Modelling, simulation, and advanced tracing for extra-functional properties in SystemC/TLM

Philipp A. Hartmann
philipp.hartmann@offis.de
OFFIS Institute for Information Technology
R&D Division Transportation

September 24, 2013
ESCUG’28
Paris, France
1 Motivation

Problem
- Power consumption becomes increasingly important for (mobile) embedded systems

Background
- Limited increase in battery capacity
- Improved power management needed

Challenges
- Complex, distributed HW/SW systems
- Heterogeneous hardware platforms
- Optimization needs real usage scenarios

Virtual Prototype based solution
- Full system simulation at high speed
- Block level power tracing
- Verification of power intent

Goal: Power-aware Virtual Platforms for Analysis and Optimisation
2 Outline

1. COMPLEX Power-Aware Virtual Prototyping
2. Scalable System-Level Power Model
3. Advanced tracing for TLM Virtual Platforms
4. Medium-term standardisation
5. Conclusion
3 Outline

COMPLEX Power-Aware Virtual Prototyping

1. COMPLEX Power-Aware Virtual Prototyping
   - Basic Approach / Enabling Technologies

2. Scalable System-Level Power Model

3. Advanced tracing for TLM Virtual Platforms

4. Medium-term standardisation

5. Conclusion
COdesign and power Mangement in PLatform-based design-space EXploration

- European Integrated Project (2010–2013)
- **Goal:** Highly efficient and productive design methodology and a holistic framework for iteratively exploring embedded HW/SW applications based on power-aware virtual prototyping in SystemC.
5 Basic Approach / Enabling Technologies
COMPLEX Power-Aware Virtual Prototyping

- Extra-functional model for timing and power
  - Explicit separation of functional and extra-functional model
  - Activity model for (dynamic) power
  - Scalable physical/technology power model for frequency, supply voltage, and temperature

- Automatic timing and power annotation techniques
  - Embedded Software: Annot. from cross-compiled binary
  - Custom Hardware: Annot. from power aware HL-synthesis
  - Black-Box Hardware IP: Power State Machines

- Scalable Timing and Power Tracing infrastructure
  - Timing and Power Tracing Streams per observable VP block
  - Processing with filters (e.g. aggregation, averaging, selection)
  - Dynamic granularity (e.g. area of interest)
6 Outline

Scalable System-Level Power Model

1. COMPLEX Power-Aware Virtual Prototyping

2. Scalable System-Level Power Model
   - Requirements
   - System-level Parameters

3. Advanced tracing for TLM Virtual Platforms

4. Medium-term standardisation

5. Conclusion
7 Requirements
Scalable System-Level Power Model

- Define **set of relevant parameters** for system-level modelling of **extra-functional properties** (power, temperature)
  - **Simplicity**: Only use those (dynamic) parameters needed for current use case (e.g. ignore area, when not looking for thermal behaviour)
  - **Composability**: Derive combined values from physical relations between individual contributors (total power, temperature-dependent power, capacitance-based power)
  - **Hierarchy**: Put parameters on “correct” geometric level, inherit parameters from context/environment
  - **Adaptivity**: Changing parameters during run-time, → support for dynamic power management

- Integration with common **extra-functional tracing/introspection**
  - Hierarchical recording of power information
  - Flexible adaptation to granularity of selected power model
  - Strongly typed (C++) physical units support, to avoid composition errors
8 System-level Parameters
Scalable System-Level Power Model

- Technology
- Process Variation
- IR Drop
- ... ??

Design param.

Geometry
- Position (x,y,z)
- Area [m²]
- Neighbourhood

Thermal
- Ambient temperature [°C]
- Module temperature [°C]

Power Modes
- Supply voltage [V]
- Clock frequency [Hz]

Static power
- Leakage [W]
- Leaking conductance [G]

Dynamic power
- Annotation
- PSM
- Switched capacitance [F]
9 System-level Parameters
Scalable System-Level Power Model

- **Building blocks** for an application-specific power model
  - Design Parameters
  - Dynamic annotation sources
- Parameters and value sources attached to structural elements (modules)
  - Parameters inherited from surrounding hierarchy, if not in local scope
- Physical units supported via **Boost.Units**
- **Hierarchical processing** of derived values by subscribing to value sources, e.g.

\[
P_{\text{dyn}}(t) = \frac{1}{2} V_{\text{dd}}^2 f \cdot C(t)
\]

- \(C(t)\) average switched capacitance per cycle (dynamic annotation)
- \(V_{\text{dd}}, f\) supply voltage, frequency (static parameters)
- \(P_{\text{dyn}}(t)\) processed function for dynamic power dissipation
10 Outline

