

SystemC 2.1 Update Tutorial - GSPx 2004



Electronic Systems Solutions

Presented by David C. Black, co-founder

www.EklecticAlly.com

info@EklecticAlly.com

Voice mailbox/FAX 888-467-4609

version 1.0

Agenda - Motivations

- Motivations
- Events
 - **sc_event_queue**
- Ports
 - **sc_export**
- Dynamic Processes
 - **sc_spawn**
 - **SC_FORK/JOIN**
- Exception Reporting
 - **sc_report**
- Environment support
 - **sc_argc/argv()**
 - callbacks
- Miscellaneous



Features of SystemC 2.0.1

- Hardware data types
 - **sc_int**<N>,...
 - **sc_logic**,...
 - **sc_fixed**<>,...
- Notion of time
 - **sc_time**
 - **wait** (D,**SC_NS**)
- Structure
 - **SC_MODULE**
- Concurrency
 - **SC_METHOD**
 - **SC_THREAD**
 - **SC_CTHREAD**
- Connectivity
 - **sc_port**
 - **sc_interface**
 - **sc_channel**
- Standard channels
 - **sc_signal**<>,...
 - **sc_fifo**<>,...
 - **sc_mutex**<>,...
- Events
 - **sc_event**
 - **notify**()
 - **wait**(evt,₁levt,₂)

Changes in SystemC 2.1

- Improved modeling features
 - Event queues, dynamic processes, **sc_export**
- Improved verification features
 - Dynamic processes, Reporting
- Improved modularity for IP
 - **sc_export**, callbacks
- Ease of use
- Bug fixes

Agenda - Events

- Motivations
- Events
 - **sc_event_queue**
- Ports
 - **sc_export**
- Dynamic Processes
 - **sc_spawn**
 - **SC_FORK/JOIN**
- Exception Reporting
 - **sc_report**
- Environment support
 - **sc_argc/argv()**
 - callbacks
- Miscellaneous

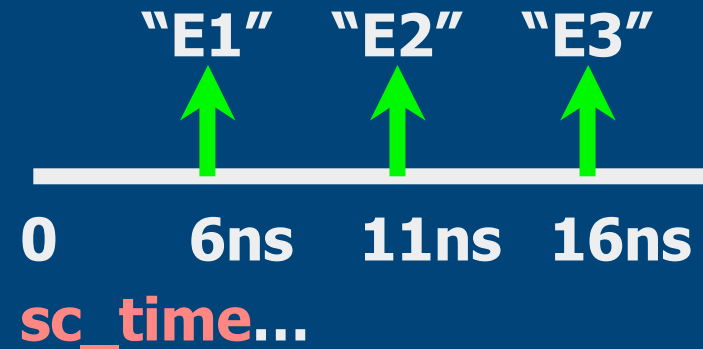


New event class

- **sc_event** class not flexible enough
 - Only one outstanding event at a time
- New **sc_event_queue** class adds queue
 - Multiple notifications result in multiple events
 - Reliably catch every notification

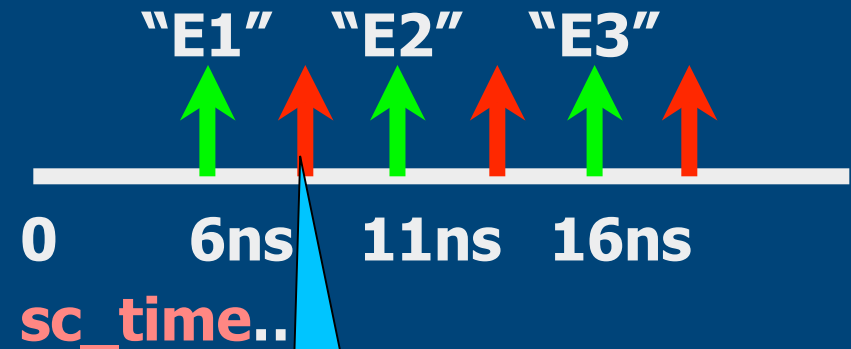
Events

```
static sc_event A_evt;
void mymod::thread1() {
    for (int i=0;i!=3;i++) {
        wait(5, SC_NS);
        A_evt.notify(1, SC_NS);
    }
}
void mymod::thread2() {
    int j = 1;
    for(;;) {
        wait(A_evt);
        cout <<"E"<<j<< endl;
    }
}
```



