everything into different libraries and some do not etc.

3.2 Accessing the UVM scoreboard from your own code

Once the UVM scoreboard is compiled with the verification environment then it is accessible either by explicit scoping:

```
class myclass;
  pk_syoscb::cl_syoscb my_new_scb;
  ...
```

or by importing the complete package into your scope:

```
import pk_syoscb::*;

class myclass;
  cl_syoscb my_new_scb;
  ...
```

3.3 Instantiating the UVM scoreboard

The UVM scoreboard itself needs to be instantiated along with the configuration object. The simplest way to to this is to add the UVM scoreboard and the configuration object to the UVM environment - note that the configuration object is passed to the scoreboard via the config_db:

```
import pk_syoscb::*;

class cl_scbtest_env extends uvm_env;

  cl_syoscb     syoscb;
  cl_syoscb_cfg syoscb_cfg;

  `uvm_component_utils_begin(cl_scbtest_env)
    `uvm_field_object(syoscb,     UVM_ALL_ON)
    `uvm_field_object(syoscb_cfg, UVM_ALL_ON)
  `uvm_component_utils_end

  ...

endclass: cl_scbtest_env

function void cl_scbtest_env::build_phase(uvm_phase phase);
  super.build_phase(phase);

  // Create the scoreboard configuration object
  this.syoscb_cfg = cl_syoscb_cfg::type_id::create("syoscb_cfg");

  // Pass the scoreboard configuration object to the config_db
```

```
    uvm_config_db #(cl_syoscb_cfg)::set(this, "syoscb", "cfg", this.syoscb_cfg);


    // Create the scoreboard
    this.syoscb = cl_syoscb::type_id::create("syoscb", this);

    ...

  endfunction: build_phase
```

3.4 Configuring the UVM scoreboard

The UVM scoreboard configuration object needs to be configured after it has been created. The following example shows how two queues Q1 and Q2 wit Q1 as the primary queue. Furthermore, one producer P1 is added to both queues:

```
  function void cl_scbtest_env::build_phase(uvm_phase phase);
    super.build_phase(phase);

    // Create the scoreboard configuration object
    this.syoscb_cfg = cl_syoscb_cfg::type_id::create("syoscb_cfg");

    // Configure the scoreboard
    this.syoscb_cfg.set_queues({"Q1", "Q2"});
    void'(this.syoscb_cfg.set_primary_queue("Q1"));
    void'(this.syoscb_cfg.set_producer("P1", {"Q1", "Q2"}));

    // Pass the scoreboard configuration object to the config_db
    uvm_config_db #(cl_syoscb_cfg)::set(this, "syoscb", "cfg", this.syoscb_cfg);


    // Create the scoreboard
    this.syoscb = cl_syoscb::type_id::create("syoscb", this);

    ...

  endfunction: build_phase
```

3.5 Function based API hook up

The function based API is very easy to use once you have done the configuration and instantiation of the scoreboard as describe above.

Whenever you need to add an UVM sequence item to a queue produced by a specified producer then you simply invoke the **cl_syoscb::add_item()** (p. 22) method:

```
  // *NOTE*: Assumes syoscb is handle to an instance of the scoreboard and
  //         item1 is a handle to a UVM sequence item

  ...

  // Insert UVM sequence item for queue: Q1, for producer: P1
  syoscb.add_item("Q1", "P1", item1);
```

Invoking the **cl_syoscb::add_item()** (p. 22) method will simply wrap the UVM sequence item in a **cl_syoscb_item** (p. 35) object, add it the correct queue and finally invoke the configured compare method.

The UVM environment will typically contain a handle to the scoreboard as described above. This can then be utilized if UVM sequences needs to be added from a test case:

```
class cl_scbtest_seq_item extends uvm_sequence_item;
  //-------------------------------------
  // Randomizable variables
  //-------------------------------------
  rand int unsigned int_a;

  //-------------------------------------
  // UVM Macros
  //-------------------------------------
  `uvm_object_utils_begin(cl_scbtest_seq_item)
    `uvm_field_int(int_a, UVM_ALL_ON)
  `uvm_object_utils_end

  //-------------------------------------
  // Constructor
  //-------------------------------------
  function cl_scbtest_seq_item::new (string name = "cl_scbtest_seq_item");
     super.new(name);
  endfunction
endclass: cl_scbtest_seq_item

class cl_scbtest_test extends uvm_test;
  //-------------------------------------
  // Non randomizable variables
  //-------------------------------------
  cl_scbtest_env scbtest_env;

  //-------------------------------------
  // UVM Macros
  //-------------------------------------
  `uvm_component_utils(cl_scbtest_test)

  //-------------------------------------
  // Constructor
  //-------------------------------------
  function new(string name = "cl_scbtest_test", uvm_component parent = null);
    super.new(name, parent);
  endfunction: new

  //-------------------------------------
  // UVM Phase methods
  //-------------------------------------
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    scbtest_env = cl_scbtest_env::type_id::create("scbtest_env", this);
  endfunction: build_phase

  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    begin
      cl_scbtest_seq_item item1;
      item1 = cl_scbtest_seq_item::type_id::create("item1");
      item1.int_a = 'h3a;
      scbtest_env.syoscb.add_item("Q1", "P1", item1);
    end
    begin
      cl_scbtest_seq_item item1;
      item1 = cl_scbtest_seq_item::type_id::create("item1");
      item1.int_a = 'h3a;
      scbtest_env.syoscb.add_item("Q2", "P1", item1);
    end
  endtask: run_phase
endclass: cl_scbtest_test
```

3.6 TLM based API hook up

The TLM API is even easier to use than the function based API. The scoreboard provides generic UVM subscribers which can be connected to anything which has a UVM analysis port (e.g. a UVM monitor). Typically, the UVM agents inside the UVM environment contain one or more monitors with UVM analysis ports which should be connected to the scoreboard. The following example has two agents which each has a monitor. The monitors are connected to Q1 and Q2 in the scoreboard:

```
import pk_syoscb::*;

class cl_scbtest_env extends uvm_env;

  cl_syoscb     syoscb;
  cl_syoscb_cfg syoscb_cfg;
  myagent       agent1;
  myagent       agent2;

  ...

  function void build_phase(uvm_phase phase);

    ...

    // Configure and create the scoreboard
    // Create and configure the agents

    ...

  endfunction: build_phase

  ...

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);

    begin
      cl_syoscb_subscriber subscriber;

      // Get the subscriber for Producer: P1 for queue: Q1 and connect it
      // to the UVM monitor producing transactions for this queue
      subscriber = this.syoscb.get_subscriber("Q1", "P1");
      this.agent1.mon.<analysis port>.connect(subscriber.analysis_export);

      // Get the subscriber for Producer: P1 for queue: Q2 and connect it
      // to the UVM monitor producing transactions for this queue
      subscriber = this.syoscb.get_subscriber("Q2", "P1");
      this.agent1.mon.<analysis port>.connect(subscriber.analysis_export);
    end
  endfunction: connect_phase
```

3.7 Factory overwrites

Finally, the wanted queue and compare algorithm implementation needs to be selected. This is done by factory overwrites since they can be changed test etc.

NOTE: This MUST be done before creating the scoreboard!

