Beyond algorithms and performance: Modelling extra-functional properties in SystemC

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

April 15, 2012
ISCUG’2013
Noida, India
1 About OFFIS
http://www.offis.de/en/

- Research & Technology Transfer in ICT
  - located in Oldenburg, Lower Saxony, Germany
  - approx. 300 employees
  - income: approx. 13 million € in 2011, 73% third-party funding

- Application Know-How & Technology Excellence
  - 3 R&D Divisions: Energy, Health, Transportation
  - 6 inter-disciplinary technology clusters
    - HW/SW System Design Methodology,
    - Dependable System Design, ...

- Hardware/Software Design Methodology Group
  - Seamless embedded HW/SW/AMS System Design Methodology
    - Multi abstraction layer modeling, simulation, refinement, synthesis
  - Impact analysis of execution platform (SoC) on functional and non-functional properties
    - Time (throughput, deadlines,...), power consumption, chip area, cost
  - Contribution to SystemC standardization
2 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
3 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
4 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
5 COMPLEX project
http://complex.offis.de

**CO**design and power **M**anagement in **P**Latform-based design-space **EX**ploration

- 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.

![Diagram](image-url)
6 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)
7 Outline

Scalable System-Level Power Model

1. COMPLEX Power-Aware Virtual Prototyping

2. Scalable System-Level Power Model
   - Requirements
   - Standardisation

3. Advanced tracing for TLM Virtual Platforms

4. Medium-term standardisation

5. Conclusion
8 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

- **Align with UPF/CPF** and the related effort towards ESL power modelling
  - Mapping of CPF/UPF concepts to SystemC/TLM simulation model
Scalable System-Level Power Model

- Design param.
  - Technology
  - Process Variation
  - IR Drop
  - … ??

- 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]

- Dynamic power
  - Module

- Power Domain

- System

- Die

- Power Domain

- Module
10 Standardisation
Scalable System-Level Power Model

- (Extra-)Functional **parametrisation of SystemC models** is under standardisation in ASI’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.

**Recommendation for 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
11 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, Hierarchical Tracing
   - Standardisation
4. Medium-term standardisation
5. Conclusion
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 (virtual platform) tools, but highly vendor-specific

- **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**
  - Filtering and data reduction (aggregation, averaging, selection)
  - Collection of user-defined performance metrics
  - Run-time configurable granularity (Region of Interest)
Stream-based tracing of extra-functional properties

Advanced tracing for TLM Virtual Platforms

- Tracing is based on \((time, value)\) streams per component
  - Streams are hierarchically named SystemC objects
  - **Strongly-typed values** for type safety with support for physical units
  - Fine granular control over (local) time offset, synchronisation with a local clock
  - push APIs for both absolute \((start, value, [duration])\), or relative \((duration, value)\) tuples
  - Non-overlapping tuples enforced by MergePolicy

- Hierarchy of (user-defined) stream preprocessors (and 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

- Automatic separation and merging of multiple source processes (initiators) optionally supported for temporal decoupling with overlaps
  - Example: Accumulate overlapping power consumptions due to loss of time resolution
    \(\rightarrow\) retains total energy consumption
Advanced tracing for TLM Virtual Platforms

- SystemC’s built-in tracing capabilities are not sufficient for extra-functional properties, yet.

- Tracing interfaces, being part of model introspection, will be handled by ASI’s SystemC CCI Working Group.

- Efficient (kernel-independent) implementation needs extended simulation callbacks, or vendor-specific instrumentation APIs.

**COMPLEX Tracing Framework**

- Flexible, but independent interface to extra-functional model.

- Suitable for state-based or annotation-based instrumentation.