Events missed

```
static sc_event A_evt;  
void mymod::thread1() {  
    for (int i=0;i!=3;i++) {  
        wait(5, SC_NS);  
        A_evt.notify(1, SC_NS);  
        A_evt.notify(3, SC_NS);  
    }  
}  
  
void mymod::thread2() {  
    int j = 1;  
    for(;;) {  
        wait(A_evt);  
        cout <<"E"<<j<< endl;  
    }  
}
```



Never
generated
events

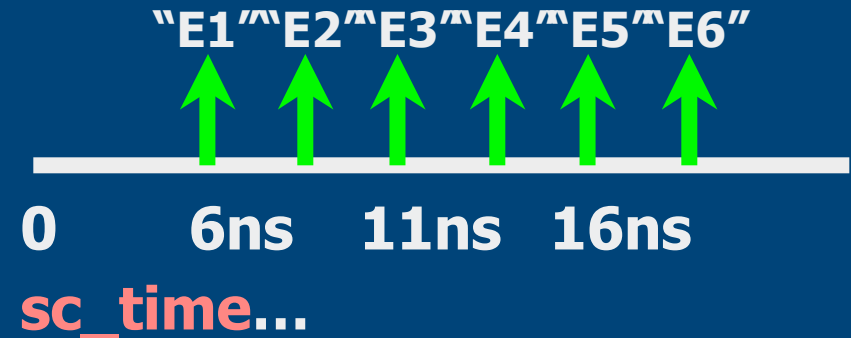
sc_event_queue class

- SYNTAX:
 - **sc_event_queue** NAME;
- METHODS:
 - **notify()** & **wait()** // of course
 - **cancel_all()**

sc_event_queue example

Events caught

```
static sc_event_que A_evt;  
void mymod::thread1() {  
    for (int i=0;i!=3;i++) {  
        wait(5, SC_NS);  
        A_evt.notify(1, SC_NS);  
        A_evt.notify(3, SC_NS);  
    }  
}  
  
void mymod::thread2() {  
    int j = 1;  
    for(;;) {  
        wait(A_evt);  
        cout <<"E"<<j<< endl;  
    }  
}
```