The following queue implementations are available:

1. Standard SV queue (**cl_syoscb_queue_std** (p. 46))

and the following compare algorithms are available:

1. Out-of-Order (cl_syoscb_compare_ooo)
2. In-Order (**cl_syoscb_compare_io** (p. 32))

The following example shows how they are configured:

```
cl_syoscb_queue::set_type_override_by_type(cl_syoscb_queue::get_type(),

                                           cl_syoscb_queue_std::get_type(),
                                           "*");

factory.set_type_override_by_type(cl_syoscb_compare_base::get_type(),
                                  cl_syoscb_compare_ooo::get_type(),
                                  "*");
```

The full build phase, including the factory overwrites, of cl_scbtest_env is shown here for completeness:

```
function void cl_scbtest_env::build_phase(uvm_phase phase);
  super.build_phase(phase);

  // Use the standard SV queue implementation as scoreboard queue
  cl_syoscb_queue::set_type_override_by_type(cl_syoscb_queue::get_type(),

                                             cl_syoscb_queue_std::get_type(),
                                             "*");

  // Set the compare strategy to be OOO
  factory.set_type_override_by_type(cl_syoscb_compare_base::get_type(),
                                    cl_syoscb_compare_ooo::get_type(),
                                    "*");

  // Create the scoreboard configuration object
  this.syoscb_cfg = cl_syoscb_cfg::type_id::create("syoscb_cfg");

  // Configure the scoreboard
  this.syoscb_cfg.set_queues({"Q1", "Q2"});
  void'(this.syoscb_cfg.set_primary_queue("Q1"));
  void'(this.syoscb_cfg.set_producer("P1", {"Q1", "Q2"}));

  // Pass the scoreboard configuration object to the config_db
  uvm_config_db #(cl_syoscb_cfg)::set(this, "syoscb", "cfg", this.syoscb_cfg);


  // Create the scoreboard
  this.syoscb = cl_syoscb::type_id::create("syoscb", this);

  ...

endfunction: build_phase
```

Chapter 4

Implementation notes

4.1 Implementation APIs

The following APIs have been defined for easy extension fo the scoreboard classes:

1. Configuration API: **cl_syoscb_cfg** (p. 23)
2. Item API: **cl_syoscb_item** (p. 35)
3. Queue API: **cl_syoscb_queue** (p. 36)
4. Compare API: **cl_syoscb_compare_base** (p. 30)
5. Subscriber API: **cl_syoscb_subscriber** (p. 49)
6. Iterator API: **cl_syoscb_queue_iterator_base** (p. 39)

4.2 General error handling

In general when a lower level method detects an error then two concepts are used. Primarily, the method will either issue a UVM info with some information about what went wrong or issue a UVM error/fatal immediately. The first one will then return **1'b0** to signal that something went wrong. Thus, it is up to the parent levels to catch the error and convert them into UVM errors/fatals etc. This method was chosen since the parent level typically provides more and better information when things go wrong.

4.3 Error categories

There are several ERROR categories. The following table lists them with some explanation:

Error Category	Description
IMPL_ERROR	Implementation error. Something is really broken
QUEUE_ERROR	A queue related error, e.g. the queue could not be found
CFG_ERROR	Configuration error. Usually, because the configuration object is missing
TYPE_ERROR	Type error. Typically issued when $cast() fails
COMPARE_ERROR	Compare error. Issued, e.g. when the in order compare fails /table>

4.4 Multiple queue references

Both the top level class **cl_syoscb** (p. 21) and the configuration class **cl_syoscb_cfg** (p. 23) contains handles to all queues. The former uses an ordinary array which provides a fast way of looping over the queues and the latter an associative which makes it easy to find a queue using only its name.

Chapter 5

Directory Hierarchy

5.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 6

Class Index

6.1 Class Hierarchy

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

Chapter 7

Class Index

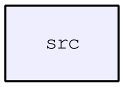
7.1 Class List

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

Chapter 8

Directory Documentation

8.1 /home/jacob/work/uvm_scoreboard/src/ Directory Reference



Files

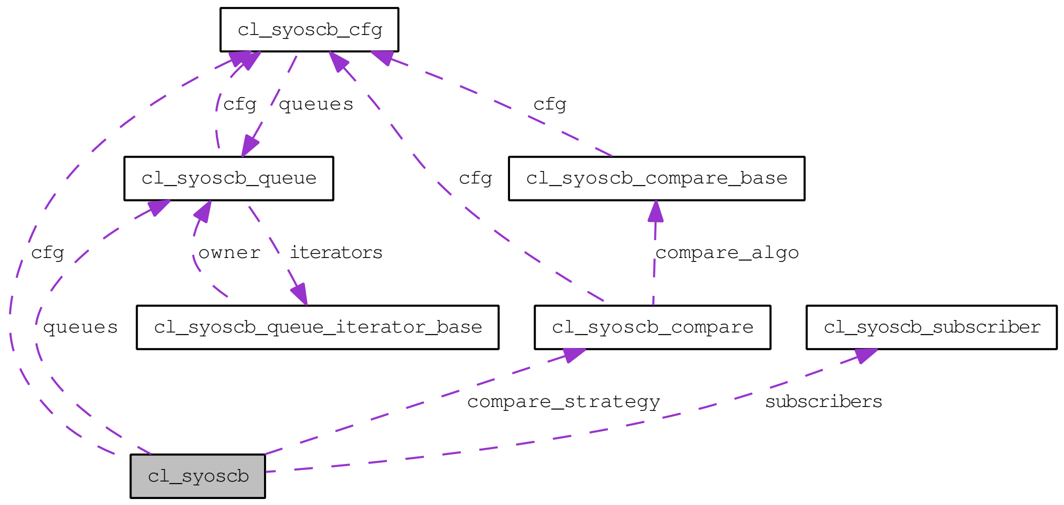
- file **cl_syoscb.svh**
- file **cl_syoscb_cfg.svh**
- file **cl_syoscb_cfg_pl.svh**
- file **cl_syoscb_compare.svh**
- file **cl_syoscb_compare_base.svh**
- file **cl_syoscb_compare_io.svh**
- file **cl_syoscb_compare_ooo.svh**
- file **cl_syoscb_item.svh**
- file **cl_syoscb_queue.svh**
- file **cl_syoscb_queue_iterator_base.svh**
- file **cl_syoscb_queue_iterator_std.svh**
- file **cl_syoscb_queue_std.svh**
- file **cl_syoscb_report_catcher.svh**
- file **cl_syoscb_subscriber.svh**
- file **pk_syoscb.sv**

Chapter 9

Class Documentation

9.1 cl_syoscb Class Reference

Top level class implementing the root of the SyoSil UVM scoreboard. Collaboration diagram for cl_syoscb:



Public Member Functions

- void **build_phase** (uvm_phase phase)

    *The build_phase gets the scoreboard configuration and forwards it to the child components (**cl_-syoscb_queue** (p. 36) and **cl_syoscb_compare** (p. 29)).*

- void **add_item** (string queue_name, string producer, uvm_sequence_item item)

    Method for adding a uvm_sequence_item to a given queue for a given producer.

- void **compare** ()

    Invokes the compare strategy.

- **cl_syoscb_subscriber get_subscriber** (string queue_name, string producer)

  Returns a UVM subscriber for a given combination of queue and producer The retrurned UVM subscriber can then be connected to a UVM monitor or similar which produces transactions which should be scoreboarded.

9.1.1 Detailed Description

Top level class implementing the root of the SyoSil UVM scoreboard.

Definition at line 2 of file cl_syoscb.svh.

9.1.2 Member Function Documentation

9.1.2.1 void cl_syoscb::add_item (string *queue_name*, string *producer*, uvm_sequence_item *item*)

Method for adding a uvm_sequence_item to a given queue for a given producer. The method will check if the queue and producer exists before adding it to the queue.

The uvm_sequence_item will be wrapped by a **cl_syoscb_item** (p. 35) along with some META data Thus, it is the **cl_syoscb_item** (p. 35) which will be added to the queue and not the uvm_sequence_item directly.

