EECS 222A: System-on-Chip Description and Modeling Lecture 1

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 1: Overview

- Course overview
  - Context, content
- Course administration
  - Schedule, assignments, communication
- Introduction to System-on-Chip design
  - Embedded computer systems
  - Levels of abstraction
  - System design flow
  - Models of computation
  - System-level description languages
  - Computation, communication, IP

Course Context

- EECS 222: Set of 4 courses on SoC Design

  A. System-on-Chip Description and Modeling
  B. System-on-Chip Design and Exploration
  C. System-on-Chip Software Synthesis
  D. System-on-Chip Hardware Synthesis

  - **Course A is prerequisite for B, C, and D, or consent of instructor**

Course Context

- EECS 222: Set of 4 courses on SoC Design

  A. System-on-Chip Description and Modeling
  Computational models for System-on-Chip (SoC). System-level specification and description languages and execution semantics. Concepts, requirements, examples. SoC modeling at different levels of abstraction (untimed, approximate time, cycle-accurate). Modeling of IP (IP wrappers), design constraints, test benches. Simulation semantics and algorithms. Co-simulation methodology.

  B. System-on-Chip Design and Exploration
  C. System-on-Chip Software Synthesis
  D. System-on-Chip Hardware Synthesis

Course Context

- EECS 222: Set of 4 courses on SoC Design
  - **A. System-on-Chip Description and Modeling**
  - **B. System-on-Chip Design and Exploration**
    System-on-Chip design flow and methodology. Design space exploration. Co-design of hardware and software, hardware/software partitioning. System-on-Chip architecture exploration and synthesis. On-chip network and communication design and synthesis. On-chip software/hardware interface generation.
  - **C. System-on-Chip Software Synthesis**
  - **D. System-on-Chip Hardware Synthesis**

Course Context

- EECS 222: Set of 4 courses on SoC Design
  - **A. System-on-Chip Description and Modeling**
  - **B. System-on-Chip Design and Exploration**
  - **C. System-on-Chip Software Synthesis**
    System-on-Chip software concepts, requirements, examples, for engineering applications such as automotive and communication. Software synthesis methodology. Algorithmic specification, design constraints. Applications using embedded operating systems. Static, dynamic scheduling. Input/output, interrupt handling. Code generation, retargetable compilation. Instruction set simulation. Debugging and prototyping.
  - **D. System-on-Chip Hardware Synthesis**

Course Context

- EECS 222: Set of 4 courses on SoC Design
  - **A. System-on-Chip Description and Modeling**
  - **B. System-on-Chip Design and Exploration**
  - **C. System-on-Chip Software Synthesis**
  - **D. System-on-Chip Hardware Synthesis**
    Hardware IP specification. Real-time constraints. Cycle-accurate languages and modeling. Target architectures, data path and control unit. Design tasks and design methodology. Behavioral synthesis. Resource allocation, operation scheduling, binding of operations and variables to functional units, storage units and busses. Communication protocol and interface synthesis. Arbiter, bridge, Transducer, Controller design and synthesis. Net list generation.

Course Content

1. Introduction to SoC concepts, computational models
2. The SpecC system-level description language
3. SoC specification, modeling guidelines, validation
4. Execution and simulation semantics
5. Abstraction levels, top-down design methodology
6. SoC architecture, processor modeling
7. SoC communication modeling
8. SoC hardware modeling, RTL
9. The SystemC system-level description language
10. UML and other system-level description languages

Course Administration

- Course web pages at **`http://eee.uci.edu/12s/18422/`**
  - Instructor information
  - Course description and policies
  - Objectives and outcomes
  - Contents and schedule
  - Resources and communication
  - Assignments

Introduction to SoC Design

- Embedded Computer Systems
- System-on-Chip (SoC) Design
- Abstraction Levels
- SoC Design Flow
- Models of Computation
- System-Level Description Languages
- Computation vs. Communication
- Intellectual Property

Embedded Computer Systems

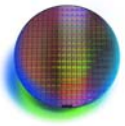
- Computers are ubiquitous, omnipresent…







- *System-on-Chip (SoC) Design*: Design of complex embedded systems on a single chip

Embedded Systems

- System embedded into another system
  - Constraints from external input (often real-time)
  - Application specific (not general purpose)
- Omnipresent in our environment
  - In many application domains
  - In 2005 [Source Netrino]
    - Only 2% of all processors in workstations
    - Remaining 8.8 billion in embedded systems
  - Pervasive


