

# SyoSil ApS UVM Scoreboard

1.0.2.1

Generated by Doxygen 1.6.1

Thu Jun 4 23:02:22 2015



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Getting started</b>	<b>3</b>
<b>3</b>	<b>How to integrate the UVM scoreboard</b>	<b>5</b>
3.1	Compiling the UVM scoreboard . . . . .	6
3.2	Accessing the UVM scoreboard from your own code . . . . .	6
3.3	Instantiating the UVM scoreboard . . . . .	6
3.4	Configuring the UVM scoreboard . . . . .	7
3.5	Function based API hook up . . . . .	7
3.6	TLM based API hook up . . . . .	9
3.7	Factory overwrites . . . . .	9
<b>4</b>	<b>Implementation notes</b>	<b>11</b>
4.1	General error handling . . . . .	12
4.2	Error categories . . . . .	12
4.3	Multiple queue references . . . . .	12
<b>5</b>	<b>Directory Hierarchy</b>	<b>13</b>
5.1	Directories . . . . .	13
<b>6</b>	<b>Class Index</b>	<b>15</b>
6.1	Class Hierarchy . . . . .	15
<b>7</b>	<b>Class Index</b>	<b>17</b>
7.1	Class List . . . . .	17
<b>8</b>	<b>Directory Documentation</b>	<b>19</b>
8.1	/home/jacob/work/uvm_scoreboard/src/ Directory Reference . . . . .	19
<b>9</b>	<b>Class Documentation</b>	<b>21</b>

9.1	cl_syoscb Class Reference . . . . .	21
9.1.1	Detailed Description . . . . .	22
9.1.2	Member Function Documentation . . . . .	22
9.1.2.1	add_item . . . . .	22
9.1.2.2	build_phase . . . . .	22
9.1.2.3	get_subscriber . . . . .	22
9.2	cl_syoscb_cfg Class Reference . . . . .	23
9.2.1	Detailed Description . . . . .	24
9.2.2	Member Function Documentation . . . . .	24
9.2.2.1	get_max_queue_size . . . . .	24
9.2.2.2	get_primary_queue . . . . .	24
9.2.2.3	set_max_queue_size . . . . .	24
9.2.2.4	set_primary_queue . . . . .	25
9.2.2.5	set_queues . . . . .	25
9.3	pk_syoscb::cl_syoscb_cfg Class Reference . . . . .	26
9.3.1	Detailed Description . . . . .	27
9.3.2	Member Function Documentation . . . . .	27
9.3.2.1	get_max_queue_size . . . . .	27
9.3.2.2	get_primary_queue . . . . .	27
9.3.2.3	set_max_queue_size . . . . .	27
9.3.2.4	set_primary_queue . . . . .	27
9.3.2.5	set_queues . . . . .	28
9.4	cl_syoscb_compare Class Reference . . . . .	29
9.4.1	Detailed Description . . . . .	29
9.5	cl_syoscb_compare_base Class Reference . . . . .	30
9.5.1	Detailed Description . . . . .	31
9.5.2	Member Function Documentation . . . . .	31
9.5.2.1	compare . . . . .	31
9.5.2.2	compare_do . . . . .	31
9.6	cl_syoscb_compare_io Class Reference . . . . .	32
9.6.1	Detailed Description . . . . .	33
9.6.2	Member Function Documentation . . . . .	33
9.6.2.1	compare . . . . .	33
9.6.2.2	compare . . . . .	33
9.6.2.3	compare_do . . . . .	33
9.6.2.4	compare_do . . . . .	33

