Pushing the Limits of Standard-Compliant Parallel SystemC Simulation

Rainer Dömer

Center for Embedded and Cyber-Physical Systems

University of California, Irvine

accellera SYSTEMS INITIATIVE

SYSTEMC™ EVOLUTION DAY

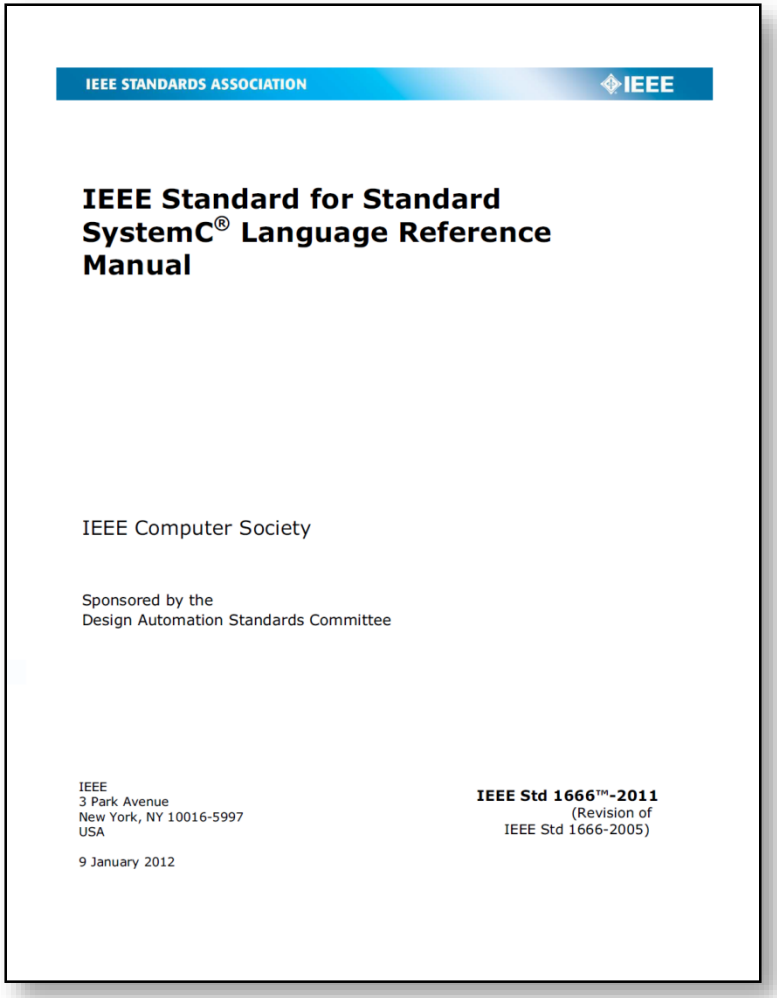
OCT 31, 2019 | MUNICH | GERMANY

Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Rainer Doemer, CECS,** 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 this standard.

Outline

4. Pushing the Limits of
3. Standard-Compliant
2. Parallel
1. SystemC Simulation


IEEE STANDARDS ASSOCIATION IEEE

IEEE Standard for Standard SystemC® Language Reference Manual

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee

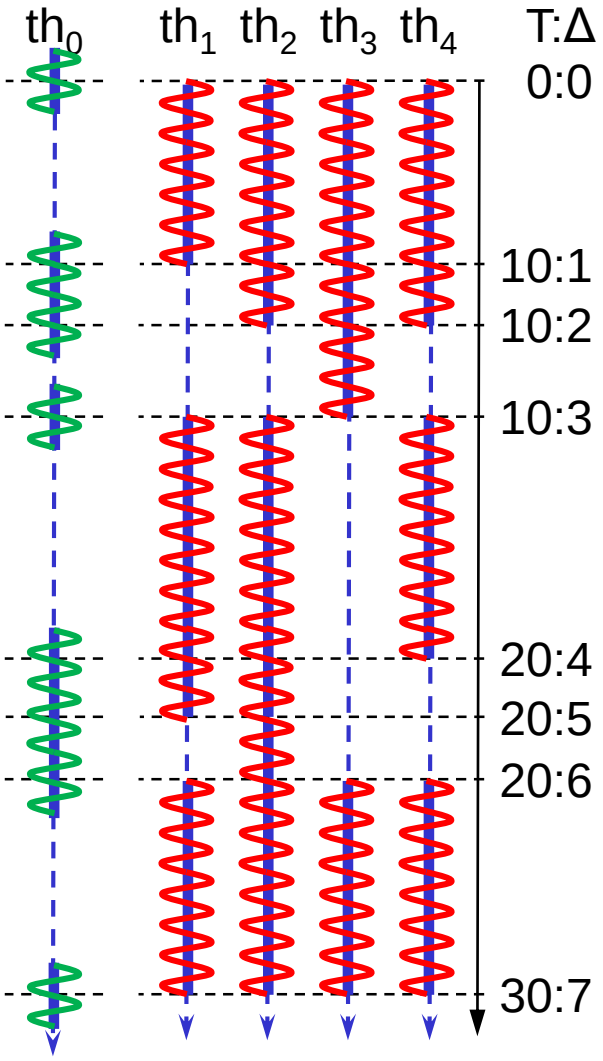
IEEE
3 Park Avenue
New York, NY 10016-5997
USA

9 January 2012

IEEE Std 1666™-2011
(Revision of
IEEE Std 1666-2005)

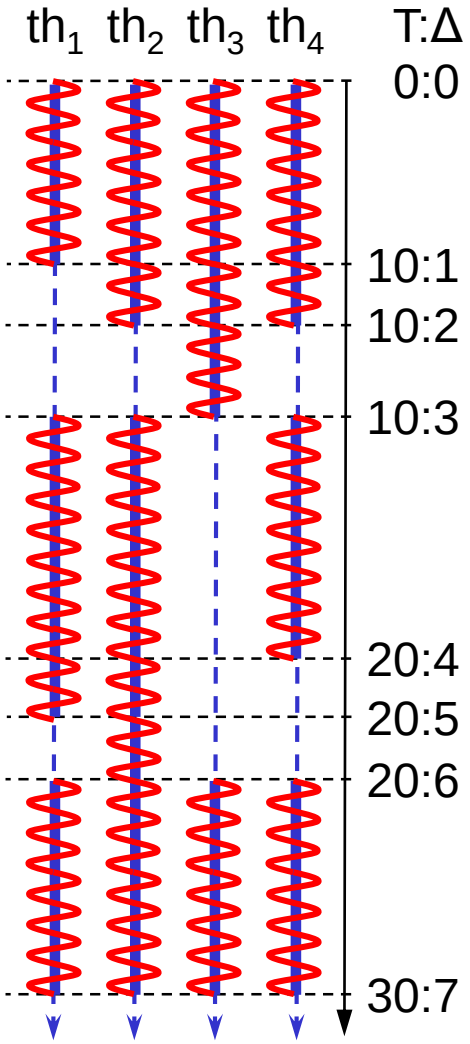
1. SystemC Simulation

- Discrete Event Simulation (DES)
  - Concurrent threads of execution
  - Managed by a central scheduler
  - Driven by events and time advances
    - Delta-cycle
    - Time-cycle
  - ➢ Partial temporal order with barriers
- Example
  - Accellera Proof-of-Concept Simulator
  - ➢ *Sequential, slow!*



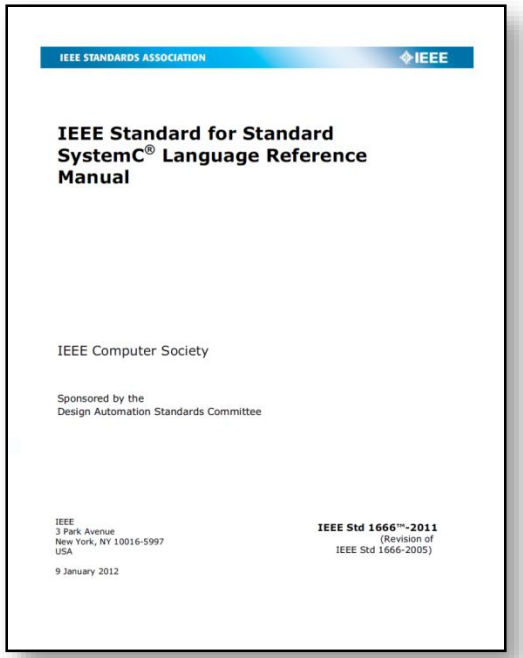
2. Parallel SystemC Simulation

- Parallel Discrete Event Simulation (PDES)
  - Concurrent threads of execution
  - Managed by a central scheduler
  - Driven by events and time advances
    - Delta-cycle
    - Time-cycle
  - ➢ Synchronous parallelism
  - ➢ Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle
  - ➢ *Order of magnitude faster!*



3. Standard-Compliant Parallel SystemC Simulation

- IEEE Standard 1666™-2011
  - Revision of IEEE Std. 1666-2005
  - Standard SystemC®
    Language Reference Manual



…unfortunately stands in the way of parallel SystemC simulation!

