EE 382V - SoC

System Level Design Methodology

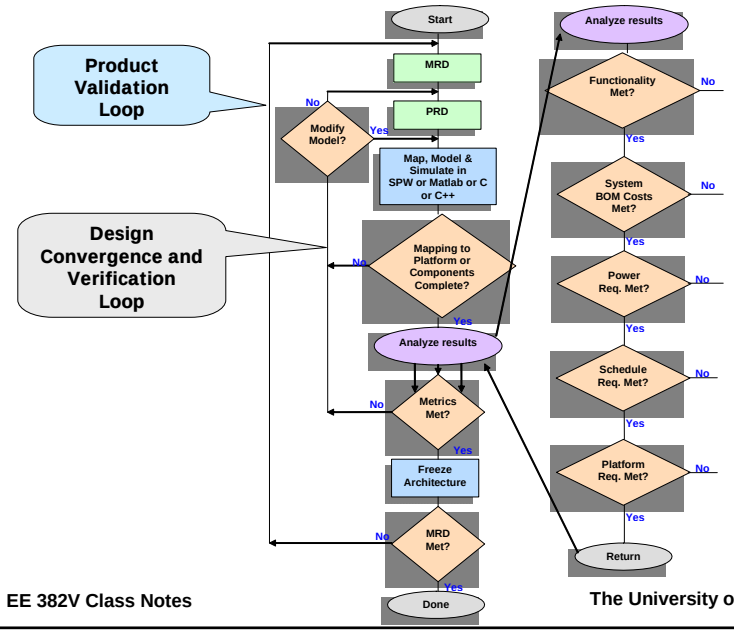
Andreas Gerstlauer
Mark McDermott
Steven Smith

