IEEE 1666-2023 UPDATE

JÉRÔME CORNET

P1666 WORKING GROUP CHAIR

SYSTEMC EVOLUTION FIKA, SEPTEMBER 2023

QUICK REFRESHER

- **SystemC**
- “Language”, Simulation library, Set of C++ classes…

- **IEEE 1666**
- Normative document governing SystemC implementation and tools, “Language Reference Manual”

- **SystemC Implementation**
- All related C++ classes (modules, ports, etc.) with simulation kernel
- Different implementation choices possible, added value
- Example implementations available: Accellera, Cadence, Siemens, Synopsys…

- **SystemC Tool**
- Example: Synthesis tool, Linter

IEEE P1666 WORKING GROUP ENTITIES

- Accellera
- Analog Devices
- ARM
- Cadence
- Intel
- Siemens (Mentor Graphics)
- Nvidia
- NXP
- STMicroelectronics
- Synopsys

IEEE P1666 WORKING GROUP OFFICERS

- Chair: Jérôme Cornet
- Vice-Chair: Praveen Wadikar
- Secretary: Mike Meredith

CONTRIBUTIONS ORIGIN

- **Previous revisions**
- Accellera (previously OSCI) + IEEE Working Group Members
- Example: sc_vector in IEEE 1666-2011: OFFIS/Philipp Hartman

- **This revision**
- Accellera, SystemC Language Working Group

- Original ideas based on forum question, LWG member need, etc.
- Cleanup of 10+ years of proposals
- Prototyping of implementation in Accellera SystemC implementation
- LRM draft written during IEEE standardization as joint effort between IEEE WG and Accellera LWG

- IEEE WG work started in November 2018

CONTRIBUTIONS CATEGORIES

- Document modernization
- Erratas
- Clarifications
- C++ Standard, Interactions with C++ compilers
- Productivity
- Future evolutions planning
- Advanced features
- Inclusive language

CONTRIBUTIONS CATEGORIES

- Document modernization
- Erratas
- Clarifications
- C++ Standard, Interactions with C++ compilers
- Productivity
- Future evolutions planning
- Advanced features
- Inclusive language

C++ STANDARD

C++ VERSION

- **New baseline for every conforming implementation and tools:**

C++17

- **Minimal version required**
- **Implementation and tools may support more recent versions!**

- **Consequences**
- Support for modern C++ code by all implementations
- Used by the SystemC standard itself!

PRODUCTIVITY: OBJECTS INITIALIZATION

OBJECTS INITIALIZATION

IEEE 1666-2011

```
SC_MODULE(my_module) {

  sc_core::sc_in<bool> clock;
  sc_core::sc_in<bool> reset;
  sc_core::sc_signal<bool> my_signal;

  …

  SC_CTOR(my_module) :
    clock("clock"),
    reset("reset"),
    my_signal("my_signal") {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

OBJECTS INITIALIZATION

IEEE 1666-2011 + C++ 11

```
SC_MODULE(my_module) {

  sc_core::sc_in<bool> clock{"clock"};
  sc_core::sc_in<bool> reset{"reset"};
  sc_core::sc_signal<bool> my_signal{"my_signal"};

  …

  SC_CTOR(my_module) {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

OBJECTS INITIALIZATION

IEEE 1666-2023

```
SC_MODULE(my_module) {
  sc_core::sc_in<bool> SC_NAMED(clock);
  sc_core::sc_in<bool> SC_NAMED(reset);
  sc_core::sc_signal<bool> SC_NAMED(my_signal);

  …

  SC_CTOR(my_module) {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

OBJECTS INITIALIZATION

IEEE 1666-2023

Supporting additional constructor parameters:

```
SC_MODULE(my_module) {

  sc_core::sc_in<bool> SC_NAMED(clock);
  sc_core::sc_in<bool> SC_NAMED(reset);
  sc_core::sc_signal<bool> SC_NAMED(my_signal, true);