➢ SystemC Evolution Day 2016
  ➢ *"Seven Obstacles in the Way of Parallel SystemC Simulation",*
    Rainer Doemer, Munich, Germany, May 2016.
  ➢ SystemC standard
    ➢ … must embrace true parallelism
    ➢ … must evolve in a major revision (3.x)

4. Pushing the Limits …

- While the SystemC standard has not changed, my group has worked hard
  - *“Let’s make the best of it!”*
- Goals
  - Accept SystemC as it is (well, most of it)
  - Build the best parallel SystemC simulator possible
  - Aim for maximum compliance with the standard

- We took this risk, and created RISC!
  - *Recoding Infrastructure for SystemC*
- RISC pushes the limits to overcome the 7 obstacles …

Obstacle 1: Co-Routine Semantics

- Fact: IEEE 1666-2011 requires *co-operative multitasking*
  - Quotes from Section "4.2.1.2 Evaluation phase" (pages 17, 18):

> Since process instances execute without interruption, only a single process instance can be running at any one time, […]. A process shall not pre-empt or interrupt the execution of another process. This is known as *co-routine* semantics or *co-operative multitasking*.
> […]
> The scheduler is not pre-emptive. An application can assume that a method process will execute in its entirety without interruption, and a thread or clocked thread process will execute the code between two consecutive calls to function **wait** without interruption.

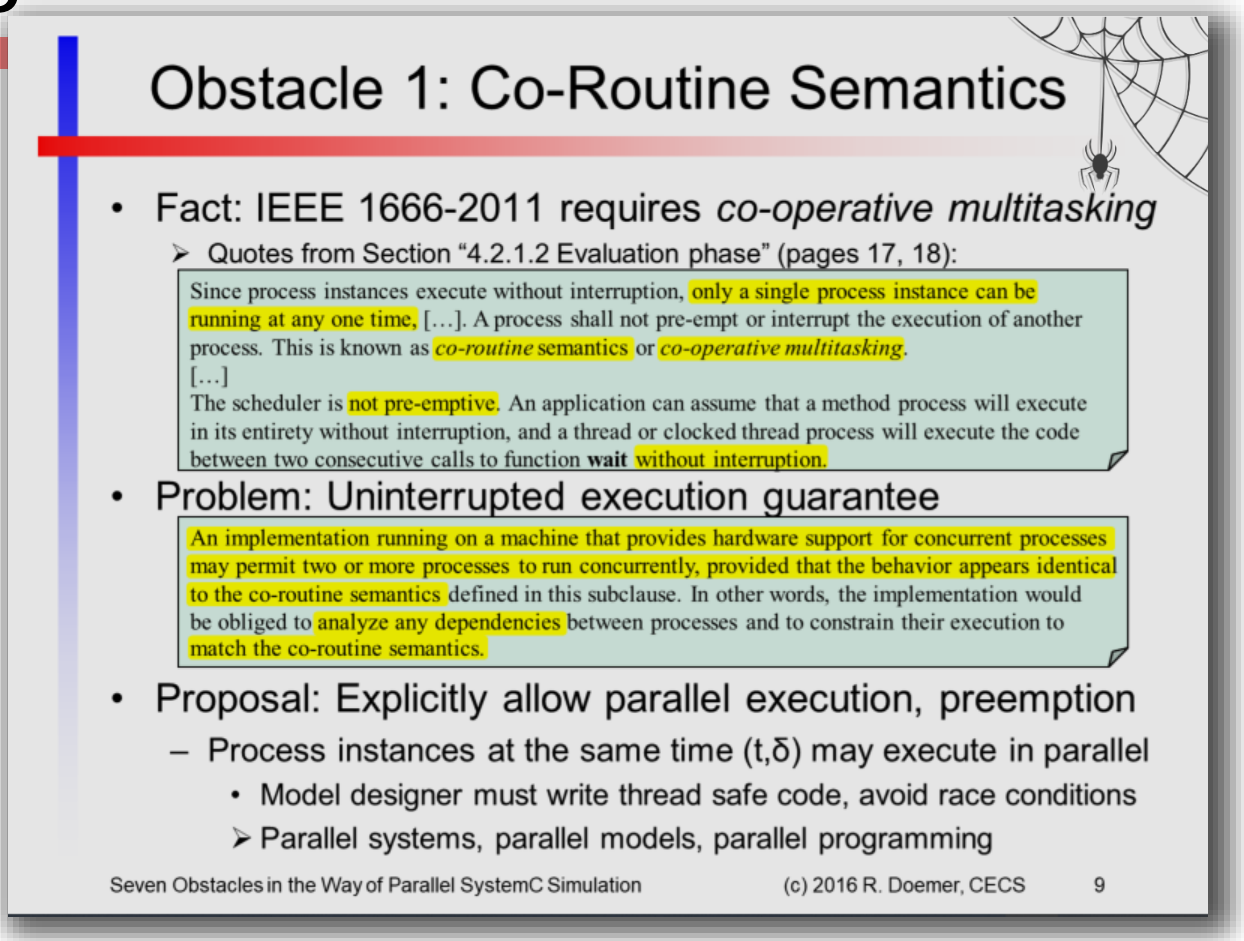
- Problem: Uninterrupted execution guarantee

> An implementation running on a machine that provides hardware support for concurrent processes may permit two or more processes to run concurrently, provided that the behavior appears identical to the co-routine semantics defined in this subclause. In other words, the implementation would be obliged to analyze any dependencies between processes and to constrain their execution to match the co-routine semantics.

- Proposal: Explicitly allow parallel execution, preemption
  - Process instances at the same time (t,δ) may execute in parallel
    - Model designer must write thread safe code, avoid race conditions
    - Parallel systems, parallel models, parallel programming

Pushing the Limits with RISC

- Obstacle 1: *Resolved!*
- ➢ Introduce a dedicated SystemC Compiler
  - ➢ Automatic analysis of parallel access conflicts
  - ➢ Run SystemC processes in parallel if there are no conflicts
  - ➢ Faster simulation
  - ➢ Results remain the same



Obstacle 2: Simulator State

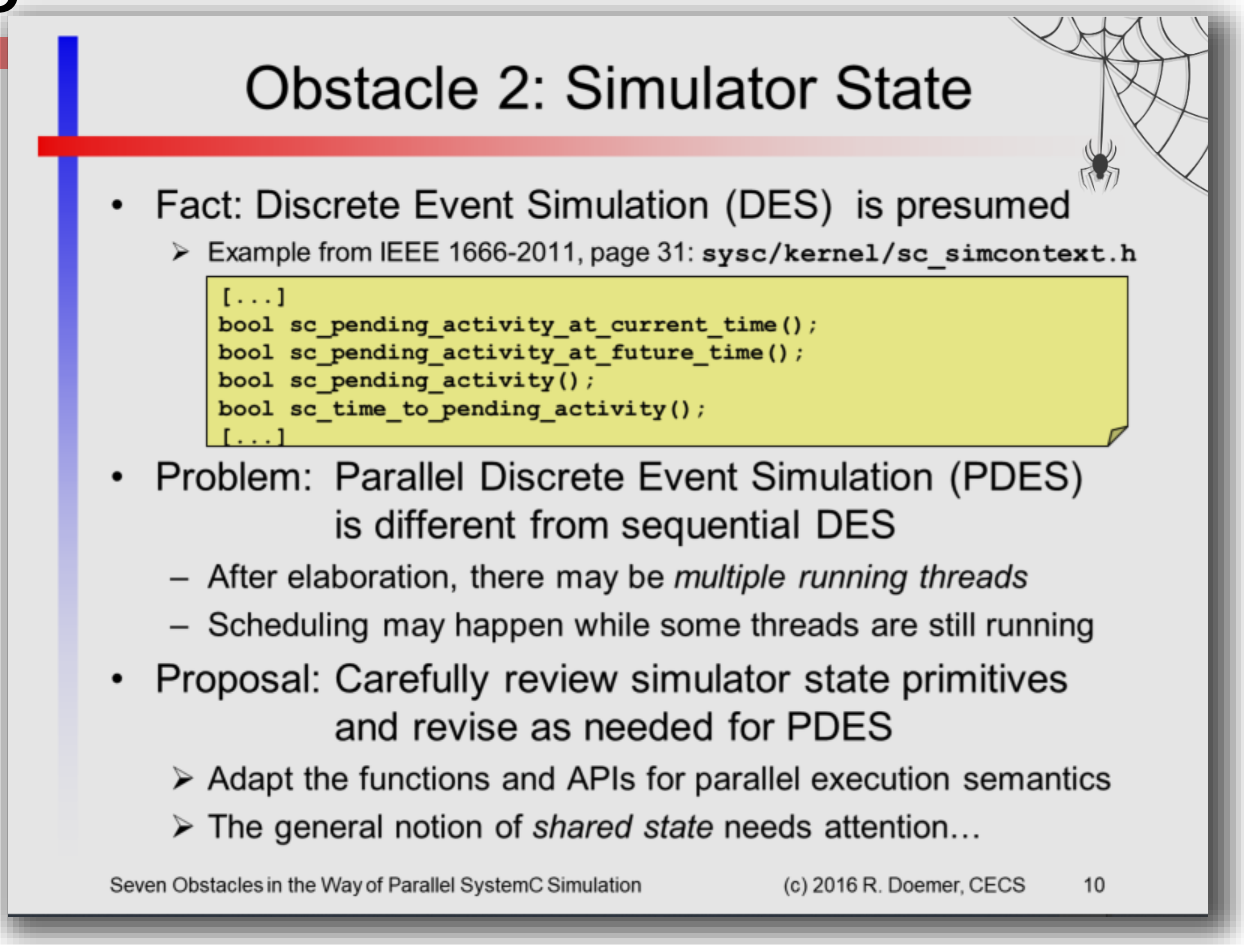
- Fact: Discrete Event Simulation (DES) is presumed
  - Example from IEEE 1666-2011, page 31: **`sysc/kernel/sc_simcontext.h`**

```
[...]
bool sc_pending_activity_at_current_time();
bool sc_pending_activity_at_future_time();
bool sc_pending_activity();
bool sc_time_to_pending_activity();
[...]
```