Fall 2009

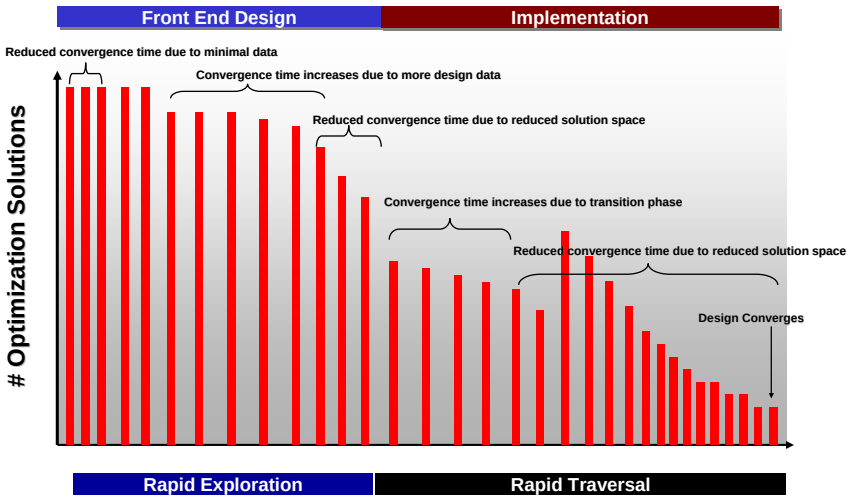
Agenda

- Design Convergence
- System Level Design
- Modeling
- Verification
- Summary

Product Design and Methodology Flow Chart

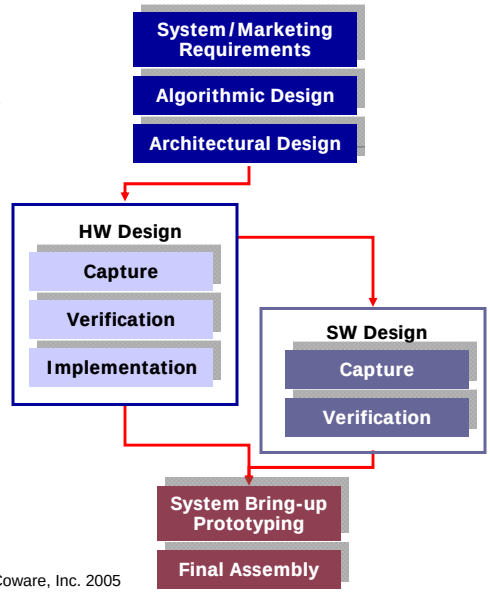


Design Convergence Iteration Profile



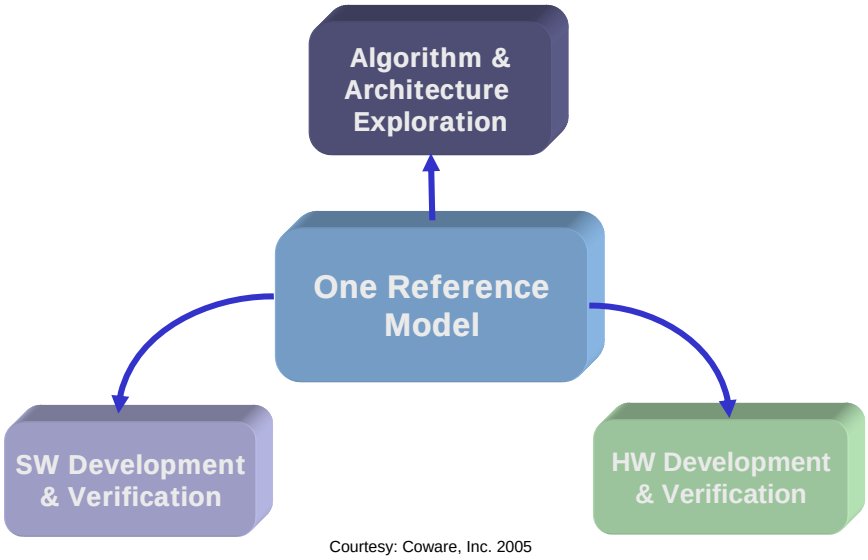
Issues with HW Centric System Design Flows

- RTL language centric
- Dysfunctional levels of abstraction
- SW Design Cycle often serial to HW Design Cycle
  - Lack of unified hardware-software representation
- Missing executable platform models early in cycle
- SW/HW integration is tough
- Simulation speed is critical
- Partitions are defined *a priori*
  - Hard to find incompatibilities across HW-SW boundary
- Lack of well-defined design flow
  - Time-to-market problems
  - Specification revision becomes difficult



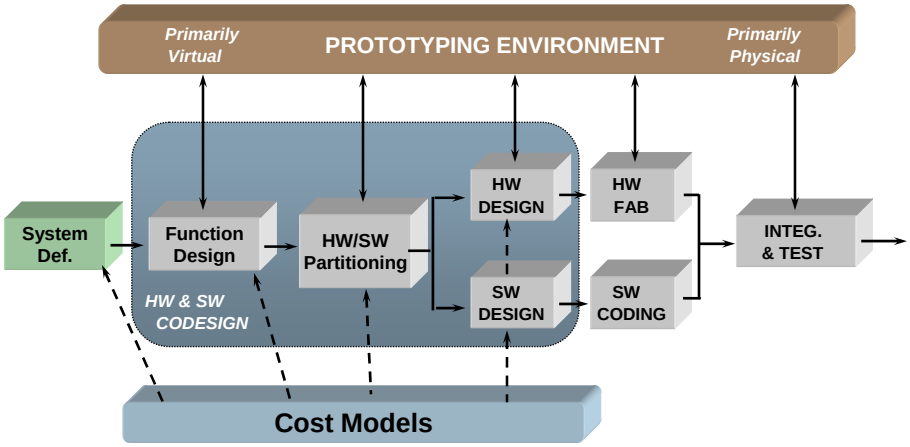
Courtesy: Coware, Inc. 2005

The ESL Solution: One Reference Model



Courtesy: Coware, Inc. 2005

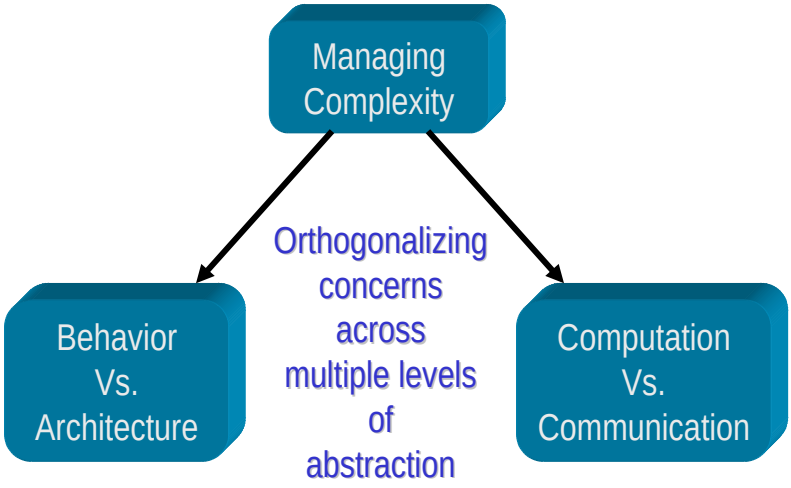
SOC Design Environment



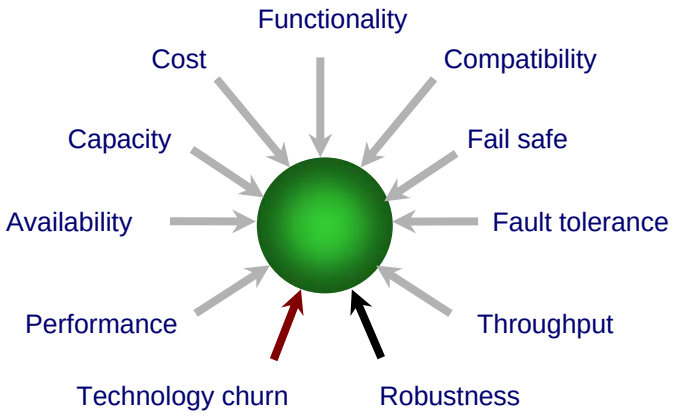
Agenda

- Design Convergence
- **System Level Design**
- Modeling
- Verification
- Summary

System Level Design



Complexity Forces



"The challenge over the next 20 years will not be speed or cost or performance; it will be a question of complexity."