This ensures that the scoreboard can easily be added to an existing testbench with already defined sequence items etc.

Definition at line 120 of file cl_syoscb.svh.

9.1.2.2 void cl_syoscb::build_phase (uvm_phase *phase*)

The build_phase gets the scoreboard configuration and forwards it to the child components (**cl_-syoscb_queue** (p. 36) and **cl_syoscb_compare** (p. 29)). Additionally, it creates all of the queues defined in the configuration object. Finally, it also creates the compare strategy via a factory create call.

Definition at line 56 of file cl_syoscb.svh.

9.1.2.3 cl_syoscb_subscriber cl_syoscb::get_subscriber (string *queue_name*, string *producer*)

Returns a UVM subscriber for a given combination of queue and producer The retrurned UVM subscriber can then be connected to a UVM monitor or similar which produces transactions which should be scoreboarded.

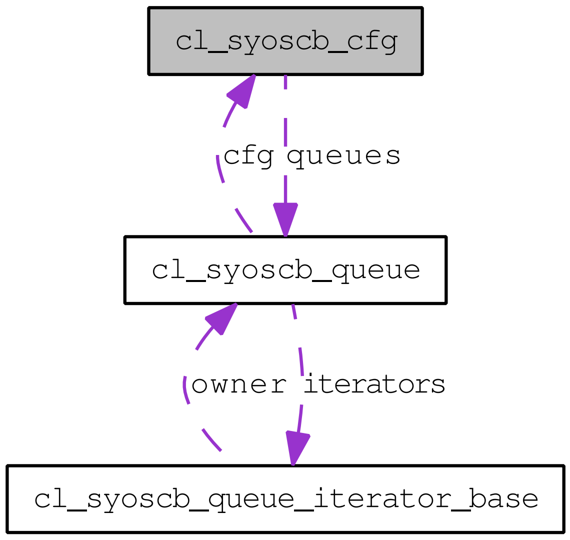
Definition at line 170 of file cl_syoscb.svh.

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

- cl_syoscb.svh

9.2 cl_syoscb_cfg Class Reference

Configuration class for the SyoSil UVM scoreboard. Collaboration diagram for cl_syoscb_cfg:



Public Member Functions

- **cl_syoscb_queue get_queue** (string queue_name)

  Configuration API: *Returns a queue handle for the specificed queue*

- void **set_queue** (string queue_name, **cl_syoscb_queue** queue)

  Configuration API: *Sets the queue object for a given queue*

- void **get_queues** (output string queue_names[ ])

  Configuration API: *Returns all queue names a string list*

- void **set_queues** (string queue_names[ ])

  Configuration API: *Will set the legal queues when provides with a list of queue names.*

- bit **exist_queue** (string queue_name)

  Configuration API: *Returns 1'b0 if the queue does not exist and 1'b1 if it exists*

- int unsigned **size_queues** ()

  Configuration API: *Returns the number of queues*

- cl_syoscb_cfg_pl **get_producer** (string producer)

  Configuration API: *Gets the given producer object for a specified producer*

- bit **set_producer** (string producer, queue_names[ ])

  Configuration API: *Sets the given producer for the listed queues*

- bit **exist_producer** (string producer)

Configuration API: Checks if a given producer exists

- void **get_producers** (output string producers[ ])

  Configuration API: *Returns all producers as string list*

- string **get_primary_queue** ()

  Configuration API: *Gets the primary queue.*

- bit **set_primary_queue** (string primary_queue_name)

  Configuration API: *Sets the primary queue.*

- void **set_disable_clone** (bit dc)

  Configuration API: *Set the value of the disable_clone member variable*

- bit **get_disable_clone** ()

  Configuration API: *Get the value of the disable_clone member variable*

- void **set_max_queue_size** (string queue_name, int unsigned mqs)

  Configuration API: *Set the maximum number of items allowed for a given queue.*

- int unsigned **get_max_queue_size** (string queue_name)

  Configuration API: *Returns the maximum number of allowed items for a given queue.*

9.2.1 Detailed Description

Configuration class for the SyoSil UVM scoreboard.

Definition at line 2 of file cl_syoscb_cfg.svh.

9.2.2 Member Function Documentation

9.2.2.1 int unsigned cl_syoscb_cfg::get_max_queue_size (string *queue_name*)

Configuration API: Returns the maximum number of allowed items for a given queue. 0 (no limit) is default

Definition at line 222 of file cl_syoscb_cfg.svh.

9.2.2.2 string cl_syoscb_cfg::get_primary_queue ()

Configuration API: Gets the primary queue. The primary queue is used by the compare algorithms to select which queue to use as the primary one.

Definition at line 180 of file cl_syoscb_cfg.svh.

9.2.2.3 void cl_syoscb_cfg::set_max_queue_size (string *queue_name*, int unsigned *mqs*)

Configuration API: Set the maximum number of items allowed for a given queue. 0 (no limit) is default

Definition at line 212 of file cl_syoscb_cfg.svh.

9.2.2.4 bit cl_syoscb_cfg::set_primary_queue (string *primary_queue_name*)

Configuration API: Sets the primary queue. The primary queue is used by the compare algorithms to select which queue to use as the primary one.

Definition at line 186 of file cl_syoscb_cfg.svh.

9.2.2.5 void cl_syoscb_cfg::set_queues (string *queue_names*[ ])

Configuration API: Will set the legal queues when provides with a list of queue names. An example could be: set_queues({"Q1", "Q2"}) Will set the max_queue_size for each queue to 0 (no limit) as default

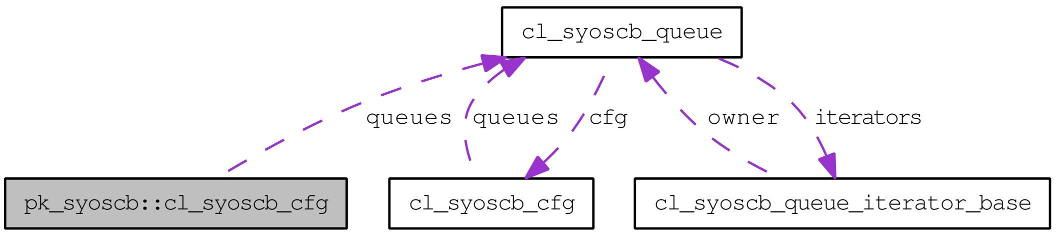
Definition at line 99 of file cl_syoscb_cfg.svh.

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

- cl_syoscb_cfg.svh

9.3 pk_syoscb::cl_syoscb_cfg Class Reference

Configuration class for the SyoSil UVM scoreboard. Collaboration diagram for pk_syoscb::cl_-syoscb_cfg:



Public Member Functions

- **cl_syoscb_queue get_queue** (string queue_name)
  Configuration API: *Returns a queue handle for the specificed queue*
- void **set_queue** (string queue_name, **cl_syoscb_queue** queue)
  Configuration API: *Sets the queue object for a given queue*
- void **get_queues** (output string queue_names[ ])
  Configuration API: *Returns all queue names a string list*
- void **set_queues** (string queue_names[ ])
  Configuration API: *Will set the legal queues when provides with a list of queue names.*
- bit **exist_queue** (string queue_name)
  Configuration API: *Returns 1'b0 if the queue does not exist and 1'b1 if it exists*
- int unsigned **size_queues** ()
  Configuration API: *Returns the number of queues*
- cl_syoscb_cfg_pl **get_producer** (string producer)
  Configuration API: *Gets the given producer object for a specified producer*
- bit **set_producer** (string producer, queue_names[ ])
  Configuration API: *Sets the given producer for the listed queues*
- bit **exist_producer** (string producer)
  Configuration API: *Checks if a given producer exists*
- void **get_producers** (output string producers[ ])
  Configuration API: *Returns all producers as string list*
- string **get_primary_queue** ()
  Configuration API: *Gets the primary queue.*

- bit **set_primary_queue** (string primary_queue_name)
  Configuration API: *Sets the primary queue.*
- void **set_disable_clone** (bit dc)
  Configuration API: *Set the value of the disable_clone member variable*
- bit **get_disable_clone** ()
  Configuration API: *Get the value of the disable_clone member variable*
- void **set_max_queue_size** (string queue_name, int unsigned mqs)
  Configuration API: *Set the maximum number of items allowed for a given queue.*
- int unsigned **get_max_queue_size** (string queue_name)
  Configuration API: *Returns the maximum number of allowed items for a given queue.*

9.3.1 Detailed Description

Configuration class for the SyoSil UVM scoreboard.

