IEEE 1666-2023
SystemC Deep Dive

Laurent Maillet-Contoz, Accellera SystemC LWG Chair

Jérôme Cornet, IEEE P1666-2023 Chair

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION DAY

NOV 16, 2023 | MUNICH | GERMANY

Previously on SystemC…

- SystemC IEEE 1666-2023 released on September, 8th

- Announcement on SystemC September Fika, “Sneak peek” presentation
  - C++ 17 baseline
  - SC_NAMED(), SC_CTOR with arguments, Deprecated SC_HAS_PROCESS
  - sc_vector additions
  - Stage Callback, Suspend mechanism
- Replay
  - https://systemc.org/events/scef202309/

SYSTEMC EVENTS

SystemC events: time out or not timeout…

IEEE 1666-2011

```
3   SC_MODULE(module_with_events_timeout) {
4     …
5
6     sc_core::sc_event my_event;
7
8     void my_thread_1() {      // Declared as SC_THREAD
9       …
10      my_event.notify(10, sc_core::SC_NS);
11    }
12
13    void my_thread_2() {      // Declared as SC_THREAD
14      // Wait with timeout
15      wait(sc_core::sc_time(20, sc_core::SC_NS), my_event);
16
17      // Did the wait timed out or was the event notified?
18      std::cout << "Something happened" << std::endl;
19    }
20
21  };
```

SystemC events: time out or not timeout…

IEEE 1666-2011

```
25  SC_MODULE(module_with_events_timeout) {
26    …
27
28    sc_core::sc_event my_event;
29    bool              my_event_notified = false;
30
31    void my_thread_1() {      // Declared as SC_THREAD
32      …
33      my_event.notify(10, sc_core::SC_NS);
34      my_event_notified = true;
35    }
36
37    void my_thread_2() {      // Declared as SC_THREAD
38      // Wait with timeout
39      wait(sc_core::sc_time(20, sc_core::SC_NS), my_event);
40
41      if (!my_event_notified) {
42        std::cout << "Waiting for my_event timed out!" << std::endl;
43      }
44      my_event_notified = false;
45    }
46
47  };
```

SystemC events: or lists

IEEE 1666-2011

```
52  SC_MODULE(module_with_events_or_list) {
53    …
54
55    sc_core::sc_event my_event, another_event;
56
57    void my_thread_1() {      // Declared as SC_THREAD
58      …
59      my_event.notify(10, sc_core::SC_NS);
60    }
61
62    void my_thread_2() {      // Declared as SC_THREAD
63      // Wait for OR list
64      wait(my_event | another_event);
65
66      std::cout << "Something happened" << std::endl;
67    }
68
69  };
```

SystemC events: or lists

IEEE 1666-2011

```
73  SC_MODULE(module_with_events_or_list) {
74    …
75
76    sc_core::sc_event  my_event, another_event;
77    sc_core::sc_event *notified_event = nullptr;
78
79    void my_thread_1() {      // Declared as SC_THREAD
80      …
81      my_event.notify(10, sc_core::SC_NS);
82      notified_event = &my_event;
83    }
84
85    void my_thread_2() {      // Declared as SC_THREAD
86      // Wait for OR list
87      wait(my_event | another_event);
88
89      if (notified_event == &my_event) {
90        std::cout << "my_event was notified!" << std::endl;
91      }
92      notified_event = nullptr;
93    }
94
95  };
```

SystemC events: “persistence”

IEEE 1666-2011

```
101  SC_MODULE(module_with_events_persistence) {
102    …
103
104    sc_core::sc_event  my_event;
105
106    void my_thread_1() {      // Declared as SC_THREAD
107      …
108      my_event.notify();
109    }
110
111    void my_thread_2() {      // Declared as SC_THREAD
112      // Simple wait for event
113      wait(my_event);
114
115      std::cout << "It worked!" << std::endl;
116    }
117
118  };
```

Did it?