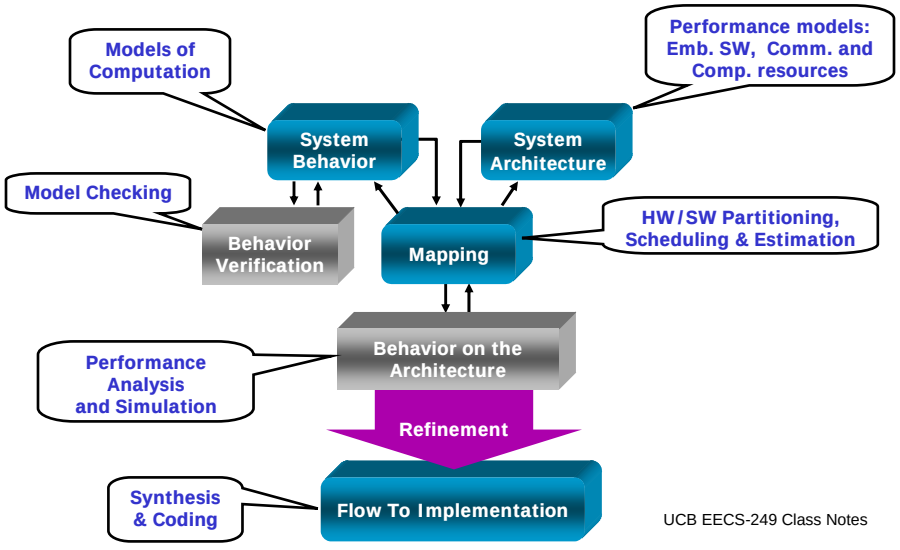
Bill Raduchel, Chief Strategy Officer, Sun Microsystems

Complexity Models

- In general reliability is inversely related to complexity
- Measures of software complexity
  - Lines of Code
  - McCabe
  - Halstead
  - Function Points

  Count branches, calls, inputs, outputs etc.
- Measure of hardware complexity
  - Number of transistors
  - Number of I/O signals
  - Silicon process

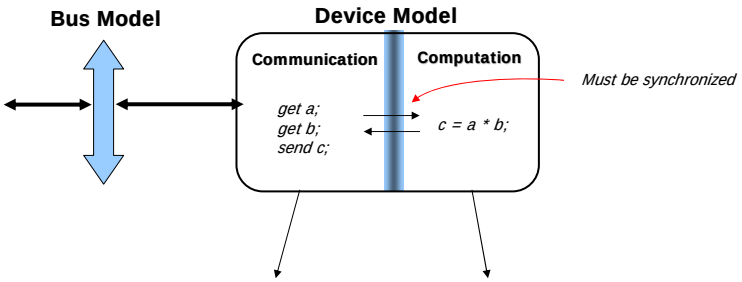
Behavior vs. Architecture



UCB EECS-249 Class Notes

Communication vs. Computation

- **Separation provides flexibility in modeling and increases IP Reuse**

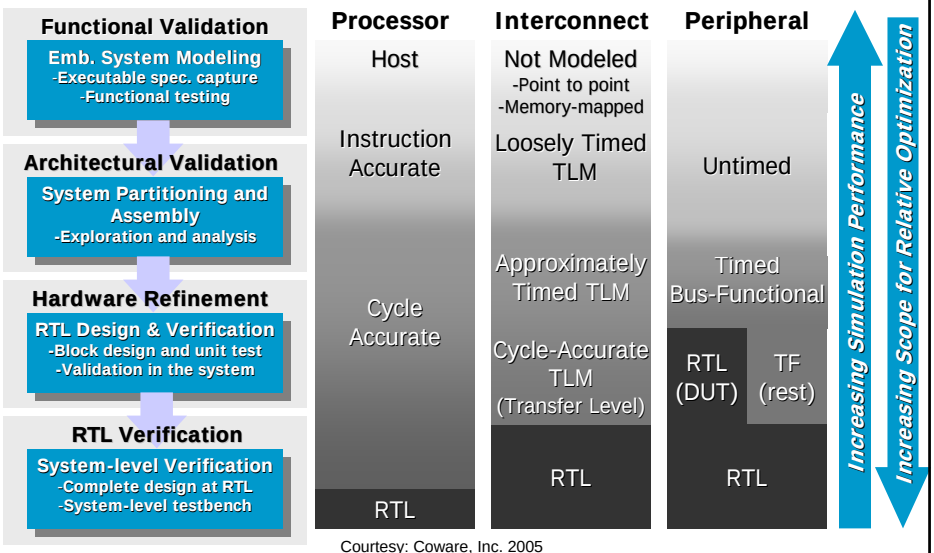


Communication can be described in a wide range of fashions, from high-level messages, to detailed signal level handshakes without impacting the behavior description.