  …

  SC_CTOR(my_module) {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

PRODUCTIVITY: MODULE WITH PARAMETERS

MODULE WITH PARAMETERS

IEEE 1666-2011

```
SC_MODULE(my_module) {

  sc_core::sc_in<bool> clock;
  sc_core::sc_in<bool> reset;
  sc_core::sc_signal<bool> my_signal;

  …

  SC_HAS_PROCESS(my_module)
  my_module(const sc_module_name & module_name,
            int my_parameter) :
    sc_core::sc_module(module_name),
    clock("clock"),
    reset("reset"),
    my_signal("my_signal") {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

MODULE WITH PARAMETERS

IEEE 1666-2023

No longer needed

```
SC_MODULE(my_module) {

  sc_core::sc_in<bool> clock;
  sc_core::sc_in<bool> reset;
  sc_core::sc_signal<bool> my_signal;

  …

  ~~SC_HAS_PROCESS(my_module)~~
  my_module(const sc_module_name & module_name,
            int my_parameter) :
    sc_core::sc_module(module_name),
    clock("clock"),
    reset("reset"),
    my_signal("my_signal") {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

MODULE WITH PARAMETERS

IEEE 1666-2023

SC_CTOR supports additional parameters

```
SC_MODULE(my_module) {

  sc_core::sc_in<bool> clock;
  sc_core::sc_in<bool> reset;
  sc_core::sc_signal<bool> my_signal;

  …

SC_CTOR(my_module, int my_parameter) :
    clock("clock"),
    reset("reset"),
    my_signal("my_signal") {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

MODULE WITH PARAMETERS

IEEE 1666-2023

SC_CTOR + SC_NAMED usage:

```
SC_MODULE(my_module) {

  sc_core::sc_in<bool> SC_NAMED(clock);
  sc_core::sc_in<bool> SC_NAMED(reset);
  sc_core::sc_signal<bool> SC_NAMED(my_signal);

  …

  SC_CTOR(my_module, int my_parameter) {
    SC_METHOD(my_method);
    sensitive << clock.pos();
  }

};
```

WRAP-UP

- **SC_HAS_PROCESS() no longer required, and now deprecated**

- **New SC_NAMED() macro**
- Automatic stringification of the first argument
- Variadic, supports extra parameters passed to the constructor of the corresponding object

- **SC_CTOR() now variadic**
- Extra parameters added to the declaration of the class constructor

PRODUCTIVITY: SC_VECTORS SENSITIVITY AND ADDITIONS

SC_VECTORS SENSITIVITY

IEEE 1666-2011

```
SC_MODULE(my_module) {

  sc_core::sc_vector<sc_core::sc_signal<bool> > signals;

  …

  SC_CTOR(my_module) :
    signals("signals", 8) {

    SC_METHOD(my_method);
    for (size_t i=0; i<signals.size(); i++) {
      sensitive << signals[i];
    }
  }

};
```

SC_VECTORS SENSITIVITY

IEEE 1666-2023

```
SC_MODULE(my_module) {

  sc_core::sc_vector<sc_core::sc_signal<bool>> SC_NAMED(signals,
                                                        8);

  …

  SC_CTOR(my_module) {
    SC_METHOD(my_method);
    sensitive << signals;
  }

};
```

WRAP-UP

- **sensitive now supports sc_vector**

- **… and actually everything that supports std::begin() and std::end()**
- Any container defining begin() and end()
- Arrays of pointers too!

SC_VECTORS ADDITIONS

IEEE 1666-2011

```
SC_MODULE(my_module) {

  sc_core::sc_vector<sc_core::sc_signal<bool> > signals;

  …

  SC_CTOR(my_module) :
    signals("signals", 8) {

    // No possibility to add elements
  }

};
```

IEEE SA STANDARDS ASSOCIATION

IEEE

SC_VECTORS ADDITIONS

IEEE 1666-2023

Adding elements with name and constructor parameters

```
SC_MODULE(my_module) {

  sc_core::sc_vector<sc_core::sc_signal<bool>> SC_NAMED(signals,
                                                         8);

  …
  SC_CTOR(my_module) {
    signals.emplace_back_with_name("signals_8", false);
    signals.emplace_back_with_name("signals_9", true);
  }

};
```

SC_VECTORS ADDITIONS

IEEE 1666-2023

Adding elements with constructor parameters only

```
SC_MODULE(my_module) {

  sc_core::sc_vector<sc_core::sc_signal<bool>> SC_NAMED(signals,
                                                         8);

