SystemC Multi-kernel support and thread safety

Contributions from SystemC LWG,
MachineWare GmbH

COSEDA Technologies GmbH

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION DAY
OCT 17, 2024 | MUNICH | GERMANY

Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **SystemC LWG members, MachineWare and Coseda** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on the standard.

Outline

Open room discussion
Material is presented,
room participation is highly appreciated

- History
- Coseda: Parallel SystemC Simulation - Does this make sense?
- MachineWare: Parallel/Distributed/Multi-Kernel Simulation is solved
- ~~SystemC LWG: Simulations With Multiple sc_simcontexts~~
  - Dropped to timeconstraints
- Summary
- Questions/Discussion

History

- Selected history of SystemC Evolution Day/Fika coverage:
  - SCED 2016

Seven Obstacles in the Way of Parallel SystemC Simulation Abstract | Presentation	Rainer Dömer, University of California
Multi-Threaded SystemC and External Interfaces Abstract | Presentation	Mark Burton, Greensocs

  - SCED 2019

Pushing the Limits of Standard-Compliant Parallel SystemC Simulation Presentation	Rainer Dömer, University of California
Synchronizing simulators, and Save and Restore Presentation	Mark Burton, Greensocs

  - Fika April 2022

Topic: Parallelization of SystemC simulations sc_during, SCale, Intel Simics, RISC	**Fika link**

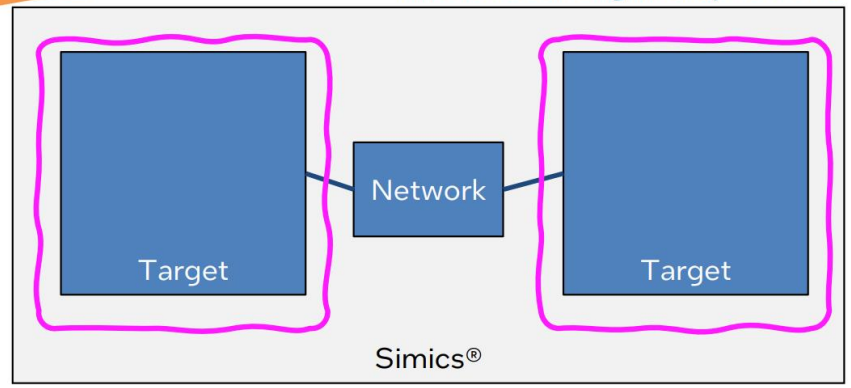
  - SCED 2022

Distributed simulation and SystemC Presentation	Mark Burton, Qualcomm

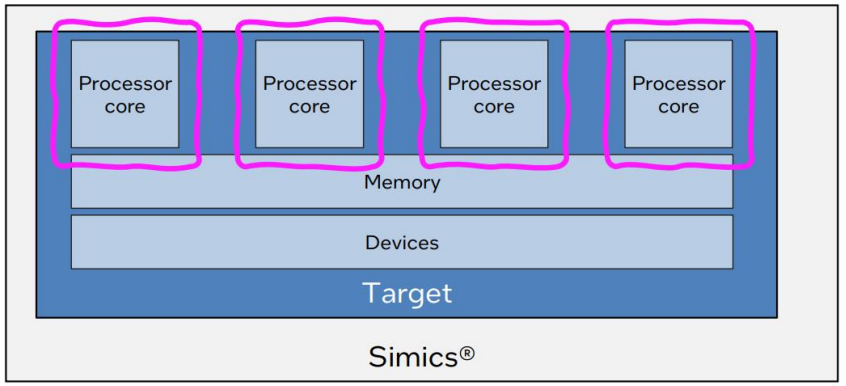
Threading in the Simics® Simulator – Use Cases

