Design for Verification in System-level Models and RTL

Anmol Mathur
Venkat Krishnaswamy
Calypto Design Systems, Inc

Outline
- What is a system-level model?
- Verification Methodology from SLM to RTL
- Sources of inconsistency between SLM and RTL
- Recommendations for keeping SLM and RTL consistent
- Conclusions

Market Trends: ESL Drivers

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Economics</th>
<th>Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Development Cost</td>
<td>Design reuse</td>
</tr>
<tr>
<td>Power / battery life</td>
<td>Time to Revenue</td>
<td>Platform design</td>
</tr>
<tr>
<td>Design updates</td>
<td>Respin reduction</td>
<td>Optimization</td>
</tr>
</tbody>
</table>

Leveraging system-level models for RTL verification can reduce the RTL verification bottleneck

Levels of Systems

- System Level Design
- Boundary assertion verification
- Hardware/software Interface verification
- Block verification

Uses of System-level Models
- Higher level of abstraction resulting in faster simulation turnaround time
- Performing architectural tradeoffs and performance validation
- Platform for software development

Functional Verification Landscape

- System level: Simulation
- RTL level: RTL-gate Equivalence Checking
- Gate level: Simulation/emulation based verification
- RTL- SL co-simulation
Co-simulation Based SLM to RTL Verification

- Input transactors to match differences in input protocols
- Output comparison needs to account for timing difference in when corresponding outputs are generated

What is SEC?

- Equivalence check designs with **temporal interface differences**
- Equivalence check designs with **internal state differences**
- Effective for eliminating bugs and ensuring consistency without testbenches

Sequential Logic Equivalence Checker (SLEC)

- SLEC comprehensively proves functional equivalence
  - System to RTL
  - RTL to RTL
- Identifies design differences (bugs)
- No testbench or assertions required

Outline

- What is a system-level model?
- Verification Methodology from SLM to RTL
- Sources of inconsistency between SLM and RTL
- Recommendations for keeping SLM and RTL consistent
- Conclusions

Computation and Communication Accuracy Requirements in SLMs

Conflicting Requirements in SLMs

- Need for speed
  - Model computations approximately
  - Reduce number of communication events by not modeling fully accurate communication
- Need for modeling accuracy
  - Architectural tradeoffs need modeling of the relevant micro-architectural components
  - Firmware development needs appropriate modeling of interface timing and architectural state
  - Functional reference models need to be computationally exact
Sources of Computational Inaccuracy

- Data agnostic models
  - Model of a network processor may only model packet headers and not packet data
  - Bus models often abstract away the actual data, since the metric of interest is bus contention
- Differences in data types
  - RTL uses bit-vectors specifying the exact data precision needed
  - C models model values using fixed width native C types (32 bit int, 64 bit long)
- Lack of bit-manipulation operations in C/C++
  - Explicit masking and shifting is needed to select bit-ranges not aligned with C native types

Issues in Fixed-point Arithmetic

- The nice algebraic properties of the +, * are not true when arithmetic computations are done using finite precision
- Addition is not associative in finite precision!
- Results in computational inconsistency between RTL models written with smaller precision bit-vectors and C models using wider native C types

Bit-approximate Models

- SLM may use floats while RTL uses fixed-point computations
- RTL may not fully model IEEE floating point standard due to high cost of full IEEE compliance
- RTL may explicitly perform rounding and truncation on intermediate computations for lowering operand widths for downstream operations

Sources of Communication Differences

- Abstraction of timing information from models to reduce communication complexity and improve simulation speed
- Lack of full micro-architectural detail in SLM
- SLMs are typically parallel implementations of an algorithm; RTL serializes algorithms for efficient hardware implementation

Serial RTL Design

- Verilog RTL implementation:
  - Throughput = 16 cycles.
  - Latency = 16 cycles

Untimed Functional Design

- Untimed DES algorithm:
  - written in C
  - wrapped in SystemC

Transaction Equivalence

- Computational view
  - Transactions are defined as the common “units” of computation between the SLM and RTL
  - For algorithmic SLM vs RTL: transactions are the computation of the function the algorithmic model computes.
  - For timed/partially timed SLM vs RTL: transactions could be finer grained — corresponding to duration between corresponding states in the two models
- End of transaction indicated by
  - Fixed number of cycles
  - Handshaking signal on output
Outline

- What is a system-level model?
- Verification Methodology from SLM to RTL
- Sources of inconsistency between SLM and RTL
- Recommendations for keeping SLM and RTL consistent
- Conclusions

Recommendations for SLM vs RTL Consistency

- Ensure SLM design team and RTL design team communicate and both realize the value of keeping SLM and RTL consistent
- Environment for comparing and keeping SLM and RTL consistent need to be in place
  - Simulation based comparison of SLM and RTL
  - Sequential equivalence checking between SLM and RTL
- SLM is not always golden
  - Inconsistency between SLM and RTL can be due to SLM bug, RTL bug or due to conditions that can never arise in the regular operation of the system (don’t cares)

Consistent Design Partitioning

- Correspondence between functional decomposition in SLM and module decomposition in RTL
  - Easier to plug RTL modules into system-level simulation
  - Allows easier correspondence between input and output interfaces of SLM sub-functions and RTL modules for sequential EC
  - Allows for hierarchical bottom-up verification
- Key design communication interfaces (buses in SOCs) form natural points of correspondence between SLMs and RTLs

Ensuring Bit-accuracy in SLMs

- Use finite bit-vector libraries (such as SystemC bit-vector and fixed-point data types) instead of native C data types, if possible
  - Simulation speed can be negatively impacted
- Do not use floating point data types in SLM if the RTL model is really a fixed-point model
  - Very hard to maintain bit-accuracy since floating-point computations use implicit rounding modes of these data types
- Ensure no inadvertent information loss when performing bit-manipulations in SLM using native C/C++ data types

Orthogonal Computation and Communication

- Keep computation and communication well separated
  - Model communication explicitly via read/write functions on communication interfaces
  - Results in ability to refine communication interfaces if more accuracy is desired without a full rewrite of SLM
  - Allows the same computational model to be re-used in multiple models with different communication accuracy
- Differences in communication accuracy between SLM and RTL are mostly intentional
  - Separation of computation and communication clearly spells out the communication interface
  - Allows for explicit development of input/output transactors to reconcile interface timing differences for co-simulation and sequential EC

Modeling for Static Design Inference

- Design analysis tools such as sequential EC need to be able to determine the sizes of design objects statically at compile time
- Use static arrays instead of dynamic memory allocation
  - This often results in faster simulation, without significantly sacrificing modeling flexibility
- Excessive use of pointer aliasing and pointer arithmetic can result in conservative static analysis
- Upper bounds on iteration count needs to be specified for data-dependant loops
  - Conditional exits from loops with static bounds can help

Summary

- System-level modeling is here to stay in hardware design teams
- Designing SLMs and RTL to facilitate keeping them functionally consistent is critical to design teams
- Cohesion and up-front planning between system-level modeling and RTL design teams can greatly help keep SLMs and RTLs consistent
- Simple coding guidelines can ensure computational and communication consistency between SLMs and RTL without sacrificing SLM modeling flexibility and execution speed