Agenda - Ports

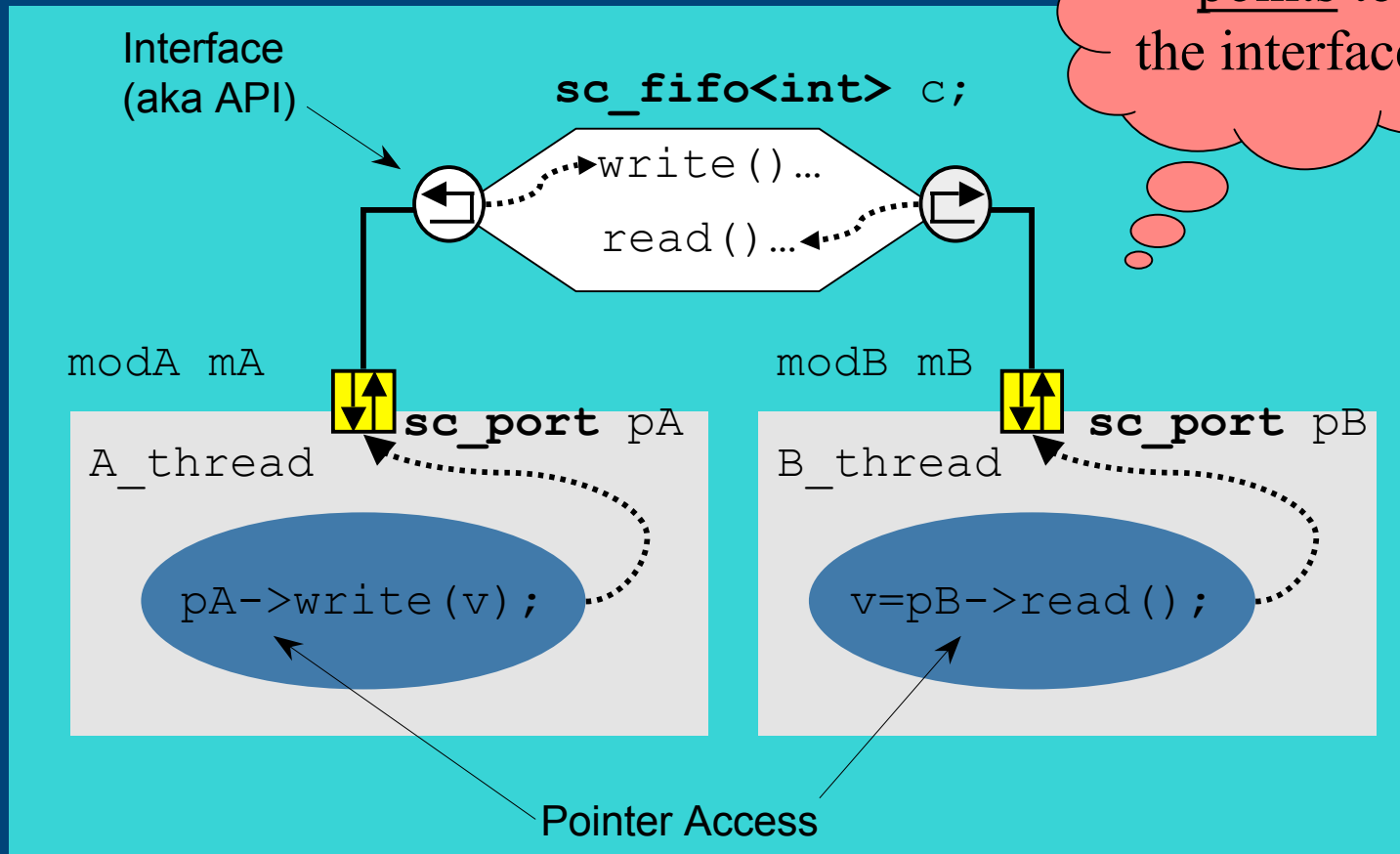
- Motivations
- Events
 - **sc_event_queue**
- Ports
 - **sc_export**
- Dynamic Processes
 - **sc_spawn**
 - **SC_FORK/JOIN**
- Exception Reporting
 - **sc_report**
- Environment support
 - **sc_argc/argv()**
 - callbacks
- Miscellaneous