Behavior can be described algorithmically, without the burden of the handshaking and control logic associated with bus communication.

Courtesy: Coware, Inc. 2005

Multiple Abstraction Levels



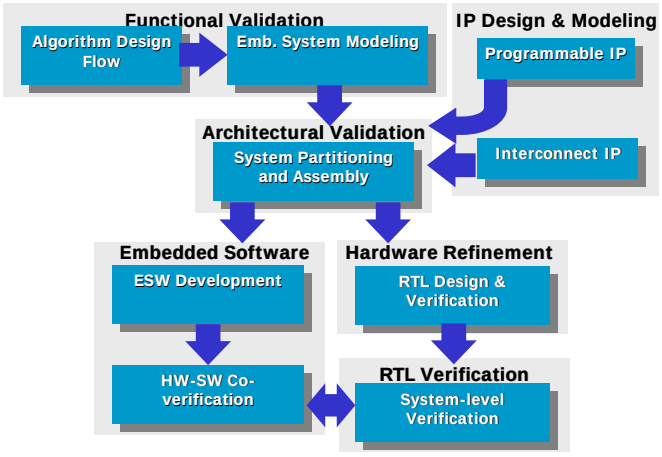
Courtesy: Coware, Inc. 2005

Two Approaches to System Level Design

- **Top down - successive refinement:**
  - Referred to as Hardware-Software Co-design
  - Connect the hardware and software design teams earlier in the design cycle.
  - Allows hardware and software to be developed concurrently
  - Starts with functional exploration
  - Goes through architectural mapping
  - The hardware and software parts are either manually coded or obtained by refinement from higher model
  - Ends with HW-SW co-verification and System Integration
- **Platform based:**
  - Hierarchical design methodology that starts at the system level
  - Enables rapid creation and verification of sophisticated SoC designs.
  - PBD uses predictable and pre-verified firm and hard blocks
  - PBD reduces overall time-to-market
    - Shorten verification time
  - Provides higher productivity through design reuse
  - PBD allows derivative designs with added functionality
  - Allows the user to focus on the part that differentiate his design

Courtesy: Coware, Inc. 2005

Top-down Design Flow

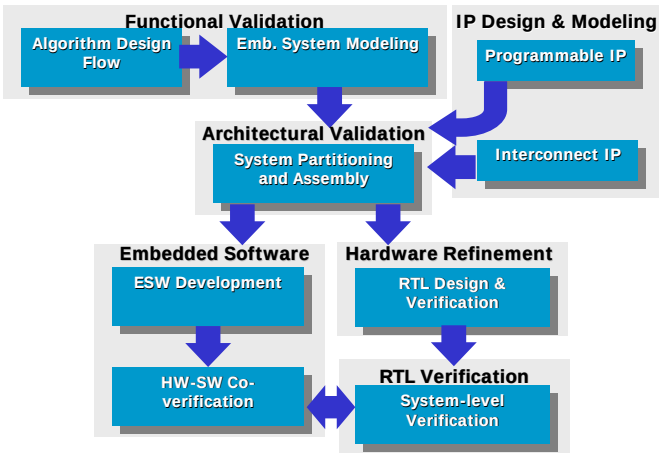


- Top-down design starts with functional validation of the system spec
- Required if you don't have a platform to start from

- Software dominates at first
- Critical need – higher performance at un-timed and "Programmers View" (PV) transaction-level abstractions

Courtesy: Coware, Inc. 2005

Platform-based Design Flow



- Platform-based design starts with architecting a processing platform for a given vertical application space
- Soft-platforms are available from various EDA vendors
- Often favored by semiconductor vendors and ASSP providers

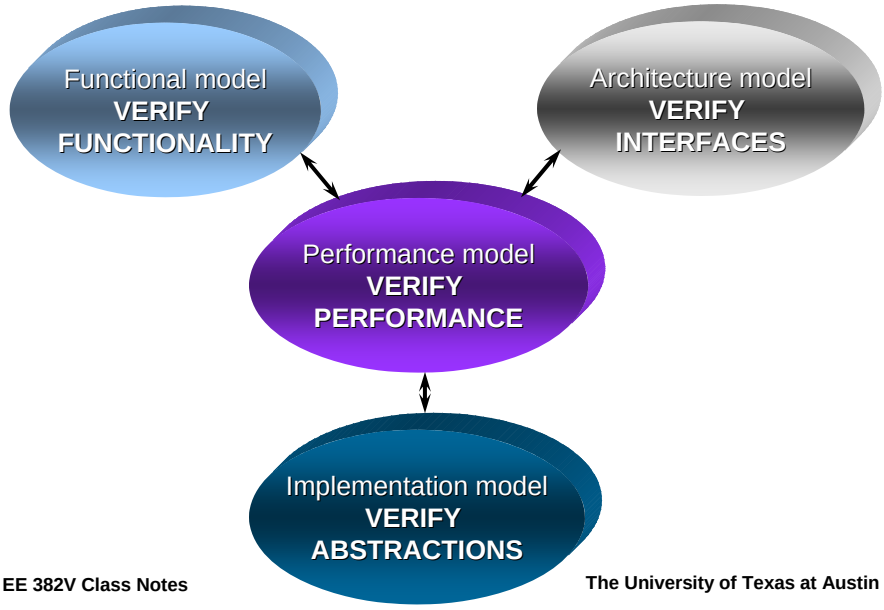
- Hardware dominates at first
- Critical need – higher performance at transfer-level TLM and cycle-accurate abstractions