9.7	pk_syoscb::cl_syoscb_item Class Reference . . . . .	34
9.7.1	Detailed Description . . . . .	34
9.8	cl_syoscb_item Class Reference . . . . .	35
9.8.1	Detailed Description . . . . .	35
9.9	cl_syoscb_queue Class Reference . . . . .	36
9.9.1	Detailed Description . . . . .	37
9.9.2	Member Function Documentation . . . . .	37
9.9.2.1	add_item . . . . .	37
9.9.2.2	empty . . . . .	37
9.9.2.3	insert_item . . . . .	38
9.10	cl_syoscb_queue_iterator_base Class Reference . . . . .	39
9.10.1	Detailed Description . . . . .	40
9.10.2	Member Function Documentation . . . . .	40
9.10.2.1	first . . . . .	40
9.10.2.2	is_done . . . . .	40
9.10.2.3	last . . . . .	41
9.10.2.4	previous . . . . .	41
9.11	pk_syoscb::cl_syoscb_queue_iterator_base Class Reference . . . . .	42
9.11.1	Detailed Description . . . . .	42
9.12	cl_syoscb_queue_iterator_std Class Reference . . . . .	43
9.12.1	Detailed Description . . . . .	44
9.12.2	Member Function Documentation . . . . .	44
9.12.2.1	first . . . . .	44
9.12.2.2	is_done . . . . .	44
9.12.2.3	last . . . . .	45
9.12.2.4	last . . . . .	45
9.12.2.5	previous . . . . .	45
9.13	cl_syoscb_queue_std Class Reference . . . . .	46
9.13.1	Detailed Description . . . . .	47
9.13.2	Member Function Documentation . . . . .	47
9.13.2.1	add_item . . . . .	47
9.13.2.2	empty . . . . .	47
9.13.2.3	insert_item . . . . .	48
9.14	cl_syoscb_subscriber Class Reference . . . . .	49
9.14.1	Detailed Description . . . . .	49



# 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**.

## 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 compiled 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;
    ...
endclass
```

or by importing the complete package into your scope:

```
import pk_syoscb::*;

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

### 3.3 Instantiating the UVM scoreboard

The UVM scoreboard itself needs to be instantiated along with the configuration object. The simplest way to do 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_scctest_env extends uvm_env;

    cl_syoscb      syoscb;
    cl_syoscb_cfg  syoscb_cfg;

    'uvm_component_utils_begin(cl_scctest_env)
        'uvm_field_object(syoscb,      UVM_ALL_ON)
        'uvm_field_object(syoscb_cfg, UVM_ALL_ON)
    'uvm_component_utils_end

    ...

endclass: cl_scctest_env

function void cl_scctest_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
endfunction
```

```

    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 000
    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 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.

## 4.2 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

## 4.3 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:

src . . . . .	19
---------------	----



# Chapter 6

## Class Index

### 6.1 Class Hierarchy

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

cl_syoscb . . . . .	21
cl_syoscb_cfg . . . . .	23
pk_syoscb::cl_syoscb_cfg . . . . .	26
cl_syoscb_compare . . . . .	29
cl_syoscb_compare_base . . . . .	30
cl_syoscb_compare_io . . . . .	32
cl_syoscb_compare_io . . . . .	32
pk_syoscb::cl_syoscb_item . . . . .	34
cl_syoscb_item . . . . .	35
cl_syoscb_queue . . . . .	36
cl_syoscb_queue_std . . . . .	46
cl_syoscb_queue_std . . . . .	46
cl_syoscb_queue_iterator_base . . . . .	39
cl_syoscb_queue_iterator_std . . . . .	43
cl_syoscb_queue_iterator_std . . . . .	43
pk_syoscb::cl_syoscb_queue_iterator_base . . . . .	42
cl_syoscb_subscriber . . . . .	49



# Chapter 7

## Class Index

### 7.1 Class List

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

<b>cl_syoscb</b> (Top level class implementing the root of the SyoSil UVM scoreboard ) . . .	21
<b>cl_syoscb_cfg</b> (Configuration class for the SyoSil UVM scoreboard ) . . . . .	23
<b>pk_syoscb::cl_syoscb_cfg</b> (Configuration class for the SyoSil UVM scoreboard ) . .	26
<b>cl_syoscb_compare</b> (Class which act as the root of the compare algorithm ) . . . .	29
<b>cl_syoscb_compare_base</b> (Base class for all comapre algorithms ) . . . . .	30
<b>cl_syoscb_compare_io</b> (Class which implements the in order compare algorithm ) .	32
<b>pk_syoscb::cl_syoscb_item</b> (The UVM scoreboard item ) . . . . .	34
<b>cl_syoscb_item</b> (The UVM scoreboard item ) . . . . .	35
<b>cl_syoscb_queue</b> (Class which base concet of a queue ) . . . . .	36
<b>cl_syoscb_queue_iterator_base</b> (Queue iterator base class defining the iterator API used for iterating queues ) . . . . .	39
<b>pk_syoscb::cl_syoscb_queue_iterator_base</b> (Queue iterator base class defining the iterator API used for iterating queues ) . . . . .	42
<b>cl_syoscb_queue_iterator_std</b> (Queue iterator class defining the iterator API used for iterating std queues ) . . . . .	43
<b>cl_syoscb_queue_std</b> (Standard implementation of a queue ) . . . . .	46
<b>cl_syoscb_subscriber</b> (Generic subscriber for the scoreboard ) . . . . .	49