SystemC events: “persistence”

IEEE 1666-2011

```
122  SC_MODULE(module_with_events_persistence) {
123    …
124
125    sc_core::sc_event  my_event;
126    bool               my_event_notified = false;
127
128    void my_thread_1() {      // Declared as SC_THREAD
129      …
130      my_event.notify();
131      my_event_notified = true;
132    }
133
134    void my_thread_2() {      // Declared as SC_THREAD
135      // Simple wait for event
136      if (!my_event_notified) {
137        wait(my_event);
138      }
139      my_event_notified = false;
140
141      std::cout << "It worked!" << std::endl;
142    }
143
144  };
```

SystemC events additions

- New method: triggered()
  - Returns true if and only if:
    - The event has been triggered in the immediately preceding delta notification phase

    OR
    - The event has been triggered in the current evaluation phase via an immediate notification.

SystemC events: time out or not timeout…

IEEE 1666-2023

```
149  SC_MODULE(module_with_events_timeout) {
150    …
151
152    sc_core::sc_event SC_NAMED(my_event);
153
154    void my_thread_1() {      // Declared as SC_THREAD
155      …
156      my_event.notify(10, sc_core::SC_NS);
157    }
158
159    void my_thread_2() {      // Declared as SC_THREAD
160      // Wait with timeout
161      wait(sc_core::sc_time(20, sc_core::SC_NS), my_event);
162
163      if (!my_event.triggered()) {
164        std::cout << "Waiting for my_event timed out!" << std::endl;
165      }
166    }
167
168  };
```

SystemC events: or lists

IEEE 1666-2023

```
173  SC_MODULE(module_with_events_or_list) {
174    …
175
176    sc_core::sc_event  SC_NAMED(my_event), SC_NAMED(another_event);
177
178    void my_thread_1() {      // Declared as SC_THREAD
179      …
180      my_event.notify(10, sc_core::SC_NS);
181    }
182
183    void my_thread_2() {      // Declared as SC_THREAD
184      // Wait for OR list
185      wait(my_event | another_event);
186
187      if (my_event.triggered()) {
188        std::cout << "my_event was notified!" << std::endl;
189      }
190    }
191
192  };
```

SystemC events: “persistence”

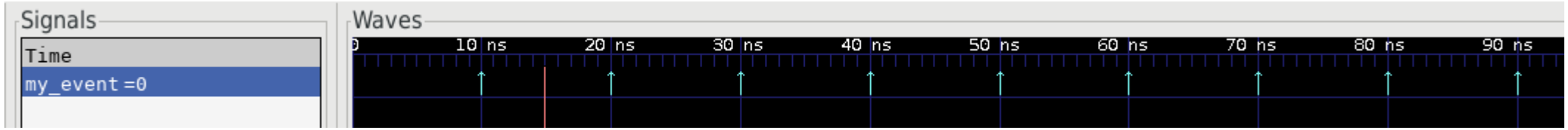
IEEE 1666-2023

```
197  SC_MODULE(module_with_events_persistence) {
198    …
199
200    sc_core::sc_event  SC_NAMED(my_event);
201
202    void my_thread_1() {      // Declared as SC_THREAD
203      …
204      my_event.notify();
205    }
206
207    void my_thread_2() {      // Declared as SC_THREAD
208      // Simple wait for event
209      if (!my_event.triggered()) {
210        wait(my_event);
211      }
212
213      std::cout << "It worked!" << std::endl;
214      // Beware of when "triggered" is reset
215    }
216
217  };
```

Other SystemC events additions

- New “None event”
  - Event guaranteed to be never notified
  - Useful for some contexts where event references are required
  - Example: TLM-1 interfaces

- Events can now be traced!



SYSTEMC TIME

SystemC time

- Main class representing simulation time values
  - sc_time and companion sc_time_unit
- Long-standing issues
  - Textual parsing
  - Load and dump of raw values
- New requests
  - Analog Mixed-Signal time scales
  - Missing operator

SystemC time: textual parsing

IEEE 1666-2023

```
222   sc_core::sc_time t1("3.5 ns");
223
224   std::cout << t1 << std::endl;
225
226   auto t2 = sc_core::sc_time::from_string("12.5 s");
227
228   std::cout << t2 << std::endl;
```

Note: new constructor and static method are using std::string_view.