- Problem: Parallel Discrete Event Simulation (PDES) is different from sequential DES
  - After elaboration, there may be *multiple running threads*
  - Scheduling may happen while some threads are still running
- Proposal: Carefully review simulator state primitives and revise as needed for PDES
  - Adapt the functions and APIs for parallel execution semantics
  - The general notion of *shared state* needs attention…

Pushing the Limits with RISC

- Obstacle 2: *Ongoing…*
  - ➢ Review and revise the SystemC API
    - ➢ Slightly adjust the semantics
    - ➢ Maximize compliance with standard
    - ➢ For APIs on the slide:
      - ➢ User's expectations can be met
      - ➢ Example: SystemC Integration with Simics VP works fine



Obstacle 3: Lack of Thread Safety

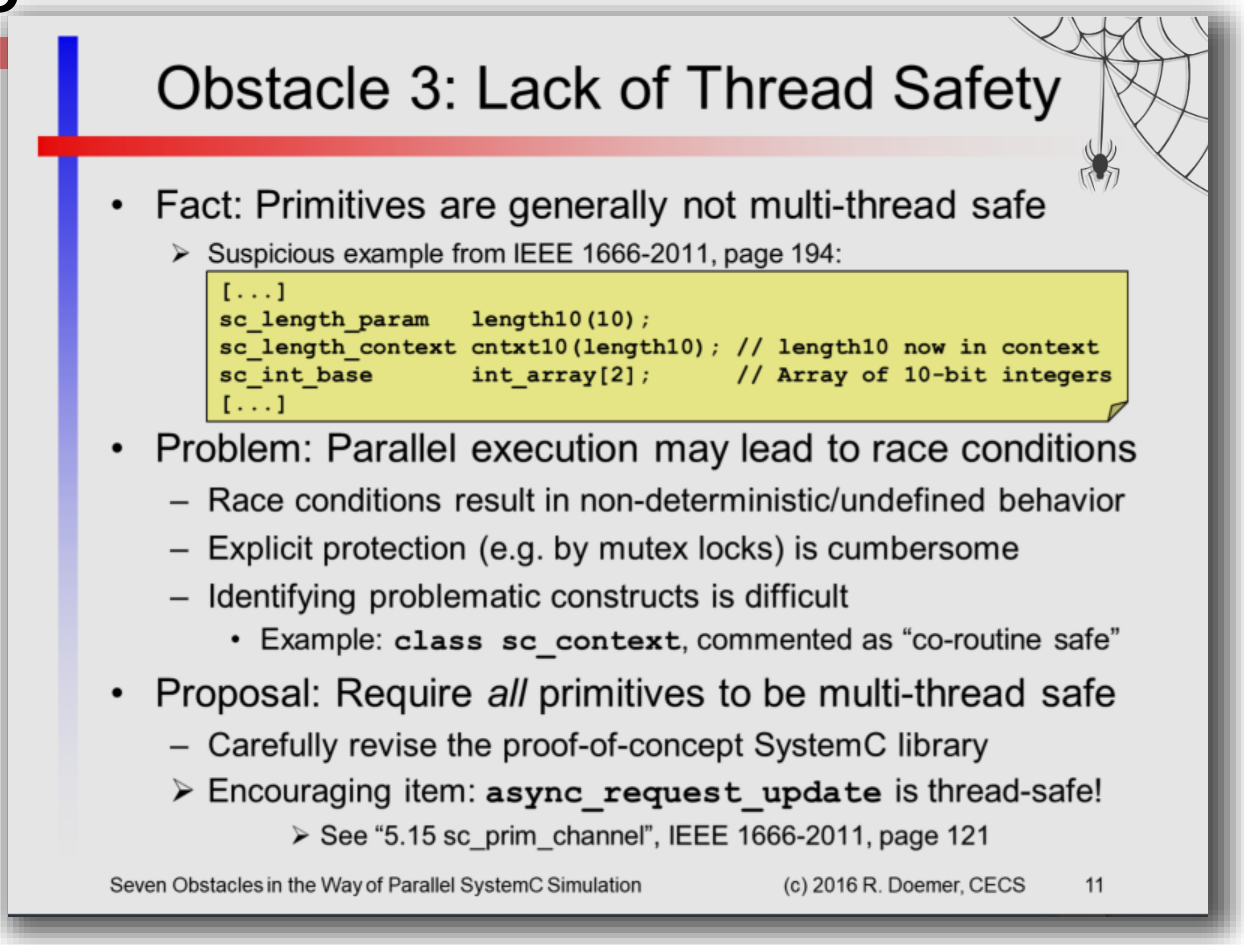
- Fact: Primitives are generally not multi-thread safe
  - Suspicious example from IEEE 1666-2011, page 194:

```
[...]
sc_length_param   length10(10);
sc_length_context cntxt10(length10); // length10 now in context
sc_int_base       int_array[2];      // Array of 10-bit integers
[...]
```

- Problem: Parallel execution may lead to race conditions
  - Race conditions result in non-deterministic/undefined behavior
  - Explicit protection (e.g. by mutex locks) is cumbersome
  - Identifying problematic constructs is difficult
    - Example: **`class sc_context`**, commented as “co-routine safe”
- Proposal: Require *all* primitives to be multi-thread safe
  - Carefully revise the proof-of-concept SystemC library
  - Encouraging item: **`async_request_update`** is thread-safe!
    - See “5.15 sc_prim_channel”, IEEE 1666-2011, page 121

Pushing the Limits with RISC

- Obstacle 3: *Ongoing…*
- ➢ Revise SystemC primitives for multi-thread safety
  - ➢ Protection by inserted locks
  - ➢ Store state in local or thread-local storage
  - ➢ For deterministic debugging, user can control number of parallel threads (e.g. set to 1)



Obstacle 3: Lack of Thread Safety

- Fact: Primitives are generally not multi-thread safe
  - ➢ Suspicious example from IEEE 1666-2011, page 194:

```
[...]
sc_length_param   length10(10);
sc_length_context cntxt10(length10); // length10 now in context
sc_int_base       int_array[2];      // Array of 10-bit integers
[...]
```

- Problem: Parallel execution may lead to race conditions
  - Race conditions result in non-deterministic/undefined behavior
  - Explicit protection (e.g. by mutex locks) is cumbersome
  - Identifying problematic constructs is difficult
    - Example: **`class sc_context`**, commented as "co-routine safe"
- Proposal: Require *all* primitives to be multi-thread safe
  - Carefully revise the proof-of-concept SystemC library
  - ➢ Encouraging item: **`async_request_update`** is thread-safe!
    - ➢ See "5.15 sc_prim_channel", IEEE 1666-2011, page 121

Seven Obstacles in the Way of Parallel SystemC Simulation (c) 2016 R. Doemer, CECS 11

Obstacle 4: Class sc_channel

- Fact: **`sc_channel`** is an alias type for **`sc_module`**
  - IEEE 1666-2011, Section "5.2.23 sc_behavior and sc_channel" (page 56):

> The typedefs **sc_behavior** and **sc_channel** are provided for users to express their intent.
> NOTE—There is no distinction between a *behavior* and a hierarchical channel other than a difference of intent. Either may include both ports and public member functions.