sc_export

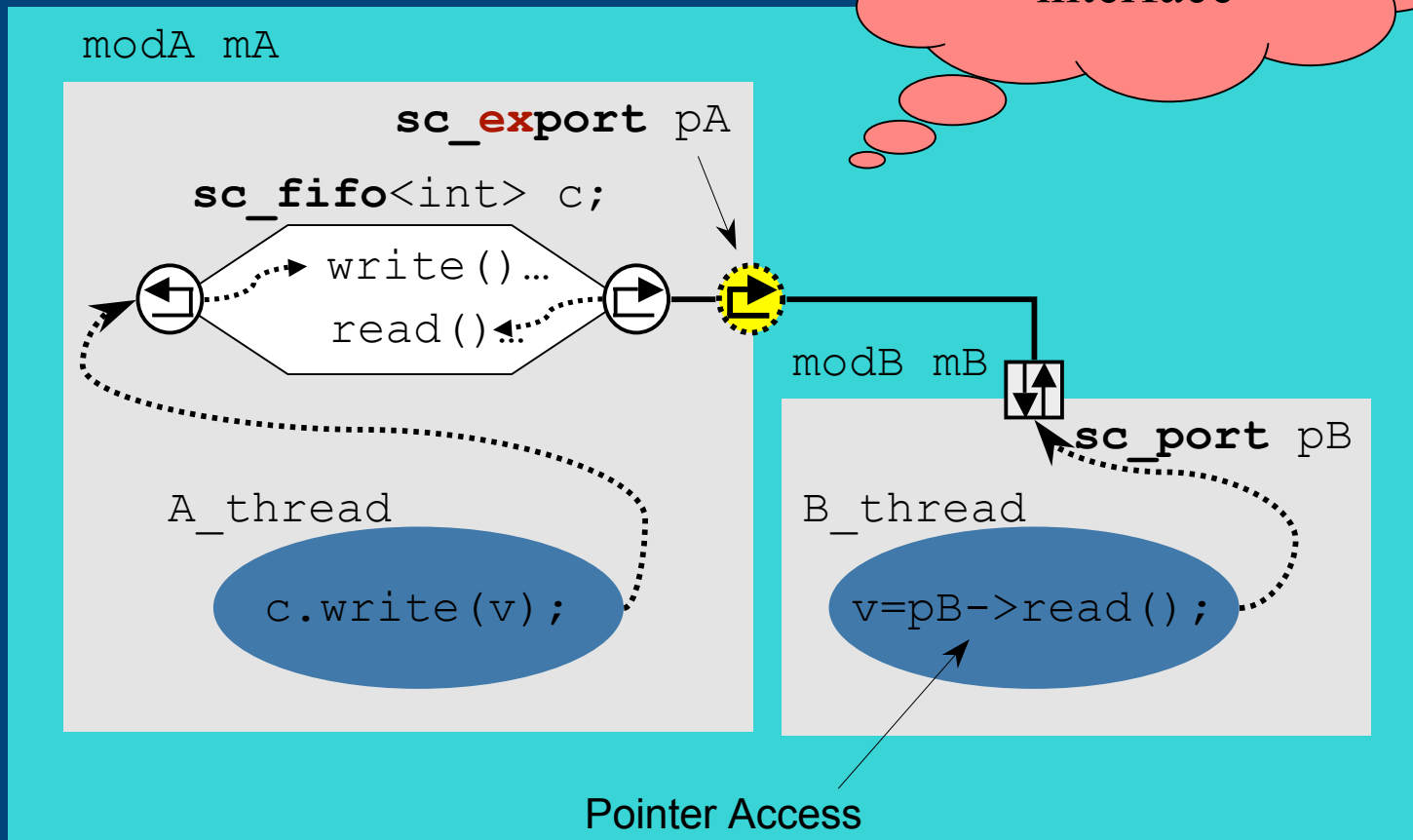
- Opposite of **sc_port** w.r.t. caller/callee
- Exposes internal channels
- More controlled hierarchical interfaces
- Can reduce context switching

sc_port (from SystemC version 2.0)



sc_export (new for SystemC version 2.1)

“exports the interface”



sc_export syntax

- DECLARATION
 - **sc_export**<INTERFACE> EXPORTNAME;
- BINDING INTERNAL
 - **SC_CTOR**(MODNAME) {
 EXPORTNAME(CHANNAME);
}
- BINDING EXTERNAL
 - INST1.PORTNAME(INST2.EXPORTNAME);

sc_export IP defn example

```
SC_MODULE(MyIP) {  
    sc_export<sc_fifo_out_if<int>> datain_xp;  
    sc_export<sc_fifo_out_if<int>> result_xp;  
    SC_CTOR(MyIP) {  
        datain_xp(datain_fifo);  
        result_xp(result_fifo);  
        SC_THREAD(main_thread);  
    }  
    void main_thread;  
private:  
    sc_fifo<int> datain_fifo;  
    sc_fifo<int> result_fifo;  
};
```