Advanced tracing for TLM Virtual Platforms

1. COMPLEX Power-Aware Virtual Prototyping
2. Scalable System-Level Power Model
3. Advanced tracing for TLM Virtual Platforms
   - Stream-based Trace Recording
   - Hierarchical Tracing & Processing
   - Configurable Backends
4. Medium-term standardisation
5. Conclusion
11 Advanced tracing for TLM Virtual Platforms

General overview

Problem: Flexible tracing of physical quantities not directly possible in SystemC

- `sc_core::sc_trace` not flexible enough (tied to simulation time)
- `sca_core::sca_trace` is SystemC AMS-specific and not widely supported
- SCV transaction recording not really appropriate
- Advanced instrumentation partly available in commercial tools

Goal: Enable flexible and configurable tracing of extra-functional properties in TLM-2-based virtual platforms

Integration with temporal-decoupling

- Independence of current simulation time needed!

Hierarchical (pre-)processing

- Compute derived quantities (→ power model)
- Filtering and data reduction (aggregation, averaging, selection)
- Collection of user-defined performance metrics
- Run-time configurable granularity (Region of Interest)
12 Stream-based tracing of extra-functional properties
Advanced tracing for TLM Virtual Platforms

- Tracing is based on \((time, value)\) streams per component
  - A \texttt{timed\_stream<T>} is a \textbf{hierarchically named} SystemC object
  - \textbf{Strongly-typed values} for type safety with support for \textbf{physical units} \((\text{Boost.Units})\)
  - Fine granular control over (local) \textit{time offset}, synchronisation with a \textit{local clock}

- \textbf{push} APIs for both \textbf{absolute} \(\text{start, value, [duration]}\), or \textbf{relative} \(\text{value,duration}\) \textbf{tuples}

  ```cpp
  void push( value_type const & value,
             time_type const & duration = sc_core::SC_ZERO_TIME ) = 0;
  void push( time_type const & offset, value_type const & value,
             time_type const & duration = sc_core::SC_ZERO_TIME ) = 0;
  ```

  - Non-overlapping tuples enforced by \textbf{MergePolicy} \((\text{e.g. accumulate, overwrite, error})\)

- \textbf{Explicit commit/sync} API to advance \textbf{stream clocks}

  ```cpp
  void commit(); // commit all pending pushes
  void commit( time_type const & offset ); // commit until offset
  time_type sync(); // commit & synchronise
  time_type sync( time_type const & offset ); // ... until offset
  ```

- \textbf{Caveat:} Determine efficiently, when all requests at a particular time have been completed!}
Hierarchical Tracing & Processing
Advanced tracing for TLM Virtual Platforms

- Hierarchy of (user-defined) **stream processors** & sinks
  - Streams can be processed by an **extensible set of (pre-)processors**
  - Hierarchically connected during simulation
  - Combine / convert / reduce values during simulation
  - Sinks can write streams to storage backends for **offline analysis**
  - Stream processors are triggered by stream updates

- A **power model** is just a specific stream processor, combining activity information with physical parameters

- Automatic stream separation and merging of **multiple initiators** optionally supported for **temporal decoupling with overlaps**
  - Example: Accumulate overlapping power consumptions due to loss of time resolution → retains total energy consumption
Several **backends** are supported

- Statistics/metrics collection, **report generation**
- `sc_trace`-based implementation (with *limited decoupling support*)
  - (processed) streams may not lag behind simulation time
- Custom **VCD** file backend (with `sc_trace` compatibility)
- **Synopsys VP Explorer** integration (based on vendor-specific instrumentation API)

**Custom backends** can be added transparently

- **Traits classes** for value types, providing conversion to numeric values, bitvectors, and string representations
- **Type registry** for specialized output handling, resolved at elaboration time
15 Outline
Medium-term standardisation

1. COMPLEX Power-Aware Virtual Prototyping
2. Scalable System-Level Power Model
3. Advanced tracing for TLM Virtual Platforms
4. Medium-term standardisation
   - Extra-functional parameters
   - Extended Simulation Callbacks
5. Conclusion
Extra-functional parameters

Medium-term standardisation

(Extra-Functional) **parametrisation of SystemC models** is under standardisation in Accellera’s **SystemC CCI Working Group**.

- Support for **physical units** is addressed, but may not be explicitly included in the standard.
- Details and granularity of a standardised **System-Level Power Model** is mostly a question of the **methodology / design flow**.
  - Align with UPF/CPF and the related effort towards ESL power modelling.

