EE382M.20: System-on-Chip (SoC) Design

Lecture 2 – Electronic System-Level (ESL) Design

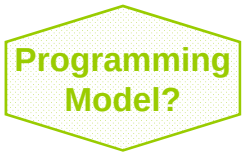
with sources from:
Christian Haubelt, Univ. of Erlangen-Nuremberg

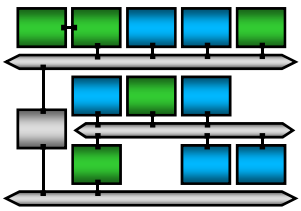
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

SoC Design Challenges

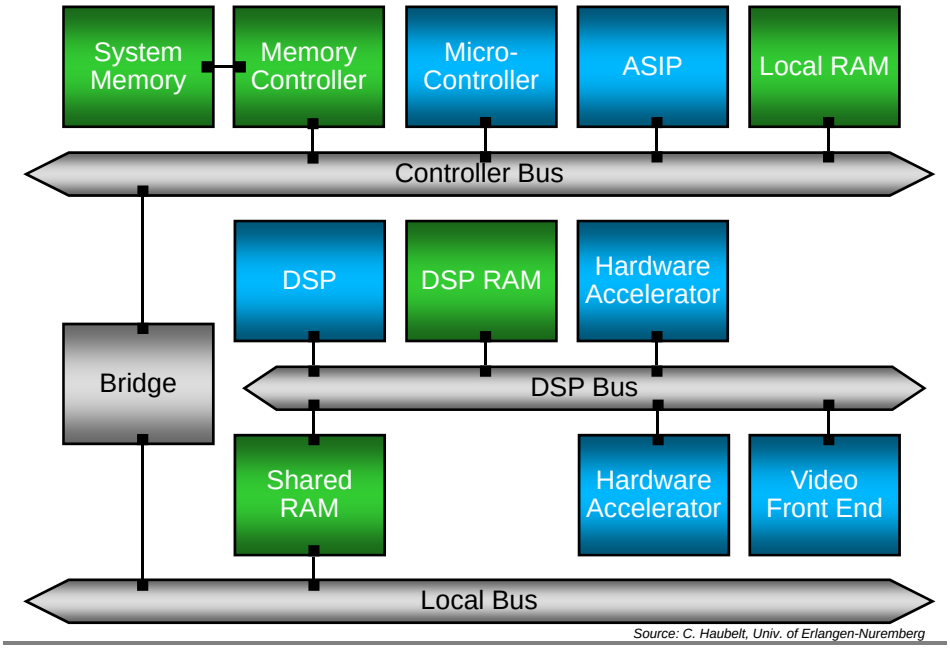
- **Complexity**
  - High degree of parallelism at various levels
- **Heterogeneity**
  - Of components
  - Of tools
- **Low-level communication mechanisms**
- **Programming model**





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

Multi-Processor System-on-Chip (MPSoC)



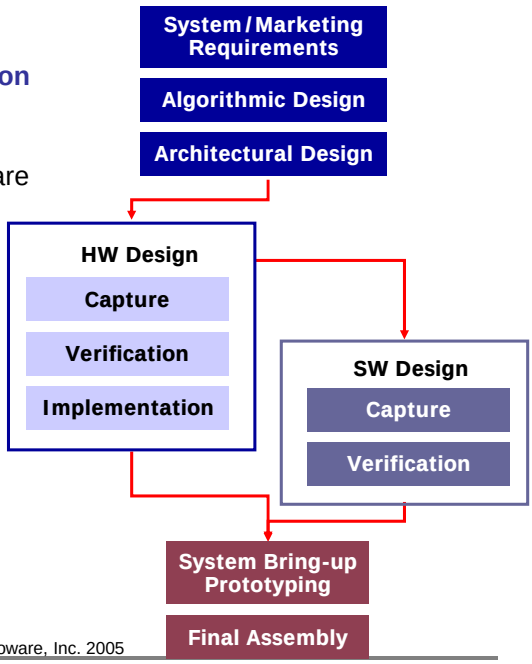
Source: C. Haubelt, Univ. of Erlangen-Nuremberg

Lecture 2: Outline

- ✓ **Introduction**
- **SoC design methodology**
  - Electronic system-level design (ESL/SLD)
- **System-level design process**
  - Synthesis
  - Modeling
- **Summary and conclusions**

