Dealing with Deprecated Features in SystemC 2.2

John Aynsley, Doulos, April 2007

In version 2.2 of the OSCI SystemC simulator, certain features have been deprecated in line with IEEE 1666™, the Standard SystemC Language Reference Manual. This document highlights which features are now marked as deprecated but still supported, which features are now unsupported, and what you can do about it.

Version 2.2 issues a warning on the first occurrence of many deprecated features. You can suppress all such warning by setting the environment variable SC_DEPRECATION_WARNINGS to the value DISABLE, although it is better practice to rewrite the deprecated code.

Starting Simulation

Functions sc_cycle and sc_initialize

Still supported in 2.2, but with warnings. These functions are a legacy from SystemC version 1, when a SystemC simulation had a single clock. If you use them, any timed event notifications will simply not work. You should switch to using sc_start() instead.

Default Time Units

Still supported in 2.2, but with warnings. Default time units are now considered a legacy from SystemC version 1, when there was but a single global clock and no proper timed notifications at all. The deprecated functions are sc_simulation_time, sc_set_default_time_unit, sc_get_default_time_unit, sc_start(double), and constructor sc_clock(const char*, double, double, double, bool). You should now use explicit time units instead of integers. In particular, sc_start() is likely to catch you out. Instead of writing

```
sc_start(-1);
```

which is now deprecated, you should write

```
sc_start();
```

Class sc_simcontext

Still supported in 2.2, but with warnings for certain functions. Class sc_simcontext will probably remain part of the OSC simulator indefinitely, but should be regarded as part of the internal implementation rather than a user-level feature. Several methods of this class have been replaced by global functions in the namespace sc_core, which you should now use instead. The new function names are sc_delta_count(), sc_is_running(), sc_get_top_level_objects(), and sc_find_objects(). Also, the method get_child_objects() has been replaced by sc_object::get_child_objects().

Module Construction and Port Binding

Constructors of class sc_port that bind ports at the time of construction

Still supported in 2.2, but with warnings. This is rather obscure, but class sc_port had a set of constructors that took a channel as an argument, and bound the port to the channel at the time of construction of the port. Since it is strongly recommended in the SystemC standard that a port should be bound at the point of instantiation of the module to which the port belongs, there is no reason to use these deprecated constructors. If you really do need to bind a port to an object within the module in which the port is defined, use the regular binding operators in the constructor of the module as follows:

```
SC_MODULE(Mod) {
        sc_in<bool>             in;
        sc_signal<bool>         b;

        SC_CTOR(Mod) {
                in.bind(b);
...
```

Method end_module of class sc_module

Still supported in 2.2. Although still part of the OSCI code, this method should be regarded as part of the implementation detail for manipulating the module name stack, and you should not call it explicitly.

operator, operator<< and operator() of class sc_module

Still supported in 2.2, but with warnings. These operators provided three alternative forms of syntax for positional port binding. The only remaining approved syntax for positional binding is the operator(). Moreover, it used to be permitted to call operator() repeatedly when binding the ports of a single module instance, but this is also deprecated with warnings. So, instead of writing

```
        mod << a << b;
```

or

```
        mod << a;
        mod << b;
```

or

```
        mod(a);
        mod(b);
```

the only permissible alternative is now

```
        mod(a, b);
```

Better still, using named binding instead:

```
        mod.port1(a);
        mod.port2(b);
```

Events and Sensitivity

operator() of class sc_sensitive

Still supported in 2.2, but with a warning. Another deprecated alternative. Instead of writing:

```
sensitive(a);
sensitive(b);
```

you should write

```
sensitive << a;
sensitive << b;
```

or

```
sensitive << a << b;
```

sensitive_pos and sensitive_neg

Still supported in 2.2, but with warnings. With the introduction of event finders, these two are now regarded as a legacy from version 1. Instead of writing:

```
sensitive_pos << a;
sensitive_neg << b;
```

you should write

```
sensitive << a.pos();
sensitive << b.neg();
```

Method notify_delayed of class sc_event

Still supported in 2.2, but with a warning. This method was originally used in the internal implementations of primitive channels, but has occasionally leaked out into application code. You should switch to using the method notify(sc_time) instead.

Global function notify(..., sc_event)

Still supported in 2.2, but with a warning. This was a global function that took the event to be notified as an argument. You should switch to using the notify() method of class sc_event instead.

Method timed_out of classes sc_module and sc_prim_channel

Still supported in 2.2, but with a warning. The timed_out() method returns true when a process resumes after a call to wait(sc_time, sc_event) or similar because of a time-out. This method was removed from the standard for theoretical reasons. In order to achieve the same effect without calling timed_out(), you would have to have a process that sets a flag when the event gets notified, which is really rather clumsy.

Function sc_get_curr_process_handle and type sc_process_b

Still supported in 2.2, but with a warning. In SystemC 2.0, function sc_get_curr_process_handle() returned a pointer to type sc_process_b. This function and type should now be regarded as part of the internal implementation. Instead, you should use sc_get_current_process_handle(), which returns an sc_process_handle. The process handle class contains some useful new features.