  - **`systemc-2.3.1/include/sysc/kernel/sc_module.h`**

```
[...]
typedef sc_module sc_channel;
typedef sc_module sc_behavior;
[...]
```

- Problem: Alias type is only another name, no new type
  - Language does not distinguish modules and channels
  - No separation of communication and computation
    - Breaks a key system-level design principle…
- Proposal: Class **`sc_channel`**, derived from **`sc_module`**
  - Module encapsulates computation (hosts threads/processes)
  - Channel encapsulates communication (implemented interfaces)

Pushing the Limits with RISC

- Obstacle 4: *Fixed!*
- ➢ Derive **`sc_channel`** from base class **`sc_module`**
  - ➢ Minimal change in SystemC headers
  - ➢ Two different types at compile-time
  - ➢ Easy distinction in static analysis
  - ➢ No known negative side-effects

Obstacle 4: Class sc_channel

- Fact: **`sc_channel`** is an alias type for **`sc_module`**
  - ➢ IEEE 1666-2011, Section "5.2.23 sc_behavior and sc_channel" (page 56):

> The typedefs **sc_behavior** and **sc_channel** are provided for users to express their intent.
> NOTE—There is no distinction between a *behavior* and a hierarchical channel other than a difference of intent. Either may include both ports and public member functions.

  - ➢ **`systemc-2.3.1/include/sysc/kernel/sc_module.h`**

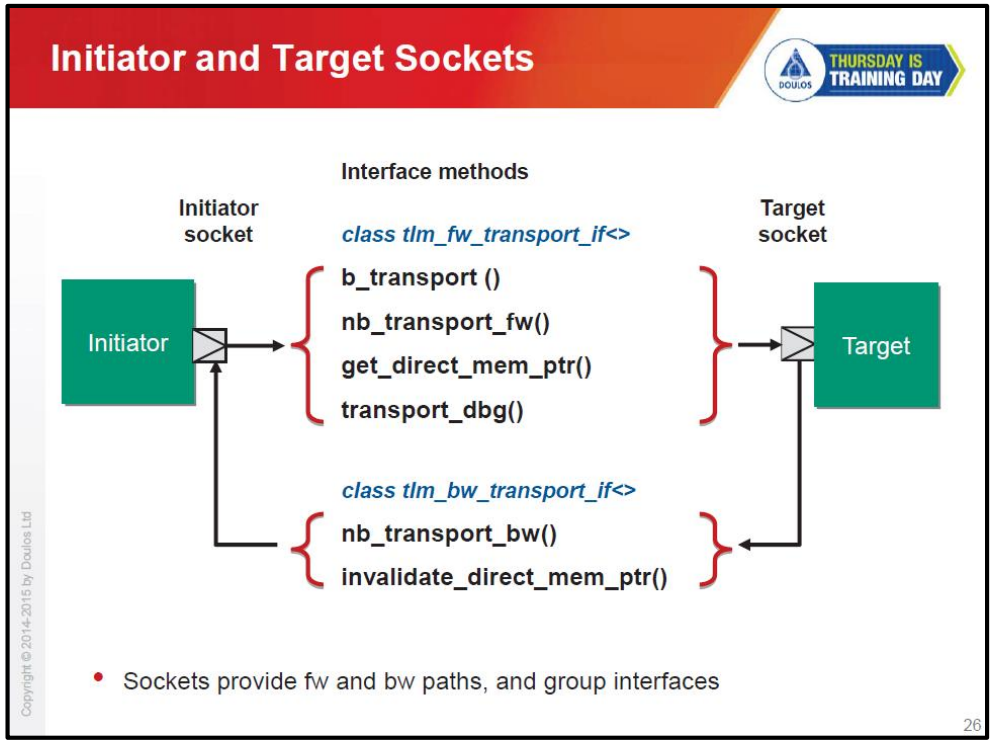
```
[...]
typedef sc_module sc_channel;
typedef sc_module sc_behavior;
[...]
```

- Problem: Alias type is only another name, no new type
  - – Language does not distinguish modules and channels
  - ➢ No separation of communication and computation
    - Breaks a key system-level design principle…
- Proposal: Class **`sc_channel`**, derived from **`sc_module`**
  - ➢ Module encapsulates computation (hosts threads/processes)
  - ➢ Channel encapsulates communication (implemented interfaces)

Seven Obstacles in the Way of Parallel SystemC Simulation (c) 2016 R. Doemer, CECS 12

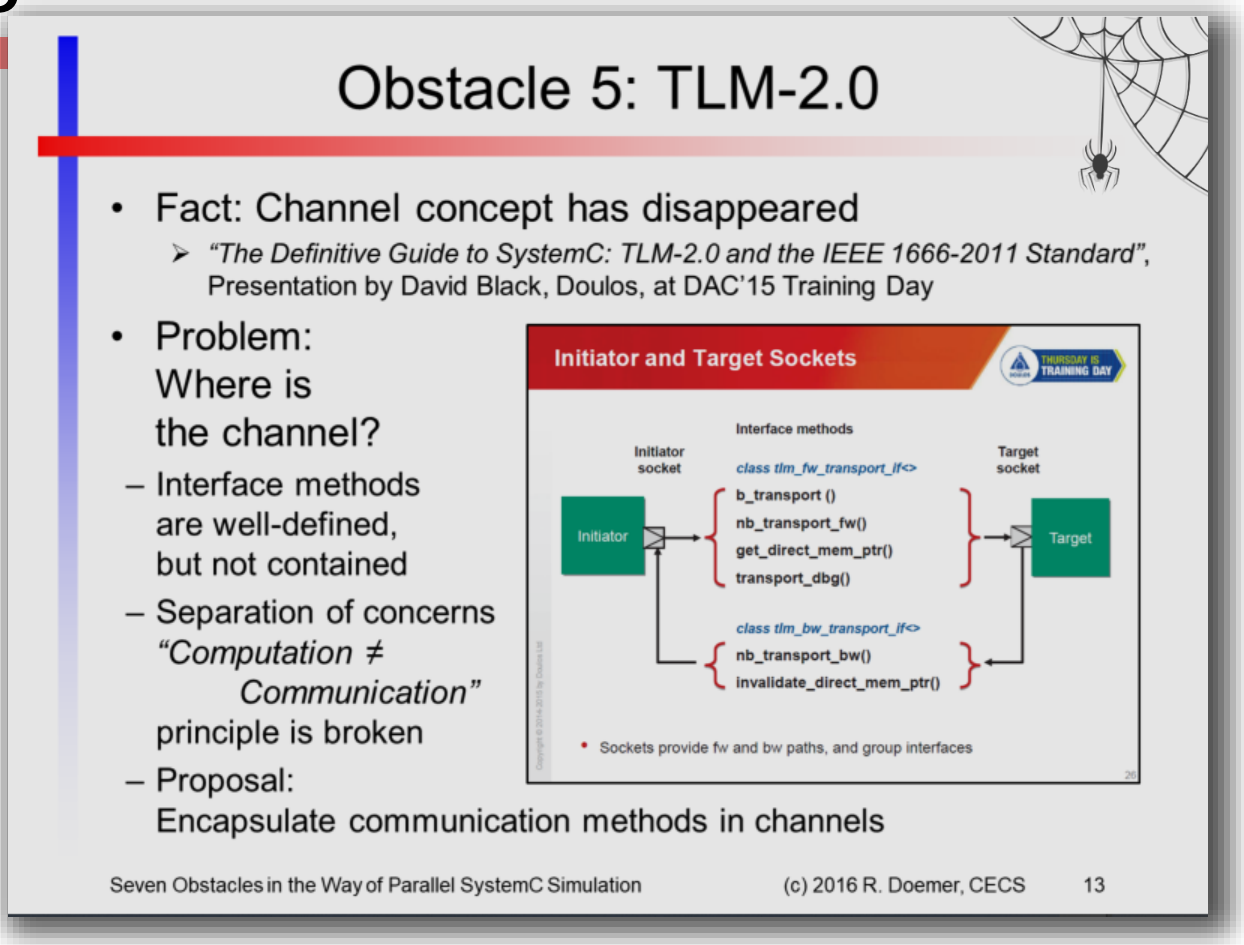
Obstacle 5: TLM-2.0

- Fact: Channel concept has disappeared
  - *"The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard"*, Presentation by David Black, Doulos, at DAC'15 Training Day
- Problem:
  Where is
  the channel?
  - Interface methods are well-defined, but not contained
  - Separation of concerns *"Computation ≠ Communication"* principle is broken
  - Proposal:
    Encapsulate communication methods in channels



Pushing the Limits with RISC

- Obstacle 5: *Reevaluated, Resolved!*
- ➢ Socket Call Path (SCP) analysis
- ➢ Variable Entanglement analysis
  - ➢ Compile-time analysis can identify target methods executed by TLM-2.0 calls
  - ➢ Support for interconnect modules and DMI [CODES+ISSS'19, ACM TECS]