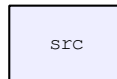




## Chapter 8

# 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`

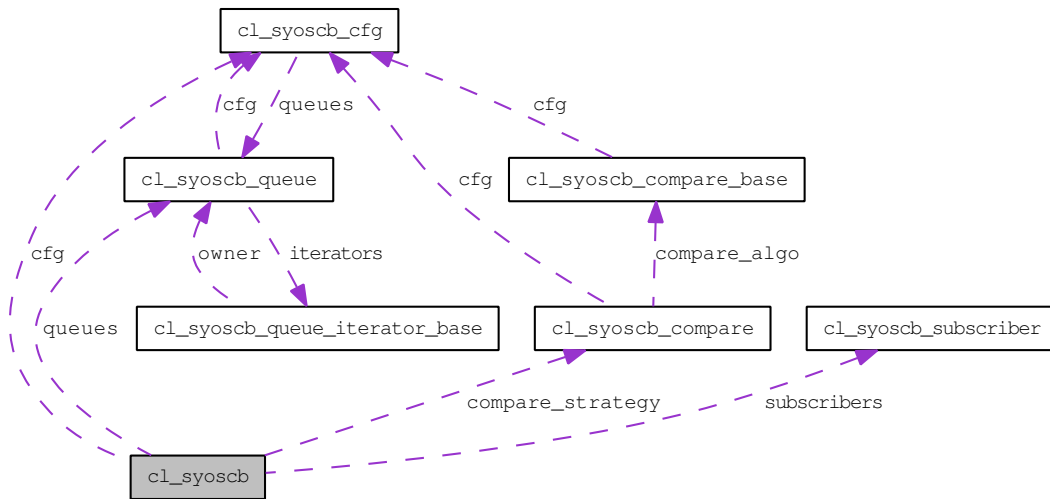


## Chapter 9

# 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 returned 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 returned UVM subscriber can then be connected to a UVM monitor or similar which produces transactions which should be scoreboarded.

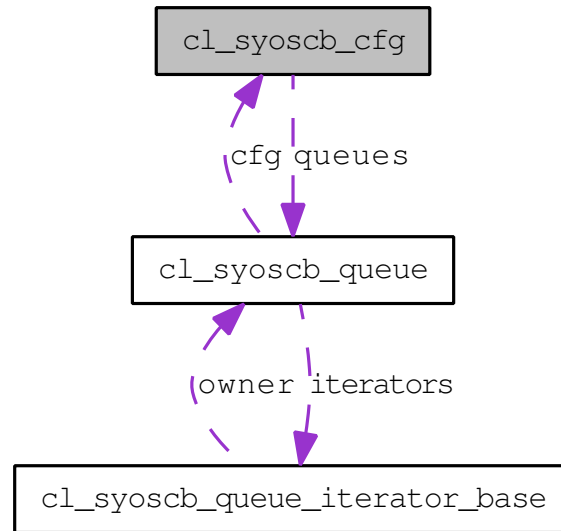
Definition at line 170 of file cl\_syoscb.svh.

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 specified 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 as a string list
- **void set\_queues** (string queue\_names[])  
*Configuration API:* Will set the legal queues when provided 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

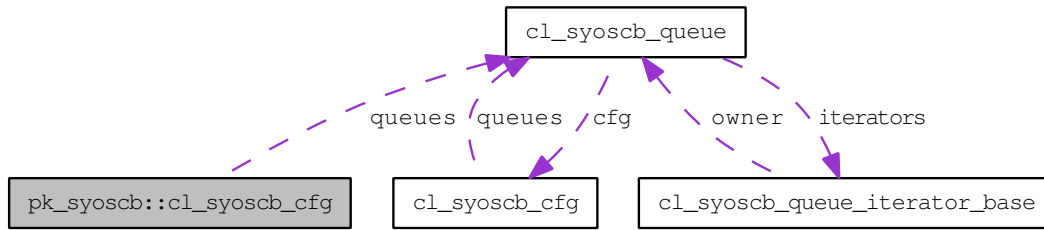
Definition at line 99 of file cl\_syoscb\_cfg.svh.