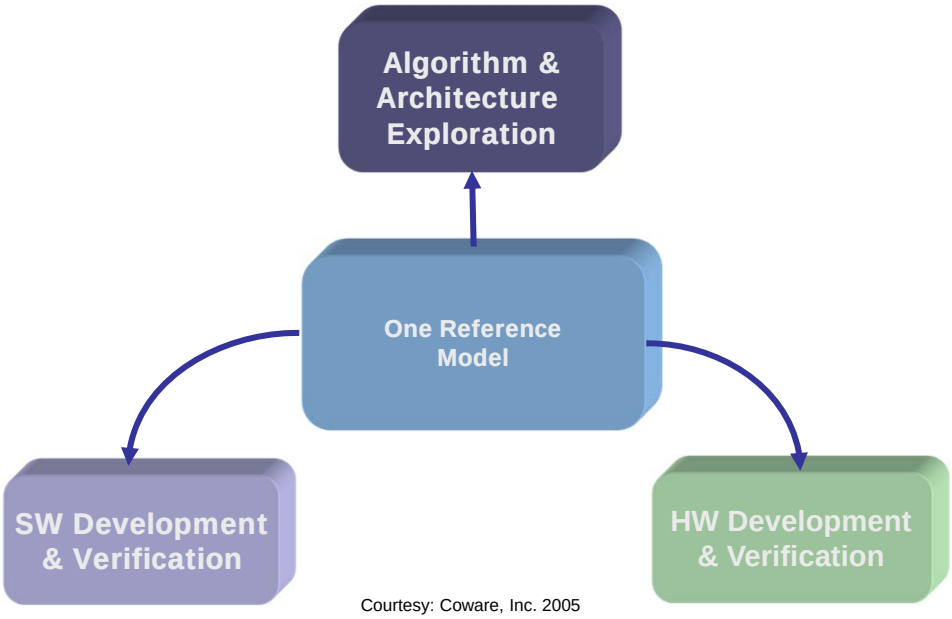
Issues in HW Centric Design Flows

- 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



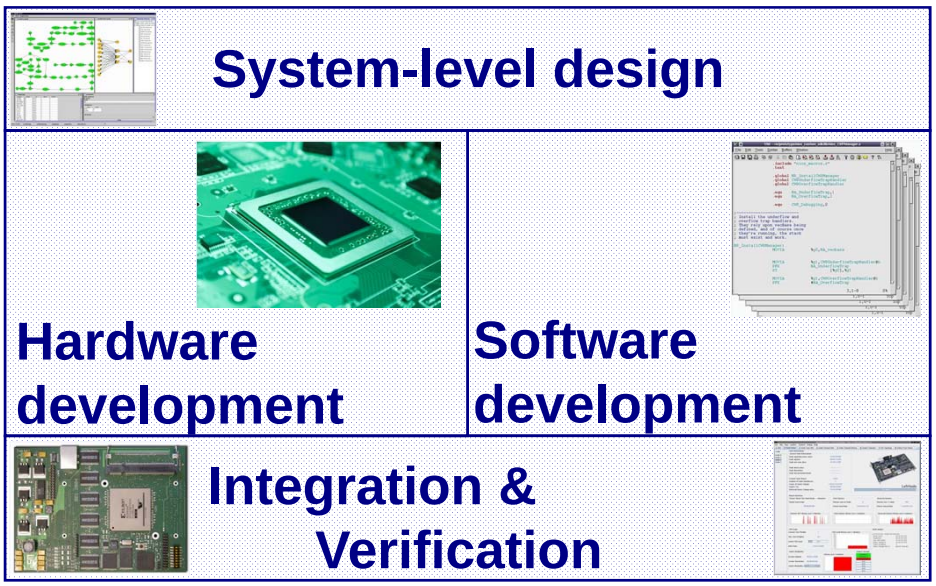
Courtesy: Coware, Inc. 2005

The ESL Solution: One Reference Model

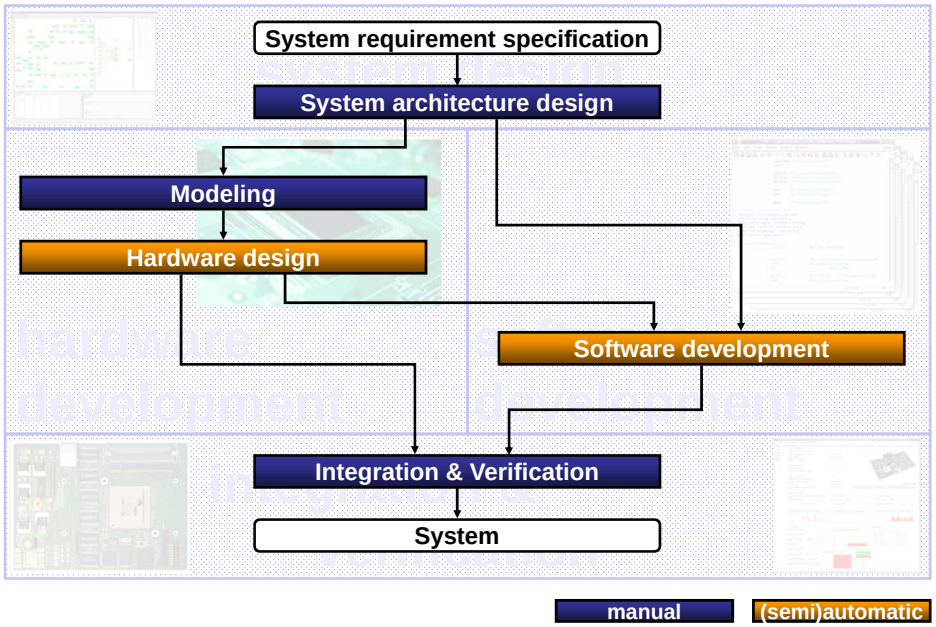


Courtesy: Coware, Inc. 2005

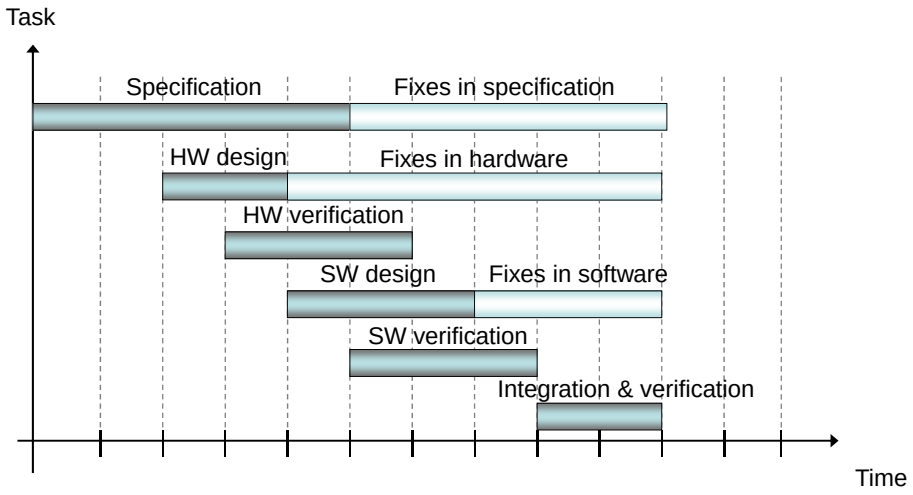