Obstacle 6: Sequential Mindset

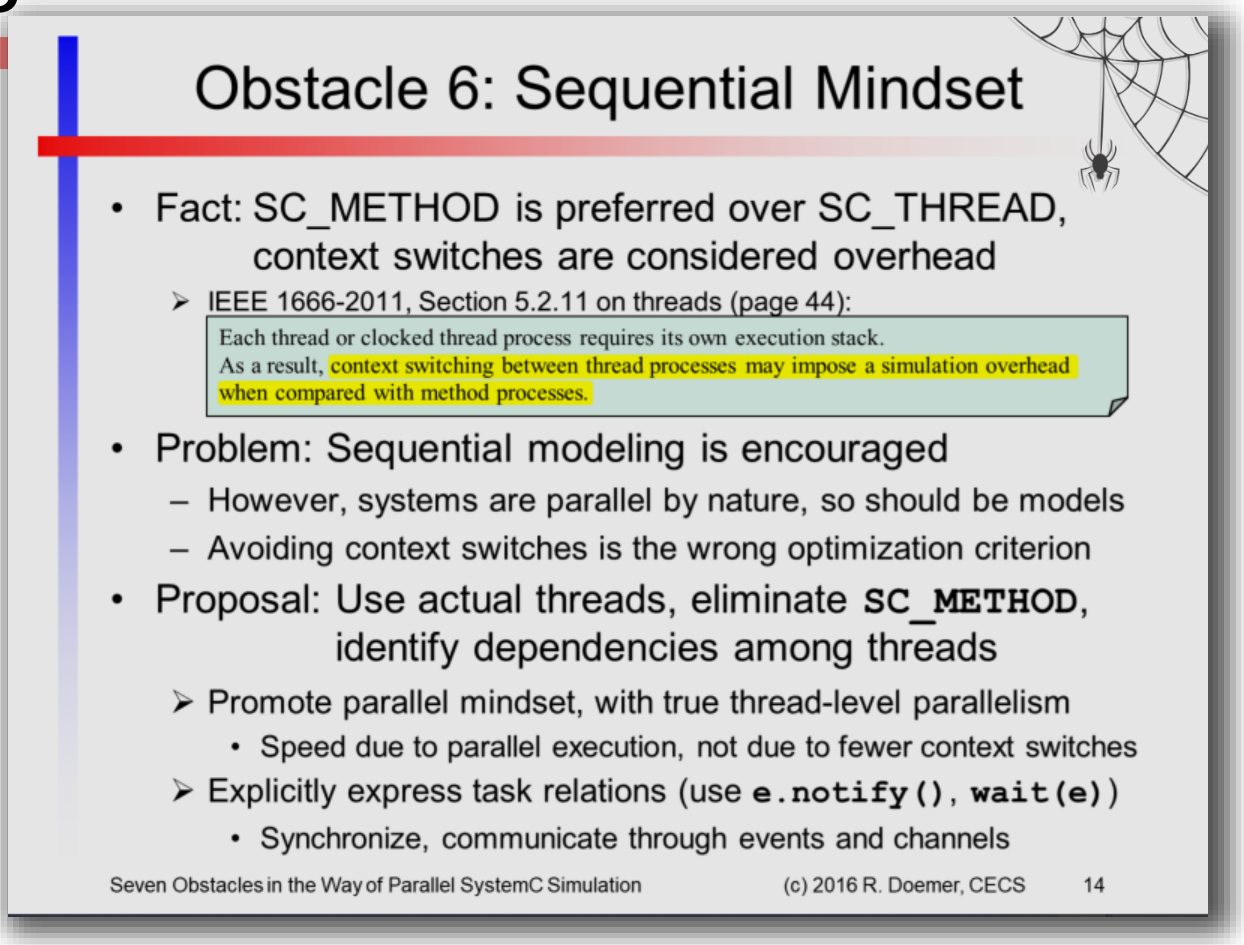
- Fact: SC_METHOD is preferred over SC_THREAD, context switches are considered overhead
  - IEEE 1666-2011, Section 5.2.11 on threads (page 44):

> Each thread or clocked thread process requires its own execution stack.
> As a result, context switching between thread processes may impose a simulation overhead when compared with method processes.

- Problem: Sequential modeling is encouraged
  - However, systems are parallel by nature, so should be models
  - Avoiding context switches is the wrong optimization criterion
- Proposal: Use actual threads, eliminate **`SC_METHOD`**, identify dependencies among threads
  - Promote parallel mindset, with true thread-level parallelism
    - Speed due to parallel execution, not due to fewer context switches
  - Explicitly express task relations (use **`e.notify()`**, **`wait(e)`**)
    - Synchronize, communicate through events and channels

Pushing the Limits with RISC

- Obstacle 6:
  Not a problem
- ➢ **`SC_METHOD`, `SC_THREAD`, `SC_CTHREAD`** can all be supported
  - ➢ Static analysis per process type
  - ➢ **`SC_METHOD`** execution by dedicated invoker threads
  - ➢ Nice optimization problem for efficient grouping with minimal conflicts



Obstacle 7: Temporal Decoupling

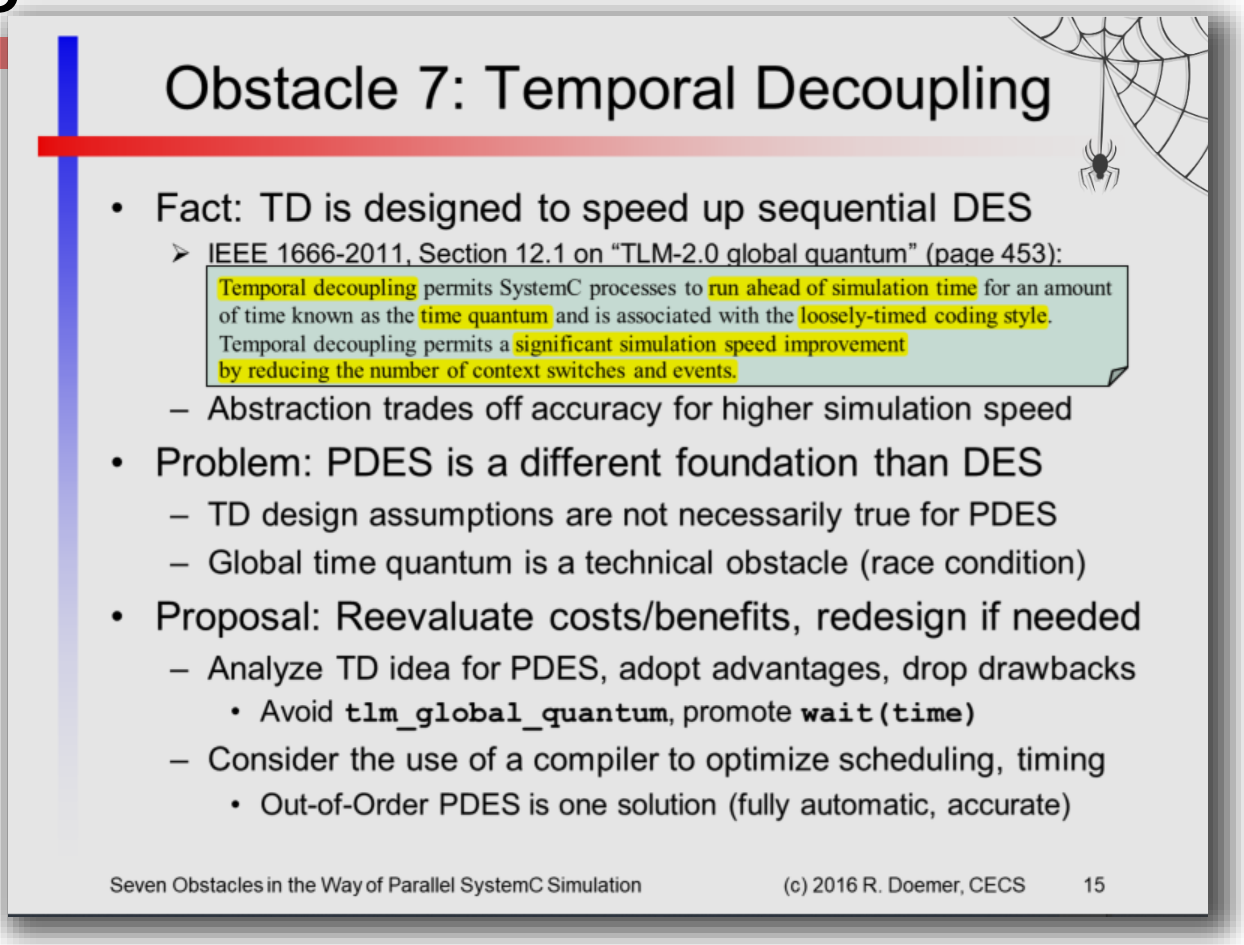
- Fact: TD is designed to speed up sequential DES
  - IEEE 1666-2011, Section 12.1 on "TLM-2.0 global quantum" (page 453):

> Temporal decoupling permits SystemC processes to run ahead of simulation time for an amount of time known as the time quantum and is associated with the loosely-timed coding style. Temporal decoupling permits a significant simulation speed improvement by reducing the number of context switches and events.