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 specified 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 as a string list
- void set\_queues (string queue\_names[])  
*Configuration API:* Will set the legal queues when provided 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 a 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 402 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 622 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 580 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 612 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 586 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

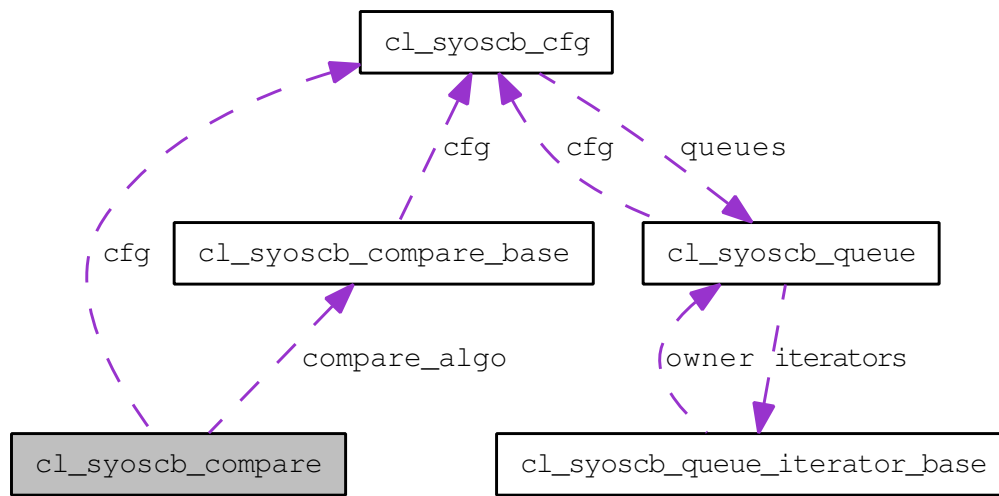
Definition at line 499 of file pk\_syoscb.sv.

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

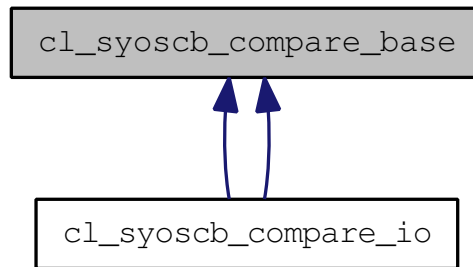
Class which act as the root of the compare algorithm. It instantiates the chosen compare algorithm. Definition at line 3 of file cl\_syoscb\_compare.svh.

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

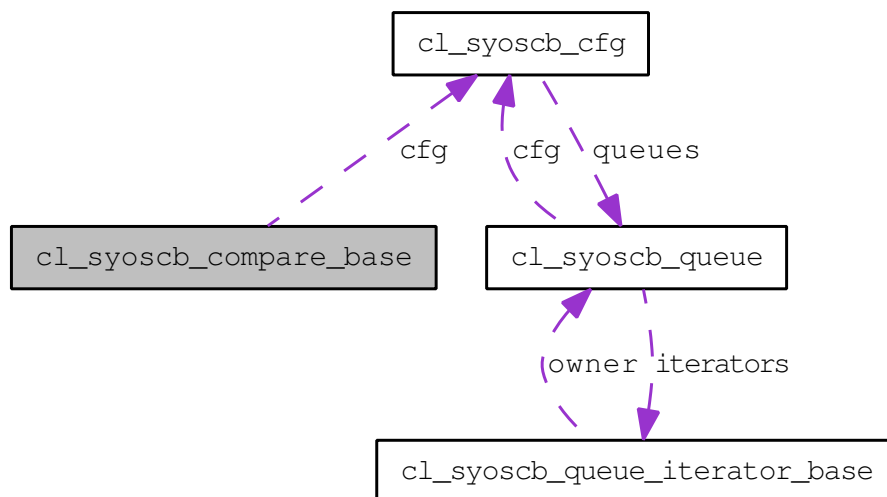
- cl\_syoscb\_compare.svh

## 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 compare 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 compute if the compare shall be done or not. If so then `do_compare()` is called.

**NOTE:** This method must be implemented.

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 implemented.

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

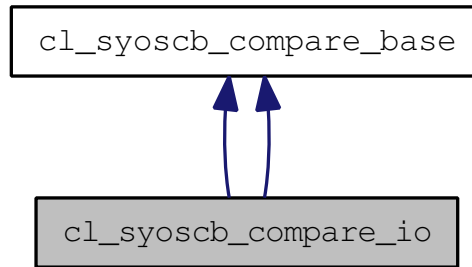
Definition at line 45 of file `cl_syoscb_compare_base.svh`.