Electronic System-Level (ESL) Design



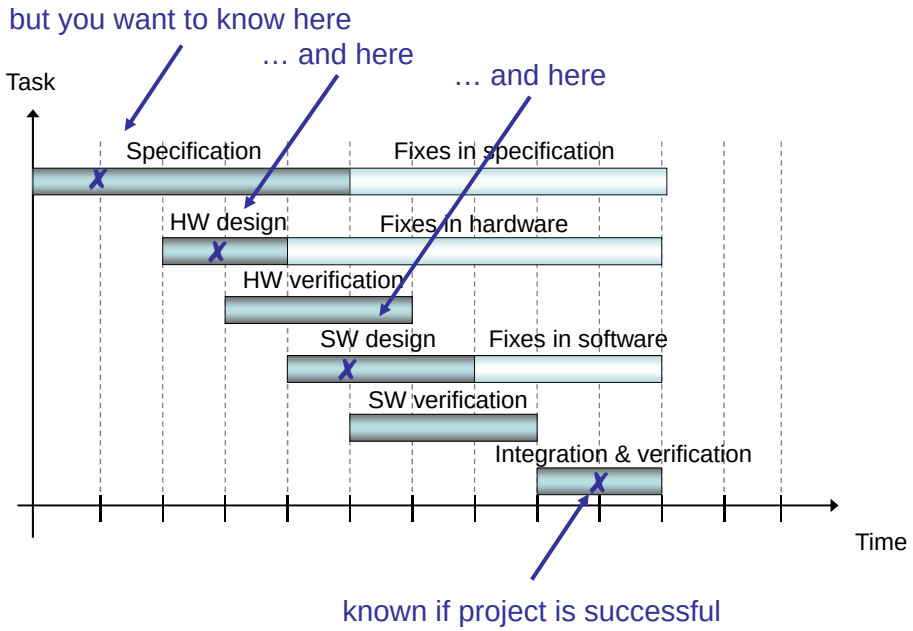
Classical System Design Flow



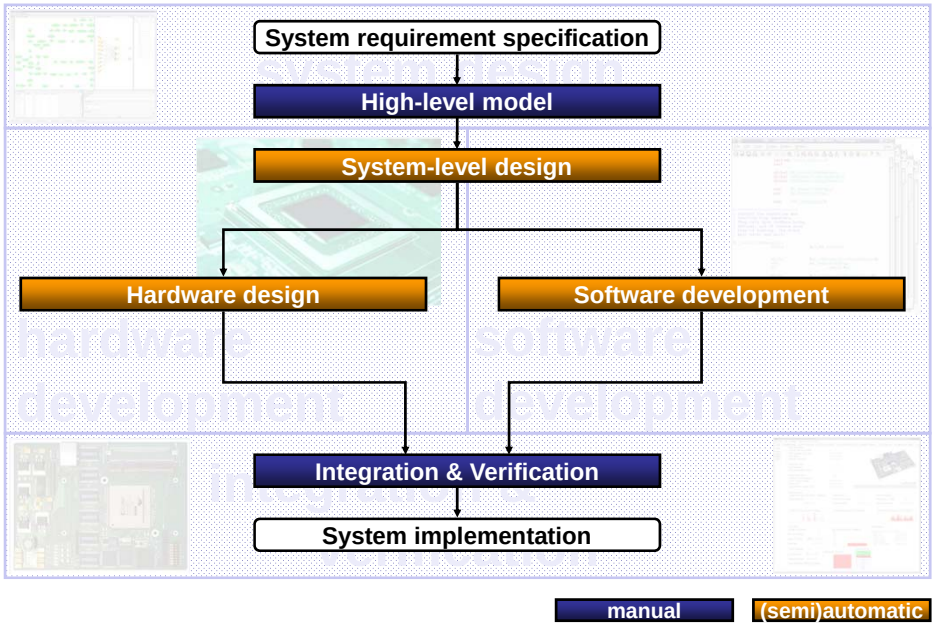
Hardware-Centric Design Cycle



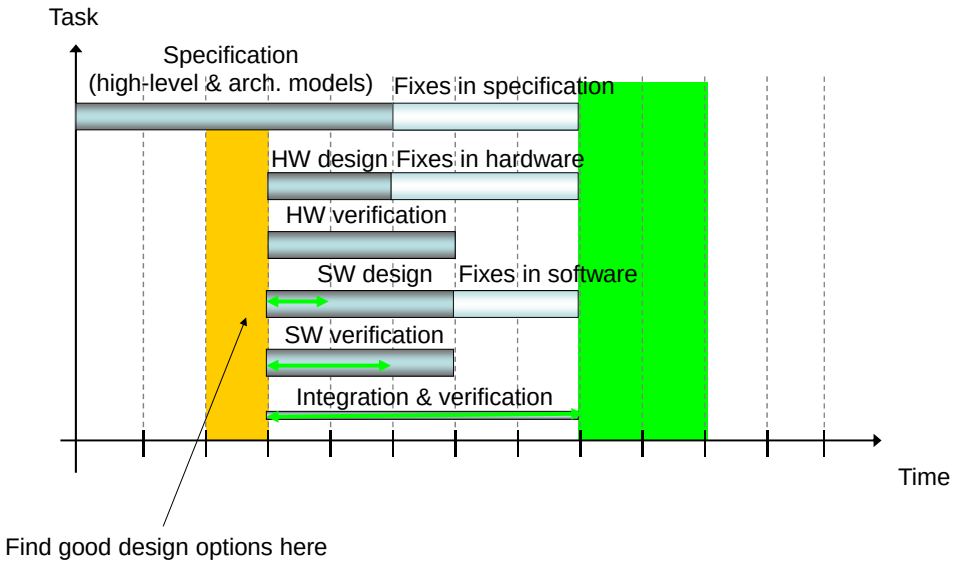
Hardware-Centric Design Cycle



Electronic System-Level (ESL) Design Flow



New ESL Design Cycle

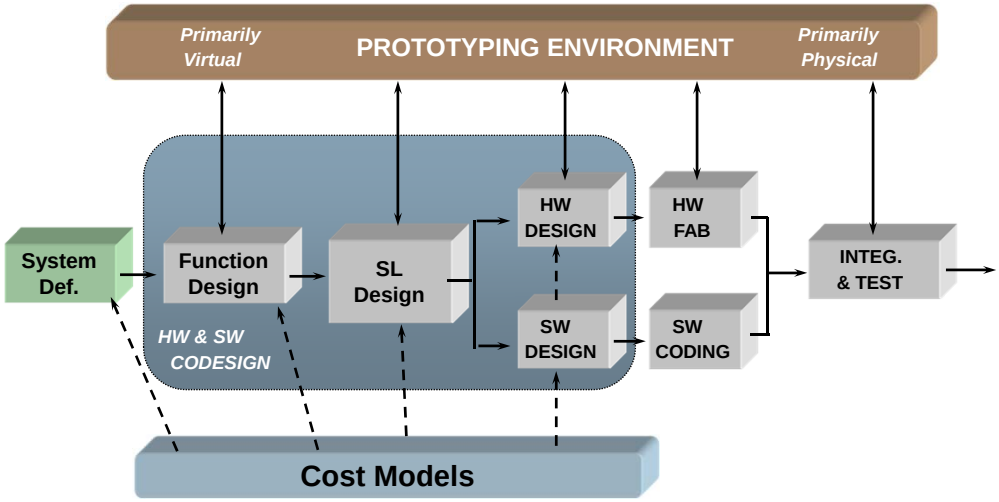