List is not exhaustive, and the different uses can be combined

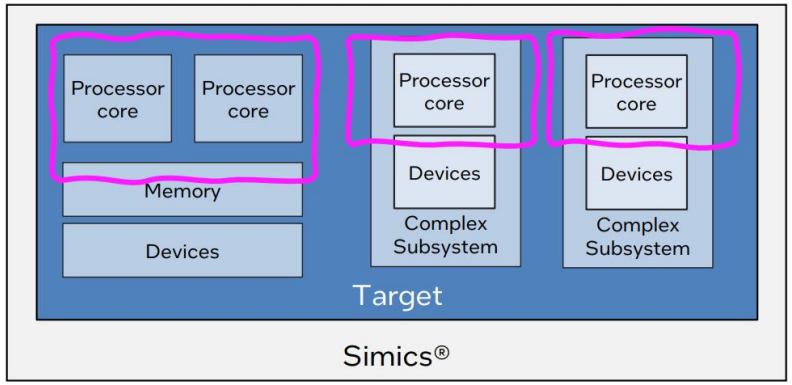
"Multimachine Accelerator"
Thread across long-latency networks



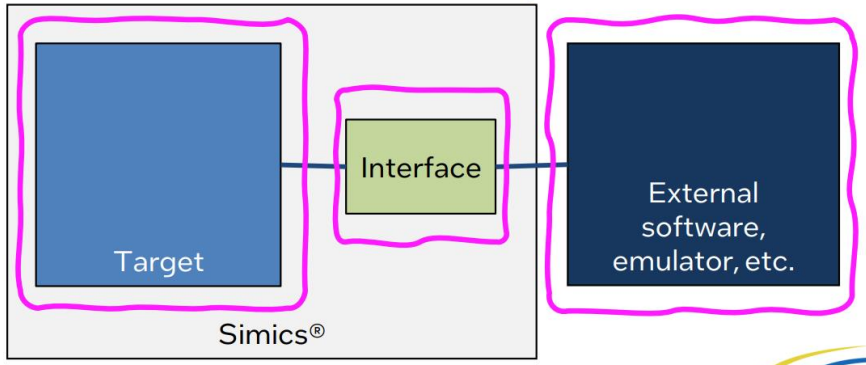
"Multicore Accelerator" (MCA)
Thread between processor cores sharing memory



"Subsystem multithreading"
Run separate (definition) subsystems on their own threads



Multithreading as coding pattern
Use a thread to interact with the outside asynchronous world

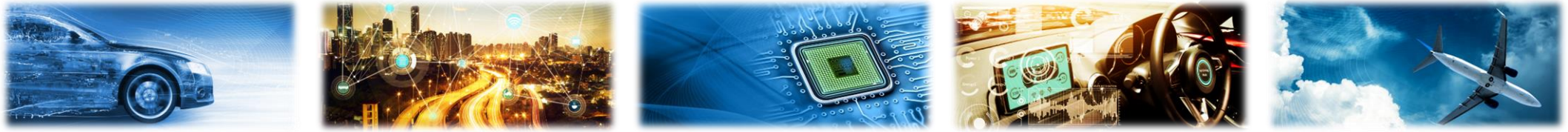


The Intel® Simics® Simulator and SystemC* and Threading, Jacob Engblom, Intel, Fika 04-2022

Parallel SystemC Simulation - Does this make sense?

COSEDA technologies

THE ANALOG AND DIGITAL SYSTEM LEVEL COMPANY



Parallel SystemC – Does this make sense?

SystemC Models

- Numerous method processes – moderate number of thread processes
- Negligible computation load per process
- Difficult to predict computation loads, changing over time loads
- Extremely high activations counts
- Tight coupling of processes
- Central/Global simulation kernel

Parallel SystemC – Does this make sense?

Requirements for successful parallelization

- High computational load per parallelized OS execution threads
- Decoupling (delay between) of parallelized OS execution threads
- Load balancing between parallelized OS execution threads
- Certain level of thread safety

Challenges:

- Causality / Reproducibility
- Debuggability
- Early speedup saturation – or slow down due synchronization overhead, cache and branch prediction effects
- Implementation effort

Parallel SystemC – Does this make sense?

Implementation Challenge – Thread Safety

- Numerous (global) memory manager
- (Global) result caching
- Numerous (global) static variables
- Which functions/resources must be accessible from different OS threads

```
int main(int argn,char* argc[])
{
   std::thread my_thread(
   []()
   {
     sc_dt::sc_uint<64> varth=42;
     std::uint64_t cnt=0;
     while(cnt++<10000000) varth.to_string();
   });


   sc_dt::sc_uint<64> varm=24;
   std::uint64_t cnt=0;
   while(cnt++<10000000) varm.to_string();

   my_thread.join();


   std::cout << "Test was successful" << std::endl;

   return 0;
}
```

-> Segmentation Fault in .to_string()

Parallel SystemC – Does this make sense?