  - Abstraction trades off accuracy for higher simulation speed
- Problem: PDES is a different foundation than DES
  - TD design assumptions are not necessarily true for PDES
  - Global time quantum is a technical obstacle (race condition)
- Proposal: Reevaluate costs/benefits, redesign if needed
  - Analyze TD idea for PDES, adopt advantages, drop drawbacks
    - Avoid **`tlm_global_quantum`**, promote **`wait(time)`**
  - Consider the use of a compiler to optimize scheduling, timing
    - Out-of-Order PDES is one solution (fully automatic, accurate)

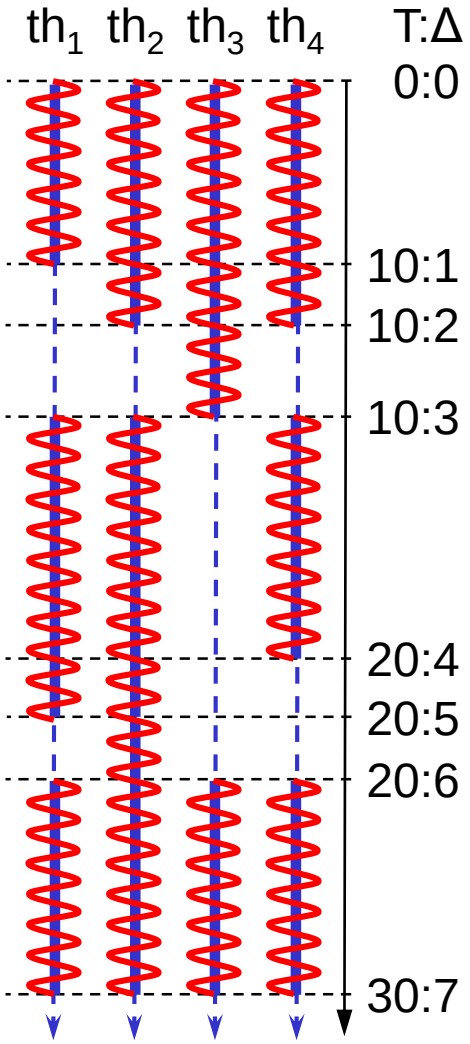
Pushing the Limits with RISC

- Obstacle 7: *TBD…*
- ➢ Investigate in future work
- ➢ Is there any need for this abstraction in PDES?
  - ➢ Out-of-order PDES
    - ➢ Likely can achieve the same benefit
    - ➢ Without loss of accuracy
  - ➢ Global time quantum (if needed) can be protected by mutex

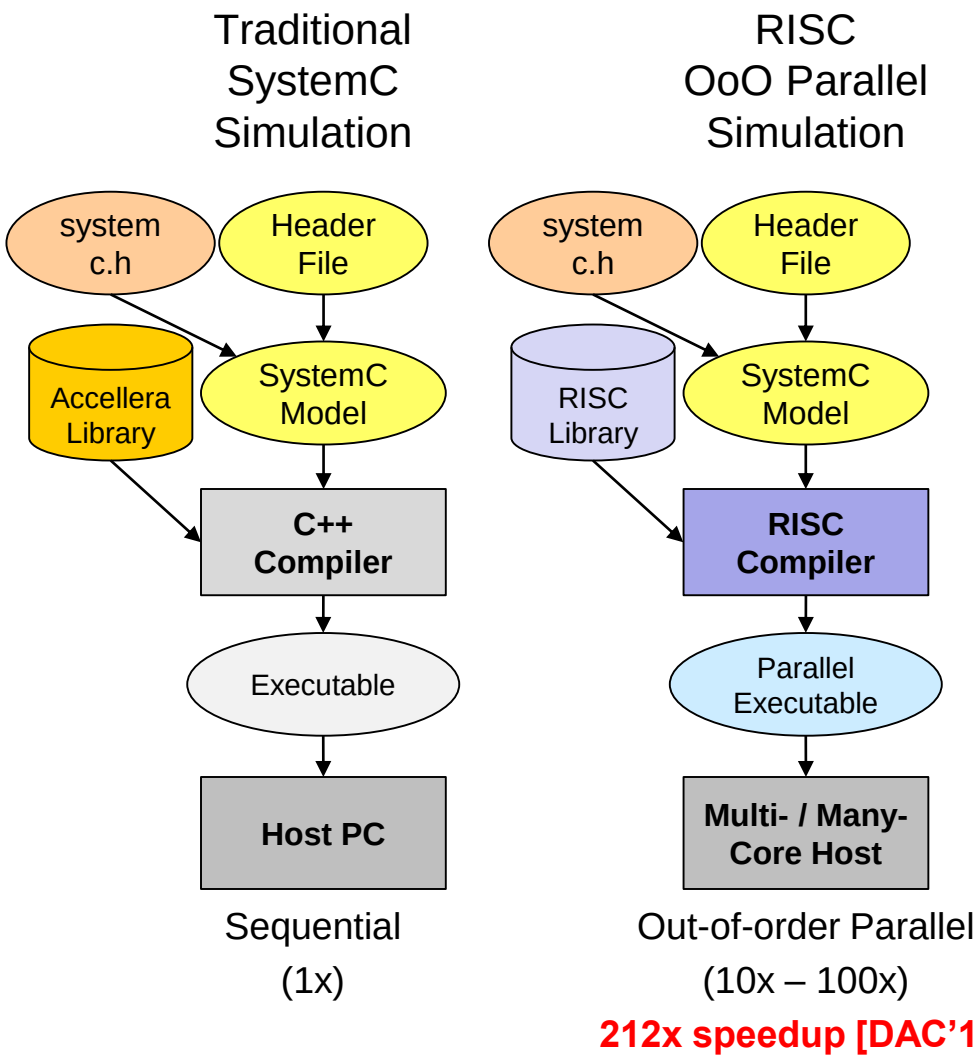


Pushing the Limits with RISC

- Out-of-Order Parallel DES
  - Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle,
    - *OR* if there are no conflicts!
  - ➢ Breaks synchronization barrier!
  - ➢ Threads run as soon as possible, even ahead of time
  - ➢ Maximum speedup!
    - Results at [DATE'12], [IEEE TCAD'14]
  - ➢ Our approach preserves…
    - ➢ Cause and effect relationship
    - ➢ Accuracy in results and timing
    - ➢ Maximum compliance with standard