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

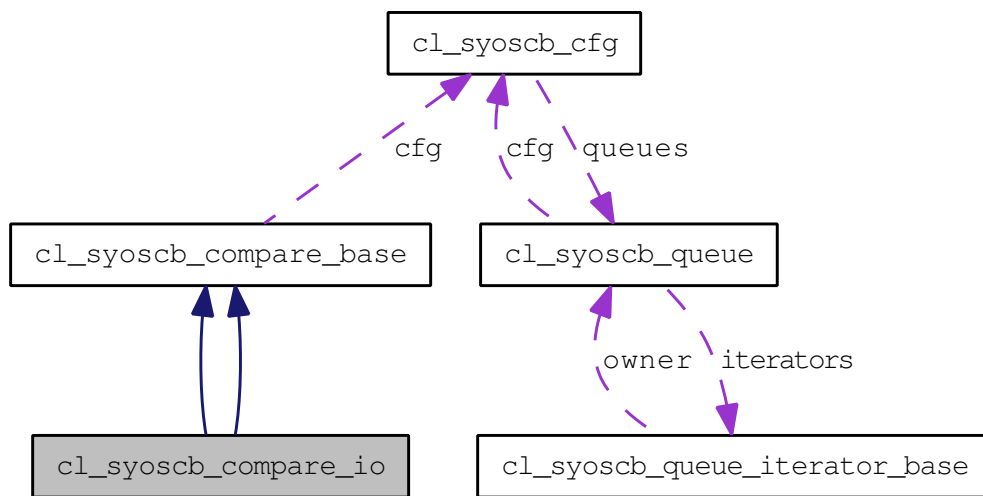
- `cl_syoscb_compare_base.svh`

## 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 compute if the compare shall be done or not. If so then do\_compare() is called.

**NOTE:** This method must be implemented.

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 implemented.

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

- 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\_sequence\_items from already existing testbenches etc. even though more META data is added to the wrapping item.

Definition at line 633 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

- 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\_squence\_items from already existing testbenches etc. even htough more META data is added to the wrapping item.

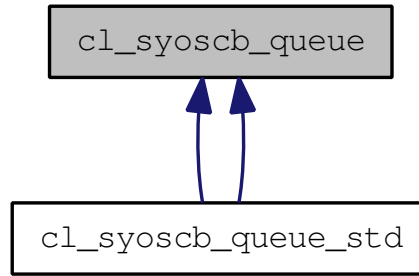
Definition at line 4 of file cl\_syoscb\_item.svh.

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

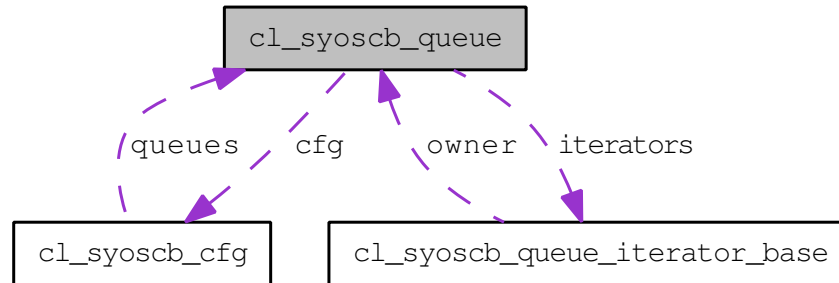
- cl\_syoscb\_item.svh

## 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 conceit 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 the 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).

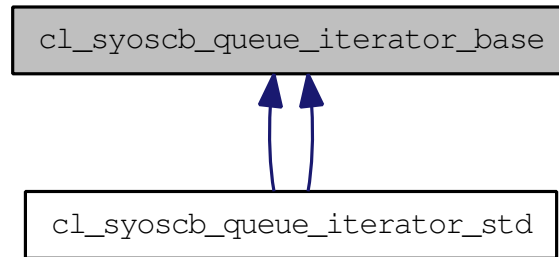
Definition at line 116 of file `cl_syoscb_queue.svh`.

