The Intel® Simics® Simulator and SystemC* and Threading

Jakob Engblom, Director of Simulation Technical Ecosystem
jakob.engblom@intel.com
Copyright Permission

• A non-exclusive, irrevocable, royalty-free copyright permission is granted by Intel 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.
Legal Notice

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, and Simics are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

© Intel Corporation.
Designing Threading in the Simics® Simulator

- Threading handled by the framework
  - Defined host-independent threading semantics and threading execution modes
  - User can enable/disable parallel execution

- Threading should affect as few modules as possible
  - Fundamentally, parallel/concurrent coding is hard and should be isolated to a few places in the code base
  - **Device models should not need to change** to enable threading – simplifies programming, simplifies deployment of threading
  - In practice, most parallelism is provided by **processors (and clocks), and a few features**

- Revert to sequential execution on demand
  - Support deterministic execution for software debug
  - Support debug of models by removing threading as a complicating factor

- Losing determinism for performance is OK
  - Fall back to slower deterministic mode if needed

- Advanced thread programming available
  - Support advanced use cases where the standard threading solutions are insufficient

Threading should be easy and automatic for most modelers
Threading in the Simics® Simulator – Use Cases

“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

List is not exhaustive, and the different uses can be combined
How the Simics® Simulator uses Threads

- **JIT threads come from a specific pool**
- **Functionality like tools and extensions can start their own host threads**
- **JIT offload**

Simics® simulator thread scheduler

- **Model simulation work is handled by a dedicated scheduler**

Host OS

- OS thread
- OS thread
- OS thread
- OS thread

Host hardware

- Core
- Core
- Core
- Core
**Thread Domains (TD)** are used to indicate parts of the model that can run concurrently with other TDs. Models in TDs are explicitly thread-aware (SystemC* models are). Very tightly coupled devices can be thread-aware and run in a TD (performance optimization).

The **Cell Thread Domain** contains the models that are not thread-aware, preserves standard serial semantics for models (only single thread modifies), and = Reuse of existing device models.

**Cells**: smallest loosely coupled unit. Typically, a single target machine. Between cells, use links to communicate. Each cell is "easy" to run in parallel to other cells (as long as there is no shared state hiding in the code).
SystemC* Models in the Simics® Simulator

- Use Accellera*-compliant SystemC models as-is
  - Including binary-compiled models
  - ABI-compatibility required

- SystemC models are compiled into/linked into a Simics module
  - Each module can contain one or more top-level adapters
  - Each adapter can be instantiated multiple times in the system model
  - The adapter provides the interface between the SystemC subsystem and the Simics simulation environment
  - The adapter provides a “co-execute” object that is scheduled by the Simics simulator core to run the model
  - Modules are loaded dynamically when used
SystemC* Models and Threading

Due to shared state in the SystemC kernel:
All instances from the same module must be in the same TD and same Cell

The entire model runs in the thread domain – not in the Cell's Thread Domain.

Other models can run in parallel to the SystemC models

Other models can run in parallel to the SystemC models

SystemC models in a separate module can run in parallel

*Other names and brands may be claimed as the property of others
The Simics® simulator can run multiple SystemC* models in parallel to each other and to other Simics models
   – Locking applied to accesses to shared state and devices, according to Simics API

Parallelism is enabled using multiple separate kernels
   – i.e., built on top, not inside of SystemC
   – Semantics in each model is standard SystemC serial semantics

Shared static state in the kernel limits available parallelism
   – Fixing it requires an ABI change for the kernel
More on the Simics® Simulator

• The Public Release of the Intel® Simics® Simulator
  – Complete Simics base product
    • Includes the Simics SystemC® Library
    • As well as Simics-native model builder tools
  – Intel-based Quick-Start Platform
  – Training materials and examples
  – Bonus: the first release of Intel ISIM, the Intel Integrated Simulation Environment with Modeling, including examples of performance, power, and thermal models

https://developer.intel.com/simics-simulator
THE END