Definition at line 415 of file pk_syoscb.sv.

9.3.2 Member Function Documentation

9.3.2.1 int unsigned cl_syoscb_cfg::get_max_queue_size (string *queue_name*)

Configuration API: Returns the maximum number of allowed items for a given queue. 0 (no limit) is default

Definition at line 635 of file pk_syoscb.sv.

9.3.2.2 string cl_syoscb_cfg::get_primary_queue ()

Configuration API: Gets the primary queue. The primary queue is used by the compare algorithms to select which queue to use as the primary one.

Definition at line 593 of file pk_syoscb.sv.

9.3.2.3 void cl_syoscb_cfg::set_max_queue_size (string *queue_name*, int unsigned *mqs*)

Configuration API: Set the maximum number of items allowed for a given queue. 0 (no limit) is default

Definition at line 625 of file pk_syoscb.sv.

9.3.2.4 bit cl_syoscb_cfg::set_primary_queue (string *primary_queue_name*)

Configuration API: Sets the primary queue. The primary queue is used by the compare algorithms to select which queue to use as the primary one.

Definition at line 599 of file pk_syoscb.sv.

9.3.2.5 void cl_syoscb_cfg::set_queues (string *queue_names*[ ])

Configuration API: Will set the legal queues when provides with a list of queue names. An example could be: set_queues({"Q1", "Q2"}) Will set the max_queue_size for each queue to 0 (no limit) as default

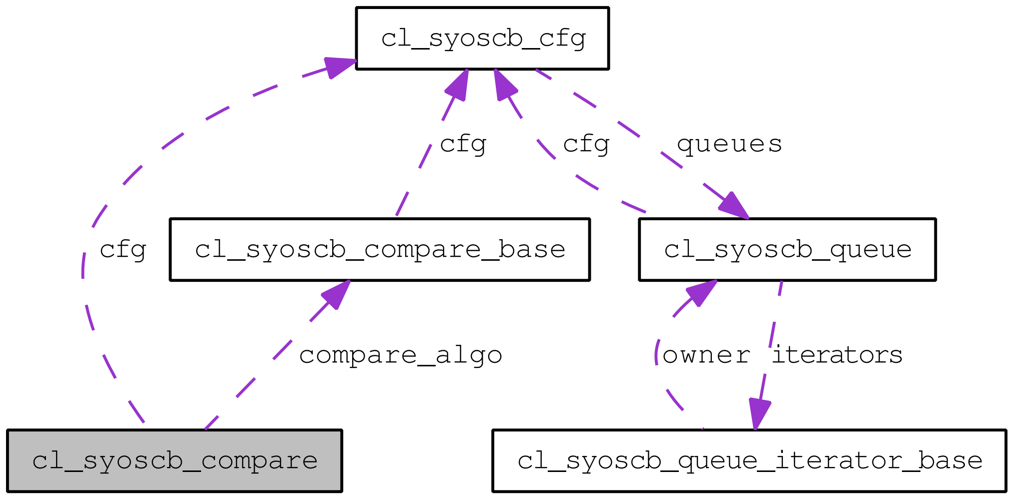
Definition at line 512 of file pk_syoscb.sv.

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

- pk_syoscb.sv

9.4 cl_syoscb_compare Class Reference

Class which act as the root of the compare algorithm. Collaboration diagram for cl_syoscb_-compare:



Public Member Functions

- void **build_phase** (uvm_phase phase)

  Gets the global scoreboard configuration and creates the compare algorithm, e.g. out-of-order.

- void **compare** ()

  Invokes the compare algorithms compare method.

9.4.1 Detailed Description

Class which act as the root of the compare algorithm. It instantiates the chosen compare algorithm.

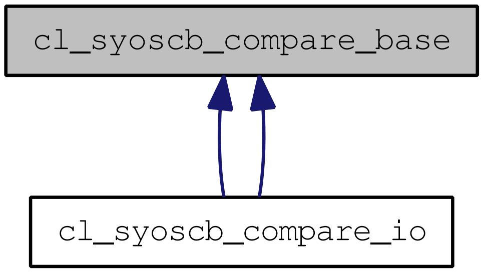
Definition at line 3 of file cl_syoscb_compare.svh.

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

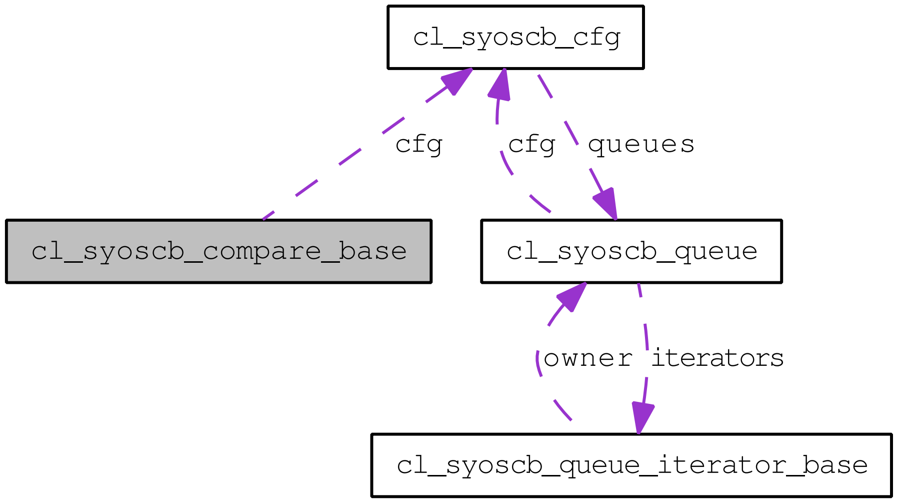
- cl_syoscb_compare.svh

9.5 cl_syoscb_compare_base Class Reference

Base class for all comapre algorithms. Inheritance diagram for cl_syoscb_compare_base:



Collaboration diagram for cl_syoscb_compare_base:



Public Member Functions

- virtual void **compare** ()

  Compare API: *This method is the compare algorithms public compare method.*

- virtual void **compare_do** ()

  Compare API: *Does the actual compare.*

- void **set_cfg** (**cl_syoscb_cfg cfg**)

  Compare API: *Passes the configuration object on to the compare algorithm for faster access.*

- **cl_syoscb_cfg get_cfg** ()

  Compare API: *Returns the configuration object*

- string **get_primary_queue_name** ()

  Compare API: *Gets the primary queue. Convinience method.*

Protected Attributes

- **cl_syoscb_cfg cfg**
  Handle to the configuration.

9.5.1 Detailed Description

Base class for all comapre algorithms.

Definition at line 2 of file cl_syoscb_compare_base.svh.