Courtesy: Coware, Inc. 2005

Agenda

- Design Convergence
- System Level Design
- **Modeling**
  - **Models of Computation**
  - **Models of Communication**
- Verification
- Summary

Taxonomy of Modeling Environments



Models of Computation

- State-oriented models
  - Finite-state machine (FSM), Petri nets, hierarchical concurrent FSM (HCFSM)
- Process-oriented models
  - Kahn process networks (KPN), Datafow, flowchart
- Heterogeneous models
  - Control/dataflow graph (CDFG), Program state machine (PSM)
- Structure-oriented models
  - Block diagram, netlist
- Programming models
  - Imperative and declarative
  - Synchronous/reactive
- Simulation models
  - Discrete event

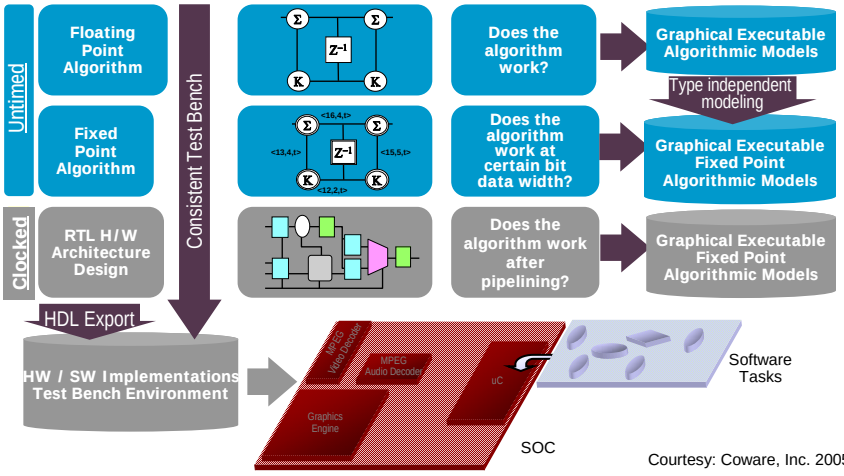
Functional Modeling & Verification

- **Model Building:**
  - Capture the relevant aspects of the system formally
  - *Abstract* model for mapping
    - No detailed wiring (busses, serial links, etc.)
    - Black-box components (ASICs, micro-controllers, DSPs, memories, etc.)
- **Model Checking:**
  - Use algorithms (i.e., tools) for model analysis, rather than for model execution (simulation)
- **Formal Hardware Verification**
  - Formalism: finite state machines
  - Algorithm: exhaustive state-space exploration

Modeling Guidelines

- **A model should capture exactly the aspects required by the system, and no more.**
  - There is not one model/algorithm/tool that fits all.
- **Being formal is a prerequisite for algorithmic analysis.**
  - Formality means having a mathematical definition for the properties of interest.
- **Being compositional is a prerequisite for scalability.**
  - Compositionality is the ability of breaking a task about A||B into two subtasks about A and B, respectively.

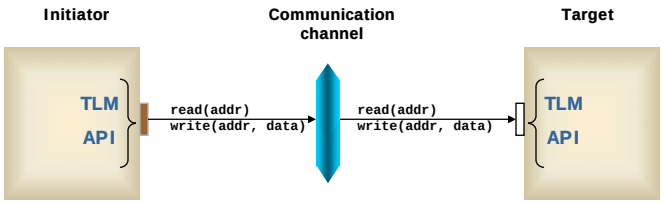
Algorithmic Modeling: SPW/MATLAB



Agenda

- Design Convergence
- System Level Design
- **Modeling**
  - Models of Computation
  - **Models of Communication**
- Verification
- Summary

Transaction Level Modeling

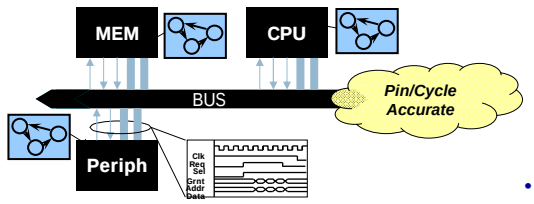


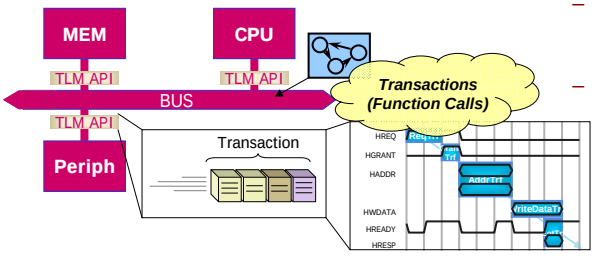
The transaction level is a higher level of abstraction for communication