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

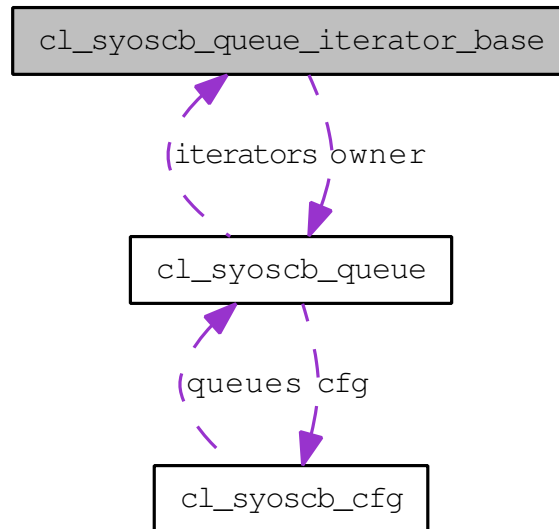
- `cl_syoscb_queue.svh`

## 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 52 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 78 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 59 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).

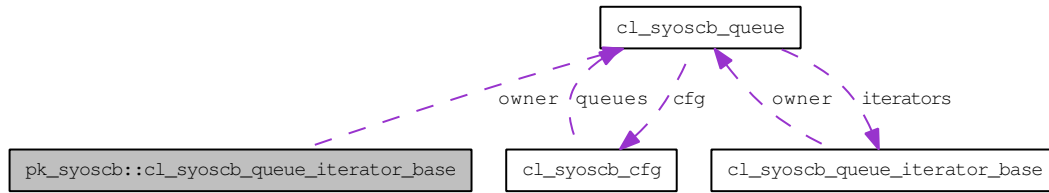
Definition at line 45 of file `cl_syoscb_queue_iterator_base.svh`.

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:



### 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 691 of file pk\_syoscb.sv.

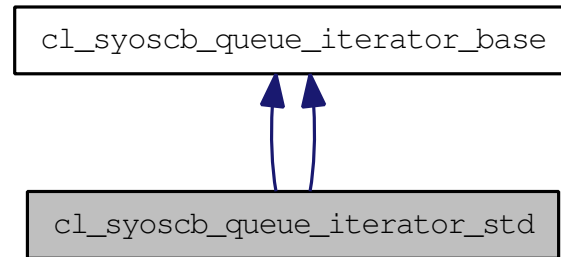
The documentation for this class was generated from the following file:

- pk\_syoscb.sv

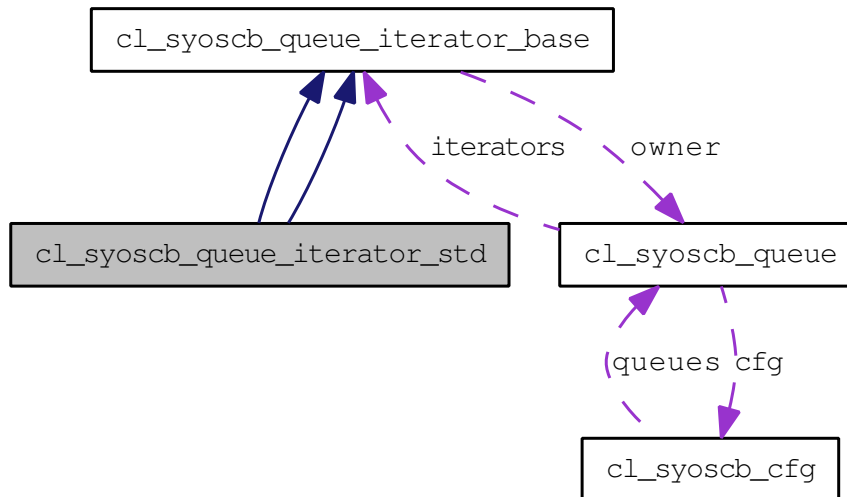


## 9.12 cl\_syoscb\_queue\_iterator\_std Class Reference

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 58 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.

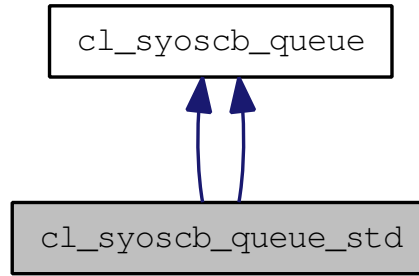
Reimplemented from `cl_syoscb_queue_iterator_base` (p. 41).