  …
  SC_CTOR(my_module) {
    signals.emplace_back(false);
    signals.emplace_back(true);
  }

};
```

WRAP-UP

- **It is now possible to add elements to a vector after a first init()**

- **emplace_back()**
- Automatically generate the name, pass arguments to the object's constructor

- **emplace_back_with_name()**
- Allows providing a custom name as first parameter, useful for collection of heterogeneous objects

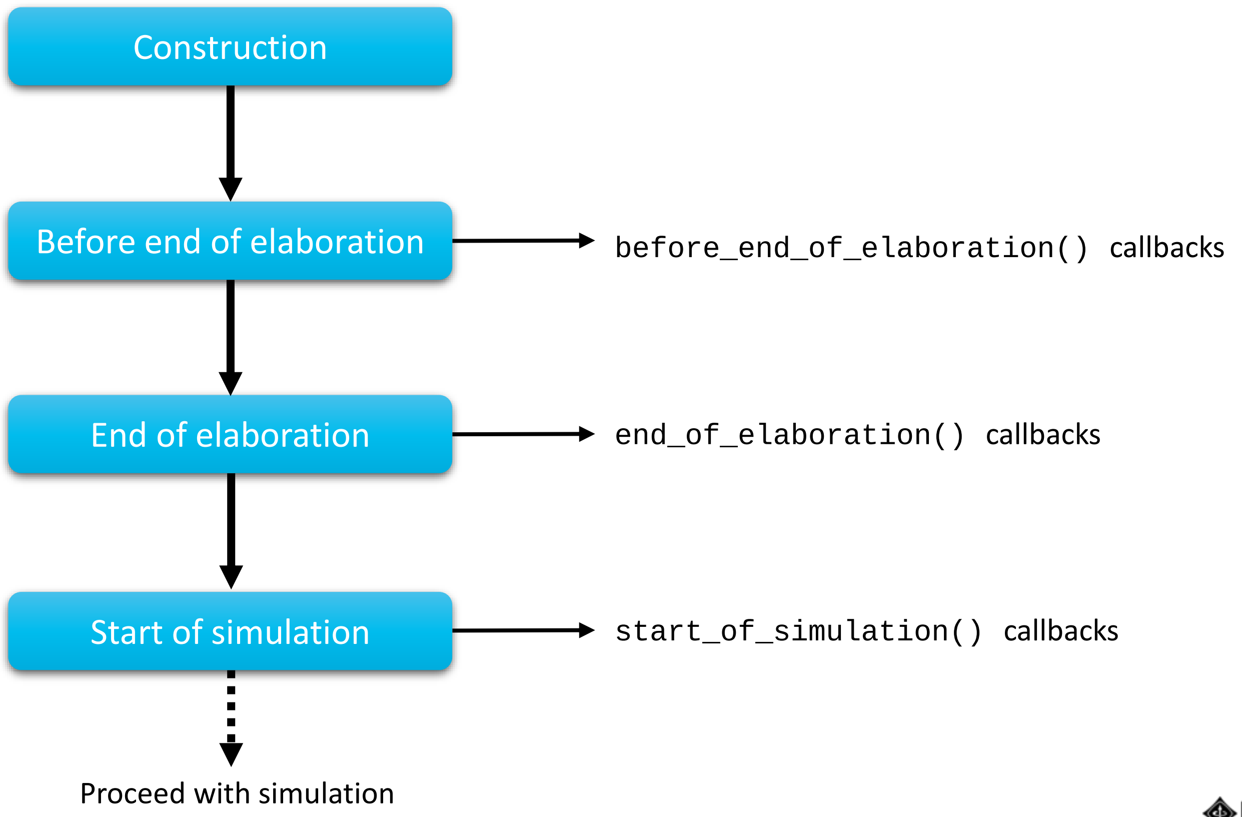
- **A vector is "locked", by default at end of init (i.e. construction time)**
- No longer possible to add elements afterwards

- **Unless a new constructor parameter "SC_VECTOR_LOCK_AFTER_ELABORATION" is used!**