For SoC, communication is dominated by the bus

Courtesy: Coware, Inc. 2005

Transaction Level Modeling - Overview



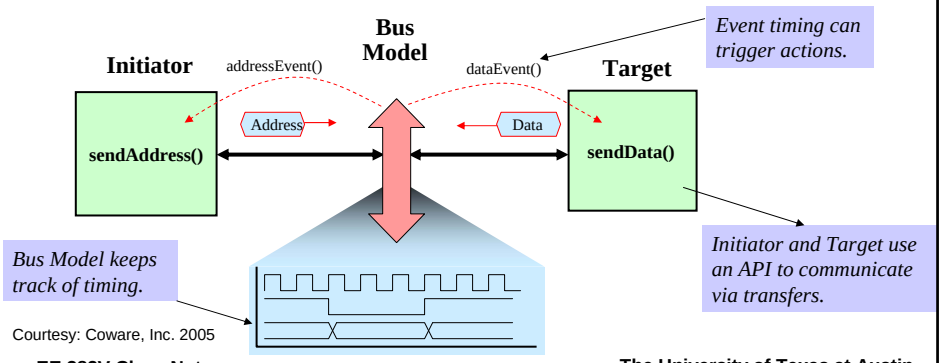


- **RTL bus: redundant complexity results in slow simulation**
  - **Each device interface must implement the bus protocol**
  - **Each device on the bus has a pin-accurate interface**
- **TLM bus: less code, fewer pins and events, yield faster simulation**
  - **Protocol is modeled as a single bus model instead of in each device**
  - **Each device communicates via transaction level API**

Courtesy: Coware, Inc. 2005

TLM - Details

- Detailed signal handshaking is reduced to series of generic events called “transfers”.
- Blocks are interconnected via a Bus Model, and communicate through an API. The Bus Model handles all the timing, and events on the bus can be used to trigger action in the peripherals.



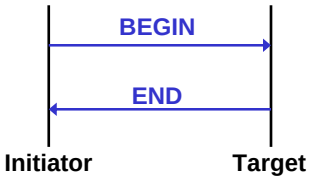
Courtesy: Coware, Inc. 2005

Goals For Standardization Of TLM Levels

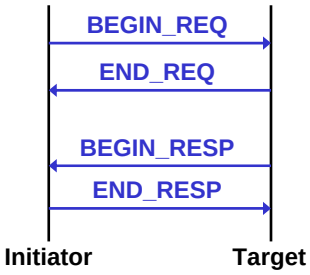
- Scope is to define a range of modeling abstraction levels for hardware and software SoC design
  - A high abstraction level enabling fast SoC models for ESW programmers and capturing system function
  - A level enabling a range of timing accuracies for SoC architects, that retains high performance
  - A level that allows full cycle-accuracy for SoC verification and HW-SW co-verification, with performance still much higher than RTL
- Levels should be defined to minimize the number of different models required
  - Minimize the number of models to provide and maintain for IP vendors (especially processors and memory models)
- Levels should be defined to minimize the amount of remodeling for the user
  - Enable a refinement process from one level to the next

SystemC/TLM 2.0 Coding Styles

- **Loosely-timed**
  - Sufficient timing detail to boot OS and simulate multi-core systems
  - Each transaction has 2 timing points: *begin* and *end*



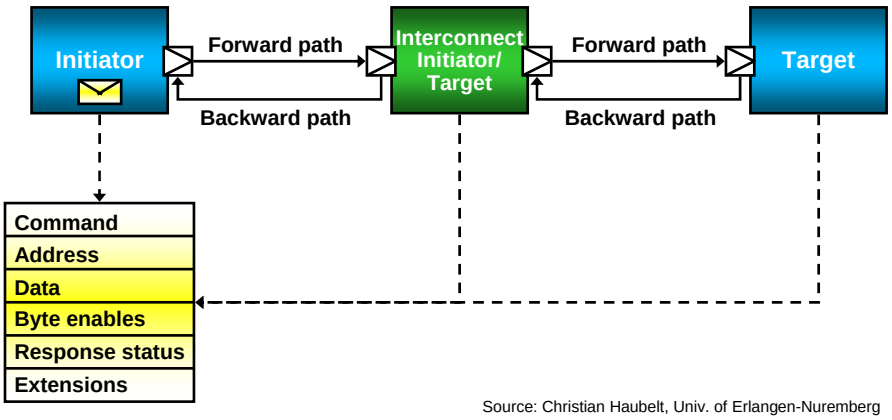
- **Approximately-timed**
  - Cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - Each transaction has at least 4 timing points



Source: Christian Haubelt, Univ. of Erlangen-Nuremberg

Initiator and Target

- **Pointer to transaction object is passed from module to module using forward and backward paths**
- **Transactions are of type generic payload**