- **Proof-of-concept implementation** may be contributed to ASI CCIWG in the future.
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
   - Extended Simulation Callbacks
   - Extensible TLM Direct Memory Interface
5. Conclusion
16 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` is explicitly inserted in simulation loop already

Proposed solution: Extended Phase Callbacks

Add optional callbacks to individual simulation phases
- (evaluation, update, simulation time advance)
17 SystemC simulation cycle
Extended Simulation Callbacks

- **initialization**
- **evaluate**
  - $R = \{\}$
- **update**
  - $U = \{\}$
- **next delta**
  - $D \rightarrow R$
  - $R \neq \{\}$
  - $R = \{\}$
- **increase simulation time**
  - $T \rightarrow T(time)$
  - $R = \{\} \lor time = end_time$
- **finished, return to sc_main()**

**Notify()**
- $R$: runnable processes
- $D$: delta notifications
- $T$: timed notifications

**Notify(0, SC_FS)**
- $D = \{\}$

**Notify(>0, SC_FS)**
- $T = \{\}$

**Information flow**
- **Control flow**
- **Phase callback**

**Note:**
- $sc\_pause()$
- $sc\_stop()$

---

Philipp A. Hartmann
Modelling extra-functional properties in SystemC
April 15, 2012
18 Interface proposal
Extended Simulation Callbacks

- Extend `sc_status` enum with new phases
  
  `SC_END_OF_EVALUATION_PHASE`, // (1)
  `SC_END_OF_UPDATE_PHASE`,    // (2)
  `SC_END_OF_TIMESTAMP_PHASE`, // (3)

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

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

- 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)

- Proof-of-concept implementation available, to be donated/discussed in LWG
19 Extensible TLM Direct Memory Interface
Medium-term standardisation

- TLM-2.0 Direct Memory Interface (DMI) is very useful for high-performance virtual platforms
  - `tlm_dmi` provides timing hints to improve accuracy
  - **Good:** Memory access without going through the virtual platform
  - **Bad:** Memory access without going through the virtual platform

- Adding additional extra-functional information is **not possible with DMI**, yet
  - Add more hints? Only Power? **Does not scale!**
  - Initiator knows nothing about internal requirements of DMI-allowing target
  → **Disable DMI** in such cases. **Is that it?**

Early Idea: Extensible DMI

- Add possibility to **add callbacks** to `tlm_dmi` response
  - Initiator is required to **trigger callbacks** after DMI access(es)
  - **Granularity/parameters** not yet decided (per R/W access, per quantum, both?)

- Extend `tlm_dmi` access specification (for backwards compatibility):

```cpp
  tlm_dmi::DMI_ACCESS_READ_WITH_CALLBACKS, tlm_dmi::DMI_ACCESS_WRITE_WITH_CALLBACKS,
```
20 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
21 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

- 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 design methodology/flow related
  - **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.
23 Additional material
Y-chart for extra-functional properties

6 Y-chart for extra-functional properties
7 Virtual Platform Power and Timing Annotation Flow
8 Power-Aware High-Level Synthesis
9 Non-invasive TLM Power State Machines
24 Y-chart (behaviour)

Y-chart for extra-functional properties

Models of computation (MoCs)

- Behavior (function)
  - Specification
  - Algorithm
  - Boolean logic \((a \lor b)\)
  - Transfer \(\frac{\partial U}{\partial t} = \ldots\)

Models of structure (MoSs)

- Structure (netlist)
  - System synthesis (task scheduling, PE & bus binding and allocation)
  - PE & communication synthesis

- Processor
- Logic
- Circuit

- Physical (layout)
  - CAD data
  - Standard cells
  - Gates
  - Transistors
Y-chart (system-level time modeling)

Y-chart for extra-functional properties

- **Behavior** (function)
- **Physical** (layout)
- **Structure** (netlist)

**Timed MoCs**
- Algorithm with timing annotations
- Specification with timing constraints

**System**
- Processor
- Logic
- Circuit

**MoS: timed resources**
- PE, Bus
- RTL
- Gates
- Transistors

**Source code annotation**
- Critical paths
- Delays
- Switching times
- Wire-load models

**f [Hz] per unit**
- ISA
- Bus/Channel configuration

Philipp A. Hartmann
Modelling extra-functional properties in SystemC
April 15, 2012
Y-chart (system-level power modeling)