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

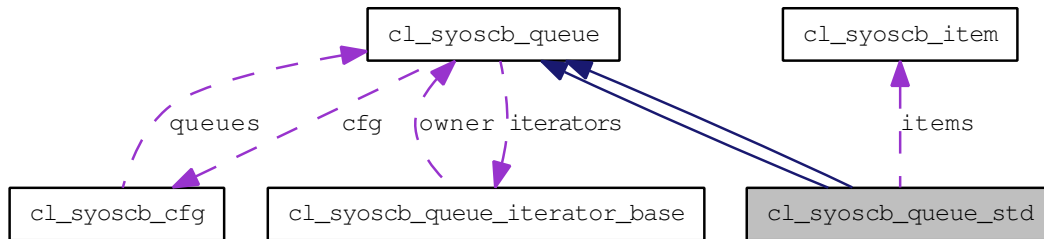
- `cl_syoscb_queue_iterator_std.svh`

### 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 that the 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 extending 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 producer 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

- /home/jacob/work/uvm\_scoreboard/src/ Directory Reference, 19
- add\_item
  - cl\_syoscb, 22
  - cl\_syoscb\_queue, 37
  - cl\_syoscb\_queue\_std, 47
- build\_phase
  - cl\_syoscb, 22
- cl\_syoscb, 21
  - add\_item, 22
  - build\_phase, 22
  - get\_subscriber, 22
- cl\_syoscb\_cfg, 23
  - get\_max\_queue\_size, 24
  - get\_primary\_queue, 24
  - set\_max\_queue\_size, 24
  - set\_primary\_queue, 25
  - set\_queues, 25
- cl\_syoscb\_compare, 29
- cl\_syoscb\_compare\_base, 30
  - compare, 31
  - compare\_do, 31
- cl\_syoscb\_compare\_io, 32
  - compare, 33
  - compare\_do, 33
- cl\_syoscb\_item, 35
- cl\_syoscb\_queue, 36
  - add\_item, 37
  - empty, 37
  - insert\_item, 37
- cl\_syoscb\_queue\_iterator\_base, 39
  - first, 40
  - is\_done, 40
  - last, 40
  - previous, 41
- cl\_syoscb\_queue\_iterator\_std, 43
  - first, 44
  - is\_done, 44
  - last, 44, 45
  - previous, 45
- cl\_syoscb\_queue\_std, 46
  - add\_item, 47
  - empty, 47
  - insert\_item, 47
- cl\_syoscb\_subscriber, 49
- compare
  - cl\_syoscb\_compare\_base, 31
  - cl\_syoscb\_compare\_io, 33
- compare\_do
  - cl\_syoscb\_compare\_base, 31
  - cl\_syoscb\_compare\_io, 33
- empty
  - cl\_syoscb\_queue, 37
  - cl\_syoscb\_queue\_std, 47
- first
  - cl\_syoscb\_queue\_iterator\_base, 40
  - cl\_syoscb\_queue\_iterator\_std, 44
- get\_max\_queue\_size
  - cl\_syoscb\_cfg, 24
  - pk\_syoscb::cl\_syoscb\_cfg, 27
- get\_primary\_queue
  - cl\_syoscb\_cfg, 24
  - pk\_syoscb::cl\_syoscb\_cfg, 27
- get\_subscriber
  - cl\_syoscb, 22
- insert\_item
  - cl\_syoscb\_queue, 37
  - cl\_syoscb\_queue\_std, 47
- is\_done
  - cl\_syoscb\_queue\_iterator\_base, 40
  - cl\_syoscb\_queue\_iterator\_std, 44
- last
  - cl\_syoscb\_queue\_iterator\_base, 40
  - cl\_syoscb\_queue\_iterator\_std, 44, 45
- pk\_syoscb::cl\_syoscb\_cfg, 26
  - get\_max\_queue\_size, 27
  - get\_primary\_queue, 27
  - set\_max\_queue\_size, 27
  - set\_primary\_queue, 27
  - set\_queues, 27
- pk\_syoscb::cl\_syoscb\_item, 34



pk\_syoscb::cl\_syoscb\_queue\_iterator\_base,  
42

previous

cl\_syoscb\_queue\_iterator\_base, 41  
cl\_syoscb\_queue\_iterator\_std, 45

set\_max\_queue\_size

cl\_syoscb\_cfg, 24  
pk\_syoscb::cl\_syoscb\_cfg, 27

set\_primary\_queue

cl\_syoscb\_cfg, 25  
pk\_syoscb::cl\_syoscb\_cfg, 27

set\_queues

cl\_syoscb\_cfg, 25  
pk\_syoscb::cl\_syoscb\_cfg, 27