There are (cheaper) alternatives?

- Checkpointing
- Shorting/split simulation scenarios – run scenarios parallel
- Optimizing single kernel performance

Parallel SystemC – Does this make sense?

Parallelizing SystemC AMS

- Dataflow (TDF) cluster have usually significant computational load (can consist a lot modules and may equation solver)
- Delays are explicitly modelled and can be analyzed and used for parallelization
- SystemC AMS time axis are still decoupled – no global data structure/scheduler
- Challenges:
  - Implementation effort
  - SystemC thread safe issues

Parallel SystemC – Does this make sense?

SystemC AMS – Low hanging fruit parallelization

- SystemC AMS tracing has may a significant effort due to time axis synchronization
- Tracing has no feedback -> can be decoupled
- Parallelization implemented in current reference implementation – can be enabled by configure option
- May crashes if SystemC datatypes are traced (thread safe issue described in previous slides)
- Depending on the use case a significant speed up possible (if tracing dominates) – at least not yet an example with a performance decrease found

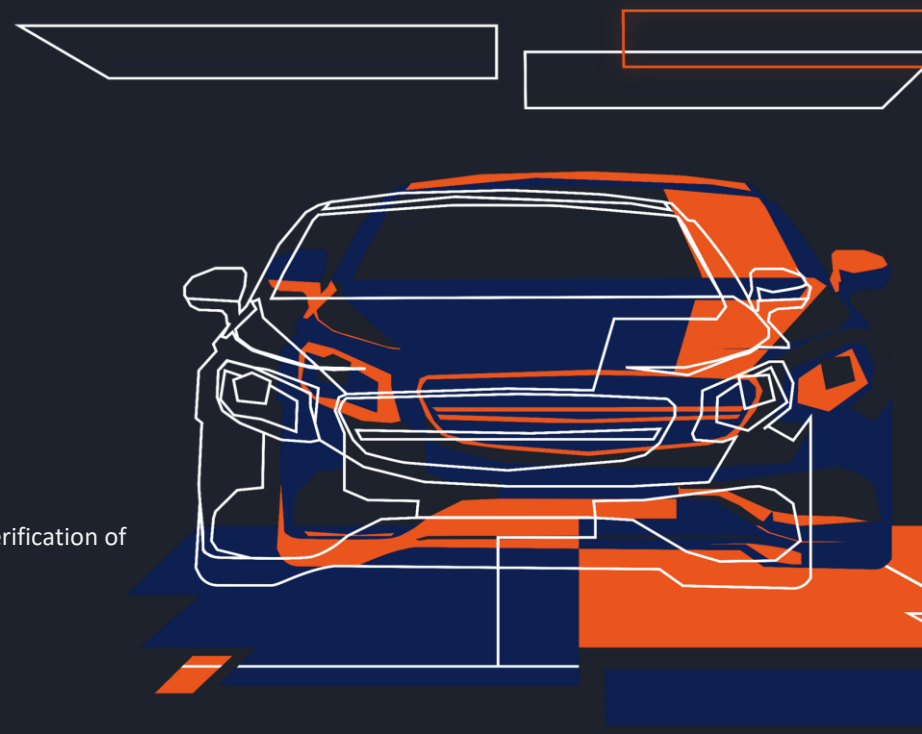
MachineWare GmbH

Virtual Hardware, Real Benefit

Hot Take:

Parallel/Distributed/Multi-Kernel Simulation is solved

theoretically…[1]
and we don't really NEED changes in SystemC



[1]: K. Chandy et al., "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," IEEE Transactions on Software Engineering, 1979.

MACHINEWARE

SystemC Multi-Kernel / Thread Safety - What does it mean?