9.5.2 Member Function Documentation

9.5.2.1 void cl_syoscb_compare_base::compare () `[virtual]`

Compare API: This method is the compare algorithms public compare method. It is called when the compare algorithm is asked to do a compare. Typically, this method is used to check state variables etc. to compte if the compare shall be done or not. If so then do_compare() is called.

NOTE: This method must be implemted.

Reimplemented in **cl_syoscb_compare_io** (p. 33), and **cl_syoscb_compare_io** (p. 33).

Definition at line 39 of file cl_syoscb_compare_base.svh.

9.5.2.2 void cl_syoscb_compare_base::compare_do () `[virtual]`

Compare API: Does the actual compare. **NOTE:** This method must be implemted.

Reimplemented in **cl_syoscb_compare_io** (p. 33), and **cl_syoscb_compare_io** (p. 33).

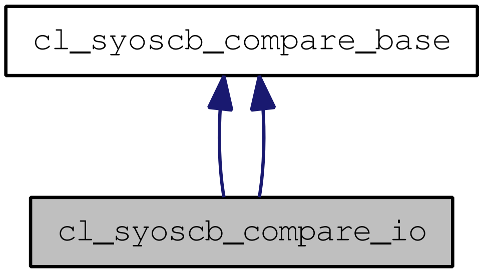
Definition at line 45 of file cl_syoscb_compare_base.svh.

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

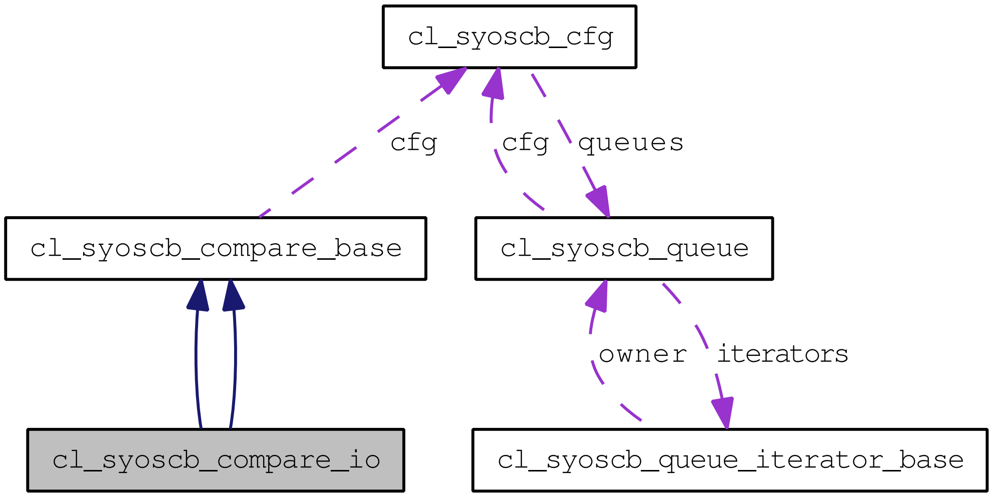
- cl_syoscb_compare_base.svh

9.6 cl_syoscb_compare_io Class Reference

Class which implements the in order compare algorithm. Inheritance diagram for cl_syoscb_-compare_io:



Collaboration diagram for cl_syoscb_compare_io:



Public Member Functions

- virtual void **compare** ()

  ***Compare API**: Mandatory overwriting of the base class' compare method.*

- void **compare_do** ()

  ***Compare API**: Mandatory overwriting of the base class' do_compare method.*

- virtual void **compare** ()

  ***Compare API**: This method is the compare algorithms public compare method.*

- void **compare_do** ()

  ***Compare API**: Does the actual compare.*

9.6.1 Detailed Description

Class which implements the in order compare algorithm.

Definition at line 2 of file cl_syoscb_compare_io.svh.

9.6.2 Member Function Documentation

9.6.2.1 virtual void cl_syoscb_compare_io::compare () `[virtual]`

Compare API: This method is the compare algorithms public compare method. It is called when the compare algorithm is asked to do a compare. Typically, this method is used to check state variables etc. to compte if the compare shall be done or not. If so then do_compare() is called.

NOTE: This method must be implemted.

Reimplemented from **cl_syoscb_compare_base** (p. 31).

9.6.2.2 void cl_syoscb_compare_io::compare () `[virtual]`

Compare API: Mandatory overwriting of the base class' compare method. Currently, this just calls do_copy() blindly

Reimplemented from **cl_syoscb_compare_base** (p. 31).

Definition at line 26 of file cl_syoscb_compare_io.svh.

9.6.2.3 void cl_syoscb_compare_io::compare_do () `[virtual]`

Compare API: Does the actual compare. **NOTE:** This method must be implemted.

Reimplemented from **cl_syoscb_compare_base** (p. 31).

9.6.2.4 void cl_syoscb_compare_io::compare_do () `[virtual]`

Compare API: Mandatory overwriting of the base class' do_compare method. Here the actual in order compare is implemented.

The algorithm gets the primary queue and then loops over all other queues to see if it can find primary item as the first item in all of the other queues. If so then the items are removed from all queues. If not then a UVM error is issued.

Reimplemented from **cl_syoscb_compare_base** (p. 31).

Definition at line 38 of file cl_syoscb_compare_io.svh.

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

- cl_syoscb_compare_io.svh

9.7 pk_syoscb::cl_syoscb_item Class Reference

The UVM scoreboard item.

Public Member Functions

- UVM_DEFAULT uvm_field_object(**item**, UVM_DEFAULT) public string **get_-producer** ()

  Item API: *Returns the producer*

- void **set_producer** (string **producer**)

  Item API: *Sets the producer*

- uvm_sequence_item **get_item** ()

  Item API: *Returns the wrapped uvm_sequence_item*

- void **set_item** (uvm_sequence_item **item**)

  Item API: *Sets the to be wrapped uvm_sequence_item*

Public Attributes

- string **producer**

  Hold the name of the producer.

- uvm_sequence_item **item**

  Handle to the wrapped uvm_sequence_item.

9.7.1 Detailed Description

The UVM scoreboard item. This item wraps the uvm_sequence_items. This ensures that future extensions to the UVM scoreboard will always be able to use all uvm_sqeuence_items from already existing testbenches etc. even though more META data is added to the wrapping item.

Definition at line 646 of file pk_syoscb.sv.

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

- pk_syoscb.sv

9.8 cl_syoscb_item Class Reference

The UVM scoreboard item.

Public Member Functions

- UVM_DEFAULT uvm_field_object(**item**, UVM_DEFAULT) public string **get_-producer** ()

  Item API: *Returns the producer*

- void **set_producer** (string **producer**)

  Item API: *Sets the producer*

- uvm_sequence_item **get_item** ()

  Item API: *Returns the wrapped uvm_sequence_item*

- void **set_item** (uvm_sequence_item **item**)

  Item API: *Sets the to be wrapped uvm_sequence_item*

Public Attributes

- string **producer**

  Hold the name of the producer.

- uvm_sequence_item **item**

  Handle to the wrapped uvm_sequence_item.

9.8.1 Detailed Description

The UVM scoreboard item. This item wraps the uvm_sequence_items. This ensures that future extensions to the UVM scoreboard will always be able to use all uvm_sqeuence_items from already existing testbenches etc. even though more META data is added to the wrapping item.

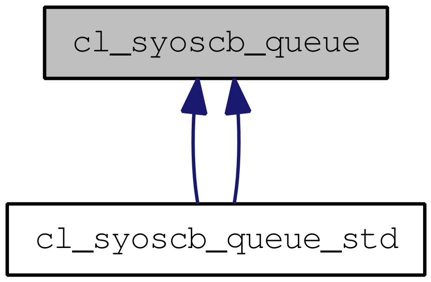
Definition at line 4 of file cl_syoscb_item.svh.