Signals

Method get_data_ref of classes sc_signal and sc_clock

Still supported in 2.2, but with a warning. You should use read() instead.

Method get_new_value of class sc_signal

Still supported in 2.2. When called following a signal assignment (write) but before the next evaluation phase, this method returns the written value of the signal, which may be different from the current value as returned by read(). This deprecated method was rarely used, but if you do use it, your code should be re-written.

Typedefs sc_inout_clk and sc_out_clk

Still supported in 2.2. If and when they are removed, you could simply provide your own typedefs or use sc_inout<bool> and sc_out<bool>

Macro sc_signal_out_if

Still supported in 2.2. If and when this macro is removed, you could simply define it yourself.

```
#define sc_signal_out_if sc_signal_inout_if
```

Clocked Threads

Certain variants of SC_CTHREAD

Still supported in 2.2. The only variant of the clocked thread macro specified in the IEEE 1666 standard is the case where the second argument is an event finder. All other forms are deprecated.

Waits in Clocked Threads

Still supported in 2.2, but with warnings. The only forms of wait() still permitted in clocked threads are wait() and wait(int). The remaining forms of wait, including waiting for a time or waiting on an event list, can still be used in a non-clocked thread.

Global and Local Watching for Clocked Threads

Not supported in 2.2. All support for local and global watching, including the methods delayed(), watching(), and lambda expressions, has been removed. In order to express the reset condition for a clocked thread, you should use the function reset_signal_is(), which provides a subset of the functionality of watching.

Trace Files

wif trace files

Still supported in 2.2. The wif trace file format has been removed from the standard, but is still supported in the OSCI simulator. The VCD trace file format remains part of the standard.

isdb trace files

Not supported in 2.2. All support for the isdb trace file format has been removed.

Method sc_set_vcd_time_unit of class vcd_trace_file

Still supported in 2.2. The class vcd_trace_file is now considered to be an implementation detail rather than part of the standard, so its public methods have been deprecated. You should use method set_time_unit() of class sc_trace_file instead. For example:

```
sc_trace_file* tf;
tf = sc_create_vcd_trace_file("foo");
tf->set_time_unit(1, SC_NS);
// ((vcd_trace_file*)tf)->sc_set_vcd_time_unit(-9); // Deprecated
```

Function sc_trace_delta_cycles

Still supported in 2.2. This function was not included in the IEEE 1666 standard because its intended behavior is unclear for designs that mix delta cycles and timed notifications. Nevertheless it remains part of the OSCI simulator, and is useful for creating VCD files for designs that do not contain timed notifications.

Methods trace and add_trace

Still supported in 2.2, but with warnings in some cases. Instead of calling the trace() method of classes sc_object, sc_signal, sc_clock and sc_fifo, and the add_trace() method of classes sc_in and sc_out, you should use the corresponding global sc_trace() function. sc_trace() is the only surviving API for tracing objects to the VCD file. For example, given

```
sc_in<bool> in;
sc_signal<bool> a;
sc_trace_file * tf;
```

then instead of writing

```
in.add_trace(tf, “in”);
a.trace(tf):
```

you should write

```
sc_trace(tf, in, “in”);
sc_trace(tf, a, “a”);
```

Function sc_trace for an Enumeration

Still supported in 2.2, but with warnings. The OSCI simulator includes one specific overloading of sc_trace that was excluded from the IEEE 1666 standard because it was considered unnecessary. This is the function to trace an integer variable given an enumeration defined by an array of C character strings. It remains part of the OSCI simulator for now.

Miscellaneous

The Reporting Mechanism Based on Integer ids

Still supported in 2.2, but with warnings. In the first version of the report handler, reports were distinguished using an integer id. This has now been replaced by a report handler that distinguishes reports using a char* message type, which you should use instead. The deprecated methods are register_id(), get_message(), is_suppressed(), suppress_id(), suppress_infos(), suppress_watchings(), make_warnings_errors(), and get_id().

Constants SC_DEFAULT_STACK_SIZE, SC_MAX_NUM_DELTA_CYCLES, and SYSTEMC_VERSION

Still supported in 2.2. These constants are regarded as being implementation details rather than part of the standard, but will probably remain part of the OSCI simulator indefinitely. The function sc_version() provides a standard way to get the version identifier.

Type sc_bit

Still supported in 2.2, but with warnings. sc_bit was deprecated because the C++ type bool provides the same functionality but with better execution speed.

Class sc_string

By default, sc_string is not supported in 2.2. You should use std::string instead. However, if you need to compile legacy code without modification, the old sc_string class can still be made available by including the following before the SystemC header:

```
#define SC_USE_SC_STRING_OLD
```

Classes sc_pvector, sc_plist, sc_phash, and sc_ppq

Still supported in 2.2. These classes only exist because compiler support for the C++ Standard Library (STL) was poor when SystemC was young. Today, you should use the C++ Standard Library instead.