- Scenario
  - Multiple entities compute in parallel with their own local simulation time
  - Data is sent between entities
  - Called: Parallel, Distributed, Multi-Kernel, Multi-Domain, Multi-Scale, …
- General Problem
  - How to move data? (easy)
  - How to synchronize time? (difficult)
  - Is it worth the effort? (difficult)
- SystemC-specific Problems
  - What does “SystemC” mean? TLM LT, TLM AT, Classic, AMS, Synthesis, …
  - What to standardize?
  - How to retroactively introduce thread-safety into an existing SW project/Standard/API?
    - E.g. what should be guaranteed thread-safe and what not?

What does it mean for SystemC?

- Goals: Faster Simulation, More Simulator Use Cases
- Similar methods published in conferences for many years (solved problem)
- Topics
  - Co-Simulation with other (SystemC) simulations
  - “Parallel” SystemC Simulation
    - Compute and Sync
    - Parallel SC_THREADS

Co-Simulation with other (SystemC) simulations

- Method
  - Instantiate many (SystemC) simulations processes (on many different hosts)
  - Connect for moving data and timing synchronization
- Examples
  - Vector SIL Kit (open-source, MIT license)
    - ❑ Synchronize time, move (bus) data between participants
    - ❑ SystemC integration fairly easy, no changes to standard required
  - FMI
  - And many more
- No kernel changes required
- What about deterministic execution?
  - Not trivial. Is it needed?

Parallel SystemC Simulation: Compute and Sync

- Method
  - Execute computation in parallel to SystemC (separate thread)
  - Synchronize time with SystemC when required
    - async_request_update, …
- Examples
  - VCML async, open-source, Apache-2.0
  - GreenSocs RunOnSysC, open-source, Apache-2.0
  - sc_during, open-source, LGPL
  - Probably many more …
- Similar to what QEMU does, but with more (SystemC) overhead
- No kernel changes required
- What about deterministic execution?
  - Not trivial. Is it needed?

Parallel SystemC Simulation: Parallel SC_THREAD

- Method
  1. Add API to SystemC to create parallel SC_THREADS and synchronize them
  2. …
  3. Profit

- Examples
  - Ventroux et al. - SCale (and more)
  - Weinstock et al. - SCope
  - And probably many more

- Needs kernel changes

- What about deterministic execution?

- What is the usecase?

What would be nice anyways? SystemC 4.0

- Thread safe SystemC API functions guaranteed by Standard
  - e.g. sc_timestamp, async_request_update, suspend, …

- Discussion: Parallel transport interface
  - Cannot call wait()
  - Do not assume SC_THREAD context
  - Do not assume called from main SystemC OS thread
  - Thread-safe
  - Can only annotate time using parameter passed by reference
  - What about event notify()?

Summary

- Running several (communicating) (SystemC) simulations in parallel is basically solved
  - Is it?
  - With limitations!
    - When running in multiple processes, interfaces are explicit and no inadvertent sharing is happening.
    - When running in the same process, it's get's ugly quickly.

- Running multiple kernels within the same process is tricky in practice
  - non-existent thread-safety of SystemC (e.g. datatypes, multiple simcontext)
  - things people use in their models (e.g. singletons, global variables, static variables etc.) that make parallelization non-straight forward in practice

- For coarse-grained parallelization in general, the FSS WG looks at the "hard" part of standardizing interfaces synchronization that works **effectively** and **efficiently** for all relevant use cases.
  - To align with FSS we need isolated SystemC 'islands' using a controlled interface (through static linking with symbol hiding of everything except the interface).

Summary

- Currently (multi-process, isolated kernels) are outside of SystemC's scope.
- However, some "asynchronous" interaction is described in the SystemC standard
  ➔ therefore, we must acknowledge processing outside of SystemC's main thread.

- Ideas for SystemC 4.0 or 3.x?
  ➔ rework the thread-safety guarantees for the kernel in certain key areas:
  - Datatypes — not thread-safe today
    - Not much standards work needed
    - Mostly implementation effort
  - Selected parts of the kernel/simcontext
    - State/Phase
    - Simulation time
    - More?
  - Tracing/Logging/Reporting
  - Combine/serialize results from multiple sources
  - (Co-)Simulation control
    - Calling sc_pause/"sc_continue" from the outside
  - Tool interfaces (CCI)

Note: we have to decide which usecase(s) we want to support! Multiple process (isolated) or multithreaded kernel

Questions?