Declare

Internal
binding

Local
channels

sc_export IP usage example

```
SC_MODULE (Top) {  
    MyIP myip_i("myip_i");  
    Other other_i("other_i");  
    SC_CTOR (Top)  
    : myip_i("myip_i")  
    , other_i("other_i")  
    {  
        other_i.data_op(myip_i.datain_xp);  
        other_i.data_ip(myip_i.result_xp);  
    }  
};
```

External
binding

Agenda - Dynamic processes

- Motivations
- Events
 - `sc_event_queue`
- Ports
 - `sc_export`
- Dynamic Processes
 - `sc_spawn`
 - `SC_FORK/JOIN`
- Exception Reporting
 - `sc_report`
- Environment support
 - `sc_argc/argv()`
 - callbacks
- Miscellaneous



Dynamically spawned processes

- Fork/join
- Required for temporal assertion checking
- Helpful for testbenches
- Automatic allocation/deallocation of threads
- Multiple threads off a single method
- Potentially challenging syntax

sc_spawn

- PRIMARY SYNTAX

- #define **SC_INCLUDE_DYNAMIC_PROCESSES**

- sc_process_handle h =

- sc_spawn(sc_bind(&FUNC[,ARG]...)[,NM[,OPTS]])

- ◆ &FUNC may be &CLASS::METHOD

- ◆ NM is process name & must be unique

- ◆ Returns sc_process_handle

- sc_spawn(RT,sc_bind(&FUNC[,ARG]...)[,NM[,OPTS]])

- ◆ RT is return type

- OTHER METHODS

- sc_spawn_options()

- sc_process_handle::wait()

- sc_ref() // reference

- sc_cref() // const reference

sc_spawn simple example 1 of 4

```
#define SC_INCLUDE_DYNAMIC_PROCESSES
#include <systemc.h>

SC_MODULE(A) {
    SC_CTOR(A) {
        SC_THREAD(main_th);
    }
    void main_th();
    void dynamic_th(double d);
};
```

Important

Declare the functions
and methods to be used
as processes

sc_spawn simple example 2 of 4

```
void A::main_th () {  
    cout << "Starting " << name() << endl;  
    wait(1, SC_NS); // delay for effect  
    sc_process_handle h1  
        =sc_spawn(sc_bind(&A::dynamic_th, this, 15.0), "d1");  
    wait(3, SC_NS); // arbitrary delay  
    sc_process_handle h2  
        =sc_spawn(sc_bind(&A::dynamic_th, this, 6.0), "d2");  
    h1.wait();  
    cout << "Finished h1" << endl;  
} //end A::main_th()
```

Spawn (launch)
the processes
from a thread

Same
method
used twice

Unique
process
name

sc_spawn simple example 3 of 4

```
void A::dynamic_th(double d);
char* me = sc_get_curr_process_handle()->name();
sc_time now = sc_time_stamp();
cout << "Starting " << me << " @ " << now << endl;
wait(d, SC_NS); // arbitrary delay
now = sc_time_stamp();
cout << "Exiting " << me << " @ " << now << endl;
} //end A::dynamic_th()
```

sc_spawn simple example 4 of 4

```
Starting A_i  
Starting A_i.d1 @ 1 ns  
Starting A_i.d2 @ 4 ns  
Exiting A_i.d2 @ 10 ns  
Exiting A_i.d1 @ 16 ns  
Finished h1
```

sc_spawn more complex example

```
bool complex_th(const Big& big, Big& rslt);

void A::main_th() {
    bool ok;
    Big big, answer;
    ...
    sc_process_handle h
        = sc_spawn(&ok,
                  sc_bind(&complex_th, sc_cref(big),
                          sc_ref(answer)),
                  "d1"
        );
} //end A::main_th()
```

Simple function

Return value

Constant reference

Reference

SC_FORK/SC_JOIN

- SYNTAX

 - SC_FORK

 - sc_spawn(...),

 - sc_spawn(...),

 - :

 - sc_spawn(...)

