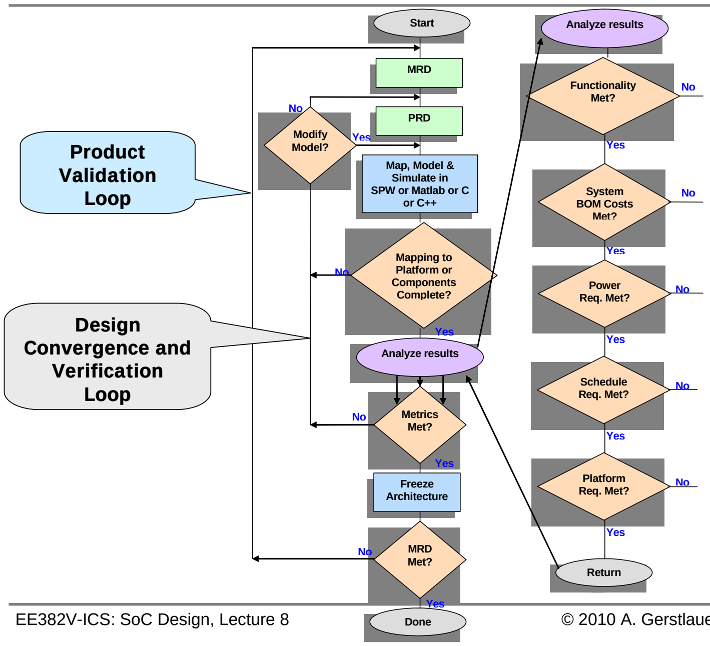
# EE382V-ICS: System-on-a-Chip (SoC) Design

## Lecture 8 - System Design Methodology

*with sources from:*
*Christian Haubelt, Univ. of Erlangen-Nuremberg*

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

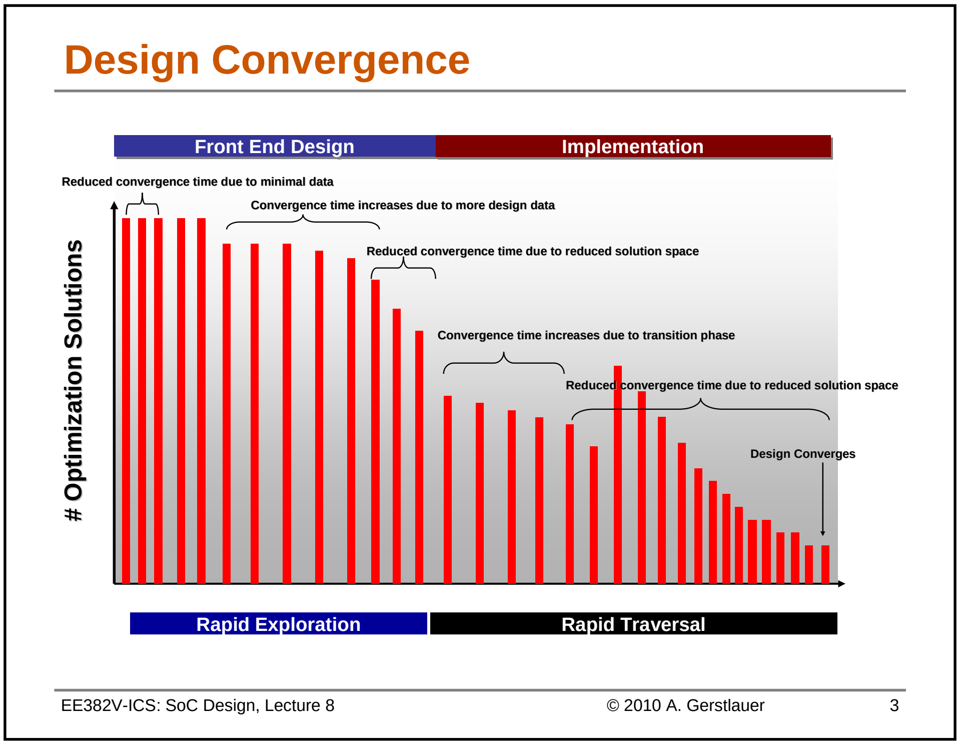
## SoC Design Flow



EE382V-ICS: SoC Design, Lecture 8

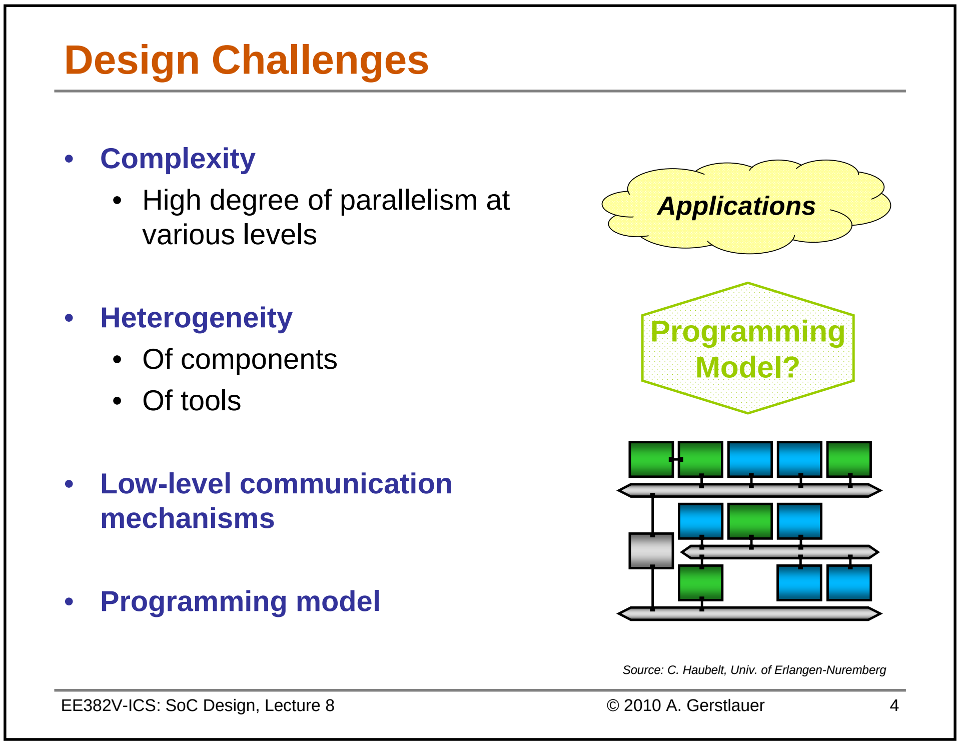
© 2010 A. Gerstlauer

2



## Design Convergence

EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 3

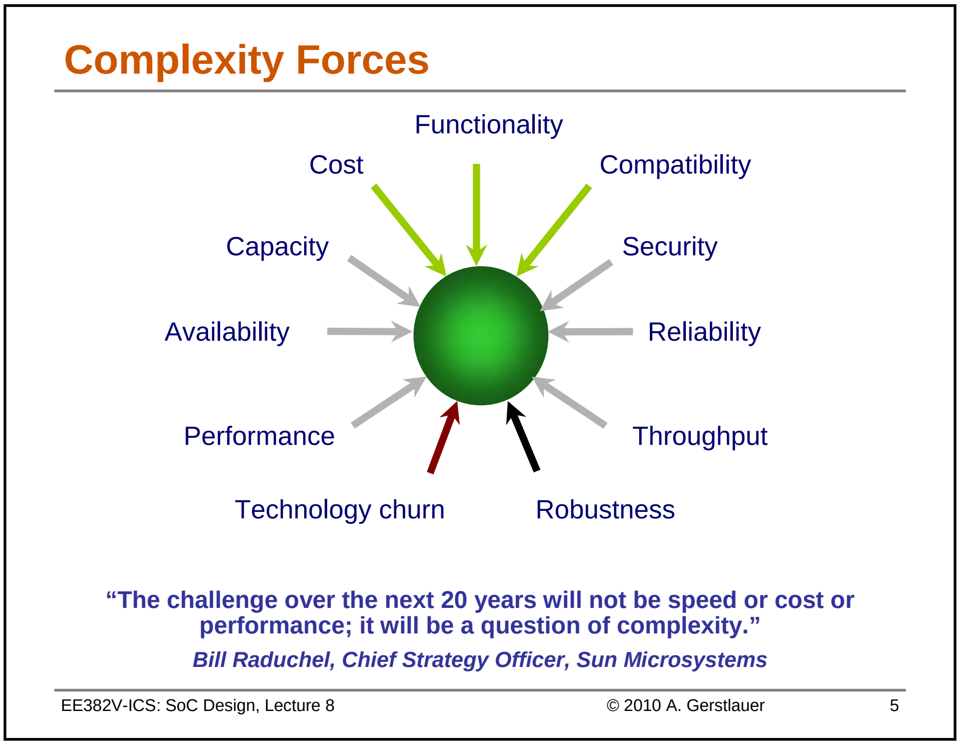


## Design Challenges

- **Complexity**
  - High degree of parallelism at various levels
- **Heterogeneity**
  - Of components
  - Of tools
- **Low-level communication mechanisms**
- **Programming model**

Source: C. Haubelt, Univ. of Erlangen-Nuremberg

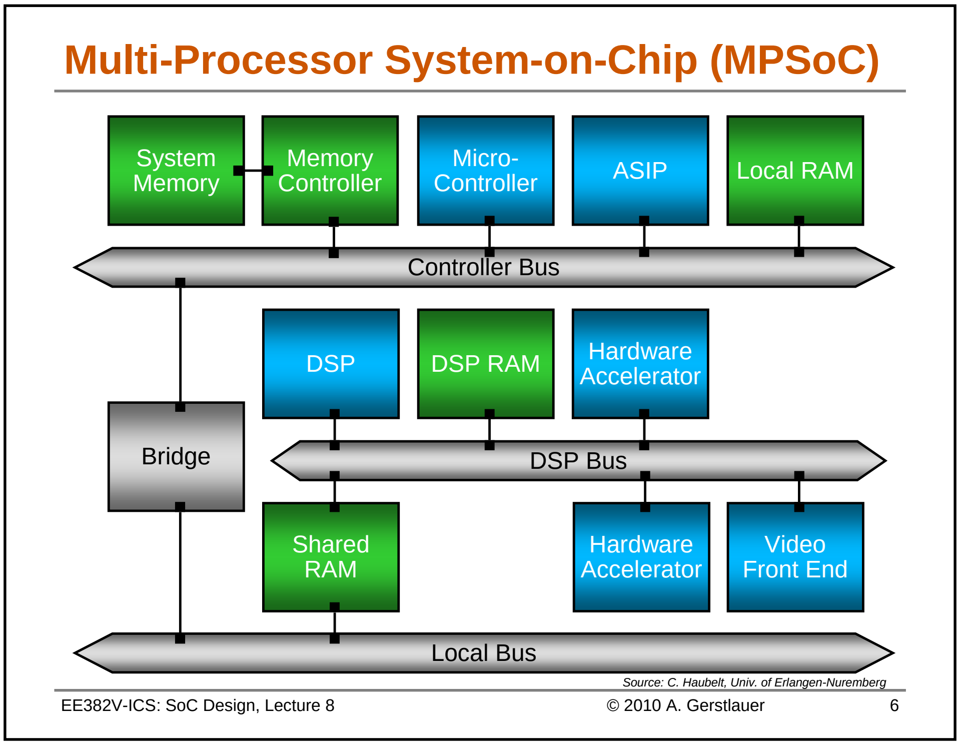
EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 4



## 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***



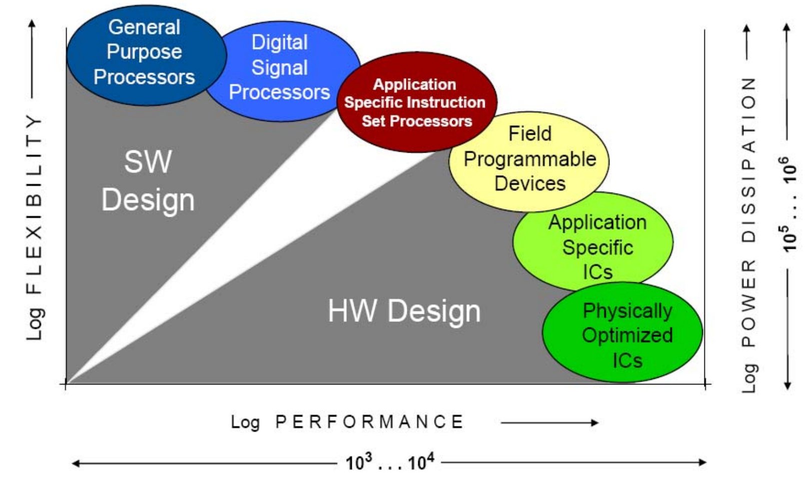
## Multi-Processor System-on-Chip (MPSoC)

*Source: C. Haubelt, Univ. of Erlangen-Nuremberg*

## MPSoC Terminology

- **Multi-processor**
  - Heterogeneous, asymmetric multi-processing (AMP)
  - Distributed memory and operating system
- **Multi-core**
  - Homogeneous, symmetric multi-processing (SMP)
  - Shared memory and operating system
  - ➢ Multi-core processors in a multi-processor system
- **Many-core**
  - > 10 cores per processor…