SystemC time: load and dump raw values

- Dump

IEEE 1666-2011

```
232  std::vector<sc_core::sc_time> time_values;
233
234  while (!finished) {
235    wait(my_event);
236
237    time_values.push_back(sc_core::sc_time_stamp());
238  }
239
240  // Dumping raw values
241  std::ofstream outfile("values.bin", std::ios::binary);
242
243  if (outfile) {
244    for (const auto & time : time_values) {
245      sc_core::sc_time::value_type raw_value = time.value();
246
247      outfile.write(reinterpret_cast<const char *>(&raw_value),
248                    sizeof(sc_core::sc_time::value_type));
249    }
250  }
```

SystemC time: load and dump raw values

- Loading

IEEE 1666-2011

```
255   std::vector<sc_core::sc_time> time_values;
256
257   // Loading raw values
258   std::ifstream infile("values.bin", std::ios::binary);
259
260   if (infile) {
261     sc_core::sc_time::value_type raw_value;
262
263     while (infile.read(reinterpret_cast<char *>(&raw_value),
264                        sizeof(sc_core::sc_time::value_type)))
265     {
266       time_values.push_back(sc_core::sc_time( ????? ));
267     }
268   }
```

SystemC time: load and dump raw values

- Loading

IEEE 1666-2023

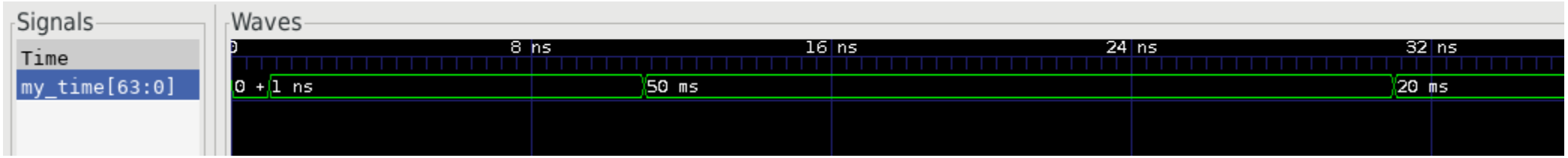
```
273   std::vector<sc_core::sc_time> time_values;
274
275   // Loading raw values
276   std::ifstream infile("values.bin", std::ios::binary);
277
278   if (infile) {
279     sc_core::sc_time::value_type raw_value;
280
281     while (infile.read(reinterpret_cast<char *>(&raw_value),
282                        sizeof(sc_core::sc_time::value_type)))
283     {
284       time_values.push_back(sc_core::sc_time::from_value(raw_value));
285     }
286   }
```

Other SystemC time additions

- New modulo operator
- Extra time units
  - Attosecond (10^{-18}): SC_AS
  - Zeptosecond (10^{-21}): SC_ZS
  - Yoctosecond (10^{-24}) : SC_YS
- Better definition of corner cases with Time Resolution

Other SystemC time additions (2)

- sc_time instances can now be traced!



SYSTEMC SIGNALS

Signals additions

IEEE 1666-2023

```
292  SC_MODULE(module_with_signals) {
293    …
294
295    sc_core::sc_signal<bool>              sig_reset{"sig_reset", true};
296
297    sc_core::sc_signal<sc_dt::sc_uint<8>> SC_NAMED(sig_value, 42);
298
299    SC_CTOR(module_with_signals) {
300    }
301
302  };
```

- Reset value at construction time

Signals additions (2)