 - SC_JOIN

- Join waits for all to finish
 - Use **wait()** for other types

SC_FORK/SC_JOIN example

```
void TB::phone_th(int id) { /* make calls */ }

void TB::call_th() {
    ...
    SC_FORK
        sc_spawn(sc_bind(&TB::phone_th, this, 1), "ph1"),
        sc_spawn(sc_bind(&TB::phone_th, this, 2), "ph2"),
        sc_spawn(sc_bind(&TB::phone_th, this, 3), "ph3")
    SC_JOIN
    ...
} //end Testbench::call_th()
```

No semi-colon

Comma separated

Agenda - Exception reporting

- Motivations
- Events
 - `sc_event_queue`
- Ports
 - `sc_export`
- Dynamic Processes
 - `sc_spawn`
 - `SC_FORK/JOIN`
- Exception Reporting
 - `sc_report`
- Environment support
 - `sc_argc/argv()`
 - callbacks
- Miscellaneous



Exception reporting

- Similar to **SCV** reporting facility
- Structured
- Consistent messaging for all components
 - Simulator core, libraries, IP, modules, testbenches
- Specify actions for severity, message type

Categorizing Exceptions

- Specify message types as either
 - **sc_string**
 - **char***

- Severity has four levels:

```
enum sc_severity {  
    SC_INFO,           // informative  
    SC_WARNING,       // potential  
    SC_ERROR,         // problem  
    SC_FATAL         // terminate  
};
```

Actions to take when handling

```
enum sc_action {  
    SC_UNSPECIFIED = 0x00,  
    SC_DO_NOTHING = 0x01,  
    SC_THROW = 0x02,  
    SC_LOG = 0x04,  
    SC_DISPLAY = 0x08,  
    SC_CACHE_REPORT = 0x10,  
    SC_INTERRUPT = 0x20,  
    SC_STOP = 0x40,  
    SC_ABORT = 0x80  
};
```

May be logically OR'd

sc_report_handler:: methods

- **report**(svr,typ,msg,fl,ln)
- **set_handler**()
- **set_actions**(typ,svr,act)
- **stop_after**(typ,svr,max)
- **suppress**([act])
- **force**([act])
- **get_new_action_id**()
- **set_log_file_name**(str)
- **get_log_file_name**()
- **set_handler**(proc)
- **default_handler**(rpt,act)

sc_report:: methods

- `get_cached_report()`
- `clear_cached_report()`
- `initialize()`
- `release()`
- `get_severity()`
- `get_msg_type()`
- `get_msg()`
- `get_file_name()`
- `get_line_number()`
- `get_time()`
- `get_process()`

Simplifying macros

- **SC_REPORT_INFO**(id, message)
- **SC_REPORT_WARNING**(id, message)
- **SC_REPORT_ERROR**(id, message)
- **SC_REPORT_FATAL**(id, message)

Simple Reporting example

my_module.cpp

```
static const char* MSGID = "MY_SIM";  
void my_module::some_thread() {  
    SC_REPORT_INFO (MSGID, "Starting");  
    ...  
    if (error_condition) {  
        SC_REPORT_ERROR (MSGID,  
            "Oops something bad happened");  
    }  
    ...  
}
```

Output from reporting

```
0 ns: Info: MY_SIM: Starting
...
4 ns: Error: MY_SIM: Oops something bad happened
In file: my_module.cpp:24
In process: my_mod_i.some_thread_0 @ 4 ns
```

Agenda - Environment support

- Motivations
- Events
 - **sc_event_queue**
- Ports
 - **sc_export**
- Dynamic Processes
 - **sc_spawn**
 - **SC_FORK/JOIN**
- Exception Reporting
 - **sc_report**
- Environment support
 - **sc_argc/argv()**
 - callbacks
- Miscellaneous



Easier access to startup arguments

- Simplifies ability of libraries and IP to use command-line control
- **sc_argc()** and **sc_argv()**

sc_argv example

```
char* SomeMod::CheckForMyopt() {
    // Scan command line for "-myopt VAL"
    for (int i=0; i<=sc_argv(); i++) {
        if (strcmp(sc_argv()[i], "-myopt") != 0) continue;
        if (++i > sc_argv()) {
            cout << "ERROR: Missing VAL for -myopt" << endl;
            break;
        } //endif
        cout << "Using -myopt " << sc_argv()[i] << endl;
        return sc_argv()[i+1];
    } //endifor
    return "";
} //end SomeMod::CheckForMyopt()
```