## Processor Implementation Options



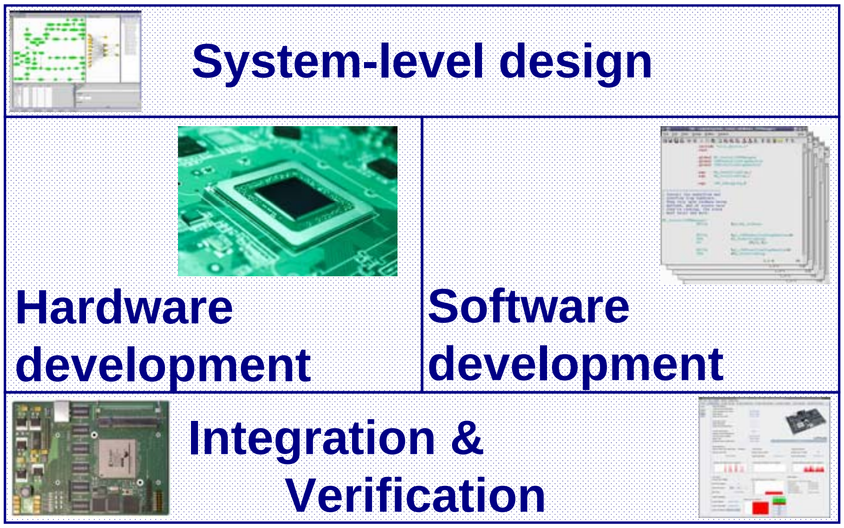
*Source: T. Noll, RWTH Aachen, via R. Leupers, "From ASIP to MPSoC", Computer Engineering Colloquium, TU Delft, 2006*

## Lecture 8: Outline

✓ Introduction

- System design methodology
  - Electronic system-level design (ESL/SLD)
- ESL design
  - Modeling
  - Synthesis
  - Verification
- ESL landscape
- Summary and conclusions

## System Design



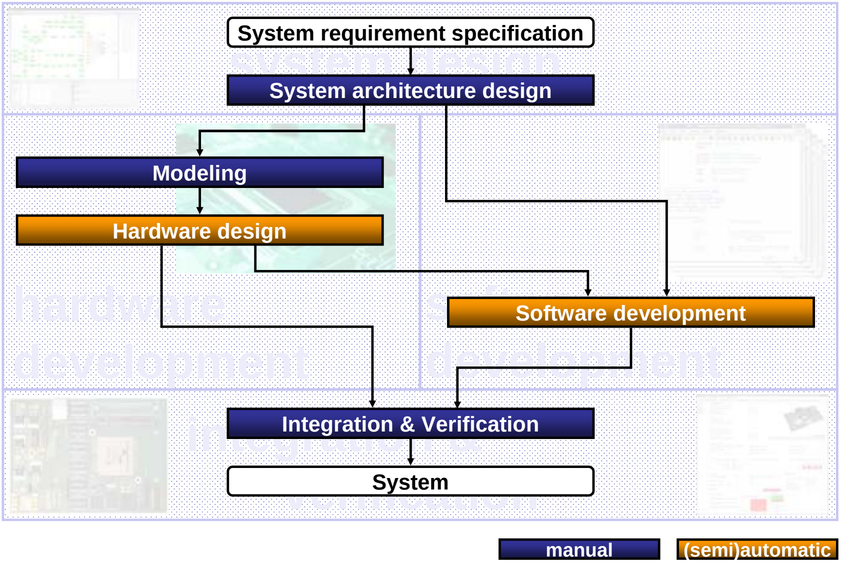
System-level design

Hardware development

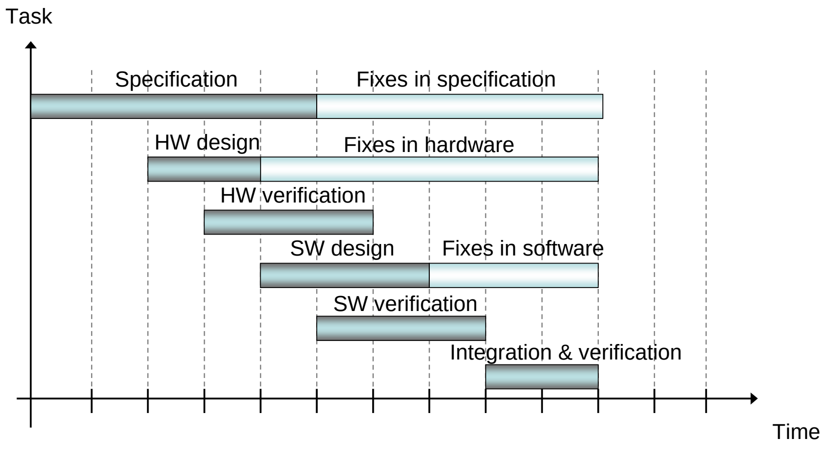
Software development

Integration & Verification

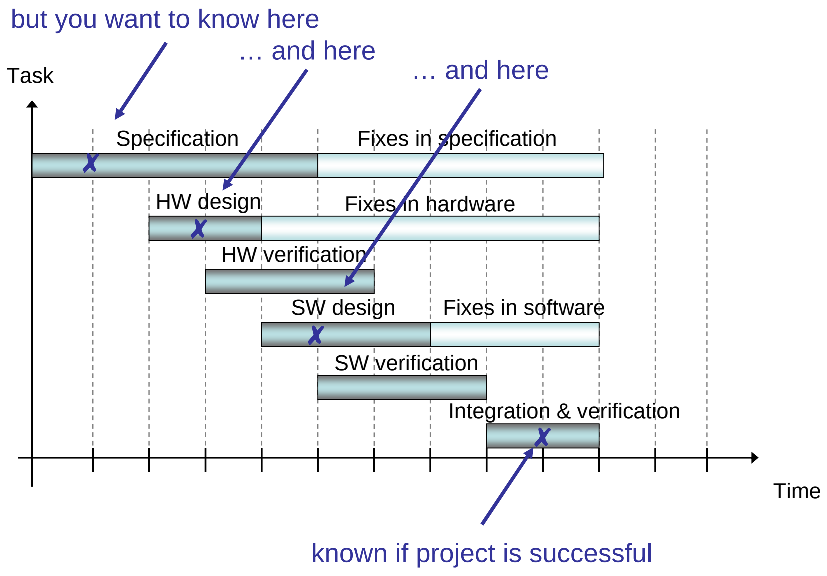
## Classical System Design Flow



## Hardware-Centric Design Cycle



## Hardware-Centric Design Cycle

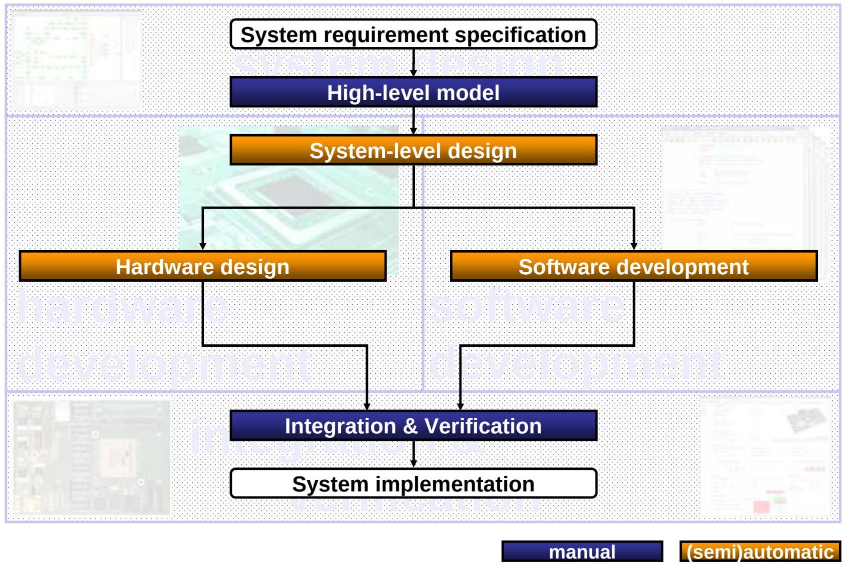

but you want to know here

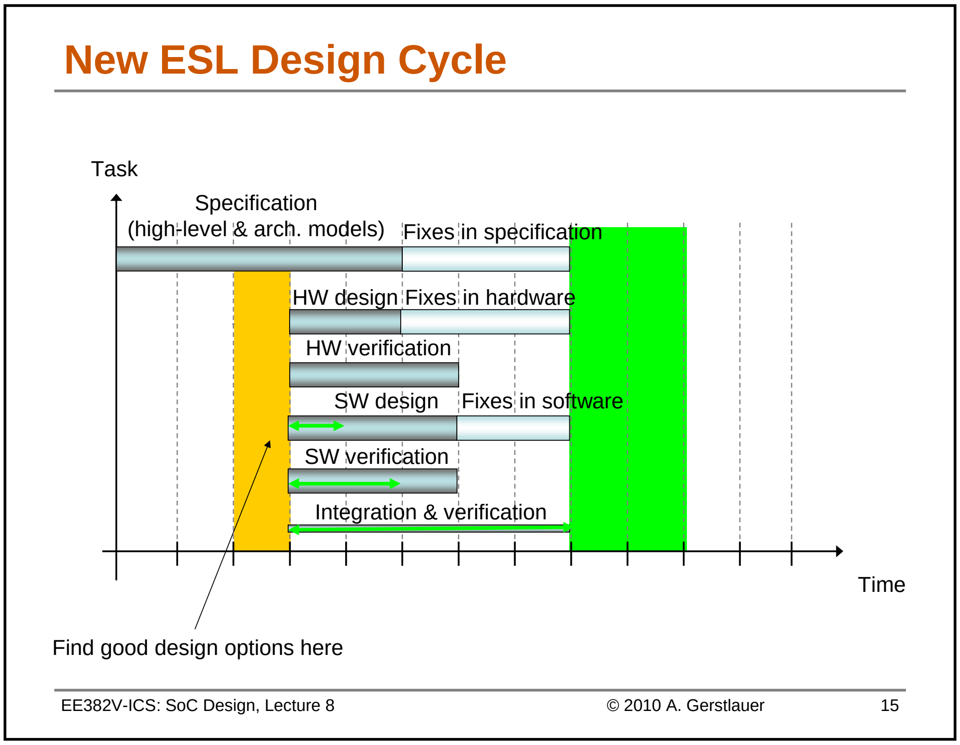
… and here

… and here

known if project is successful

## Electronic System-Level (ESL) Design Flow

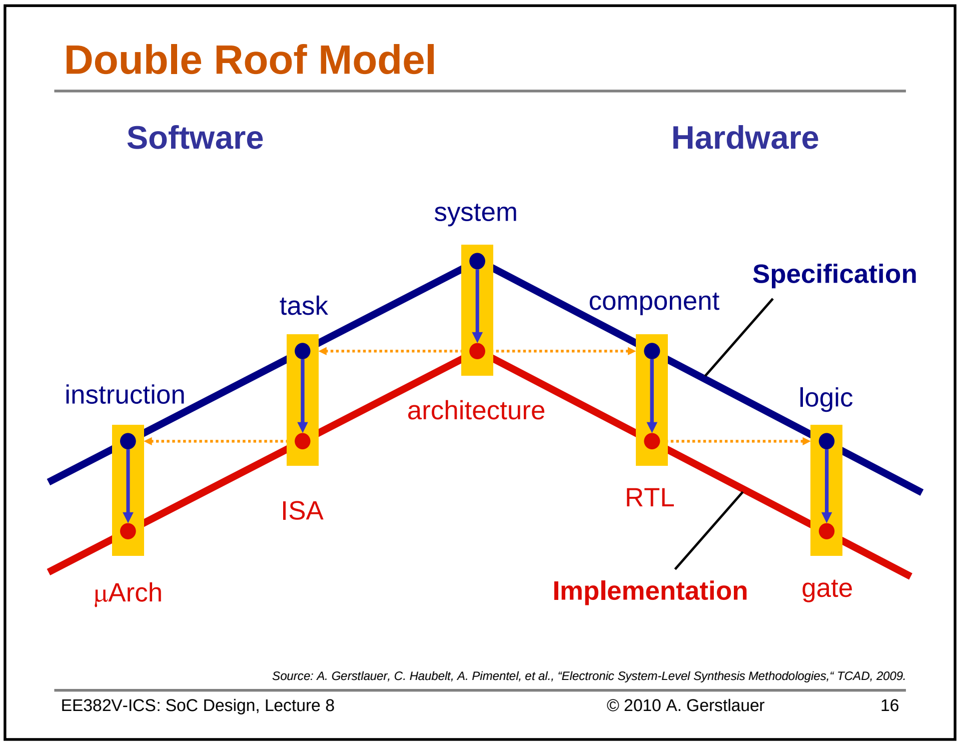




## New ESL Design Cycle

Find good design options here

EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 15



## Double Roof Model

*Source: A. Gerstlauer, C. Haubelt, A. Pimentel, et al., "Electronic System-Level Synthesis Methodologies," TCAD, 2009.*

EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 16

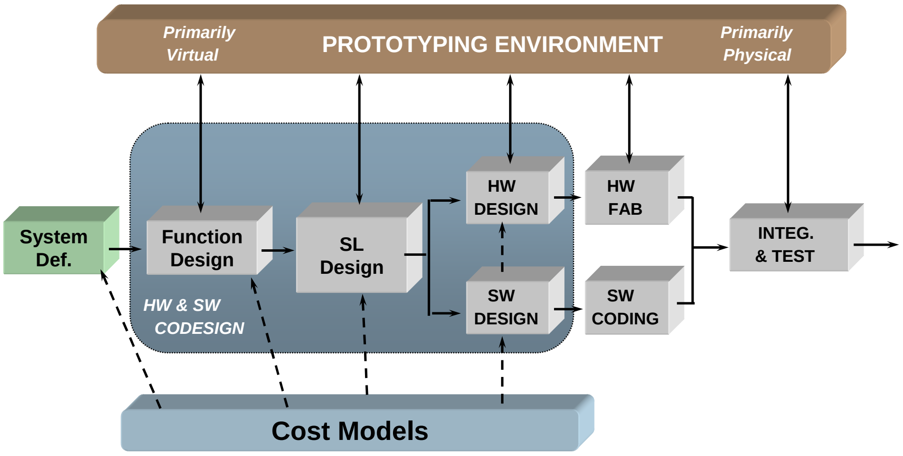
# Design Methodologies

➢ Set of models and design steps (transformations)