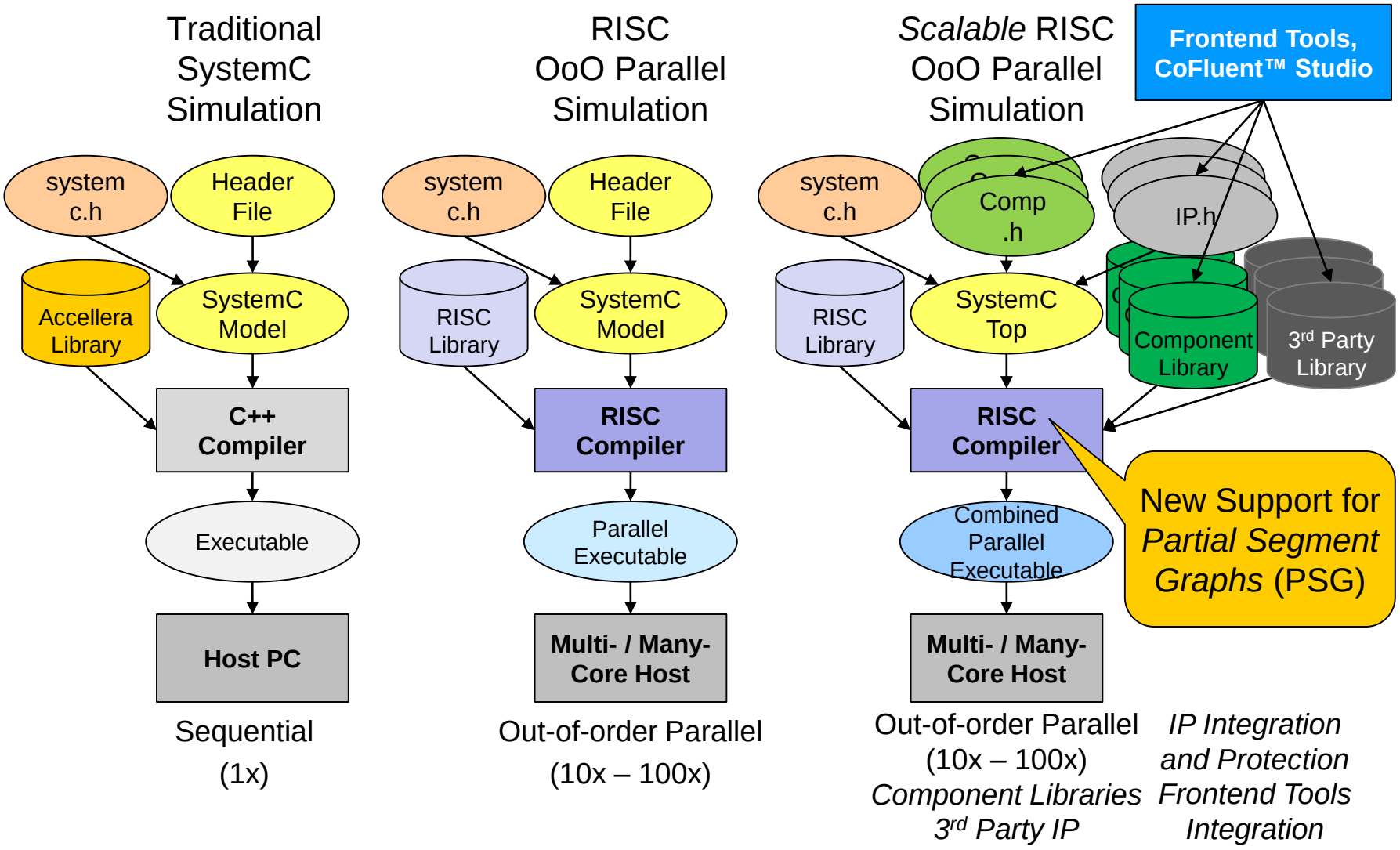


Recoding Infrastructure for SystemC



- RISC Infrastructure
  - Dedicated RISC compiler tool chain
  - Compliance with standard SystemC semantics
  - Open source available from CECS
- ➢ Out-of-order Parallel Simulation
  - ➢ Fully accurate
  - ➢ Two orders of magnitude faster

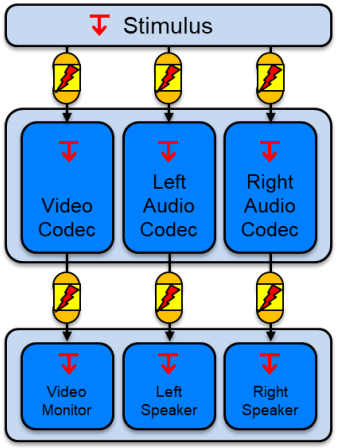
Scaling RISC: File Hierarchies, 3rd Party IP