New callbacks

- **Motivation**

- allow IP integration without code in **sc_main**
- module specific setup/cleanup

- **METHODS**

- **before_end_of_elaboration()**
- **start_of_simulation()**
- **end_of_simulation()**

- **NOTE**

- Called for every instance!

Callback example 1 of 3

```
static sc_uint<32> instances = 0;
static bool report_setup = false;
void SomeMod::end_of_elaboration() {
    instances++;
    if (!report_setup) {
        sc_set_stop_mode(SC_STOP_FINISH_DELTA);
        sc_report_handler::set_log_file_name("sim.log");
        sc_report_handler::stop_after(SC_ERROR, 100);
        sc_report_handler::set_actions(SC_ERROR,
            SC_DISPLAY | SC_CACHE_REPORT | SC_LOG);
        report_setup = true;
    } //endif
} //end SomeMod::end_of_elaboration()
```

Callback example 2 of 3

```
static char* RTAG = "SomeMod";
static char* VERSION = "$Id: SomeMod.cpp,v 1.1 ... $";
static bool displayed = false;
void SomeMod::start_of_simulation() {
    if (!displayed) {
        SC_REPORT_INFO(RTAG, VERSION);
        SC_REPORT_INFO(RTAG,
            instances.to_str() + " instances");
        displayed = true;
    }
} //end SomeMod::start_of_simulation()
```

Callback example 3 of 3


```
static int cleaned = 0;
void SomeMod::end_of_simulation() {
    if (!cleaned++) {
        sc_report *rp =
            sc_report_handler::get_cached_report();
        if ( rp ) {
            cout << rp->get_msg() << endl;
            SC_REPORT_INFO("END", "Simulation FAILED");
        } else {
            SC_REPORT_INFO("END", "Simulation PASSED");
        } //endif
    } //endif
} //end SomeMod::end_of_simulation()
```

Agenda - Miscellaneous

- Motivations
- Events
 - **sc_event_queue**
- Ports
 - **sc_export**
- Dynamic Processes
 - **sc_spawn**
 - **SC_FORK/JOIN**
- Exception Reporting
 - **sc_report**
- Environment support
 - **sc_argc/argv()**
 - callbacks
- Miscellaneous



Ease of use features

- Signal and port specializations
 - Directly assign to/from part selects of signals/ports
- Mixed concatenation
 - Concatenations of **sc_int**, **sc_uint**, **sc_big_int**, and **sc_big_uint** can now be mixed without casts
- Object code release tagging
 - Link-time detection between incompatible object files
- POSIX thread support
 - Allows use of code coverage and memory leak checking tools
- Support for MacOS X 

Bug fixes (1 of 2)

- **sc_start()** after simulation has reached its internal maximum time value would overflow simulation time.
- **sc_trace** for uint64, int64 missing
- **sc_set_time_resolution** not properly affecting VCD dump information.
- The value of **sc_clock** needs to be updated during update phase not evaluate (execution) phase to prevent race conditions.
- **sc_string** subscript operator may modify multiple instance because of copy semantics.
- Cpu risc example not shipped anymore

Bug fixes (2 of 2)

- Error in **sc_bv** char constructor
- **sc_biguint** partial selection bug
- Missing terminating null char in >> operator for **sc_string**.
- The constructor **sc_module(const sc_module&)** is not defined
- Signal initialized in constructor not registered with its module.
- Deletion of main fiber should not occur in **~sc_cor_fiber**
- Need ability to compile with -Wno-deprecated
- tracing ports after end of elaboration had no effect
- **wait** statements in constructor led to crashes

For more information

- OSCI Release notes
- Book
- Simulator vendors

Questions?