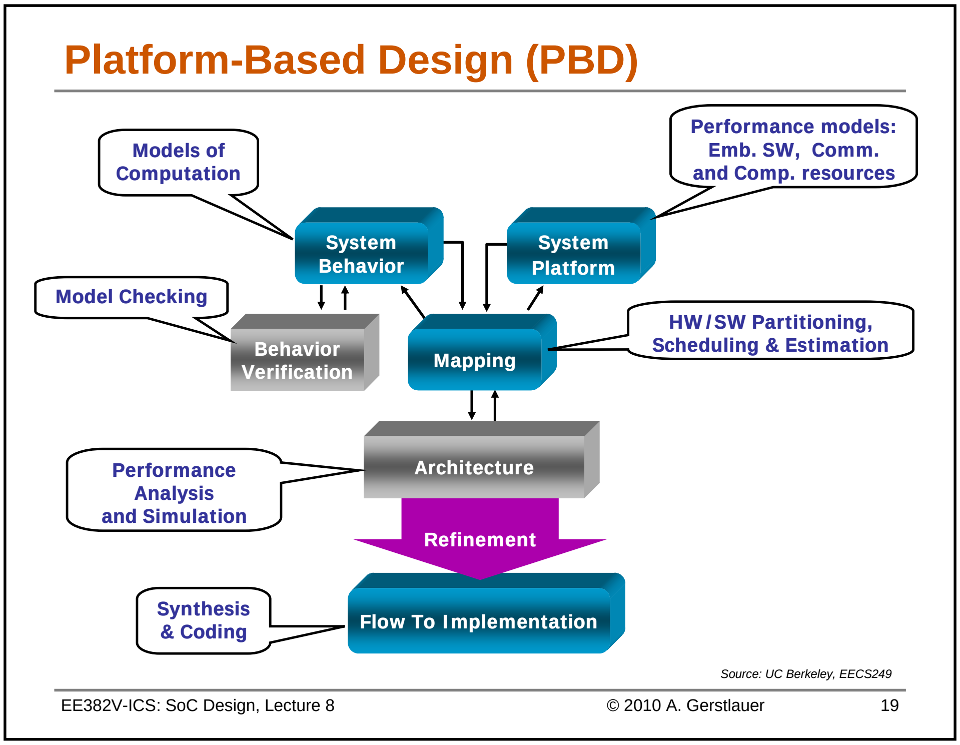
- **Top down design**
  - Starts with functional system specification
    - Application behavior
    - Models of Computation (MoC)
  - Successive refinement
  - Connect the hardware and software design teams earlier in the design cycle.
  - Allows hardware and software to be developed concurrently
  - 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 design**
  - Starts with architecting a processing platform for a given vertical application space
    - Semiconductor, ASSP vendors
  - Enables rapid creation and verification of sophisticated SoC designs variants
  - 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

*Source: Coware, Inc., 2005*

# Top-Down ESL Design Environment

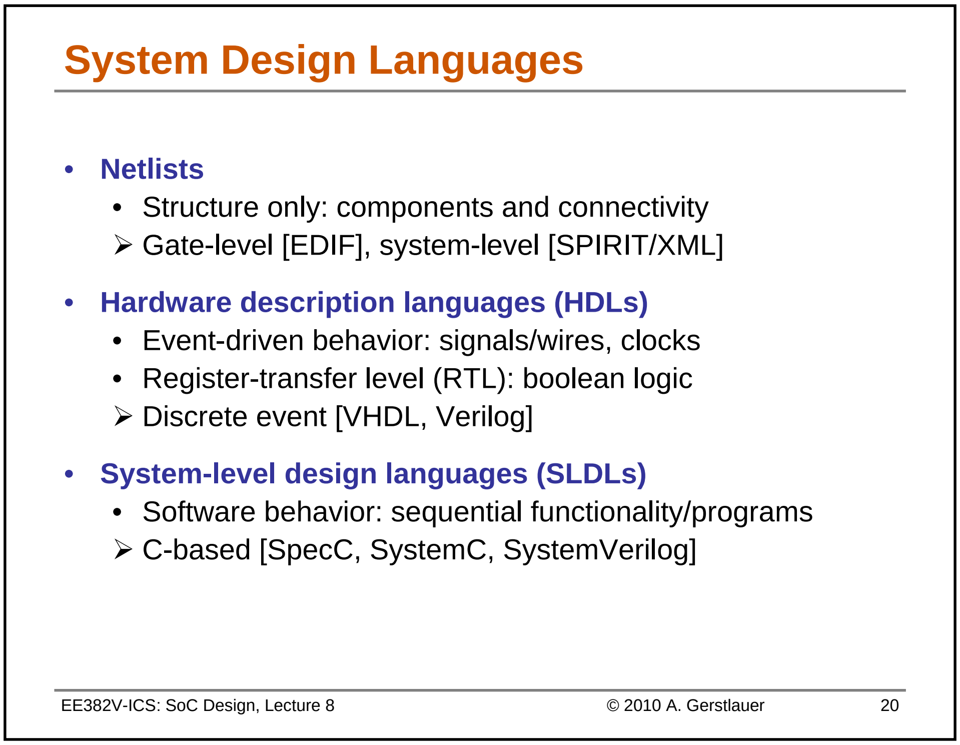


*Copyright © 1995-1999 SCRA Used with Permission*



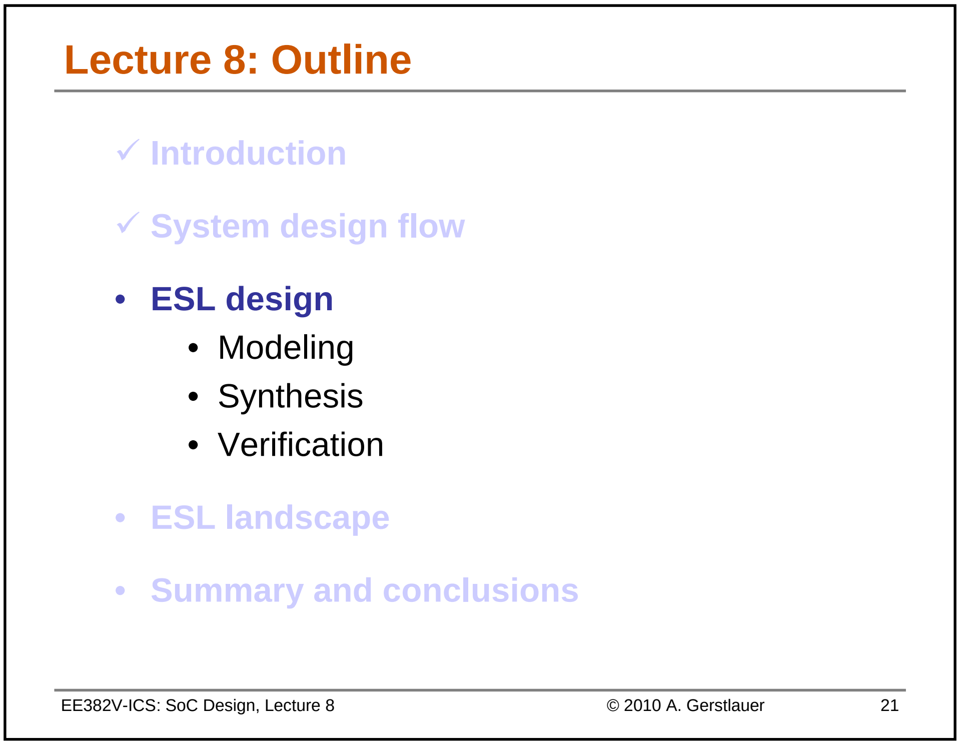
## Platform-Based Design (PBD)

*Source: UC Berkeley, EECS249*



## System Design Languages

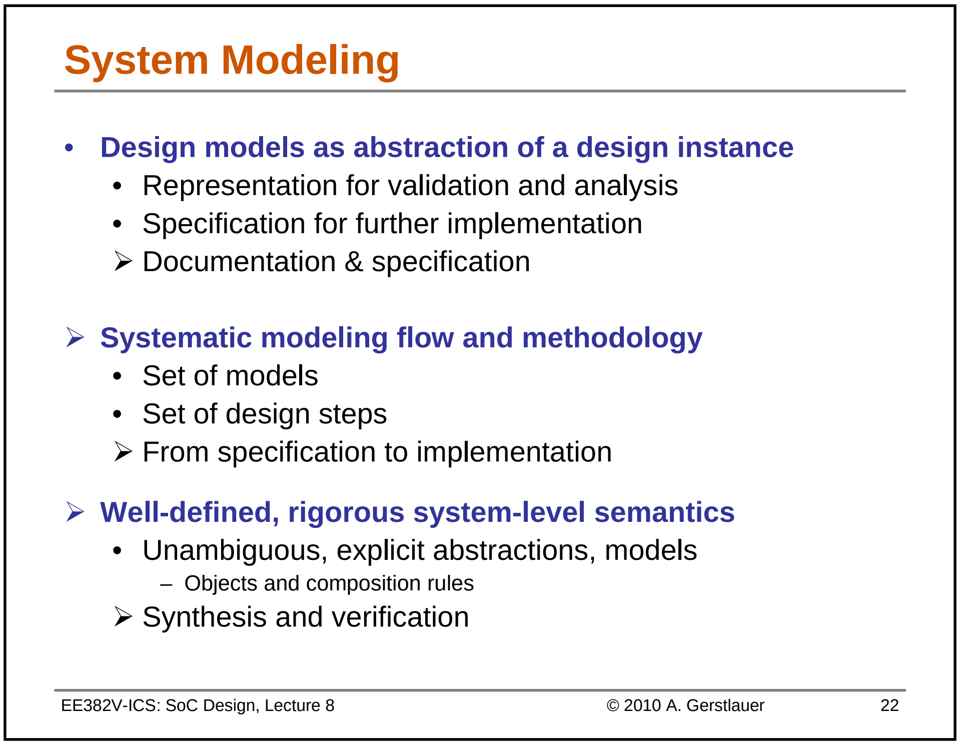
- **Netlists**
  - Structure only: components and connectivity
  - ➢ Gate-level [EDIF], system-level [SPIRIT/XML]
- **Hardware description languages (HDLs)**
  - Event-driven behavior: signals/wires, clocks
  - Register-transfer level (RTL): boolean logic
  - ➢ Discrete event [VHDL, Verilog]
- **System-level design languages (SLDLs)**
  - Software behavior: sequential functionality/programs
  - ➢ C-based [SpecC, SystemC, SystemVerilog]



## Lecture 8: Outline

- ✓ Introduction
- ✓ System design flow
- **ESL design**
  - Modeling
  - Synthesis
  - Verification
- ESL landscape
- Summary and conclusions

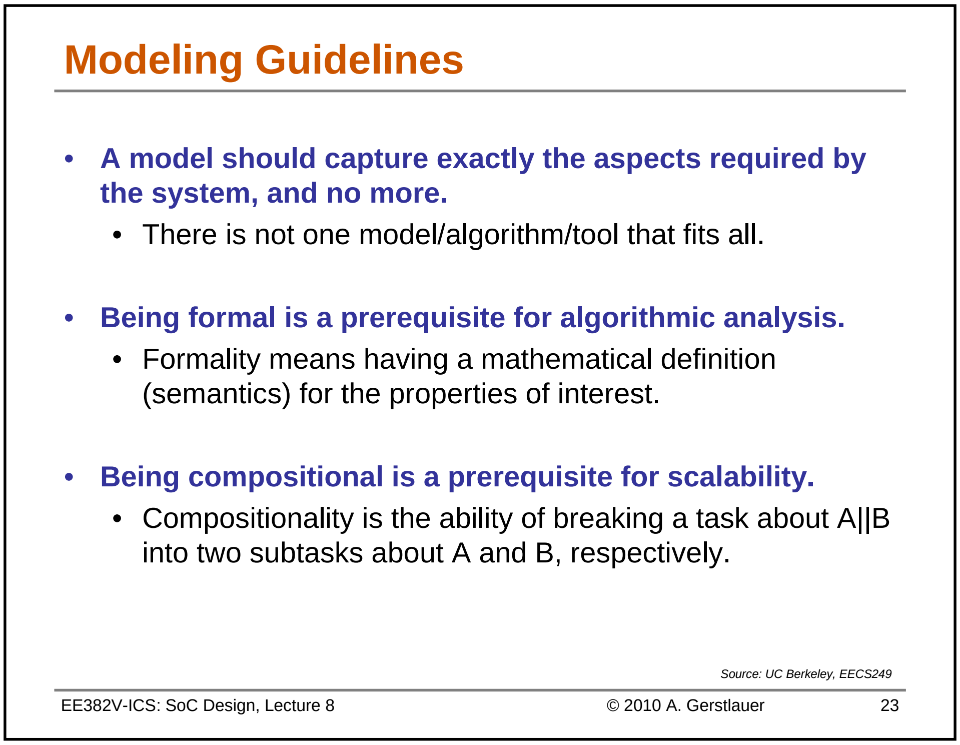
EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 21



## System Modeling

- **Design models as abstraction of a design instance**
  - Representation for validation and analysis
  - Specification for further implementation
  - ➢ Documentation & specification
- ➢ **Systematic modeling flow and methodology**
  - Set of models
  - Set of design steps
  - ➢ From specification to implementation
- ➢ **Well-defined, rigorous system-level semantics**
  - Unambiguous, explicit abstractions, models
    - – Objects and composition rules
  - ➢ Synthesis and verification