Path towards Standardisation

- Collect existing practices and requirements for potential future **Extra-Functional Base Model** (c.f. TLM-2.0 base protocol)
  - Common set of parameters (names, units)
  - Rules for composing, using, (run-time) changing
- Postpone standardisation after **completion of CCI configuration** aspects
  - Standardised parametrisation addresses most technical showstoppers
  - Parameter selection can be aligned between model providers/integrators
### 17 Extended Simulation Callbacks

**Motivation**

- SystemC’s capabilities for adding **custom introspection** code (like a power-model) are limited
- In TLM-2.0 virtual platforms, events and **transactions may arrive out-of-order**
  - Temporal decoupling required for today’s complex platforms
  - Difficult to determine, when all conflicting transactions from different initiators have arrived
- No safe state to **extract model information** available
  - Process scheduling is **implementation-defined**
  - `sc_trace` updates are explicitly inserted in simulation loop internally
  - Workarounds based on `sc_pending_activity_` functions are inefficient and non-interoperable

**Proposed solution: Extended Phase Callbacks**

Add callback mechanism to individual simulation phases: `initialization`, `(evaluation)`, `update`, `simulation time advance`
18 SystemC simulation cycle
Extended Simulation Callbacks

- Initialization
  - R: runnable processes
  - D: delta notifications
  - T: timed notifications

- Notify
  - notify(0, SC_FS)
  - notify(>0, SC_FS)

- Evaluate
  - R = {}

- Update
  - U = {}

- Next Delta
  - D → R
  - R ≠ {}

- Increase Simulation Time
  - T → T(time) R
  - R = {} ∨ time = end_time

- SC_stop()

- SC_pause()

- Information flow
- Control flow
- Phase callback

Finished, return to sc_main()
19 Interface proposal
Extended Simulation Callbacks

- Extend `sc_status` enum with new phases
  
  ```
  SC_END_OF_INITIALIZATION, // (I)
  /* SC_END_OF_EVALUATION, // (E) [TBD] */
  SC_END_OF_UPDATE,  // (U)
  SC_BEFORE_TIMESTEP, // (T)
  ```

- Add new `callback` function to `sc_object`
  ```
  virtual void simulation_phase_callback()
  {
      /* warn, if base implementation is called */
  }
  ```

- Explicitly subscribe to specific phase(s)
  ```
  obj.register_simulation_phase_callback( SC_BEFORE_TIMESTEP );
  ```

- Subscription to multiple phases (e.g. `(SC_PAUSED|SC_STOPPED)`) → determine current phase via `sc_get_status` within callback

- Definition of extended callback semantics for all simulation phases (consistency)

- Callback body may not call arbitrary notification functions (→ keep kernel scheduling stable)
Performance overhead

Extended Simulation Callbacks

- **Proof-of-concept implementation** available, to be discussed in LWG
- **Performance overhead** seems to be **acceptable**
  - If feature is disabled in the kernel, no overhead visible (< 1%)
  - Without registered callbacks, but active infrastructure, overhead below 3% on all platforms (64-bit seems to be slightly more affected)
  - Active callback significantly faster than additional **SC_METHOD**

- **Artificial benchmark**
  - Two SystemC processes
  - 1,000,000 timed notifications
  - 100 delta cycles per timestep

- **Simple TLM-2.0 Virtual Platform**
  - ISS, AT Interconnect, Memory (with DMI), HW accelerator (DCT)
21 Outline

Conclusion

1. COMPLEX Power-Aware Virtual Prototyping
2. Scalable System-Level Power Model
3. Advanced tracing for TLM Virtual Platforms
4. Medium-term standardisation
5. Conclusion
Conclusion
Modelling extra-functional properties in SystemC

- Considering **extra-functional properties** is increasingly important in early phases of the design flow
  - Trade-off: performance ↔ power ↔ design effort
  - Optimisation of power management
  - Verification of power intent
  - **COMPLEX simulation framework** provides customizable infrastructure

- Existing approaches rely on custom/vendor-specific extensions
  - **Reduced model interoperability**!
  - Start the standardisation process **related to SystemC**
  - Aligned with other activities at e.g. Si2 Low Power Coalition (CPF) and IEEE P1801 (UPF)

- **But: Standardisation takes time!**
  - Highly methodology/flow dependant
  - **Long-term goal**: Converging on a common, flexible **Extra-Functional Base Model**
  - **Next steps**: Provide **enabling infrastructure** for custom extra-functional modelling
  - Target medium-term extensions to **improve model expressiveness** and **integration** of custom introspection techniques (LWG+CCIWG)
Thank you! Any questions?

Philipp A. Hartmann, <philipp.hartmann@offis.de>

This work has been partially supported by the COMPLEX FP7 European Integrated Project, funded by the European Commission under Grant Agreement 247999, and the EnerSave project, funded by the German Ministry of Research (BMBF), registration number 1BE1100.