Y-chart for extra-functional properties

Power-Annotated MoCs

Behavior (function)
- specification with power constraints
- algorithm with power annotations

Structure (netlist)
- CGD [V] per unit R_{leak}
- input vectors macro models
- switching activity
- switched capacitance

Physical (layout)
- area [µm^2] per component, bus length prediction

System
Processor
Logic
Circuit

MoS: power resources

Source code annotation
Y-chart (system-level thermal modeling)

Y-chart for extra-functional properties

Behavior (function)

Physical (layout)

Structure (netlist)

System

Processor

Logic

Circuit

Temperature-Annotated MoCs

PE & communication Synthesis

System synthesis (task scheduling, PE & bus binding and allocation)

package & environment structure

component placement

package selection

power over time per component

algorithm with power and timing

specification with thermal constraints

Philipp A. Hartmann Modelling extra-functional properties in SystemC April 15, 2012
28 Y-tube (multi-objective Y-chart)
Y-chart for extra-functional properties
29 Additional material
Virtual Platform Power and Timing Annotation Flow

6 Y-chart for extra-functional properties

7 Virtual Platform Power and Timing Annotation Flow
   - Hardware/Software Task separation
   - Component-level Power & Timing Estimation
   - Virtual System Generation

8 Power-Aware High-Level Synthesis

9 Non-invasive TLM Power State Machines
### 30 Virtual Platform Power and Timing Annotation Flow

**Overview**

- **Executable specification**
  a) Task graph model of application
  b) Application scenario stimuli
  c) Task to platform resource mapping
  d) Processing, Communication and Memory blocks

- **Estimation & model generation**
  e) Extraction of task’s behaviour
  f) **Power & Timing Estimation** and back-annotation to input model
  g) **HW IP components** with power and timing information
  h) Assemble Virtual Platform

- **Simulation**
  i) Executable power-aware VP
  j) Configurable tracing
Starting from
- Executable task model
- Platform / resource model

Task mapping
- Assigning application tasks to platform resources
- Custom Hardware, Software, IP blocks

Automatic extraction of task behaviour for estimation
- Prepare input component models for power and timing estimation point-tools
- Manual specification of Power State Machines for IP components
- Back-annotation for Virtual Platform simulation
32 Hardware/Software Task Estimation and Back-Annotation
Virtual Platform Power and Timing Annotation Flow

Forward individual task behaviour to appropriate estimation tool for timing and power back-annotation.
33 Additional material
Power-Aware High-Level Synthesis

6 Y-chart for extra-functional properties

7 Virtual Platform Power and Timing Annotation Flow

8 Power-Aware High-Level Synthesis
   - Hardware Basic Blocks
   - Back-Annotation and Model Generation

9 Non-invasive TLM Power State Machines
34 Custom Hardware Estimation Flow
Power-Aware High-Level Synthesis

Integration into PowerOpt HLS technology
(now provided by OFFIS)

1. Perform “classical” High-Level synthesis
2. Power characterisation of design properties
3. Identification and characterisation of individual control-steps
4. Generation of fast and accurate functional models with power and timing
characterisation of design properties
Power-Aware High-Level Synthesis

Leakage
- Leakage estimate is provided by PowerOpt base technology, after data path is fixed
- Modelled as apparent resistance to enable voltage scaling

Clock-tree power
- Assumed to be constant offset to the power dissipation
- Modelled as additional switched capacitance offset per cycle

Controller power
- Estimation provided by PowerOpt as average activity
- Assumed to be uniformly distributed per (active) control-step
Power-simulation of each individual RT component would be prohibitively slow.