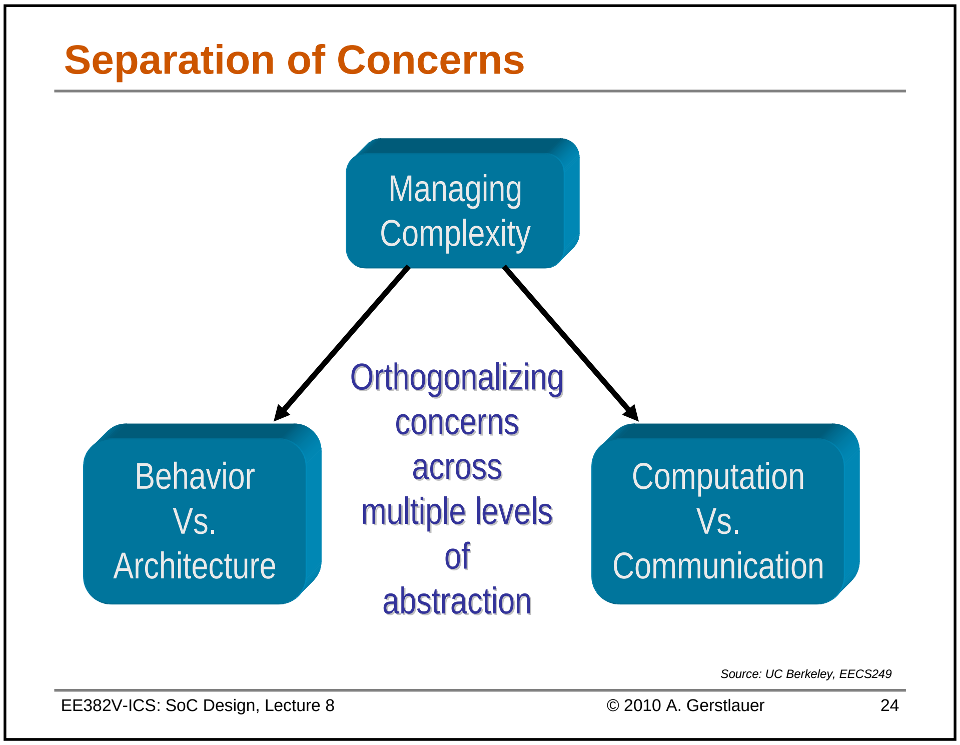
EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 22



## 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 (semantics) 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.

*Source: UC Berkeley, EECS249*



## Separation of Concerns

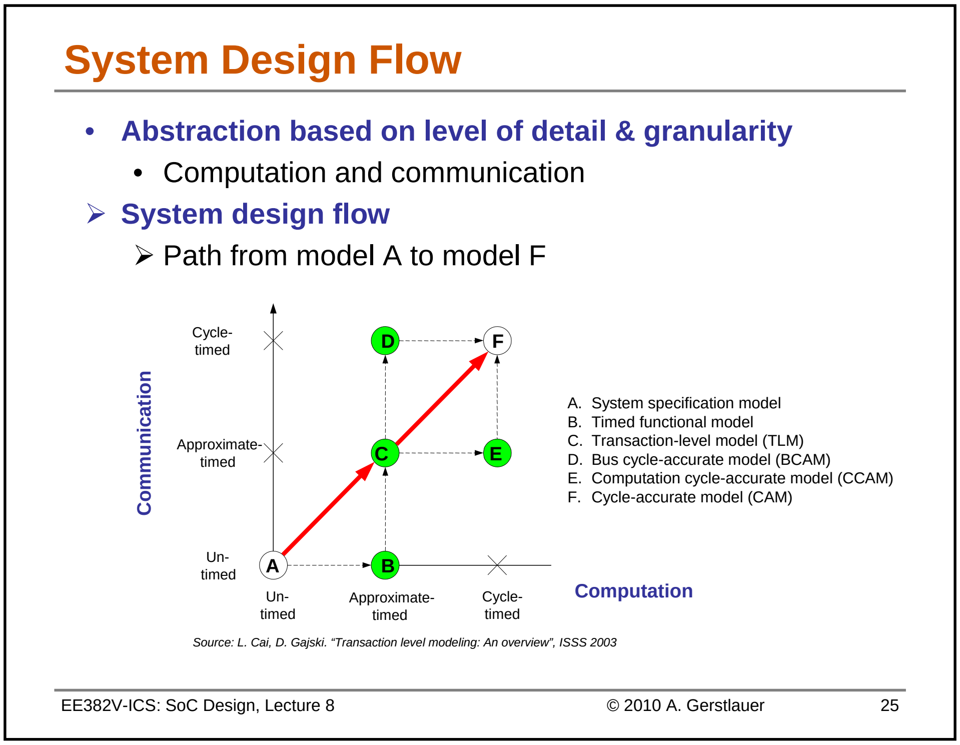
Managing Complexity

Behavior Vs. Architecture

Orthogonalizing concerns across multiple levels of abstraction

Computation Vs. Communication

*Source: UC Berkeley, EECS249*

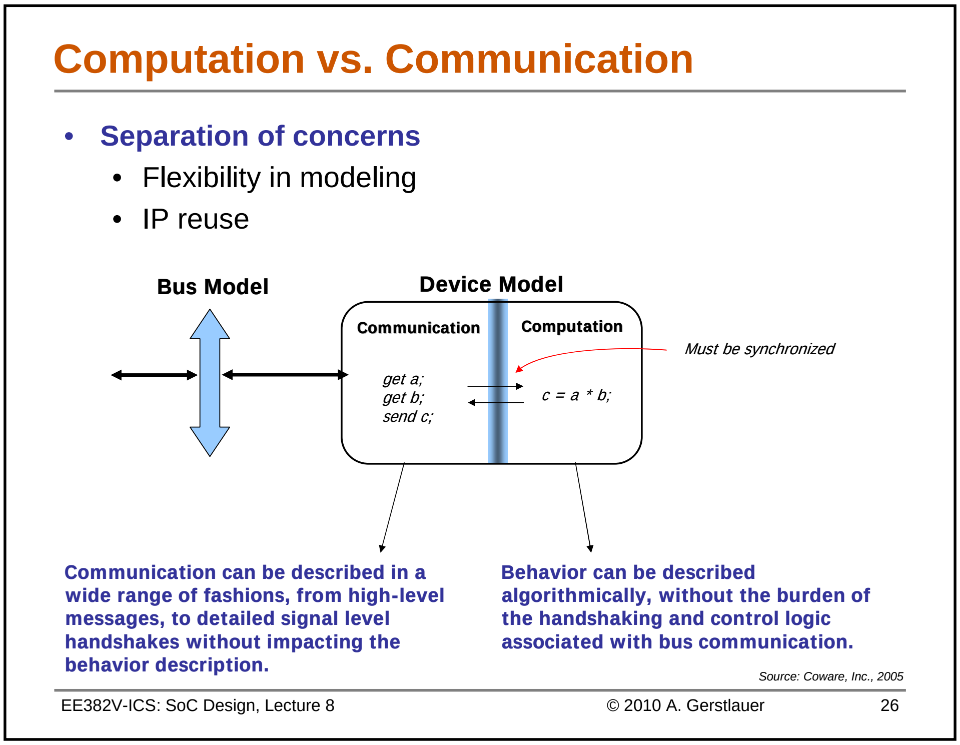


## System Design Flow

- **Abstraction based on level of detail & granularity**
  - Computation and communication
- ➢ **System design flow**
  - ➢ Path from model A to model F

A. System specification model
B. Timed functional model
C. Transaction-level model (TLM)
D. Bus cycle-accurate model (BCAM)
E. Computation cycle-accurate model (CCAM)
F. Cycle-accurate model (CAM)

*Source: L. Cai, D. Gajski. "Transaction level modeling: An overview", ISSS 2003*



## Computation vs. Communication

- **Separation of concerns**
  - Flexibility in modeling
  - IP reuse

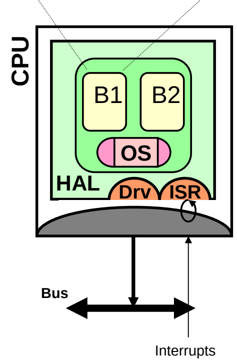
**Communication can be described in a wide range of fashions, from high-level messages, to detailed signal level handshakes without impacting the behavior description.**

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

*Source: Coware, Inc., 2005*

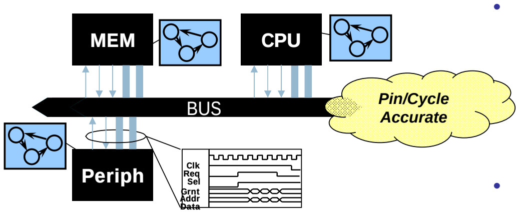
## Computation Models

```
Process B1()
{
   …
   waitfor(15000);
   …
   waitfor(25000);
   …
};
```

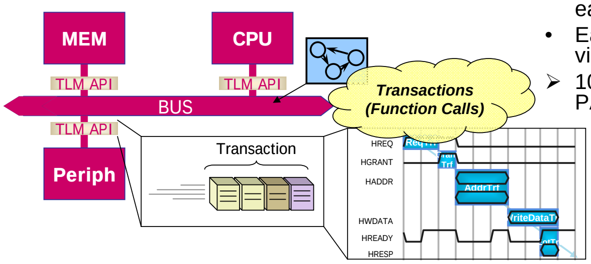


- **Application model**
  - Model of Computation (MoC)
    - – Process-/state-based [KPN, SDF, FSM, …]
  - Back-annotated execution timing
    - – Timing granularity (basic block level)
- **Processor model**
  - Operating system
    - – Real-time multi-tasking (RTOS), drivers
  - Hardware abstraction layer (HAL)
    - – Media accesses
  - Processor hardware
    - – Bus I/O & interrupts
- **Instruction-set model**
  - Instruction-set or micro-architecture
    - – Down to cycle-accurate behavior

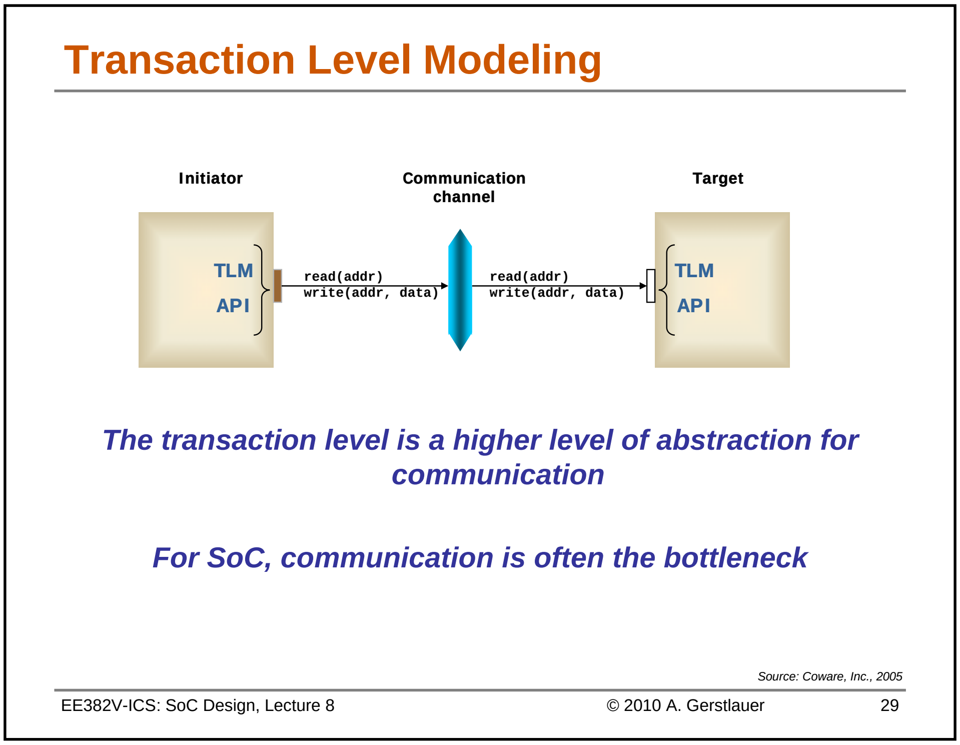
## Communication Models



- **Pin-Accurate Model (PAM)**
  - Redundant RTL complexity results in slow simulation
  - Each device interface must implement the bus protocol
  - Each device on the bus has a pin-accurate interface
- **Transaction-Level Model (TLM)**
  - Less code, no wires, fewer events yield faster simulation
  - Protocol is modeled as a single bus model instead of in each device
  - Each device communicates via transaction-level API
  - ➢ 100x-10,000x faster than PAM



Source: Coware, Inc., 2005



## Transaction Level Modeling

Initiator

Communication channel

Target

TLM API

read(addr)
write(addr, data)

read(addr)
write(addr, data)

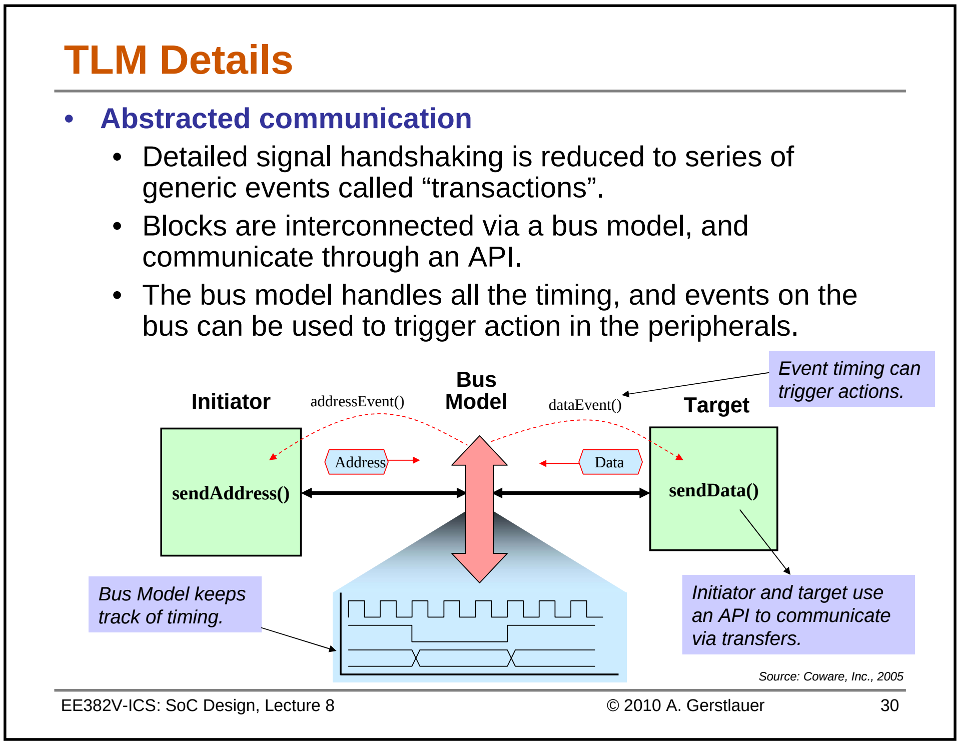
TLM API

*The transaction level is a higher level of abstraction for communication*

*For SoC, communication is often the bottleneck*

*Source: Coware, Inc., 2005*

EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 29



## TLM Details

- **Abstracted communication**
  - Detailed signal handshaking is reduced to series of generic events called “transactions”.
  - 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.