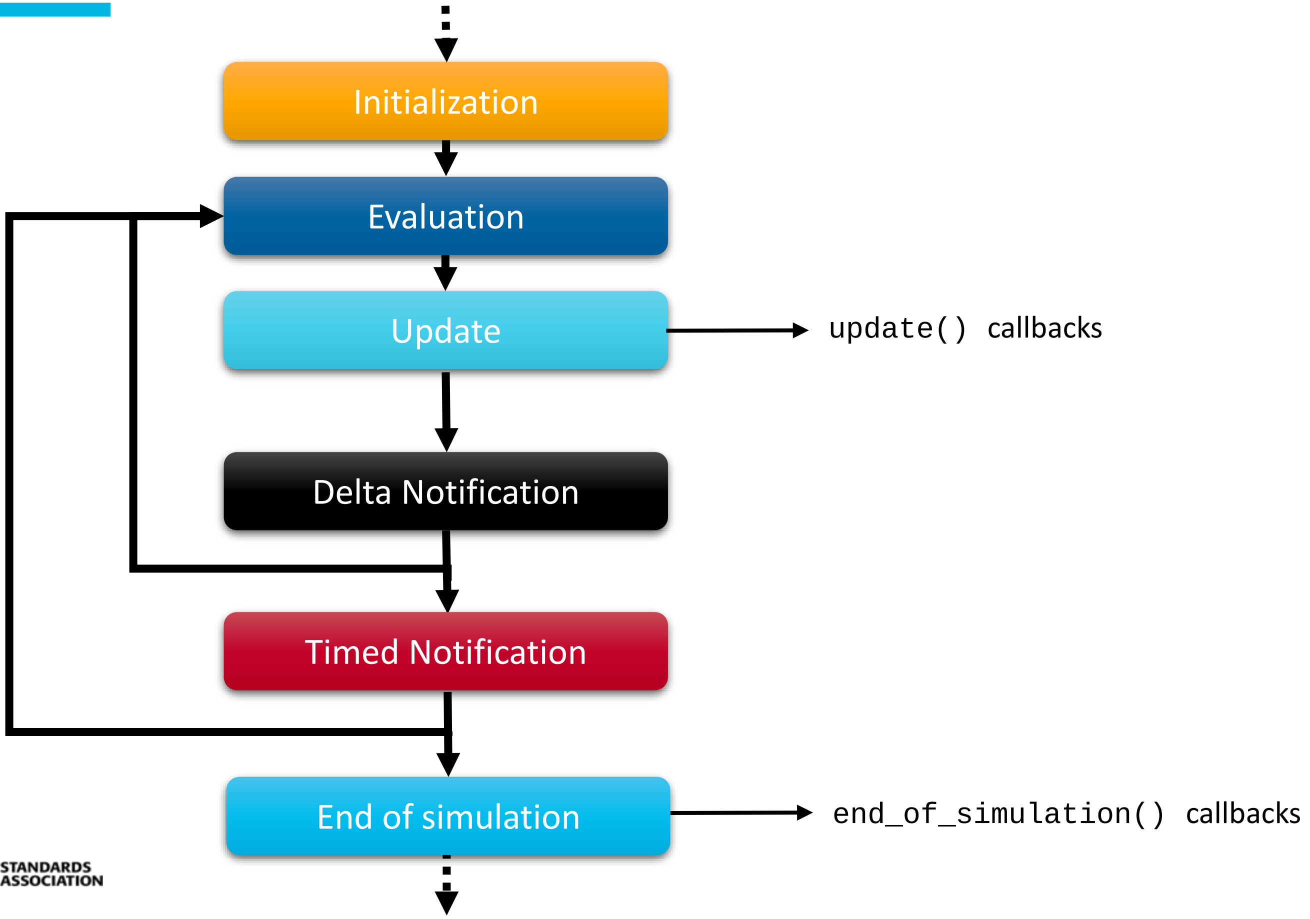
ADVANCED FEATURES

ADVANCED FEATURES: SIMULATION STAGE CALLBACKS

CLASSICAL SCHEDULER PHASES: ELABORATION


Construction
Before end of elaboration
End of elaboration
Start of simulation
before_end_of_elaboration() callbacks
end_of_elaboration() callbacks
start_of_simulation() callbacks
Proceed with simulation

CLASSICAL SCHEDULER PHASES: SIMULATION



PHASE VS STAGE CALLBACKS

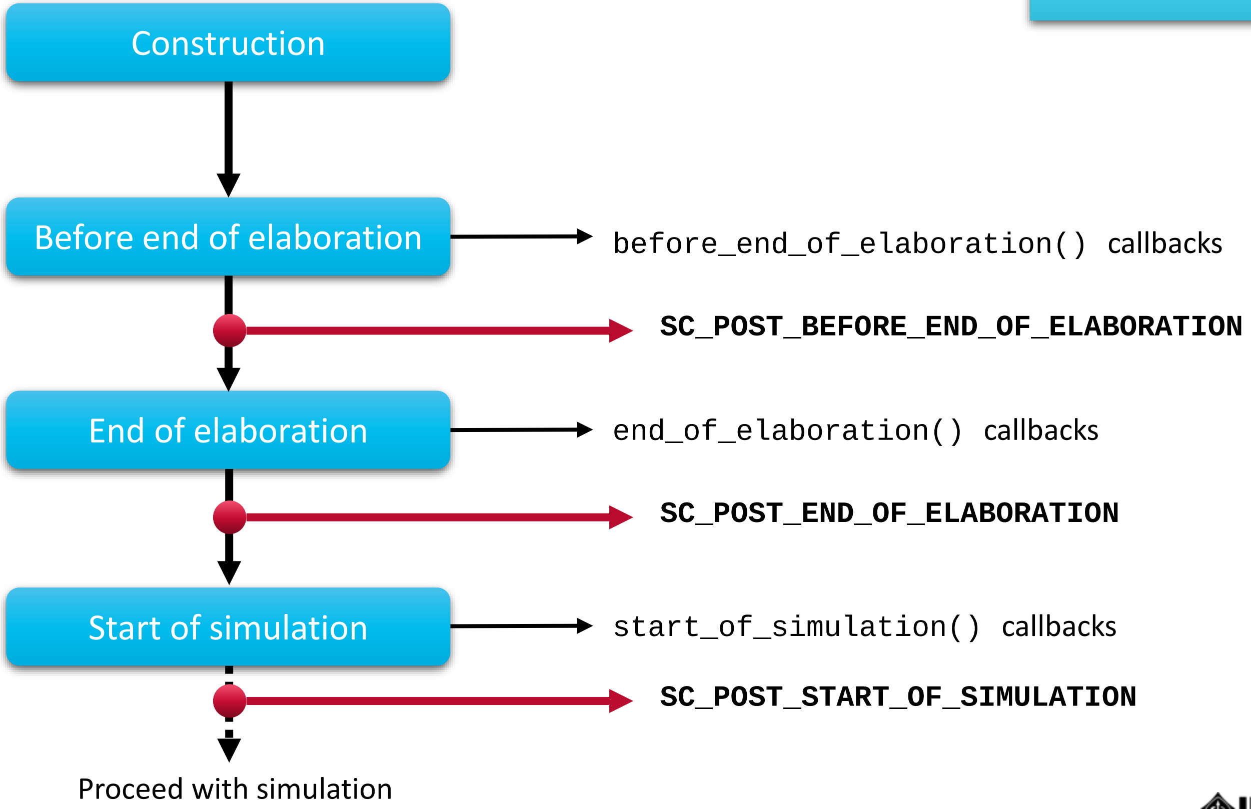
- **Classical “phases” callbacks**
- Used by models for implementation purposes
- Examples:
  - Instantiating new modules at before_end_of_elaboration