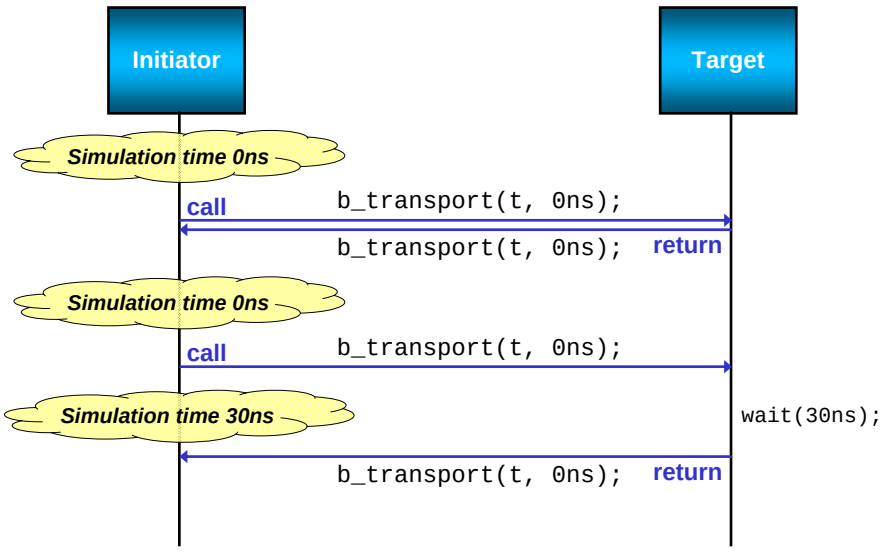
Source: Christian Haubelt, Univ. of Erlangen-Nuremberg

Blocking and Non-Blocking Transports

- **Blocking transport interface**
  - Typically used with loosely-timed coding style
  - tlm_blocking_transport_if
    ```
    void b_transport(TRANS&, sc_time&);
    ```
- **Non-blocking transport interface**
  - Typically used with approximately-timed coding style
  - Includes transaction phases
  - tlm_fw_nonblocking_transport_if
    ```
    tlm_sync_enum nb_transport_fw(TRANS&, PHASE&, sc_time&);
    ```
  - tlm_bw_nonblocking_transport_if
    ```
    tlm_sync_enum nb_transport_bw(TRANS&, PHASE&, sc_time&);
    ```

Source: Christian Haubelt, Univ. of Erlangen-Nuremberg

Blocking Transport



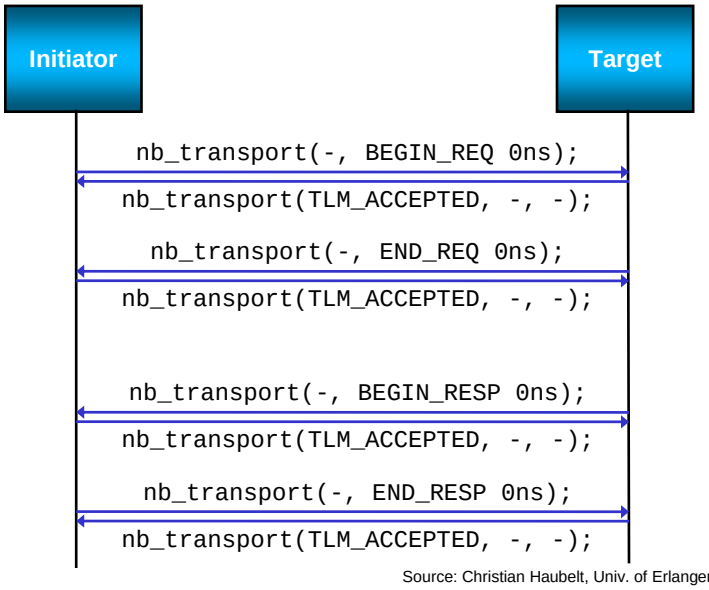
Source: Christian Haubelt, Univ. of Erlangen-Nuremberg

Transaction Phases (tlm_sync_enum)

- TLM_ACCEPTED
  - Transaction, phase and timing arguments unmodified (ignored) on return
  - Target may respond later (depending on protocol)
- TLM_UPDATED
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine to the next state
- TLM_COMPLETED
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine straight to the final phase