Initiator

addressEvent()

Bus Model

dataEvent()

Target

sendAddress()

Address

Data

sendData()

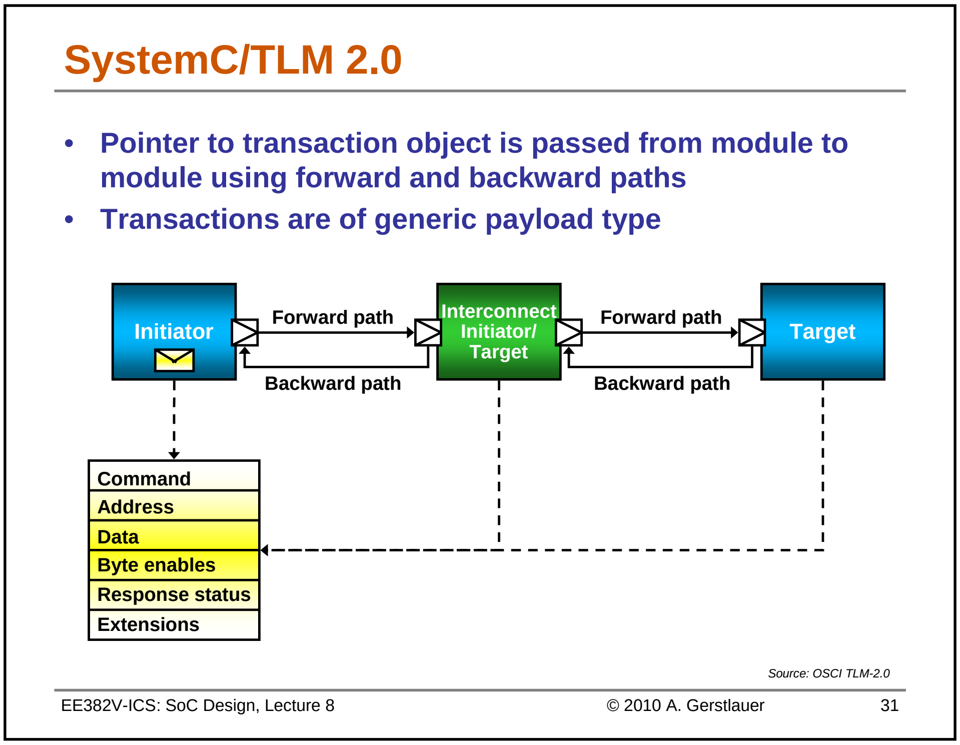
*Event timing can trigger actions.*

*Bus Model keeps track of timing.*

*Initiator and target use an API to communicate via transfers.*

*Source: Coware, Inc., 2005*

EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 30

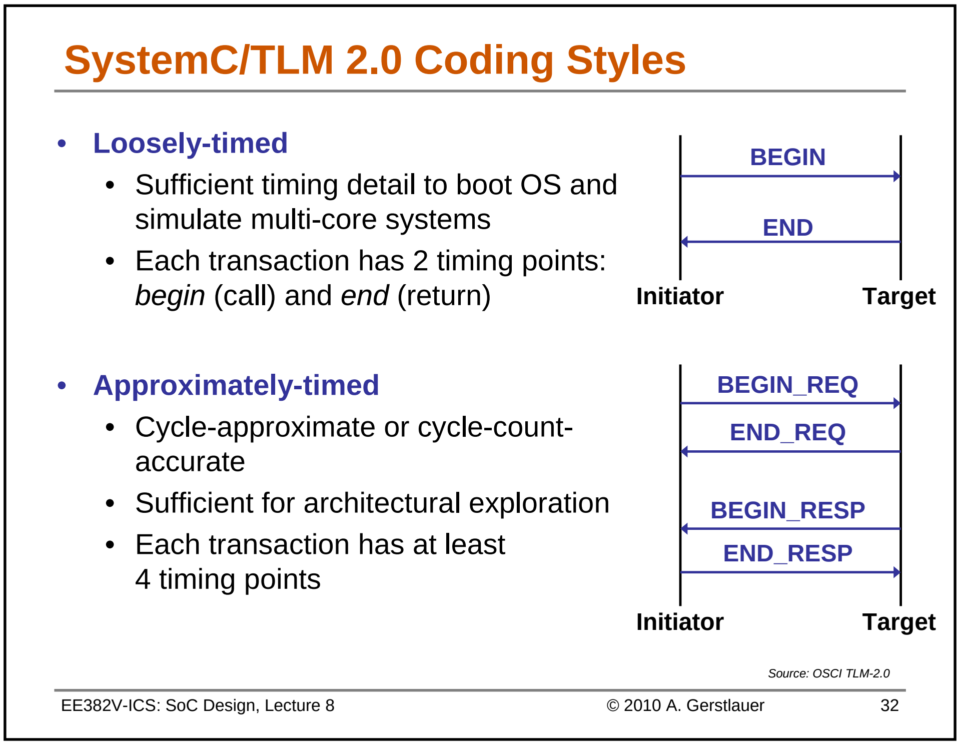


## SystemC/TLM 2.0

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

Initiator
Forward path
Backward path
Interconnect Initiator/ Target
Forward path
Backward path
Target
Command
Address
Data
Byte enables
Response status
Extensions

Source: OSCI TLM-2.0



## SystemC/TLM 2.0 Coding Styles

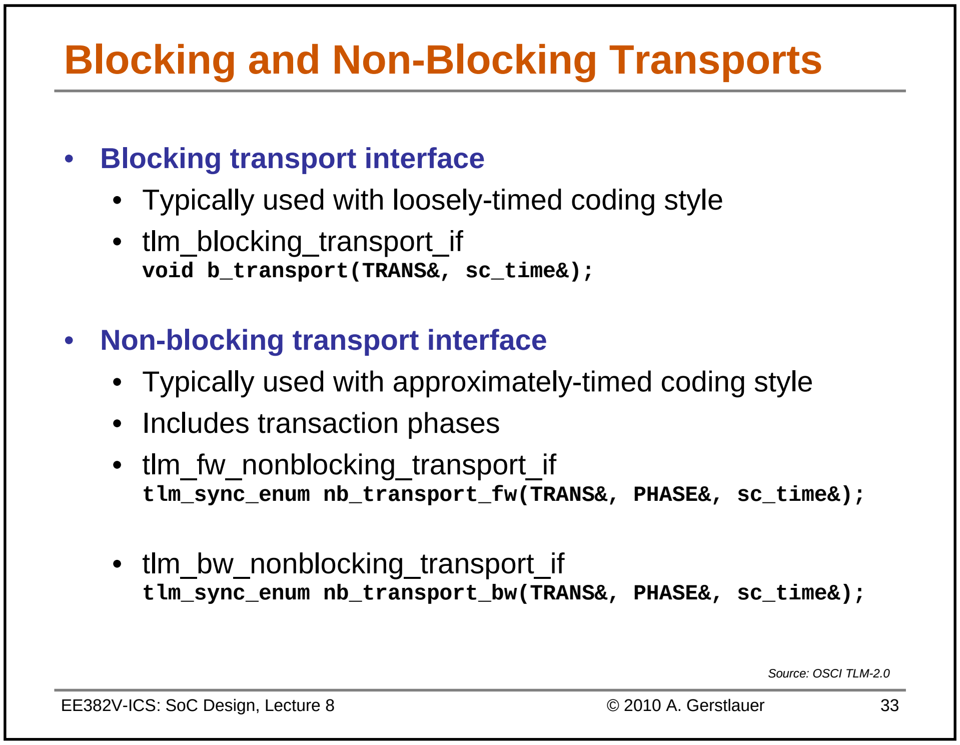
- Loosely-timed
  - Sufficient timing detail to boot OS and simulate multi-core systems
  - Each transaction has 2 timing points: *begin* (call) and *end* (return)

BEGIN
END
Initiator
Target

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

BEGIN_REQ
END_REQ
BEGIN_RESP
END_RESP
Initiator
Target

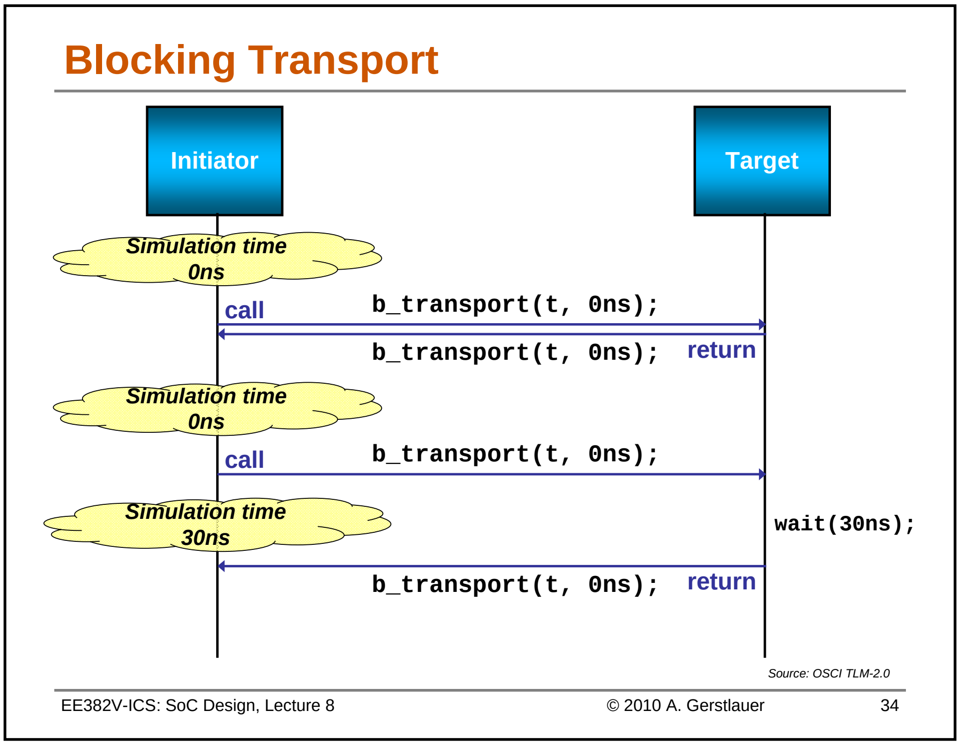
Source: OSCI TLM-2.0



## Blocking and Non-Blocking Transports

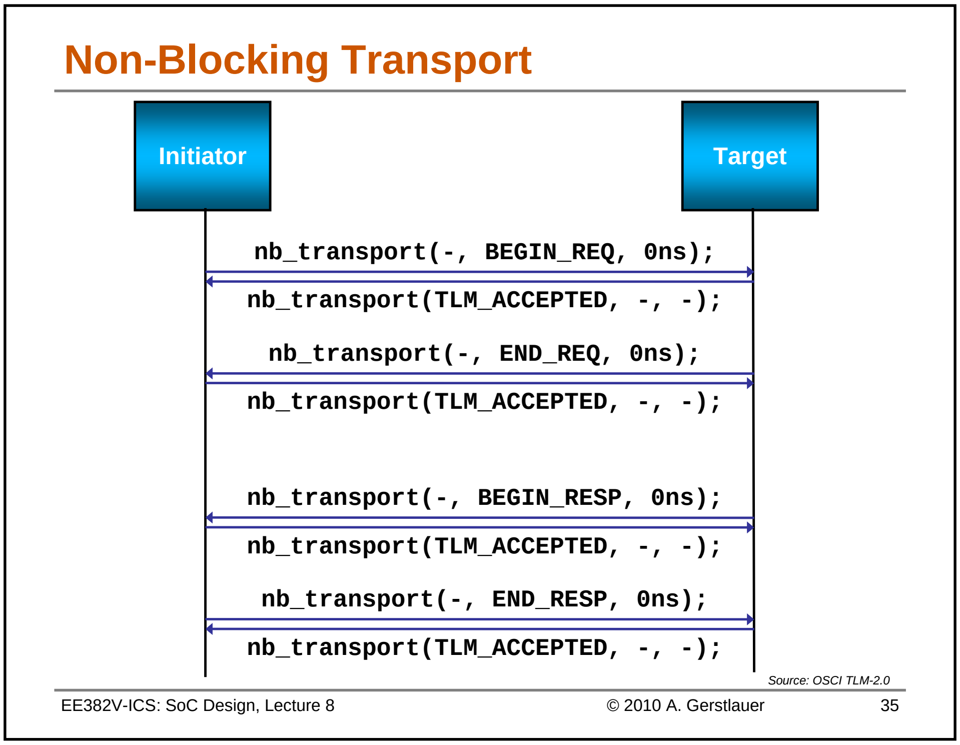
- **Blocking transport interface**
  - Typically used with loosely-timed coding style
  - tlm_blocking_transport_if
    `void b_transport(TRANS&, sc_time&);`
- **Non-blocking transport interface**
  - Typically used with approximately-timed coding style
  - Includes transaction phases
  - tlm_fw_nonblocking_transport_if
    `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: OSCI TLM-2.0*