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

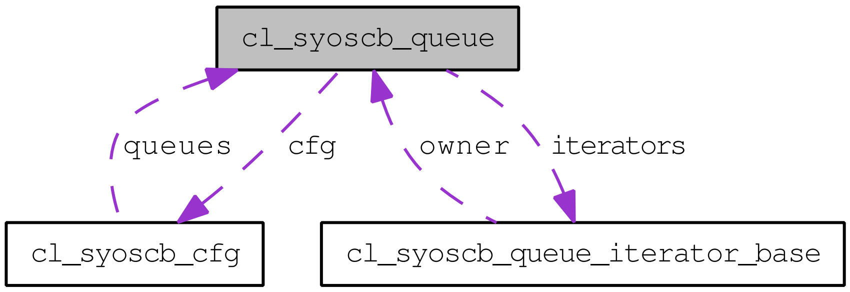
- cl_syoscb_item.svh

9.9 cl_syoscb_queue Class Reference

Class which base concet of a queue. Inheritance diagram for cl_syoscb_queue:



Collaboration diagram for cl_syoscb_queue:



Public Member Functions

- void **build_phase** (uvm_phase phase)
  Gets the global scoreboard configuration.
- void **check_phase** (uvm_phase phase)
  Checks if the queue is empty. If not then a UVM error is issued.
- virtual bit **add_item** (string producer, uvm_sequence_item item)
  ***Queue API:** Adds an uvm_sequence_item.*
- virtual bit **delete_item** (int unsigned idx)
  ***Queue API:** Deletes the item at index idx from the queue*
- virtual **cl_syoscb_item get_item** (int unsigned idx)
  ***Queue API:** Gets the item at index idx from the queue*
- virtual int unsigned **get_size** ()
  ***Queue API:** Returns the current size of the queue*
- virtual bit **empty** ()
  ***Queue API:** Returns whether or not the queue is empty.*

- virtual bit **insert_item** (string producer, uvm_sequence_item item, int unsigned idx)
  Queue API: *Inserts a uvm_sequence_item at index idx.*

- virtual **cl_syoscb_queue_iterator_base create_iterator** ()
  Queue API: *Creates an iterator for this queue.*

- virtual bit **delete_iterator** (**cl_syoscb_queue_iterator_base** iterator)
  Queue API: *Deletes a given iterator for this queue.*

Protected Attributes

- **cl_syoscb_cfg cfg**
  Handle to the configuration.

- **cl_syoscb_queue_iterator_base iterators** [**cl_syoscb_queue_iterator_base**]
  List of iterators registered with queue.

- int unsigned **iter_idx**
  Current number of iterators.

- semaphore **iter_sem**
  Semaphore guarding exclusive access to the queue when multiple iterators are in play.

9.9.1 Detailed Description

Class which base concet of a queue. All queues must extend this class and implement the queue API.

Definition at line 3 of file cl_syoscb_queue.svh.

9.9.2 Member Function Documentation

9.9.2.1 bit cl_syoscb_queue::add_item (string *producer*, uvm_sequence_item *item*) `[virtual]`

Queue API: Adds an uvm_sequence_item. The implementation must wrap this in a **cl_-syoscb_item** (p. 35) object before the item is inserted

Reimplemented in **cl_syoscb_queue_std** (p. 46), and **cl_syoscb_queue_std** (p. 47).

Definition at line 84 of file cl_syoscb_queue.svh.

9.9.2.2 bit cl_syoscb_queue::empty () `[virtual]`

Queue API: Returns whether or not the queue is empty. 1'b0 means thet te queue is not empty. 1'b1 means that the queue is empty

Reimplemented in **cl_syoscb_queue_std** (p. 46), and **cl_syoscb_queue_std** (p. 47).

Definition at line 109 of file cl_syoscb_queue.svh.

9.9.2.3 bit cl_syoscb_queue::insert_item (string *producer*, uvm_sequence_item *item*, int unsigned *idx*) [virtual]

Queue API: Inserts a uvm_sequence_item at index idx. The implementation must wrap the uvm_sequence_item in a **cl_syoscb_item** (p. 35) before it is inserted.

Reimplemented in **cl_syoscb_queue_std** (p. 46), and **cl_syoscb_queue_std** (p. 48).

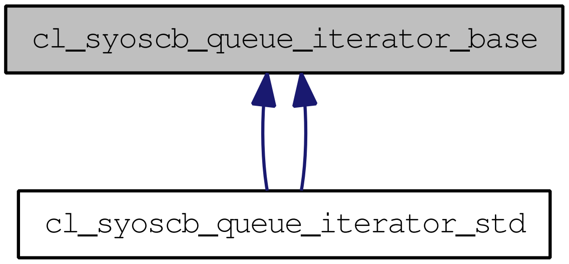
Definition at line 116 of file cl_syoscb_queue.svh.

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

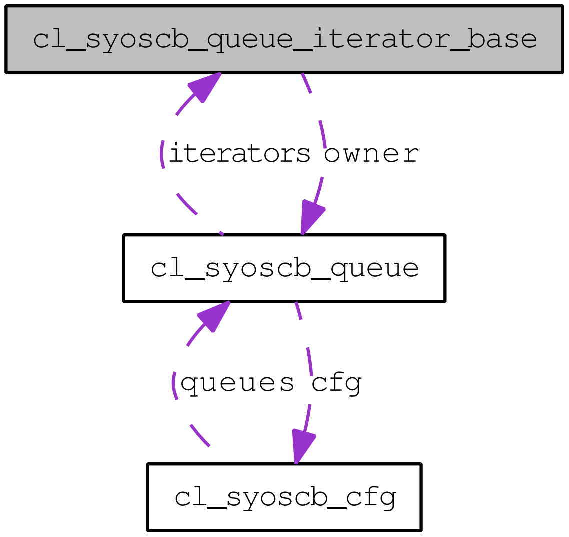
- cl_syoscb_queue.svh

9.10 cl_syoscb_queue_iterator_base Class Reference

Queue iterator base class defining the iterator API used for iterating queues. Inheritance diagram for cl_syoscb_queue_iterator_base:



Collaboration diagram for cl_syoscb_queue_iterator_base:



Public Member Functions

- virtual bit **previous** ()

  Iterator API: *Moves the iterator to the previous item in the queue.*

- virtual bit **first** ()

  Iterator API: *Moves the iterator to the first item in the queue.*

- virtual bit **last** ()

  Iterator API: *Moves the iterator to the last item in the queue.*

- virtual int unsigned **get_idx** ()

  Iterator API: *Returns the current index*

- virtual **cl_syoscb_item get_item** ()

  Iterator API: *Returns the current* ***cl_syoscb_item*** *(p. 35) object at the current index*

- virtual bit **is_done** ()

  Iterator API: *Returns 1'b0 as long as the iterator has not reached the end.*

- protected **cl_syoscb_queue get_queue** ()

  Iterator API: *Returns releated queue*

- virtual bit **set_queue** (**cl_syoscb_queue owner**)

  Iterator API: *Sets releated queue*

Protected Attributes

- **cl_syoscb_queue owner**

  The owner of this iterator.

- int unsigned **position** = 0

  Current position in the queue.

9.10.1 Detailed Description

Queue iterator base class defining the iterator API used for iterating queues.

Definition at line 2 of file cl_syoscb_queue_iterator_base.svh.