Design Methodologies

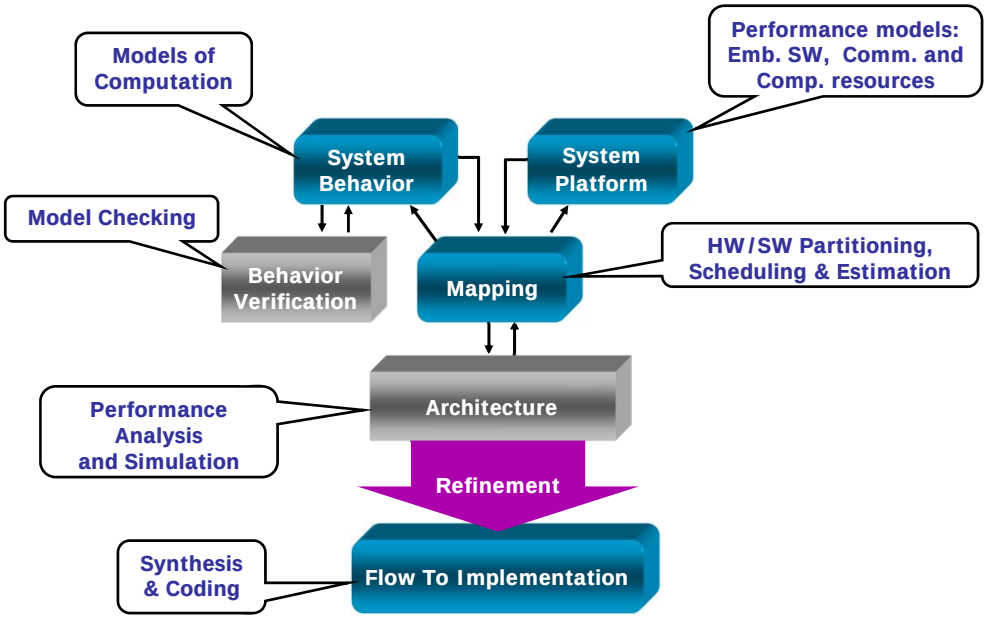
- **Top down design**
  - Starts with functional system specification
    - Application behavior
    - Models of Computation (MoC)
  - Successive refinement
  - Connect the hardware and software design teams earlier in the design cycle.
  - Allows hardware and software to be developed concurrently