## Blocking Transport

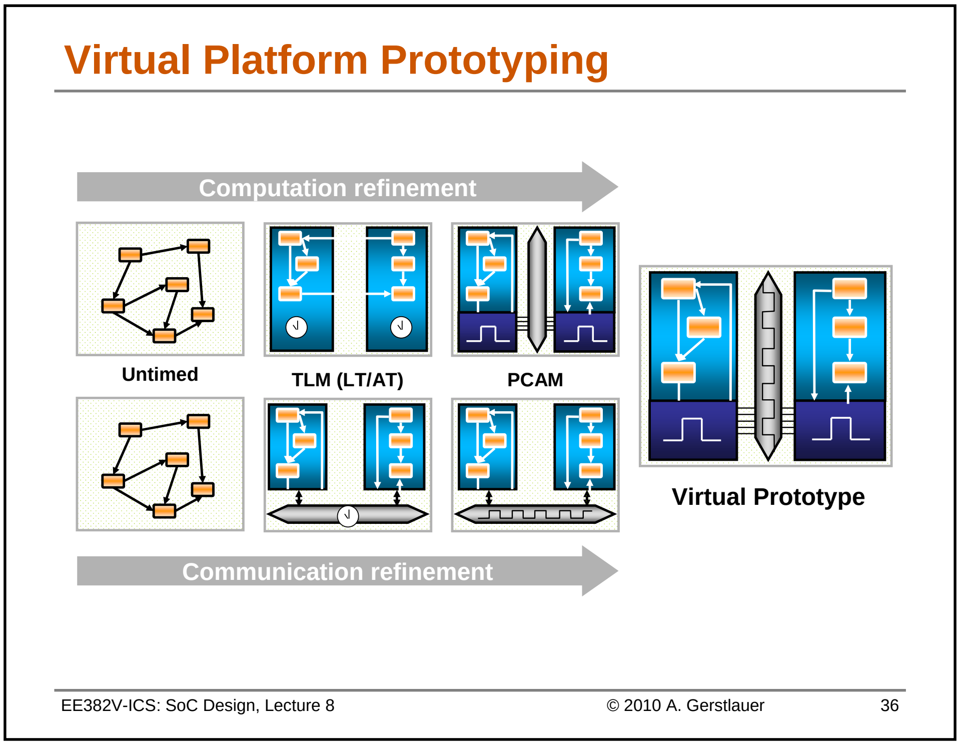
*Source: OSCI TLM-2.0*



## Non-Blocking Transport

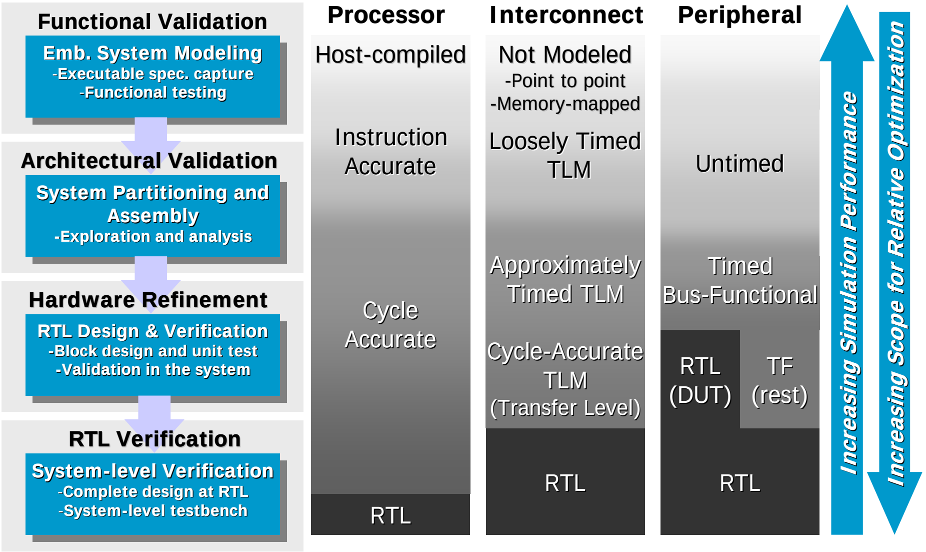
```
nb_transport(-, BEGIN_REQ, 0ns);
nb_transport(TLM_ACCEPTED, -, -);
nb_transport(-, END_REQ, 0ns);
nb_transport(TLM_ACCEPTED, -, -);

nb_transport(-, BEGIN_RESP, 0ns);
nb_transport(TLM_ACCEPTED, -, -);
nb_transport(-, END_RESP, 0ns);
nb_transport(TLM_ACCEPTED, -, -);
```

*Source: OSCI TLM-2.0*



## Virtual Platform Prototyping

## Abstraction Levels



Functional Validation

Emb. System Modeling
-Executable spec. capture
-Functional testing

Architectural Validation

System Partitioning and Assembly
-Exploration and analysis

Hardware Refinement

RTL Design & Verification
-Block design and unit test
-Validation in the system

RTL Verification

System-level Verification
-Complete design at RTL
-System-level testbench

| Processor | Interconnect | Peripheral |
|---|---|---|
| Host-compiled | Not Modeled<br>-Point to point<br>-Memory-mapped | |
| Instruction Accurate | Loosely Timed TLM | Untimed |
| | Approximately Timed TLM | Timed Bus-Functional |
| Cycle Accurate | Cycle-Accurate TLM (Transfer Level) | RTL (DUT) / TF (rest) |
| RTL | RTL | RTL |

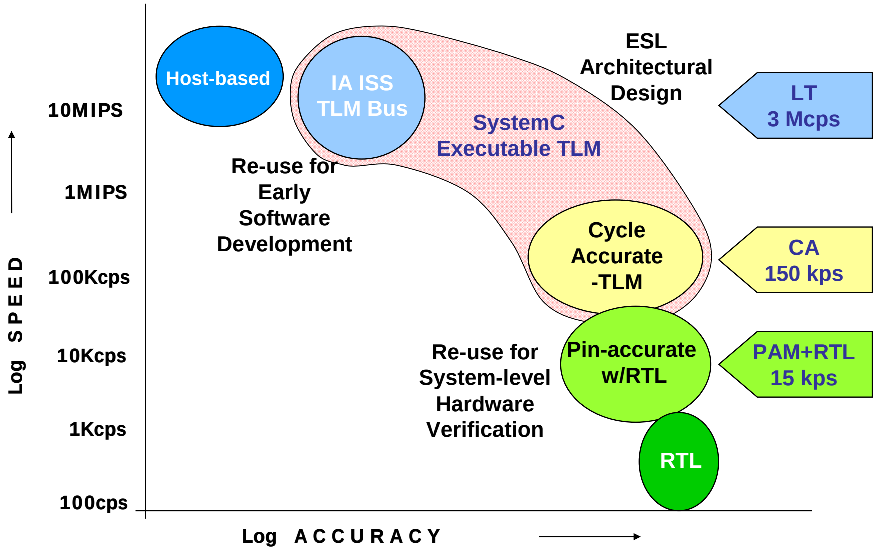
*Increasing Simulation Performance*

*Increasing Scope for Relative Optimization*

*Source: Coware, Inc., 2005*

EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 37

## Speed vs. Accuracy



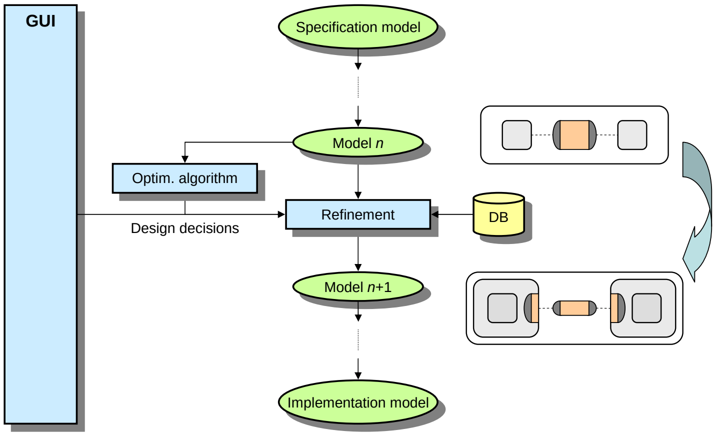
EE382V-ICS: SoC Design, Lecture 8 © 2010 A. Gerstlauer 38

## Lecture 8: Outline

- ✓ Introduction
- ✓ System design methodology
- ESL design
  - ✓ Modeling
  - Synthesis
  - Verification
- ESL landscape
- Summary and conclusions

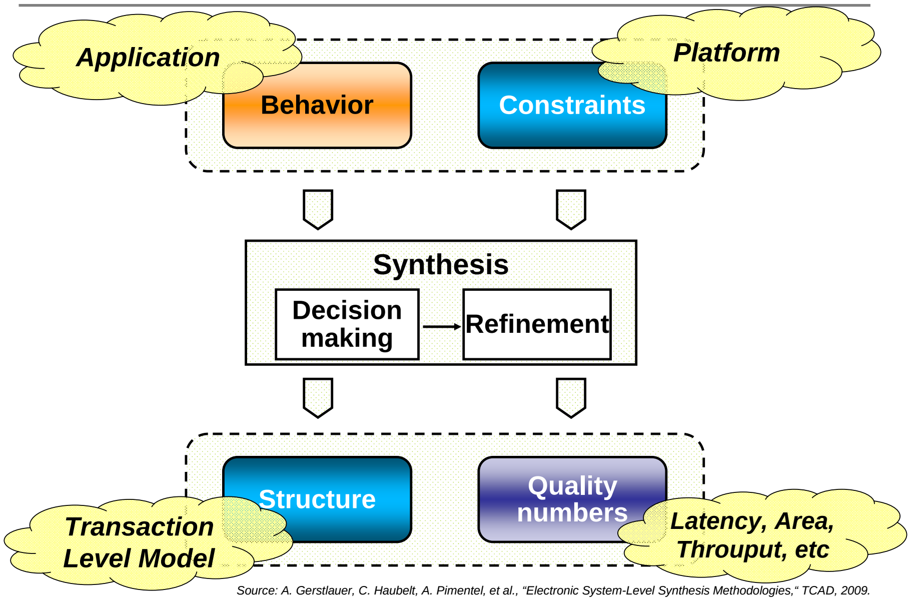
## Design Automation

- Synthesis = Decision making + model refinement



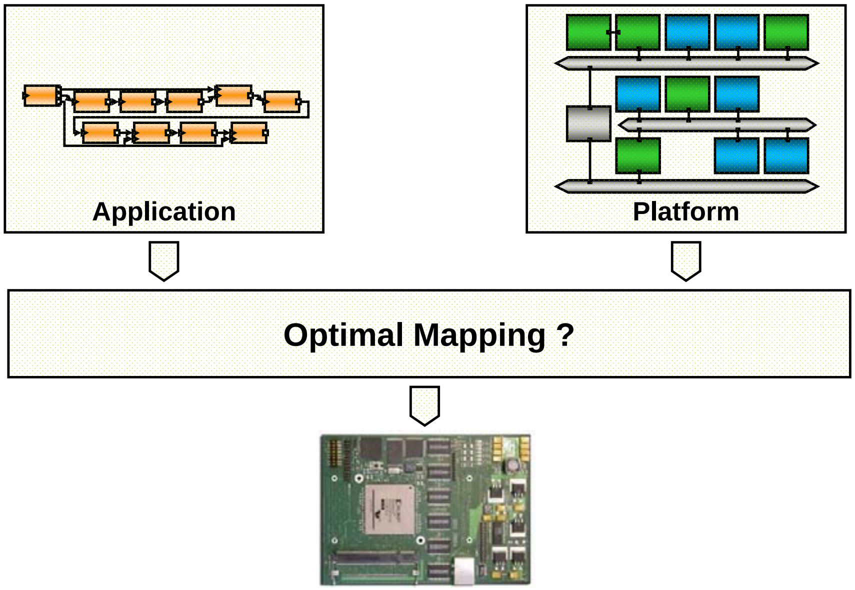
- ➢ Successive, stepwise model refinement
- ➢ Layers of implementation detail