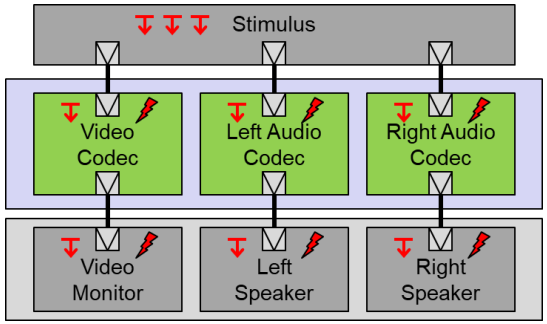
Scaling RISC: Support for TLM-2.0

- Various Modeling Styles Supported by RISC v0.6.0

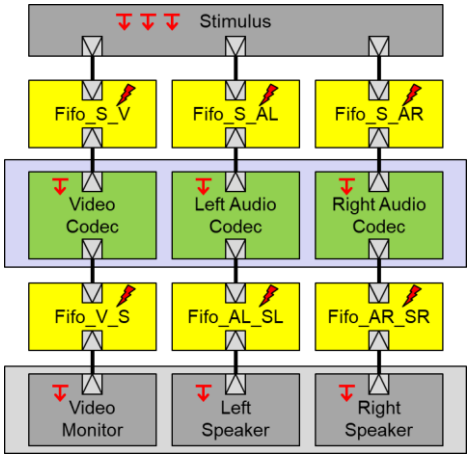
Structural Composition



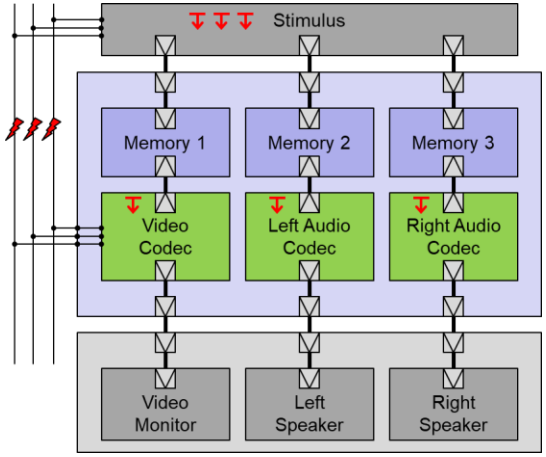
Synchronization



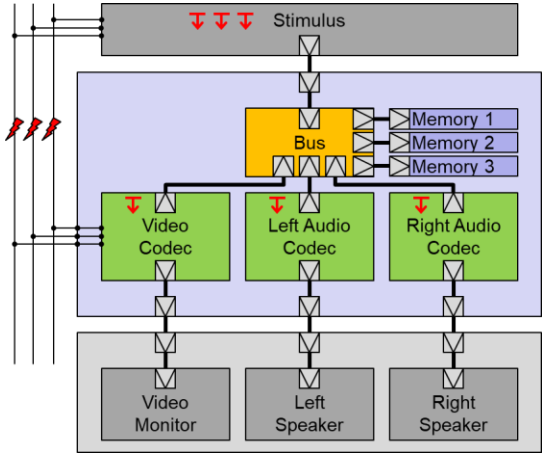
Connectivity



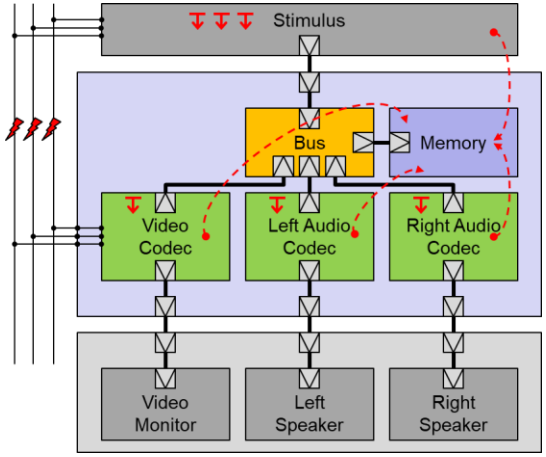
Explicit Memories



Interconnect Modules

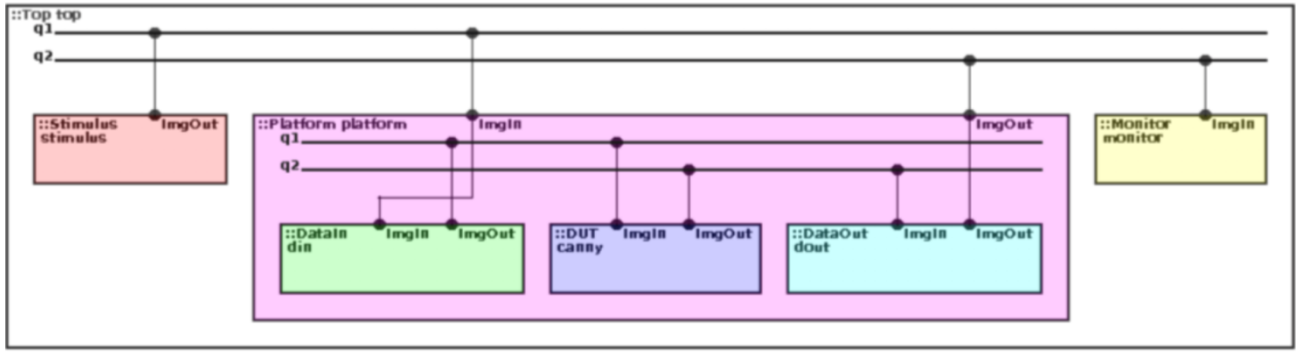


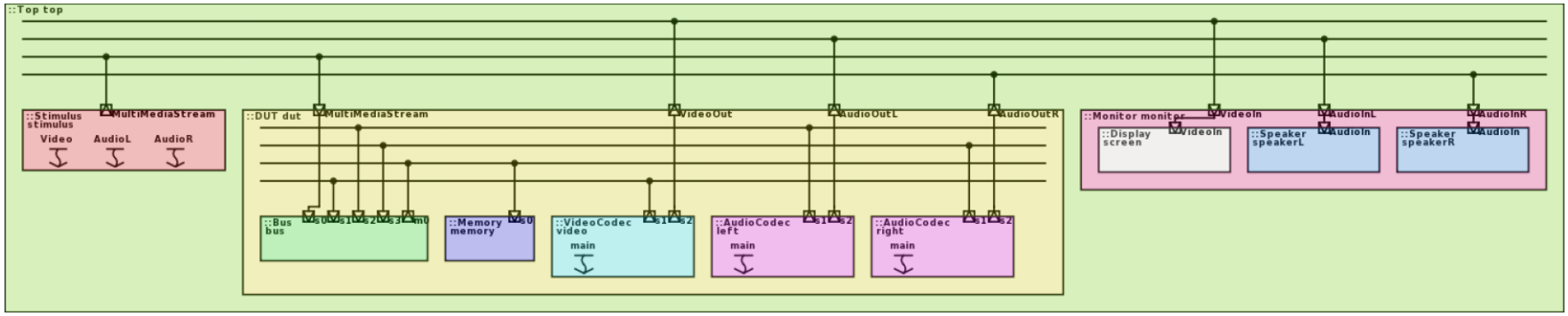
DMI



Scaling RISC: Analysis and Transformation

- Example: Model Visualization
  - Hierarchy and connectivity
    - Ports and sockets
  - Threads in modules





RISC Open Source

- RISC Compiler and Simulator, Release V0.6.0
  - ➢ http://www.cecs.uci.edu/~doemer/risc.html#RISC060
    - Installation notes and script: **`INSTALL`**, **`Makefile`**
    - Open source tar ball: **`risc_v0.6.0.tar.gz`**
    - Docker script and container: **`Dockerfile`**
    - Doxygen documentation: RISC API, OOPSC API
    - Tool manual pages: **`risc`**, **`simd`**, **`visual`**, …
    - BSD license terms: **`LICENSE`**
  - – Companion Technical Report
    - CECS Technical Report 19-04: **`CECS_TR_19_04.pdf`**
  - ➢ Docker container:

```
bash# docker pull ucirvinelecs/risc060
bash# docker run -it ucirvinelecs/risc060
[dockeruser]# cd demodir
[dockeruser]# make play_demo
```

    - ➢ https://hub.docker.com/r/ucirvinelecs/risc060/

Conclusion

- Overcoming Obstacles towards Parallel SystemC
  1. Co-Routine Semantics: *Resolved*
  2. Simulator State: *Ongoing…*
  3. Lack of Thread Safety: *Ongoing…*
  4. Class sc_channel: *Fixed*
  5. TLM-2.0: *Reevaluated, Resolved*
  6. Sequential Mindset: *Not a problem*
  7. Temporal Decoupling: *TBD…*
- Recoding Infrastructure for SystemC
  - Introduction of a dedicated SystemC compiler
  - Out-of-order parallel simulation on multi- and many-core hosts
  - Maximum compliance with IEEE SystemC semantics
- Open Source
  - Thanks to Intel Corporation!

References (1)

- [IESS'19a] Z. Cheng, T. Schmidt, R. Dömer: *"Enabling IP Reuse and Protection in Out-of-Order Parallel SystemC Simulation"*, Proceedings of IESS, Springer, Friedrichshafen, Germany, Sep. 2019.
- [IESS'19b] E. Arasteh, R. Dömer: *"An Untimed SystemC Model of GoogLeNet"*, Proceedings of IESS, Springer, Friedrichshafen, Germany, Sep. 2019.
- [CODES+ISSS'19] Z. Cheng, R. Dömer: *"Analyzing Variable Entanglement for Parallel Simulation of SystemC TLM-2.0 Models"*, accepted at CODES+ISSS, New York, Oct. 2019.
  - Reprint to appear as journal article in ACM Transactions on Embedded Computer Systems!
- [DVCon'19] D. Mendoza, A. Dingankar, Z. Cheng, R. Dömer: *"Integrating Parallel SystemC Simulation into Simics® Virtual Platform"*, accepted at DVCon Europe, Munich, Germany, Oct. 2019.
- [ASPDAC'20] Z. Cheng, A. Arasteh, R. Dömer: *"Event Delivery using Prediction for Faster Parallel SystemC Simulation"*, accepted at ASPDAC, Beijing, China, Jan. 2020.

References (2)

- [HLDVT'17] Z. Cheng, T. Schmidt, G. Liu, R. Dömer: *"Thread- and Data-Level Parallel Simulation in SystemC, a Bitcoin Miner Case Study"*, Proceedings of HLDVT, Santa Cruz, California, Oct. 2017.
- [CECS'17] D. Mendoza, R. Dömer: *"A Tool for Visualization of SystemC Models"*, CECS Technical Report 17-06, Nov. 2017.
- [DATE'18] T. Schmidt, Z. Cheng, R. Dömer: *"Port Call Path Sensitive Conflict Analysis for Instance-Aware Parallel SystemC Simulation"*, Proceedings of DATE, Dresden, Germany, March 2018.
- [FDL'18] Z. Cheng, T. Schmidt, R. Dömer: *"SystemC Coding Guideline for Faster Out-of-Order Parallel Discrete Event Simulation"*, Proceedings of FDL, Munich, Germany, Sep. 2018.
- [CECS'18] G. Liu, T. Schmidt, Z. Cheng, D. Mendoza, R. Dömer: *"RISC Compiler and Simulator, Release V0.5.0: Out-of-Order Parallel Simulatable SystemC Subset"*, CECS TR 18-03, Sep. 2018.
- [TCAD'TBD] T. Schmidt, Z. Cheng, G. Liu, D. Mendoza, A. Dingankar, R. Dömer: *"RISC: A Static Analysis Framework for Parallel Simulation of SystemC"*, submitted to IEEE Transactions on CAD.

References (3)

- [DATE'18] T. Schmidt, Z. Cheng, R. Dömer: *"Port Call Path Sensitive Conflict Analysis for Instance-Aware Parallel SystemC Simulation"*, Proceedings of DATE, Dresden, Germany, March 2018.
- [DAC'17] T. Schmidt, G. Liu, R. Dömer: *"Towards Ultimate Parallel SystemC Simulation through Thread and Data Level Parallelism"*, Proceedings DAC, Austin, TX, June 2017.
- [Springer'17] R. Dömer, G. Liu, T. Schmidt: *"Parallel Simulation", chapter 17 in "Handbook of Hardware/Software Codesign"* by S. Ha and J. Teich, Springer Netherlands, June 2016.
- [ASPDAC'17] T. Schmidt, G. Liu, R. Dömer: *"Hybrid Analysis of SystemC Models for Fast and Accurate Parallel Simulation"*, Proceedings ASPDAC, Tokyo, Japan, January 2017.
- [IEEE ESL'16] R. Dömer: *"Seven Obstacles in the Way of Standard-Compliant Parallel SystemC Simulation"*, IEEE Embedded Systems Letters, vol. 8, no. 4, pp. 81-84, Dec. 2016.
- [DAC'15] R. Dömer: *"Towards Parallel Simulation of Multi-Domain System Models"*, Keynote, DAC workshop on System-to-Silicon Performance Modeling and Analysis, June 2015.
- [IEEE TCAD'14] W. Chen, X. Han, C. Chang, G. Liu, R. Dömer: *"Out-of-Order Parallel Discrete Event Simulation for Transaction Level Models"*, IEEE Transactions on CAD, vol. 33, no. 12, pp. 1859-1872, December 2014.
- [DATE'14] W. Chen, X. Han, R. Dömer: *"May-Happen-in-Parallel Analysis based on Segment Graphs for Safe ESL Models"*, Proceedings of DATE, Dresden, Germany, March 2014.
- [DATE'13] W. Chen, R. Dömer: *"Optimized Out-of-Order Parallel Discrete Event Simulation Using Predictions"*, Proceedings of DATE, Grenoble, France, March 2013.
- [DATE'12] W. Chen, X. Han, R. Dömer: *"Out-of-Order Parallel Simulation for ESL Design"*, Proceedings of DATE, Dresden, Germany, March 2012.

Acknowledgments

- For solid work, fruitful discussions, and honest feedback, I would like to thank:
  - My team at UCI
    - Zhongqi Cheng, Daniel Mendoza, Emad Arasteh
    - Tim Schmidt, Guantao Liu
    - Farah Arabi, Spencer Kam
  - Our collaborators at Intel
    - Ajit Dingankar
    - Desmond Kirkpatrick
    - Abhijit Davare
    - Philipp Hartmann
  - And many others…
- This work has been supported in part by substantial funding from Intel Corporation. Thank you!