  - Delta notifications in the update callback of a sc_prim_channel instance
- Order of execution: implementation-defined

- **Need for other callbacks for “observing” the simulation**
- First attempt during 2011 revision
- Second attempt backed by implementation done by Philipp Hartmann (OFFIS)
- Access to “points in the simulation” previously not reachable
- “Observation” spirit: no possibility to modify the simulation (no object instantiation, event notifications, etc.)

$\longrightarrow$ New **Simulation Stage Callbacks**!

STAGE CALLBACKS DURING ELABORATION

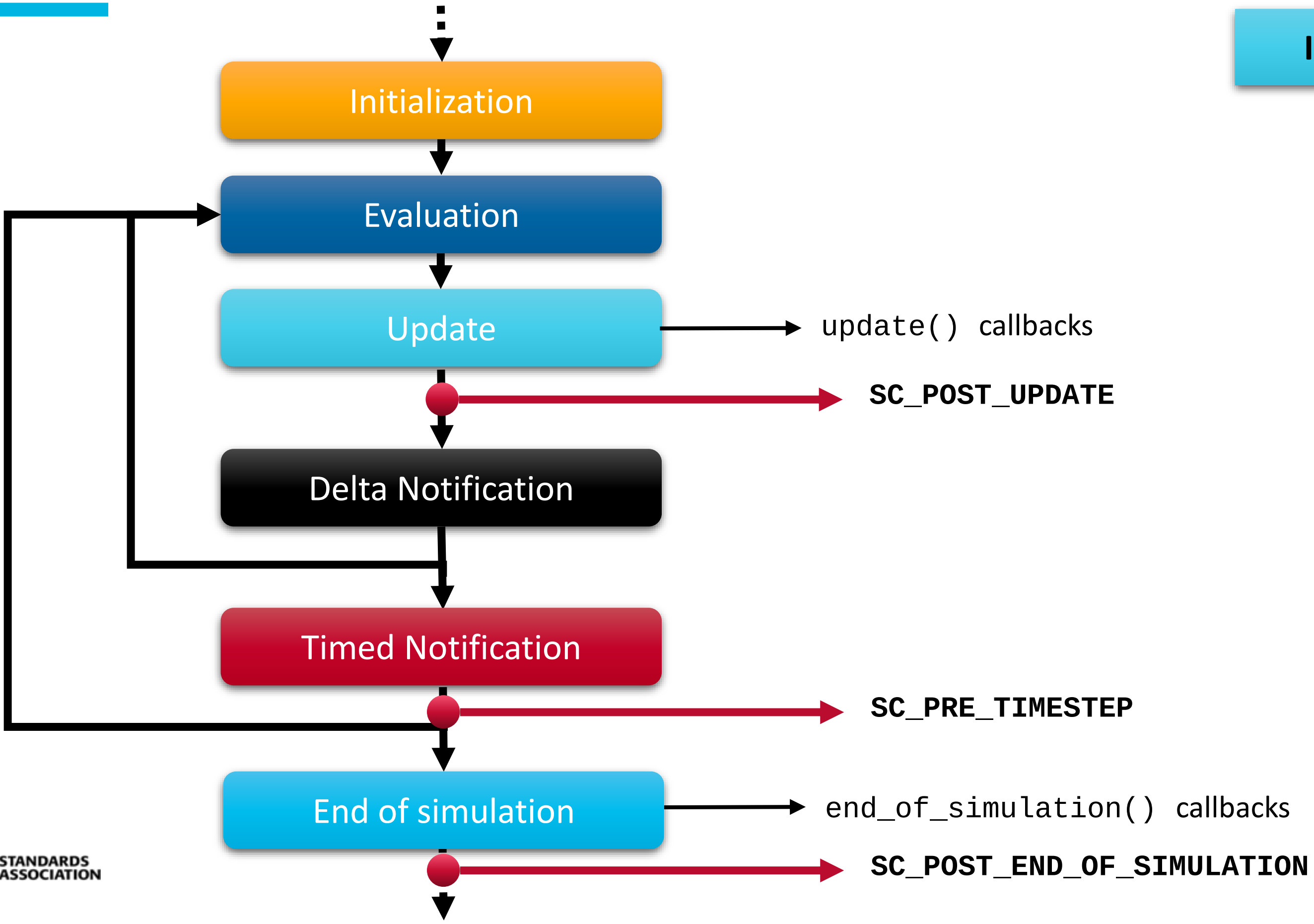
IEEE 1666-2023



Construction
Before end of elaboration
before_end_of_elaboration() callbacks
SC_POST_BEFORE_END_OF_ELABORATION
End of elaboration
end_of_elaboration() callbacks
SC_POST_END_OF_ELABORATION
Start of simulation
start_of_simulation() callbacks
SC_POST_START_OF_SIMULATION
Proceed with simulation

STAGE CALLBACKS DURING SIMULATION

IEEE 1666-2023



STAGE CALLBACKS

- **No need for callback to be located in a sc_module**
- sc_stage_callback_if C++ interface
- One virtual method to implement receiving a sc_stage ENUM (with value SC_POST_UPDATE, etc.)

- **Registration/unregistration using a single function with ENUM flag combination**
- Example:

```
    sc_register_stage_callback(*this, SC_POST_UPDATE | SC_PRE_TIMESTEP)
```

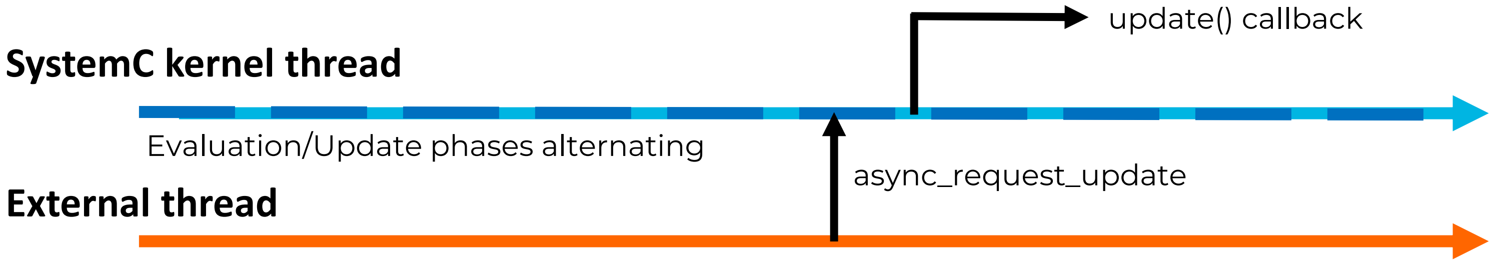
- **Also available before simulation "state" changes**
- SC_PRE_PAUSE, SC_PRE_STOP, etc.

ADVANCED FEATURES: SUSPENSION

SUSPENSION

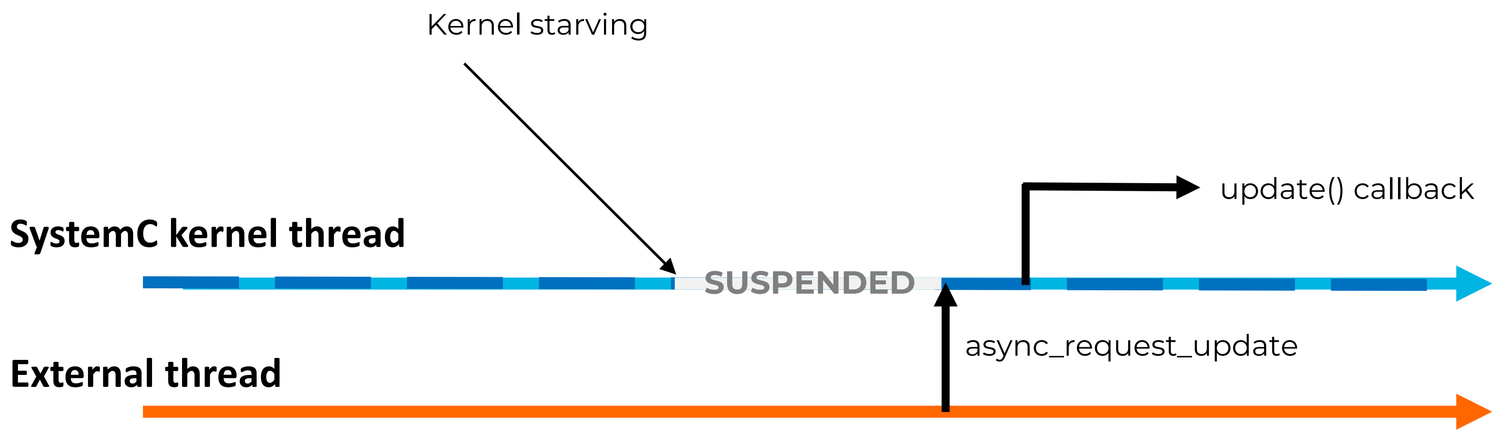
- **Context: interactions of the SystemC kernel with other OS threads**
- 2011 Revision introduced async_request_update() thread-safe call

- **Classical issue: kernel starving before receiving any async_request_update()**
- Workarounds: “keep-alive” mechanisms
- Problem: keep-alives not aware of each other



SUSPENSION

- **Idea: sc_prim_channel instances can opt-in for suspension in case of starvation**
- Call to async_attach_suspending
- Kernel enters a special Suspended state
- Simulation resumes on first async_request_update() received



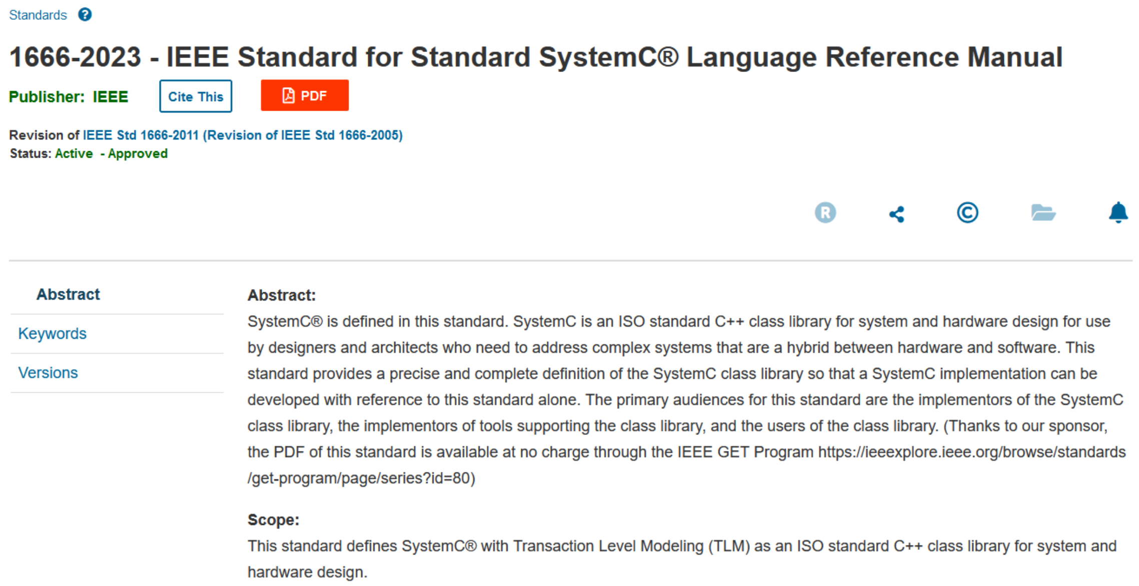
SUSPENSION

- Suspended state also possible through explicit request: sc_suspend_all, sc_unsuspend_all
- Possibility to opt-out for suspension at process level

CONCLUSION

AVAILABILITY

- **IEEE 1666-2023 is available!**
  - https://www.accellera.org/downloads/ieee
  - https://ieeexplore.ieee.org/document/10246125



Standards

1666-2023 - IEEE Standard for Standard SystemC® Language Reference Manual

Revision of IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)
Status: Active - Approved

Abstract:
SystemC® is defined in this standard. SystemC is an ISO standard C++ class library for system and hardware design for use by designers and architects who need to address complex systems that are a hybrid between hardware and software. This standard provides a precise and complete definition of the SystemC class library so that a SystemC implementation can be developed with reference to this standard alone. The primary audiences for this standard are the implementors of the SystemC class library, the implementors of tools supporting the class library, and the users of the class library. (Thanks to our sponsor, the PDF of this standard is available at no charge through the IEEE GET Program https://ieeexplore.ieee.org/browse/standards/get-program/page/series?id=80)

Scope:
This standard defines SystemC® with Transaction Level Modeling (TLM) as an ISO standard C++ class library for system and hardware design.

CONCLUSION

- **Many features, fixes in domains from productivity to advanced features**

- **Lessons learned**
- Commits with code in github is not enough!
- Contributions should include:
  - A sample implementation…
  - … well separated from a rigorous specification.
  - Specification should provision at least some corner cases

- Thanks to all contributors which made it possible to turn initial code into proper LRM elements

THANK YOU

Jérôme Cornet

jerome.cornet [a t] st.com