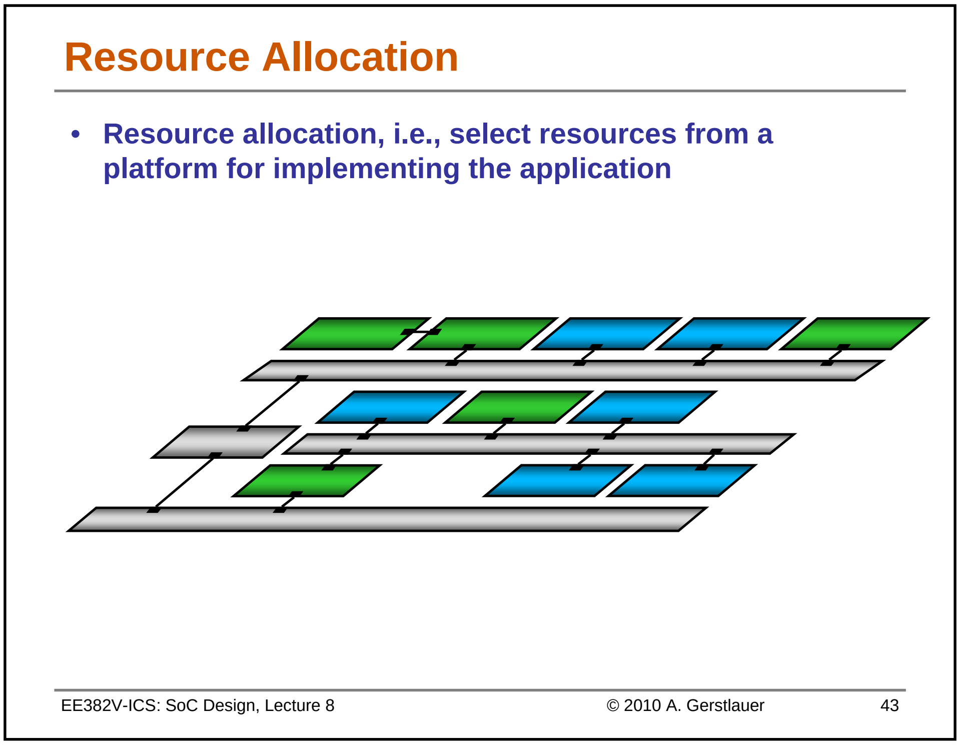
## X-Chart



*Source: A. Gerstlauer, C. Haubelt, A. Pimentel, et al., "Electronic System-Level Synthesis Methodologies," TCAD, 2009.*

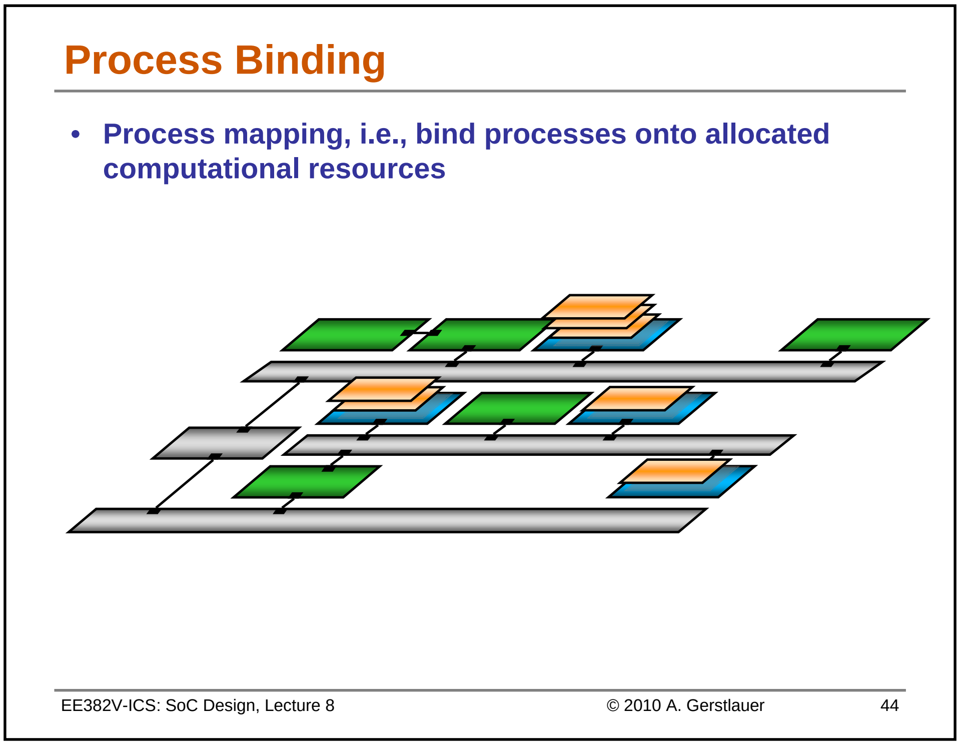
## Platform-Based System Synthesis





## Resource Allocation

- Resource allocation, i.e., select resources from a platform for implementing the application

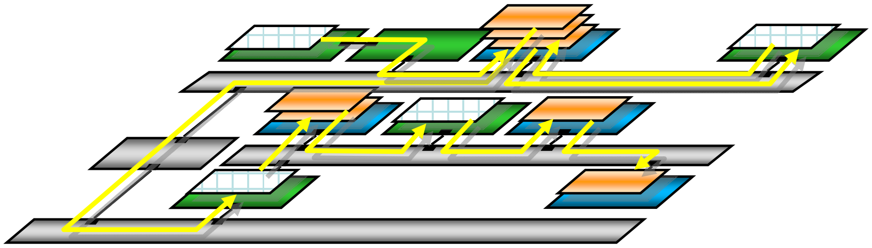


## Process Binding

- Process mapping, i.e., bind processes onto allocated computational resources

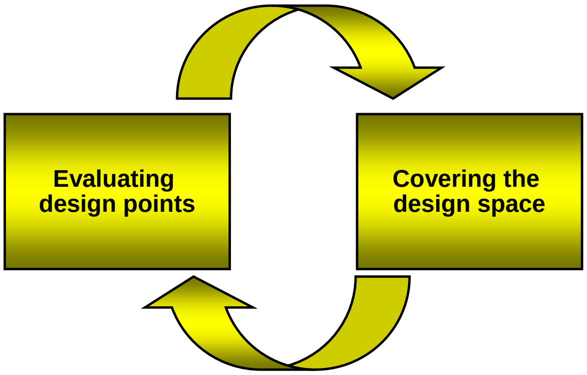
## Channel Routing

- Channel mapping, i.e., assign channels to paths over busses and address spaces



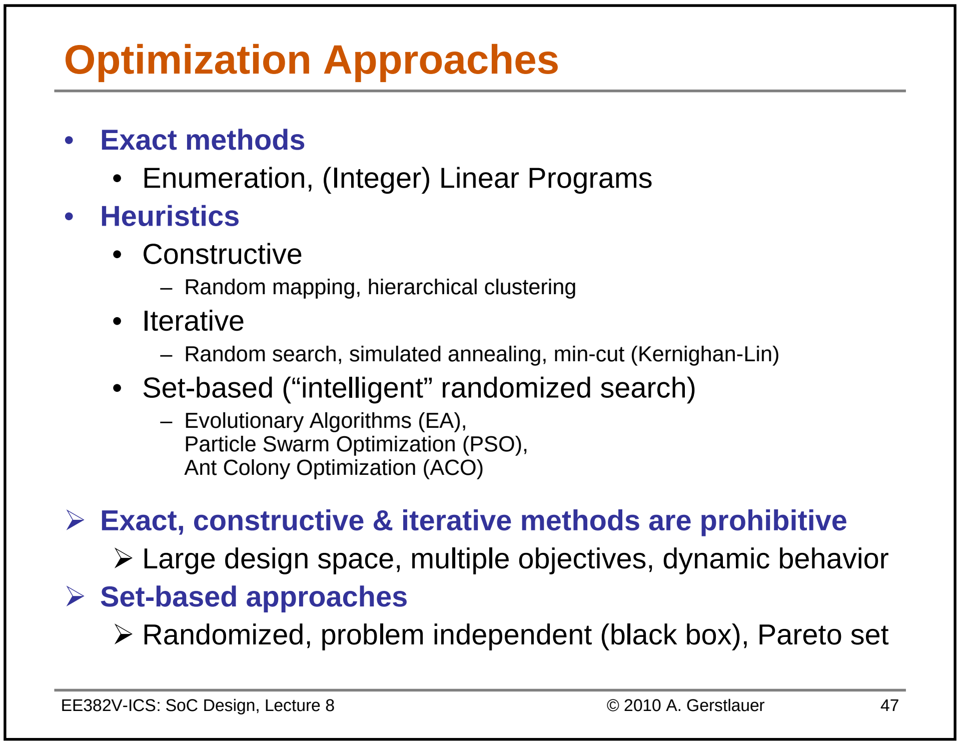
## Design Space Exploration

- Design Space Exploration is an iterative process:
  - How can a single design point be evaluated?
  - How can the design space be covered during the exploration process?



Evaluating design points

Covering the design space

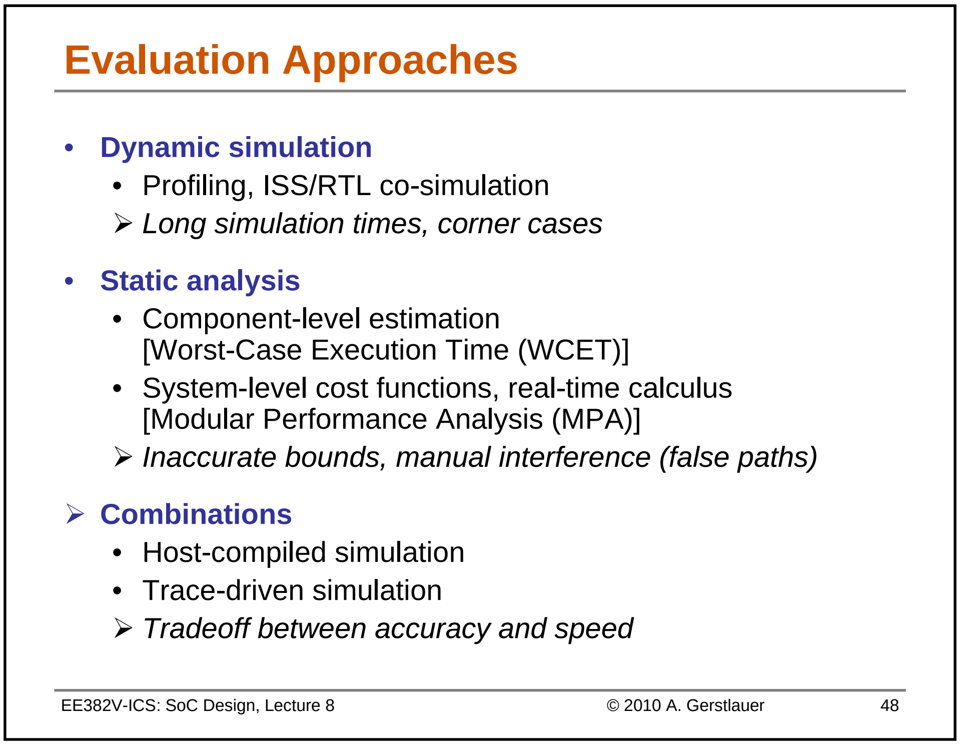


## Optimization Approaches

- **Exact methods**
  - Enumeration, (Integer) Linear Programs
- **Heuristics**
  - Constructive
    - Random mapping, hierarchical clustering
  - Iterative
    - Random search, simulated annealing, min-cut (Kernighan-Lin)
  - Set-based ("intelligent" randomized search)
    - Evolutionary Algorithms (EA),<br>Particle Swarm Optimization (PSO),<br>Ant Colony Optimization (ACO)

➢ **Exact, constructive & iterative methods are prohibitive**
- ➢ Large design space, multiple objectives, dynamic behavior

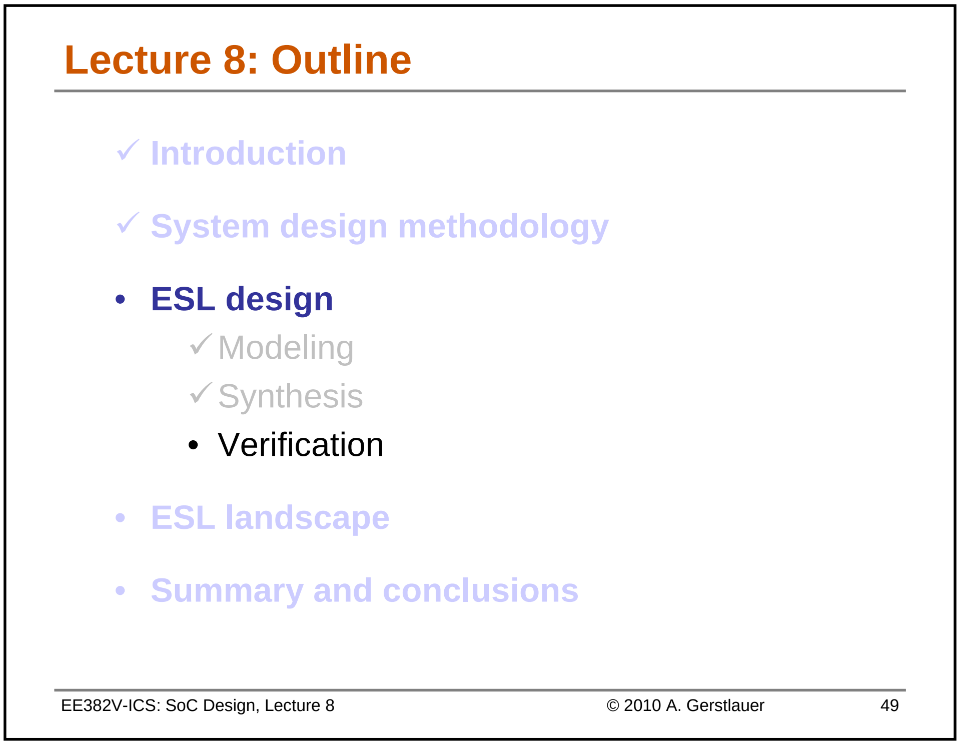
➢ **Set-based approaches**
- ➢ Randomized, problem independent (black box), Pareto set