**Idea:** Combine and characterize all activated RT components in each control-step to an **Hardware Basic Block**.

**Identification** of active RT components:
- Determine *target registers*, enabled for storing updated inputs.
- Traverse backwards towards *source registers*, tracking multiplexer selects.
- Traverse forward to collect *extra functionality* that is active, but not needed for the result.

Strictly **sequential control-steps** can be optionally combined to **multi-cycle HBBs**, further improving simulation performance.
37 Characterisation of individual control-steps (II)

Hardware Basic Blocks

Dynamic power model

- Simplest (and fastest) dynamic HBB power model uses average activity
- Based on stimuli provided during synthesis

- Control-step activity is assumed to be the sum of average switched capacitance of each activated component: 
  \[ A = \sum_{n=1}^{N} \frac{1}{M_n} \sum_{m=1}^{M_n-1} \alpha (\nu_n, \text{pattern}_{m-1}, \text{pattern}_m) \]
  - Active components \( \nu_1, \ldots, \nu_N \)
  - \( M_n \) is the number of stimuli applied to \( \nu_n \)
  - \( \alpha(\nu_n, \ldots) \) is the switched capacity by applying given patterns consecutively

- On-going research is evaluating probabilistic models to address internal correlations (data dependencies, inter-block dependencies)

- Multi-cycle HBBs can be combined by averaging (with loss of resolution)
Back-Annotation and Model Generation
Power-Aware High-Level Synthesis

- **Generated module container for generic white-box IP blocks**
  - Functional and power models
  - Communication & Tracing

- **The functional model implements the function-call of the original module**
  - Execute the behaviour
  - Perform (extra-functional) simulation steps, until functional model finishes

- **The functional model** consists of hierarchical, plain C++ “processes”
  - RT data path (hardware basic blocks)
  - Controller (switch statement)
  - Possibly sub-processes
39 Additional material
Non-invasive TLM Power State Machines

6 Y-chart for extra-functional properties

7 Virtual Platform Power and Timing Annotation Flow

8 Power-Aware High-Level Synthesis

9 Non-invasive TLM Power State Machines
   - TLM Observable Sockets
   - Protocol State Machine
   - Power State Machine
**Goal:** Add extra-functional power model to **black-box IP components** (memories, interconnects, accelerators, ...)

Ingredients: Power State Machine (**PSM**), Protocol State Machine (**PrSM**), Transaction Snooping

---

![Diagram showing the relationship between IP Component, TLM-2.0 Functional Model, PrSM, and PSM with associated power information and transaction snooping.](image-url)
Observation of TLM-2 communication
Non-invasive TLM Power State Machines

- Approximate internal state by observing the interaction with environment
  - Generation of transparent PSM wrapper with special observable sockets
  - Transaction forwarding to/from component
  - Bookkeeping and protocol handling (LT/AT)

- Two new convenience socket types
  - `tlm_utils::tlm_observable_initiator_socket<BUSWIDTH>`
  - `tlm_utils::tlm_observable_target_socket<BUSWIDTH>`

- Observer infrastructure to register and trigger Protocol State Machine transition conditions
  - Other use-cases supported as well
  - Details presented at ISCUG’2013 tutorial
Following the register interface description, a Protocol State Machine is defined.

Abstracts from TLM-2 artefacts towards application.

PrSM is triggered by transaction conditions, matching specific transaction properties:
- Address, data, command, phase, ...
- User-defined conditions supported.

Triggers PSM and may update extended state.

Separate address ranges into categories:
- Configuration data
- Control data
- Payload data
Current states of Power State Machine provides **average switched capacitance per cycle**

States (and transitions) report updates to the tracing, according to current power model

Determine relevant power states from the **vendor’s datasheet**
- Or by using **low-level power simulations**
- Needs to be done manually

Transitions are triggered
- Externally via explicit **PSM events**
- Internally via **timeouts**

Extended **state variables** can be read to influence
- Current activity
- Timeout expressions