  - 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



Platform-Based Design (PBD)

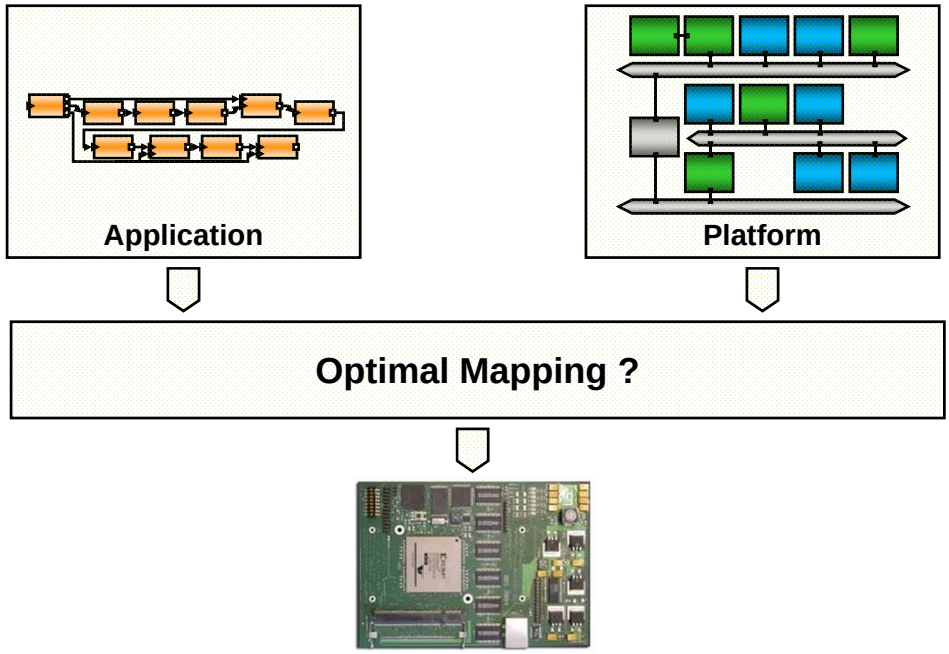


Source: UC Berkeley, EECS249

Lecture 2: Outline

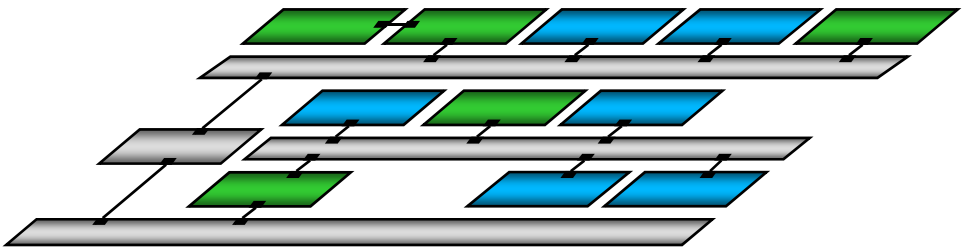
- ✓ Introduction
- ✓ SoC design methodology
- **System-level design**
  - Synthesis
  - Modeling
- Summary and conclusions