## Evaluation Approaches

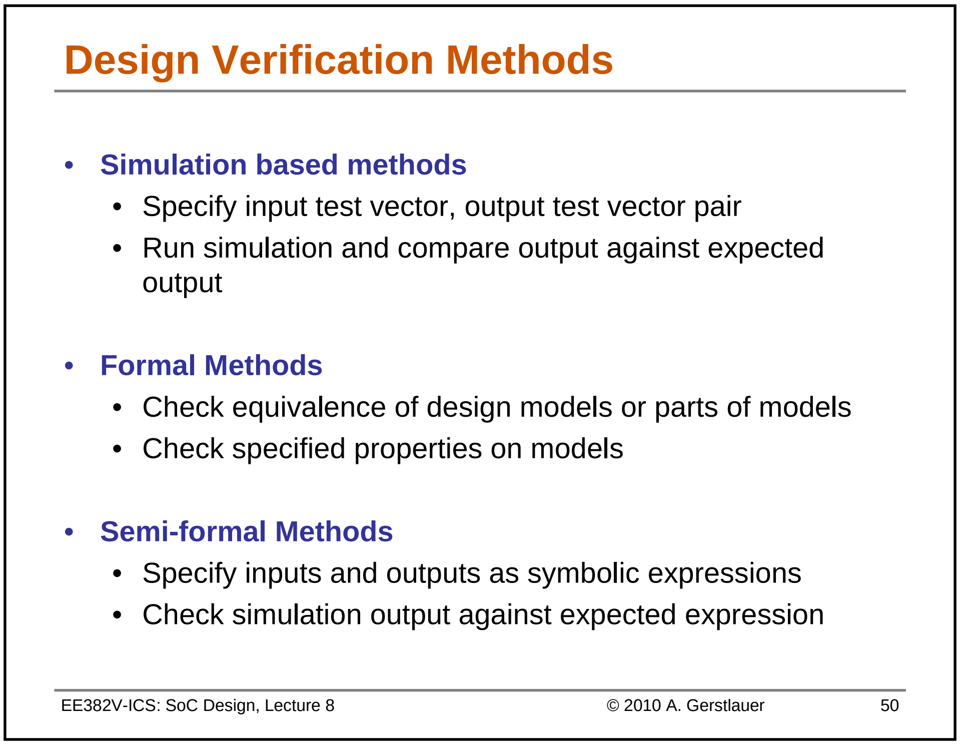
- **Dynamic simulation**
  - Profiling, ISS/RTL co-simulation
  - ➢ *Long simulation times, corner cases*
- **Static analysis**
  - Component-level estimation<br>[Worst-Case Execution Time (WCET)]
  - System-level cost functions, real-time calculus<br>[Modular Performance Analysis (MPA)]
  - ➢ *Inaccurate bounds, manual interference (false paths)*

➢ **Combinations**
- Host-compiled simulation
- Trace-driven simulation
- ➢ *Tradeoff between accuracy and speed*



## Lecture 8: Outline

- ✓ Introduction
- ✓ System design methodology
- ESL design
  - ✓ Modeling
  - ✓ Synthesis
  - Verification
- ESL landscape
- Summary and conclusions

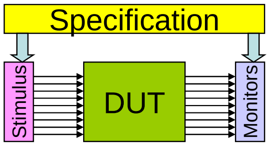


## Design Verification Methods

- **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

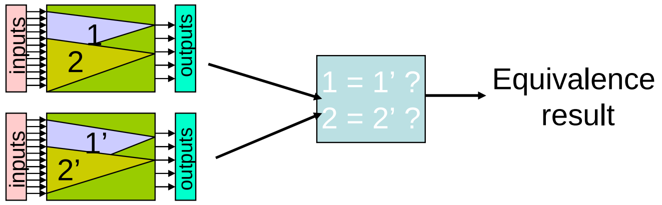
- **Create test vectors and simulate model**
  - Simulation, debugging and visualization tools [Synopsys VCS, Mentor ModelSim, Cadence NC-Sim]



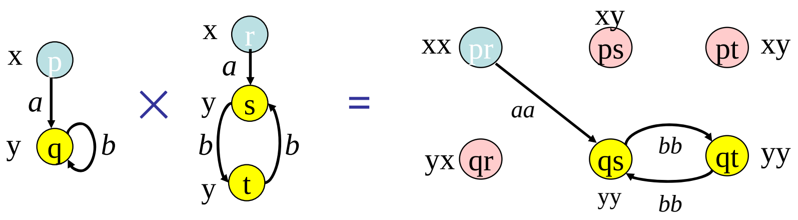
- **Inputs**
  - Specification
    - Used to create interesting stimuli and monitors
  - Model of DUT
    - Typically written in HDL or C or both
- **Output**
  - Failed test vectors
    - Pointed out in different design representations by debugging tools

## Equivalence Checking

- **LEC uses boolean algebra to check for logic equivalence**

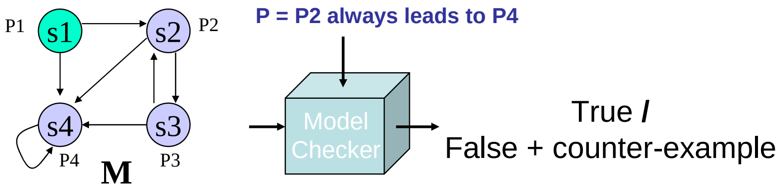


- **SEC uses FSMs to check for sequential equivalence**



## 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)



**P = P2 always leads to P4**

True / False + counter-example

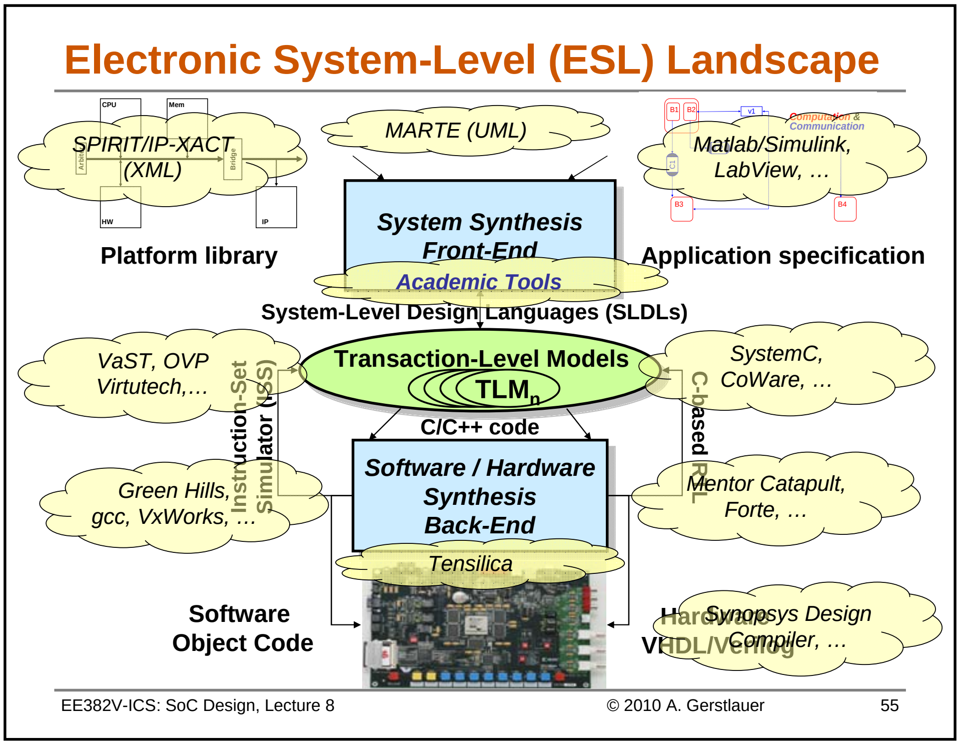
## Lecture 8: Outline

✓ Introduction

✓ System design methodology

✓ ESL design

- **ESL landscape**
  - Commercial tools
  - Academic tools
- Summary and conclusions



## Electronic System-Level (ESL) Landscape

SPIRIT/IP-XACT (XML)

Platform library

MARTE (UML)

System Synthesis Front-End

Academic Tools

Matlab/Simulink, LabView, …

Application specification

System-Level Design Languages (SLDLs)

VaST, OVP Virtutech,…

Instruction-Set Simulator (ISS)

Transaction-Level Models

$TLM_n$

SystemC, CoWare, …

C-based RTL

C/C++ code

Green Hills, gcc, VxWorks, …

Software / Hardware Synthesis Back-End

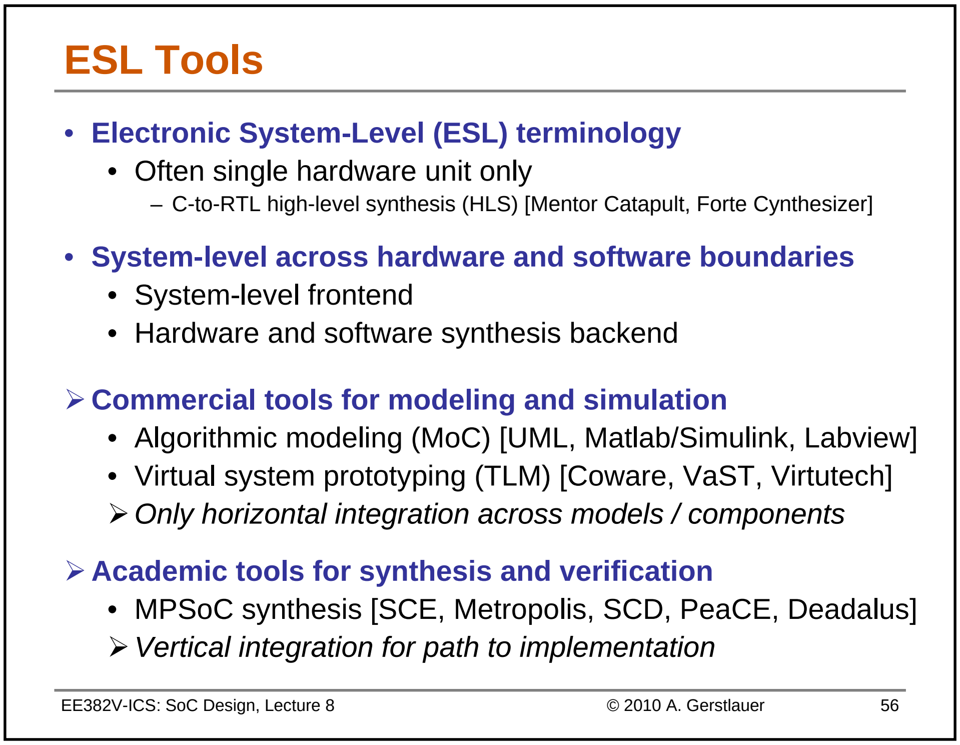
Mentor Catapult, Forte, …

Tensilica

Software Object Code

Hardware VHDL/Verilog

Synopsys Design Compiler, …



## ESL Tools

- **Electronic System-Level (ESL) terminology**
  - Often single hardware unit only
    - – C-to-RTL high-level synthesis (HLS) [Mentor Catapult, Forte Cynthesizer]
- **System-level across hardware and software boundaries**
  - System-level frontend
  - Hardware and software synthesis backend

➢ **Commercial tools for modeling and simulation**
  - Algorithmic modeling (MoC) [UML, Matlab/Simulink, Labview]
  - Virtual system prototyping (TLM) [Coware, VaST, Virtutech]
  - ➢ *Only horizontal integration across models / components*

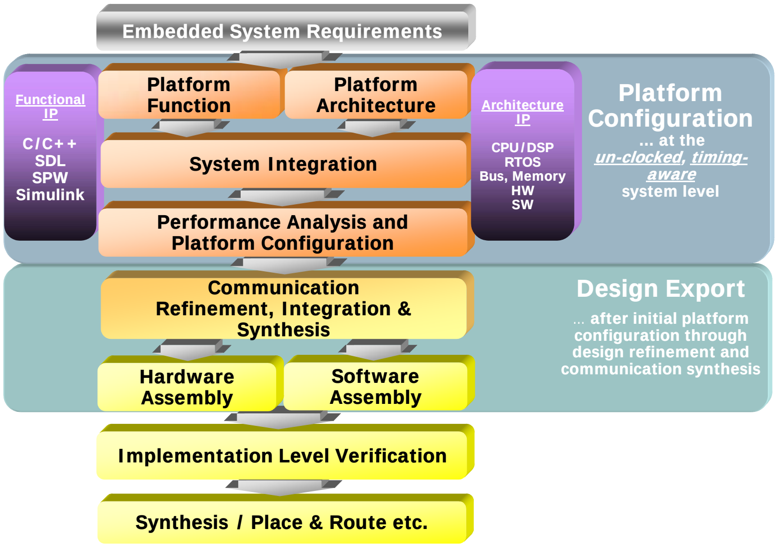
➢ **Academic tools for synthesis and verification**
  - MPSoC synthesis [SCE, Metropolis, SCD, PeaCE, Deadalus]
  - ➢ *Vertical integration for path to implementation*

## Academic MPSoC Design Tools

| Approach | DSE | Comp. decision | Comm. decision | Comp. refine | Comm. refine |
|---|---|---|---|---|---|
| Daedalus | ● | ● | ○ | ● | ○ |
| Koski | ● | ● | ○ | ● | ○ |
| Metropolis |  | ○ |  | ○ |  |
| PeaCE/HoPES | ○ | ○ |  | ● | ○ |
| SCE |  |  |  | ● | ● |
| SystemCoDesigner | ● | ● | ● | ○ |  |

*Source: A. Gerstlauer, C. Haubelt, A. Pimentel, et al., "Electronic System-Level Synthesis Methodologies," TCAD, 2009.*

## System Design Flow Summary


Embedded System Requirements
Functional IP
C/C++
SDL
SPW
Simulink
Platform Function
Platform Architecture
System Integration
Performance Analysis and Platform Configuration
Architecture IP
CPU/DSP
RTOS
Bus, Memory
HW
SW
Platform Configuration
... at the *un-clocked*, *timing-aware* system level
Communication Refinement, Integration & Synthesis
Hardware Assembly
Software Assembly
Design Export
... after initial platform configuration through design refinement and communication synthesis
Implementation Level Verification
Synthesis / Place & Route etc.