IEEE 1666-2023

```
307  SC_MODULE(signal_module) {
308    sc_core::sc_in<bool>              reset_in{"reset_in"};
309    sc_core::sc_in<sc_dt::sc_uint<8>> SC_NAMED(value_in);
310
311    sc_core::sc_out<bool>             SC_NAMED(output);
312
313    …
314  };
315
316  SC_MODULE(test_unbound_tie) {
317    signal_module                          SC_NAMED(SIGNAL_MODULE);
318
319    SC_CTOR(test_unbound_tie) {
320      SIGNAL_MODULE.reset_in(sc_core::sc_tie::value(false));
321      SIGNAL_MODULE.value_in(sc_core::sc_tie::value(sc_dt::sc_uint<8>(38)));
322
323      SIGNAL_MODULE.output(sc_core::sc_unbound);
324    }
325  };
```

- Tie and unbound

SYSTEMC HIERARCHY API

Object Hierarchy in SystemC

- sc_object-and-its-descendants hierarchy
  - sc_module, sc_port, sc_export, sc_prim_channel, tlm_initiator_socket, etc.
  - Fixed at construction time
  - New objects are hierarchically nested in their parent
    (except for special cases, see TLM-2 sockets, port classes aggregating other ports)
- Process & sc_event hierarchy
  - Two separate hierarchies
  - Parallel but unified!
  - Evolve with creation/destruction of instances during simulation

Hierarchical names in practice

- Method name() returns the full hierarchical name of hierarchical object
  - Companion basename() method (NEW: basename() in sc_process_handle)

- Hierarchical name also appears in introspection tools

- Conflict between names are checked upon object creation
- sc_gen_unique_name() allows avoiding duplicates

Issues with hierarchy

- Multiple corner cases
  - Need to know current hierarchy outside of any proper sc_object contexts
  - Need to create objects
    - At construction time
    - But outside the constructor of the corresponding object/module

- Other object hierarchies also existing!
  - Configuration Parameter (ex: CCI)
  - Other simulations (ex: UVM SystemC, other languages)
  - How to reconcile everyone?

Additions

- Getting hierarchical context from anywhere
  - sc_get_current_sc_object() returns current “parent”
- Controlling hierarchical point of instantiation for objects
  - Allows instantiation at top level…
  - … or from a given hierarchical point
  - Use a new class: sc_hierarchy_scope
    - sc_object::get_hierarchy_scope()
    - sc_hierarchy_scope::get_root()

Hierarchy scope example

IEEE 1666-2011

```
330  SC_MODULE(test_hierarchy_scope) {
331    my_module   *OPTIONAL_MODULE;
332
333    SC_CTOR(test_hierarchy_scope) :
334      OPTIONAL_MODULE(0) {
335    }
336
337    void enable_optional() {
338      OPTIONAL_MODULE = new my_module("OPTIONAL_MODULE");
339    }
340
341    ~test_hierarchy_scope() {
342      delete OPTIONAL_MODULE;
343    }
344
345  };
346
347  //------------------------------
348  int sc_main(int, char **) {
349    test_hierarchy_scope HIERARCHY_SCOPE_MODULE;
350
351    HIERARCHY_SCOPE_MODULE.enable_optional();
352
353    sc_core::sc_start();
354
355    return 0;
356  }
```

Hierarchy scope example

IEEE 1666-2023

```
361 SC_MODULE(test_hierarchy_scope) {
362   my_module   *OPTIONAL_MODULE;
363
364   SC_CTOR(test_hierarchy_scope) :
365     OPTIONAL_MODULE(0) {
366   }
367
368   void enable_optional() {
369     sc_core::sc_hierarchy_scope scope = get_hierarchy_scope();
370
371     OPTIONAL_MODULE = new my_module("OPTIONAL_MODULE");
372   }
373
374   ~test_hierarchy_scope() {
375     delete OPTIONAL_MODULE;
376   }
377
378 };
379
380 //--------------------------------
381 int sc_main(int, char **) {
382   test_hierarchy_scope SC_NAMED(HIERARCHY_SCOPE_MODULE);
383
384   HIERARCHY_SCOPE_MODULE.enable_optional();
385
386   sc_core::sc_start();
387
388   return 0;
389 }
```