Platform-Based System Synthesis



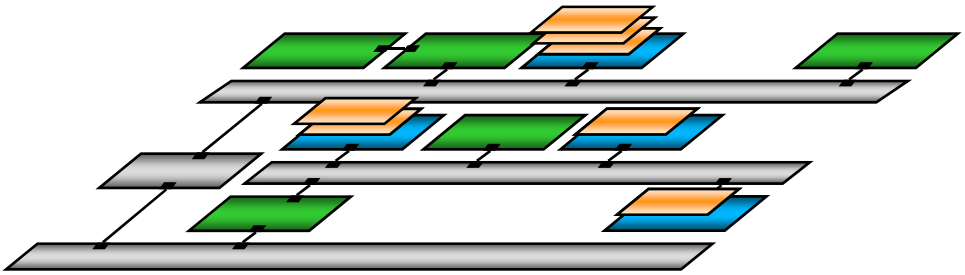
Resource Allocation

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



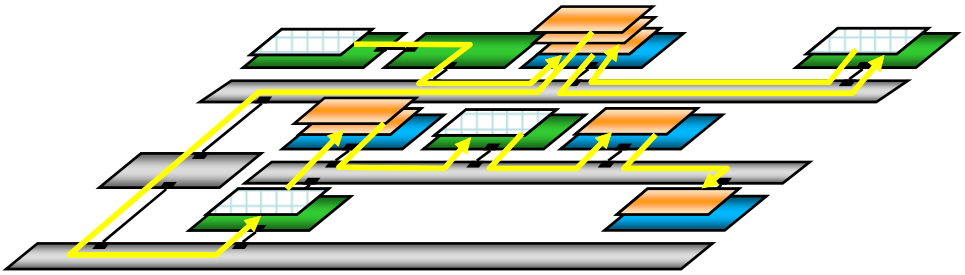
Process Binding

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



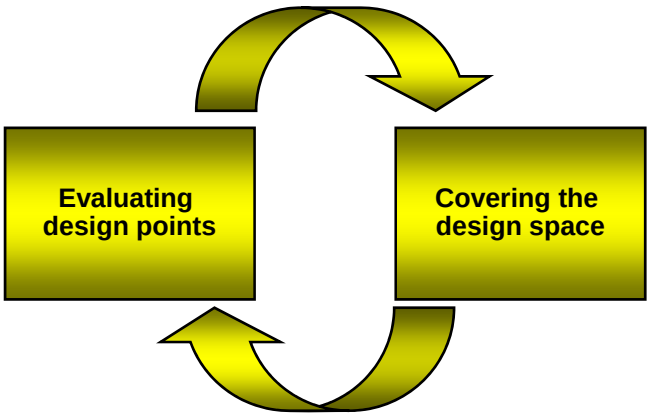
Channel Routing

- **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?



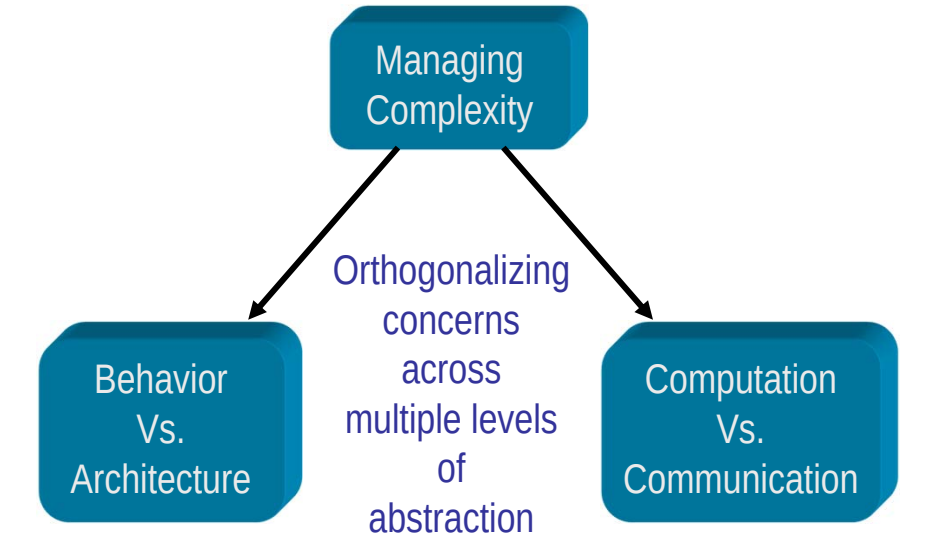
Lecture 2: Outline

- ✓ Introduction
- ✓ SoC design methodology
- **System-level design**
  - ✓ Synthesis
  - Modeling
- Summary and conclusions

System Modeling

- **Design models as abstraction of a design instance**
  - Representation for validation and analysis
  - Specification for further implementation
  - ➢ Documentation & specification
- ➢ **Systematic methodology**
  - Set of models and transformations (design steps)
  - Modeling flow defines the design process
  - ➢ From specification to implementation
- ➢ **Well-defined, rigorous semantics**
  - Unambiguous, explicit abstractions
  - Formal models
  - ➢ Synthesis and verification