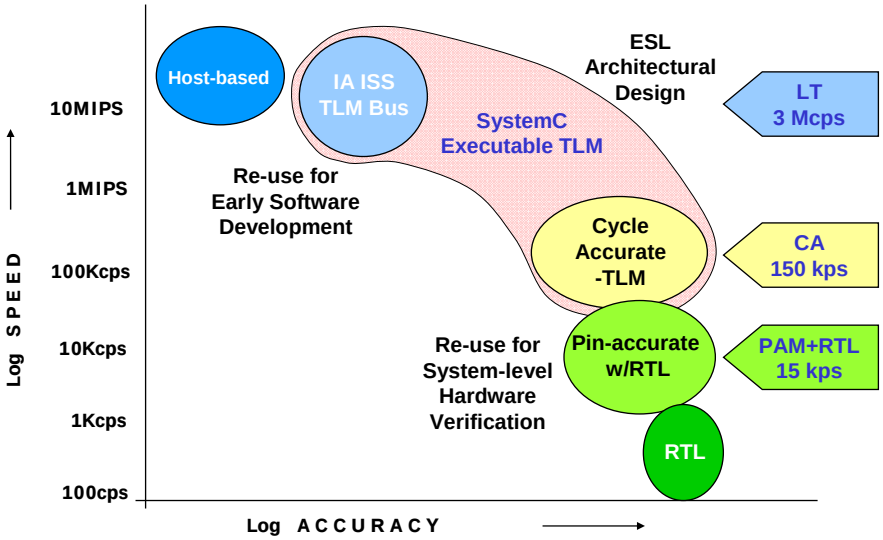
Source: Christian Haubelt, Univ. of Erlangen-Nuremberg

Non-Blocking Transport



Source: Christian Haubelt, Univ. of Erlangen-Nuremberg

Speed vs. Accuracy



Agenda

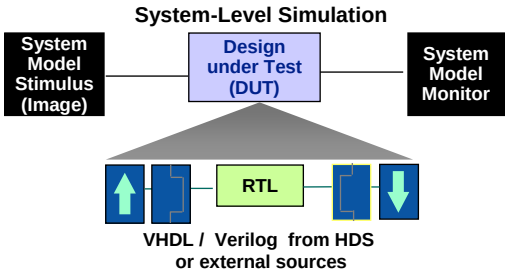
- Design Convergence
- System Level Design
- Modeling
- **Verification**
  - **Simulation**
  - **Formal methods**
- Summary

Design Verification Methods

- **Simulation based methods**
  - Specify input test vector, output test vector pair
  - Run simulation and compare output against expected output
- **Formal Methods**
  - Check equivalence of design models or parts of models
  - Check specified properties on models
- **Semi-formal Methods**
  - Specify inputs and outputs as symbolic expressions
  - Check simulation output against expected expression

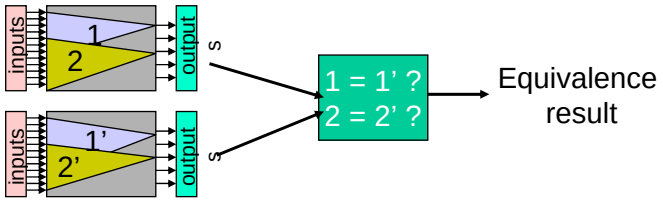
Simulation

- RTL model is imported directly into system simulation model
- Blocks may be required to interface the RTL model with the system simulation model
- Benefits - Only one testbench. Reduce number and size of files containing stimulus/expected results and number of testbenches
- Better testing is possible

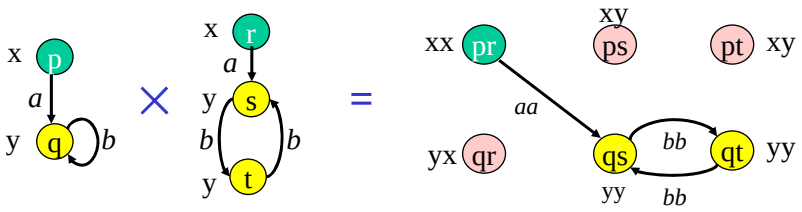


Equivalence Checking

- LEC uses boolean algebra to check for logic equivalence

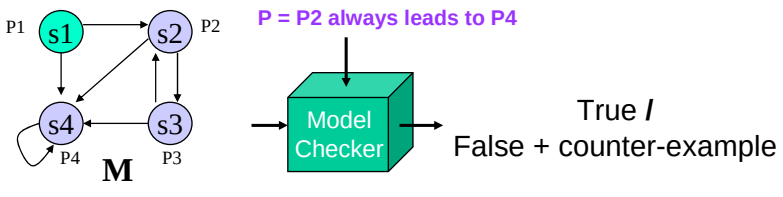


- SEC uses FSMs to check for sequential equivalence



Model Checking

- Model *M* satisfies property *P*? [Clarke, Emerson '81]
- Inputs
  - State transition system representation of *M*
  - Temporal property *P* as formula of state properties
- Output
  - True (property holds)
  - False + counter-example (property does not hold)



Agenda

- Design Convergence
- System Level Design
- Modeling
- Verification
- **Summary**

Desirable Design Methodology

- **Design should be based on the use of one or more formal models to describe the behavior of the system at a high level of abstraction**
  - **such behavior should be captured on an unbiased way, that is, before a decision on its decomposition into hardware and software components is taken**
- **The final implementation of the system should be generated as much as possible using automatic synthesis from this high level of abstraction**
  - **to ensure implementations that are "correct by construction"**
- **Validation (through simulation or verification) should be done as much as possible at the higher levels of abstraction**

Flow Summary