Hierarchy scope example

IEEE 1666-2023

modern version

```
395  SC_MODULE(test_hierarchy_scope) {
396    std::unique_ptr<my_module>  OPTIONAL_MODULE;
397
398    SC_CTOR(test_hierarchy_scope) {
399    }
400
401    void enable_optional() {
402      sc_core::sc_hierarchy_scope scope = get_hierarchy_scope();
403
404      OPTIONAL_MODULE = std::make_unique<my_module>("OPTIONAL_MODULE");
405    }
406
407  };
408
409  //------------------------------
410  int sc_main(int, char **) {
411    test_hierarchy_scope SC_NAMED(HIERARCHY_SCOPE_MODULE);
412
413    HIERARCHY_SCOPE_MODULE.enable_optional();
414
415    sc_core::sc_start();
416
417    return 0;
418  }
```

Additions (2)

- Reconciling with other hierarchies
  - Query whether a hierarchical name already exist
    - sc_hierarchical_name_exists()
  - “Book” a given hierarchical name
    - sc_register_hierarchical_name()
    - sc_unregister_hierarchical_name()

ISO C++17 IN SYSTEMC API

C++17 in SystemC API

- C++17 new baseline for SystemC

- What about C++17 features use in the API?
  - Indirect uses
    - SC_NAMED (in-class direct initialization)
    - sc_assert ([[noreturn]] attribute)
  - Direct uses
    - sc_time (std::string_view)
    - sc_[un]register_hierarchical_name() (std::string_view)

C++17 in SystemC API (2)

- Why isn't there string_view everywhere?
  - Compatible with const char *… more or less, what about NULL/nullptr?
- Paving the way for the future
  - Used string_view in new APIs
  - Retained const char * in several locations for this revision
  - Introduced several restrictions to prepare for string_view
    - get_log_file_name() no longer returns NULL
    - Passing NULL/nullptr to SystemC APIs using const char * as parameter is now illegal
- More to do on other topics in next revisions!

GRAB BAG

Grab bag

- Logic vector & Fixed-point datatypes bit references
- Thorough test and update of code examples in LRM
- Inclusive language
- Cleanup of definition for sc_argc & sc_argv to match ISO C++
- Cleanup of const qualifiers in return values

Grab bag (2)

- sc_object missing virtual destructor
- Proper restrictions on sc_start(float)
- TLM clarifications & bug fixes
  - TLM-1 tlm::tlm_fifo_get_if disambiguation
  - TLM-2 Request/Response rule clarified for multi-sockets
  - TLM-2 set_extension/set_auto_extension errata
  - TLM-2 Non-blocking Transport State Diagram fix

Grab bag (3)

- Report handlers
  - New get_handler() to return current handler
  - Proper definition of SC_DEFAULT_INFO_ACTIONS, etc. in sc_core namespace
  - sc_assert() now leveraging [[noreturn]]

- sc_get_status() can now be called from external thread
- Base classes and virtual functions
  - sc_port_base, sc_export_base: new get_interface(), get_interface_type()
  - New template-free base class for TLM-2 sockets: tlm::tlm_base_socket_if

Grab bag (4)

- Removal of all numeric values for enum constants
  - Actuel numeric value is implementation-defined
  - Examples: sc_starvation_policy, sc_stop_mode, sc_status, …
  - Allows proper addition of new constants without ordering problems
  - Retain proper logical wise operations when needed

Errata

- Some errata yet to be published on IEEE side
  - Stage callbacks (p51 &60):
    ```
    void sc_register_stage_callback( const sc_stage_callback_if&, int );
    void sc_unregister_stage_callback( const sc_stage_callback_if&, int );
    ```
    should be replaced with:
    ```
    void sc_register_stage_callback( sc_stage_callback_if&, int );
    void sc_unregister_stage_callback( sc_stage_callback_if&, int );
    ```
  - None event, definition p124 should read:
    ```
    static const sc_event & none();
    ```
  - Version number, IEEE_1666_SYSTEMC should read 202301L