9.10.2 Member Function Documentation

9.10.2.1 bit cl_syoscb_queue_iterator_base::first () `[virtual]`

Iterator API: Moves the iterator to the first item in the queue. It shall return 1'b0 if there is no first item (Queue is empty).

Reimplemented in **cl_syoscb_queue_iterator_std** (p. 43), and **cl_syoscb_queue_-iterator_std** (p. 44).

Definition at line 56 of file cl_syoscb_queue_iterator_base.svh.

9.10.2.2 bit cl_syoscb_queue_iterator_base::is_done () `[virtual]`

Iterator API: Returns 1'b0 as long as the iterator has not reached the end. When the iterator has reached the end then it returns 1'b1.

Reimplemented in **cl_syoscb_queue_iterator_std** (p. 44), and **cl_syoscb_queue_-iterator_std** (p. 44).

Definition at line 82 of file cl_syoscb_queue_iterator_base.svh.

9.10.2.3 bit cl_syoscb_queue_iterator_base::last () `[virtual]`

Iterator API: Moves the iterator to the last item in the queue. It shall return 1'b0 if there is no last item (Queue is empty).

Reimplemented in **cl_syoscb_queue_iterator_std** (p. 45), and **cl_syoscb_queue_-iterator_std** (p. 45).

Definition at line 63 of file cl_syoscb_queue_iterator_base.svh.

9.10.2.4 bit cl_syoscb_queue_iterator_base::previous () `[virtual]`

Iterator API: Moves the iterator to the previous item in the queue. It shall return 1'b0 if there is no previous item, e.g. when it is either empty or the iterator has reached the very beginning of the queue.

Reimplemented in **cl_syoscb_queue_iterator_std** (p. 43), and **cl_syoscb_queue_-iterator_std** (p. 45).

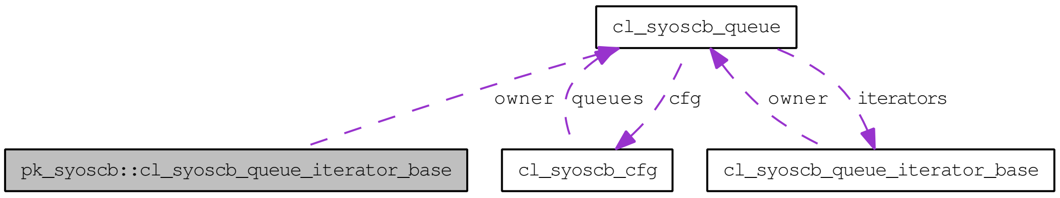
Definition at line 49 of file cl_syoscb_queue_iterator_base.svh.

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

- cl_syoscb_queue_iterator_base.svh

9.11 pk_syoscb::cl_syoscb_queue_iterator_base Class Reference

Queue iterator base class defining the iterator API used for iterating queues. Collaboration diagram for pk_syoscb::cl_syoscb_queue_iterator_base:


cl_syoscb_queue
owner
queues
cfg
owner
iterators
pk_syoscb::cl_syoscb_queue_iterator_base
cl_syoscb_cfg
cl_syoscb_queue_iterator_base

Protected Attributes

- **cl_syoscb_queue owner**
  The owner of this iterator.
- int unsigned **position** = 0
  Current position in the queue.

9.11.1 Detailed Description

Queue iterator base class defining the iterator API used for iterating queues.

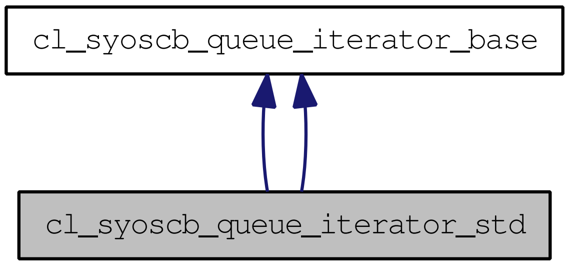
Definition at line 737 of file pk_syoscb.sv.

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

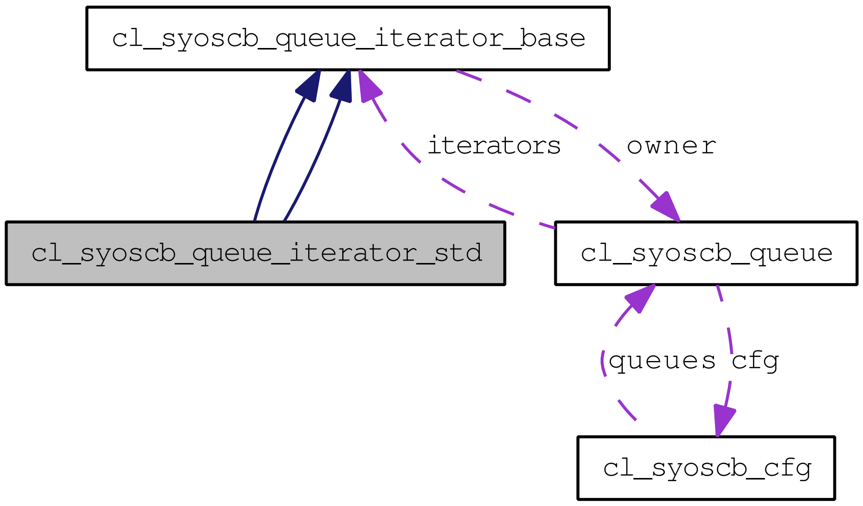
- pk_syoscb.sv

9.12 cl_syoscb_queue_iterator_std Class Reference

Queue iterator class defining the iterator API used for iterating std queues. Inheritance diagram for cl_syoscb_queue_iterator_std:



Collaboration diagram for cl_syoscb_queue_iterator_std:



Public Member Functions

- virtual bit **previous** ()

  Iterator API: *See **cl_syoscb_queue_iterator_base** (p. 39) for details*

- virtual bit **first** ()

  Iterator API: *See **cl_syoscb_queue_iterator_base** (p. 39) for details*

- virtual bit **last** ()

  Iterator API: *Moves the iterator to the last item in the queue.*

- virtual int unsigned **get_idx** ()

  Iterator API: *See **cl_syoscb_queue_iterator_base** (p. 39) for details*

- virtual **cl_syoscb_item get_item** ()

*Iterator API: See **cl_syoscb_queue_iterator_base** (p. 39) for details*

- virtual bit **is_done** ()

  *Iterator API: See **cl_syoscb_queue_iterator_base** (p. 39) for details*

- virtual bit **set_queue** (**cl_syoscb_queue owner**)

  *Iterator API: See **cl_syoscb_queue_iterator_base** (p. 39) for details*

- virtual bit **previous** ()

  Iterator API: Moves the iterator to the previous item in the queue.

- virtual bit **first** ()

  Iterator API: Moves the iterator to the first item in the queue.

- virtual bit **last** ()

  Iterator API: Moves the iterator to the last item in the queue.

- virtual int unsigned **get_idx** ()

  Iterator API: Returns the current index

- virtual **cl_syoscb_item get_item** ()

  *Iterator API: Returns the current **cl_syoscb_item** (p. 35) object at the current index*

- virtual bit **is_done** ()

  Iterator API: Returns 1'b0 as long as the iterator has not reached the end.

- virtual bit **set_queue** (**cl_syoscb_queue owner**)

  Iterator API: Sets releated queue

9.12.1 Detailed Description

Queue iterator class defining the iterator API used for iterating std queues.

Definition at line 2 of file cl_syoscb_queue_iterator_std.svh.