Separation of Concerns



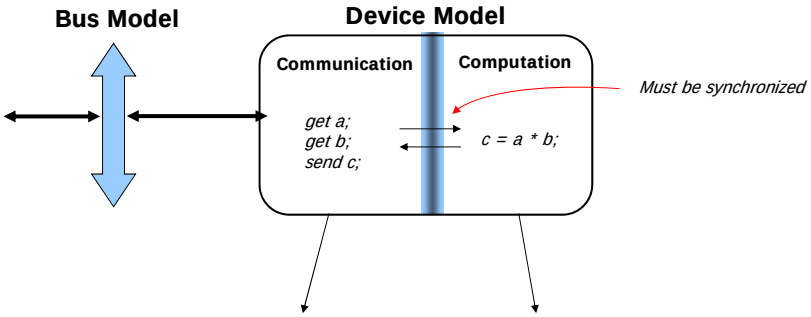
Platform-based design (PBD)

Transaction-level modeling (TLM)

Source: UC Berkeley, EECS249

Computation vs. Communication

- **Separation of concerns**
  - Flexibility in modeling, IP reuse
  - Design computation & communication separately

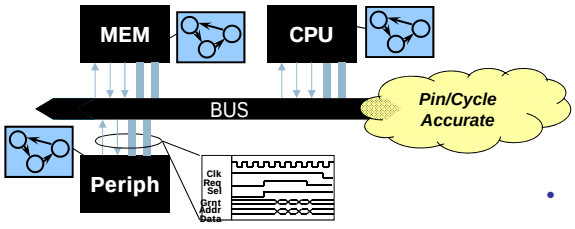


Communication can be described in a wide range of fashions, from high-level messages, to detailed signal level handshakes without impacting the behavior description.

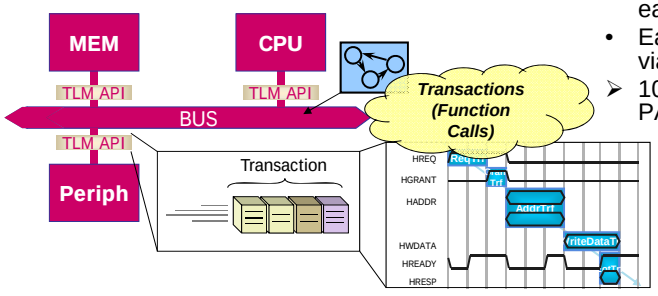
Behavior can be described algorithmically, without the burden of the handshaking and control logic associated with bus communication.

Source: Coware, Inc., 2005

Communication Models

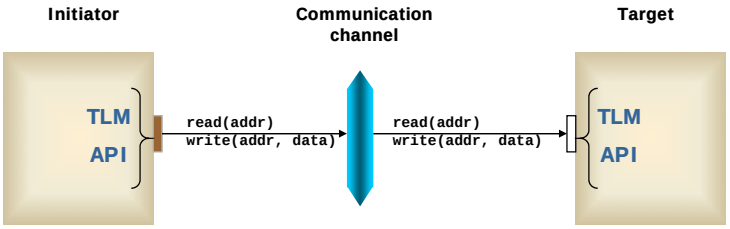


- **Pin-Accurate Model (PAM)**
  - Redundant RTL complexity results in slow simulation
  - Each device interface must implement the bus protocol
  - Each device on the bus has a pin-accurate interface
- **Transaction-Level Model (TLM)**
  - Less code, no wires, fewer events yield faster simulation
  - 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



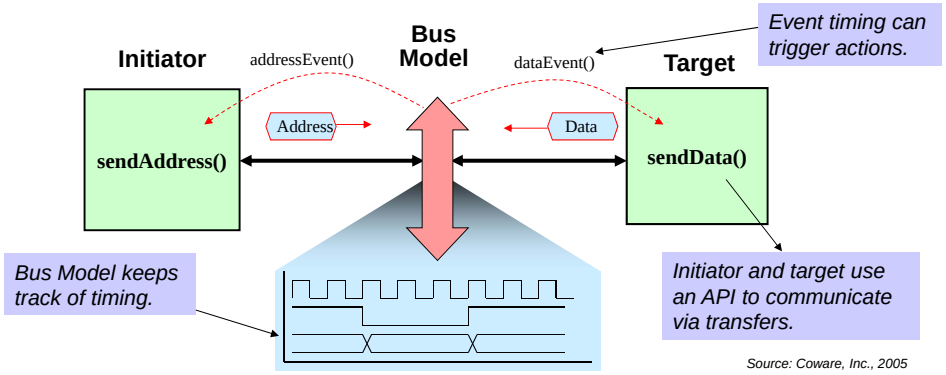
The transaction level is a higher level of abstraction for communication

For SoC, communication is often the bottleneck

Source: Coware, Inc., 2005

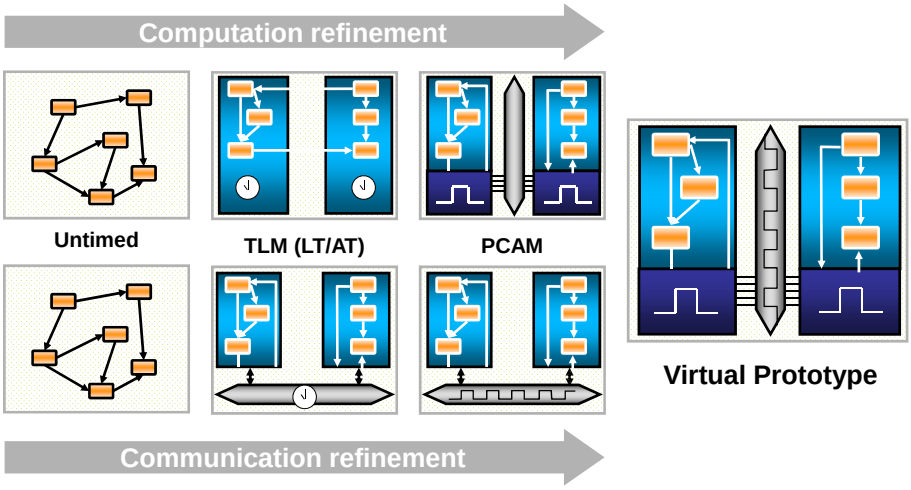
TLM Details

- **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.

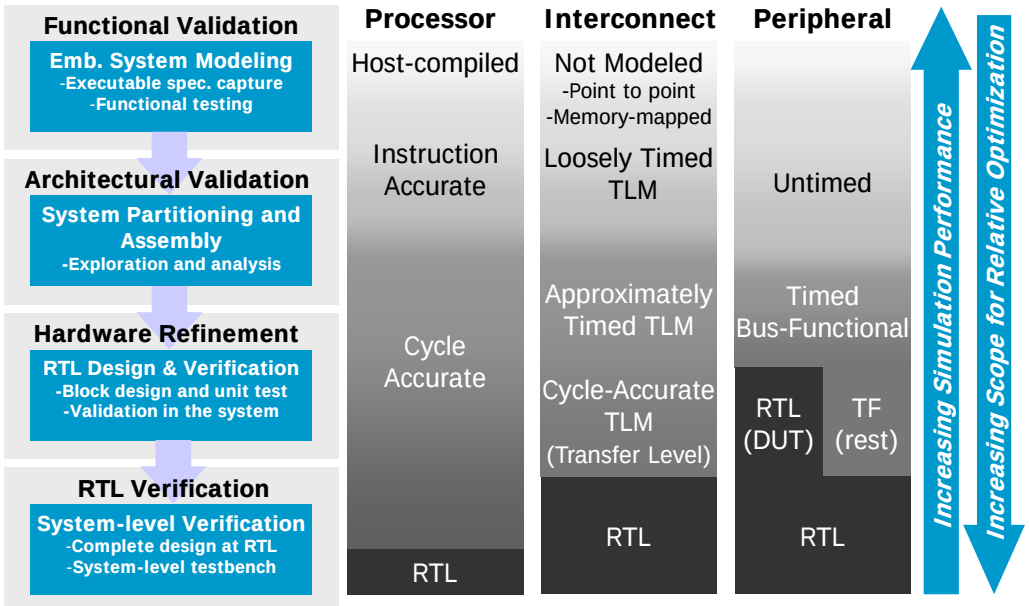


Source: Coware, Inc., 2005

Virtual Platform Prototyping

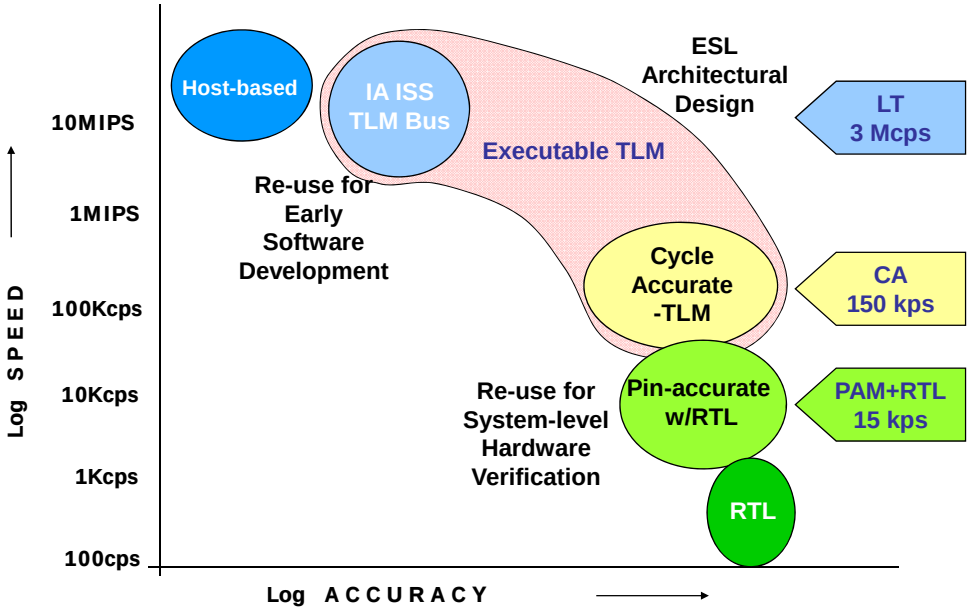


Abstraction Levels



Source: Coware, Inc., 2005

Speed vs. Accuracy



System Design Flow Summary