9.12.2 Member Function Documentation

9.12.2.1 virtual bit cl_syoscb_queue_iterator_std::first () `[virtual]`

Iterator API: Moves the iterator to the first item in the queue. It shall return 1'b0 if there is no first item (Queue is empty).

Reimplemented from **cl_syoscb_queue_iterator_base** (p. 40).

9.12.2.2 virtual bit cl_syoscb_queue_iterator_std::is_done () `[virtual]`

Iterator API: Returns 1'b0 as long as the iterator has not reached the end. When the iterator has reached the end then it returns 1'b1.

Reimplemented from **cl_syoscb_queue_iterator_base** (p. 40).

9.12.2.3 virtual bit cl_syoscb_queue_iterator_std::last () [virtual]

Iterator API: Moves the iterator to the last item in the queue. It shall return 1'b0 if there is no last item (Queue is empty).

Reimplemented from **cl_syoscb_queue_iterator_base** (p. 41).

9.12.2.4 bit cl_syoscb_queue_iterator_std::last () [virtual]

Iterator API: Moves the iterator to the last item in the queue. It shall return 1'b0 if there is no last item (Queue is empty).

Reimplemented from **cl_syoscb_queue_iterator_base** (p. 41).

Definition at line 62 of file cl_syoscb_queue_iterator_std.svh.

9.12.2.5 virtual bit cl_syoscb_queue_iterator_std::previous () [virtual]

Iterator API: Moves the iterator to the previous item in the queue. It shall return 1'b0 if there is no previous item, e.g. when it is either empty or the iterator has reached the very beginning of the queue.

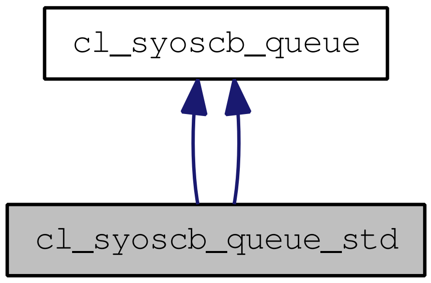
Reimplemented from **cl_syoscb_queue_iterator_base** (p. 41).

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

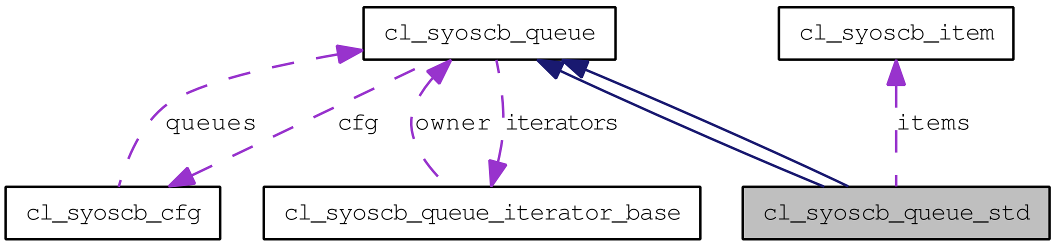
- cl_syoscb_queue_iterator_std.svh

9.13 cl_syoscb_queue_std Class Reference

Standard implementation of a queue. Inheritance diagram for cl_syoscb_queue_std:



Collaboration diagram for cl_syoscb_queue_std:



Public Member Functions

- virtual bit **add_item** (string producer, uvm_sequence_item item)
  Queue API: *See* ***cl_syoscb_queue*** (p. *36) for more details*
- virtual bit **delete_item** (int unsigned idx)
  Queue API: *See* ***cl_syoscb_queue*** (p. *36) for more details*
- virtual **cl_syoscb_item get_item** (int unsigned idx)
  Queue API: *See* ***cl_syoscb_queue*** (p. *36) for more details*
- virtual int unsigned **get_size** ()
  Queue API: *See* ***cl_syoscb_queue*** (p. *36) for more details*
- virtual bit **empty** ()
  Queue API: *See* ***cl_syoscb_queue*** (p. *36) for more details*
- virtual bit **insert_item** (string producer, uvm_sequence_item item, int unsigned idx)
  Queue API: *See* ***cl_syoscb_queue*** (p. *36) for more details*
- virtual **cl_syoscb_queue_iterator_base create_iterator** ()
  Queue API: *See* ***cl_syoscb_queue*** (p. *36) for more details*

- virtual bit **delete_iterator** (**cl_syoscb_queue_iterator_base** iterator)

  Queue API: *See* ***cl_syoscb_queue*** *(p. 36) for more details*

- virtual bit **add_item** (string producer, uvm_sequence_item item)

  Queue API: *Adds an uvm_sequence_item.*

- virtual bit **delete_item** (int unsigned idx)

  Queue API: *Deletes the item at index idx from the queue*

- virtual **cl_syoscb_item get_item** (int unsigned idx)

  Queue API: *Gets the item at index idx from the queue*

- virtual int unsigned **get_size** ()

  Queue API: *Returns the current size of the queue*

- virtual bit **empty** ()

  Queue API: *Returns whether or not the queue is empty.*

- virtual bit **insert_item** (string producer, uvm_sequence_item item, int unsigned idx)

  Queue API: *Inserts a uvm_sequence_item at index idx.*

- virtual **cl_syoscb_queue_iterator_base create_iterator** ()

  Queue API: *Creates an iterator for this queue.*

- virtual bit **delete_iterator** (**cl_syoscb_queue_iterator_base** iterator)

  Queue API: *Deletes a given iterator for this queue.*

9.13.1 Detailed Description

Standard implementation of a queue. Uses a normal SystemVerilog queue as implementation. The class implements the queue API as defined by the queue base class.

Definition at line 4 of file cl_syoscb_queue_std.svh.

9.13.2 Member Function Documentation

9.13.2.1 virtual bit cl_syoscb_queue_std::add_item (string *producer*, uvm_sequence_item *item*) `[virtual]`

Queue API: Adds an uvm_sequence_item. The implementation must wrap this in a **cl_-syoscb_item** (p. 35) object before the item is inserted

Reimplemented from **cl_syoscb_queue** (p. 37).

9.13.2.2 virtual bit cl_syoscb_queue_std::empty () `[virtual]`

Queue API: Returns whether or not the queue is empty. 1'b0 means thet te queue is not empty. 1'b1 means that the queue is empty

Reimplemented from **cl_syoscb_queue** (p. 37).

9.13.2.3 virtual bit cl_syoscb_queue_std::insert_item (string *producer*, uvm_sequence_item *item*, int unsigned *idx*) `[virtual]`

Queue API: Inserts a uvm_sequence_item at index idx. The implementation must wrap the uvm_sequence_item in a **cl_syoscb_item** (p. 35) before it is inserted.

Reimplemented from **cl_syoscb_queue** (p. 38).

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

- cl_syoscb_queue_std.svh

9.14 cl_syoscb_subscriber Class Reference

Generic subscriber for the scoreboard.

Public Member Functions

- void **write** (uvm_sequence_item t)

  The write method which must be implemented when extening uvm_subscriber.

- string **get_queue_name** ()

  ***Subscriber API**: Returns the name of the queue which this subscriber is connected to.*

- void **set_queue_name** (string qn)

  ***Subscriber API**: Sets the name of the queue which this subscriber is connected to.*

- string **get_producer** ()

  ***Subscriber API**: Returns the name of the produer which this subscriber is connected to.*

- void **set_producer** (string p)

  ***Subscriber API**: Sets the name of the producer which this subscriber is connected to.*

9.14.1 Detailed Description

Generic subscriber for the scoreboard. It provides the write method for UVM monitors and utilizes the function based API of the scb to insert the items received through the write method.

Definition at line 4 of file cl_syoscb_subscriber.svh.

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

- cl_syoscb_subscriber.svh

Index