Source: P. Chou, UCI


Source: Edumicator


Source: Miele


Source: Philips


Source: www.trouper.com


Source: www.medicacorp.com/

Embedded System Design

- Design challenges
  - Often mobile
    - Battery powered (low power)
  - Often highly reliable
    - Extreme environment (e.g. temperature)
  - High performance constraints
    - Often real-time requirements
  - High complexity
    - E.g. Mercedes Benz E-class
      - 55 electronic control units
      - 5 communication busses
  - Tightly coupled
    - Software
    - Hardware
  - Rapid development for low price…

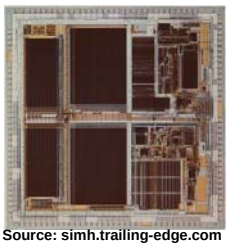
Source: Motorola Inc

Source: Daimler

EECS222A: SoC Description and Modeling, Lecture 1 (c) 2012 R. Doemer 13

Embedded System Design

- Design Advantages
  - *Application known at design time*
  - *Environment known at design time*
  - Allows for customized / optimized solution
    - Improved performance
    - More functionality
    - At lower power



Source: simh.trailing-edge.com

- Custom Platform, SW and HW components
  - Multi-Processor System-on-Chip (MPSoC),
    - Complete embedded system integrated on a chip
  - General-purpose and application-specific processors
  - Application Specific Integrated Circuit (ASIC)
  - Field Programmable Gate Array (FPGA)
  - Circuit board with off-the-shelf-components

Source: Xilinx

EECS222A: SoC Description and Modeling, Lecture 1 (c) 2012 R. Doemer 14

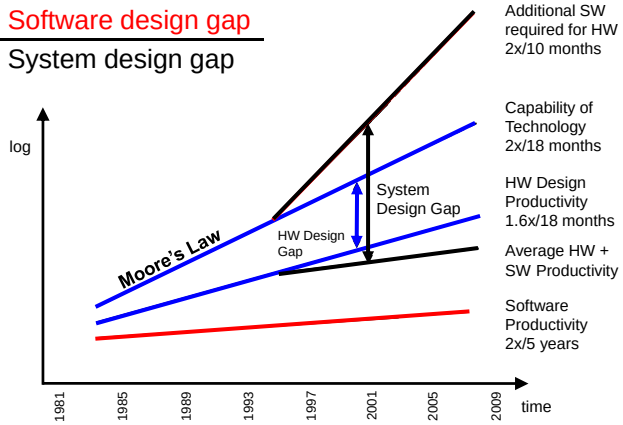
Design Complexity Challenge

- Productivity Gap

  Hardware design gap

  \+ Software design gap

  = System design gap



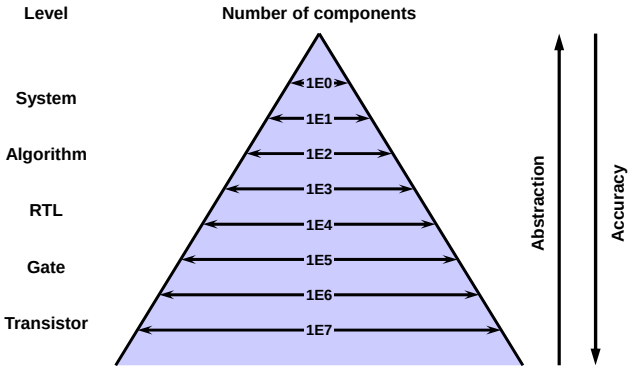
(source: *"Hardware-dependent Software"*, Ecker et al., 2009)

Design Complexity Challenge

- Productivity Gaps
  - Hardware productivity gap
    - Capacities in chip size outpace capabilities in chip design
    - Moore's law: chip capacity doubles every 18 months
    - HW design productivity estimated at 1.6x over 18 months
  - Software productivity gap
    - Growth of SW productivity estimated at 2x every 5 years
    - Needs in embedded SW estimated at 2x over 10 months
  - System productivity gap
    - HW gap + SW gap

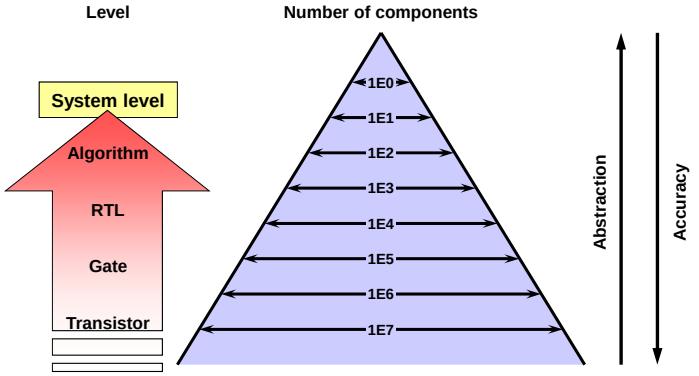
Abstraction Levels

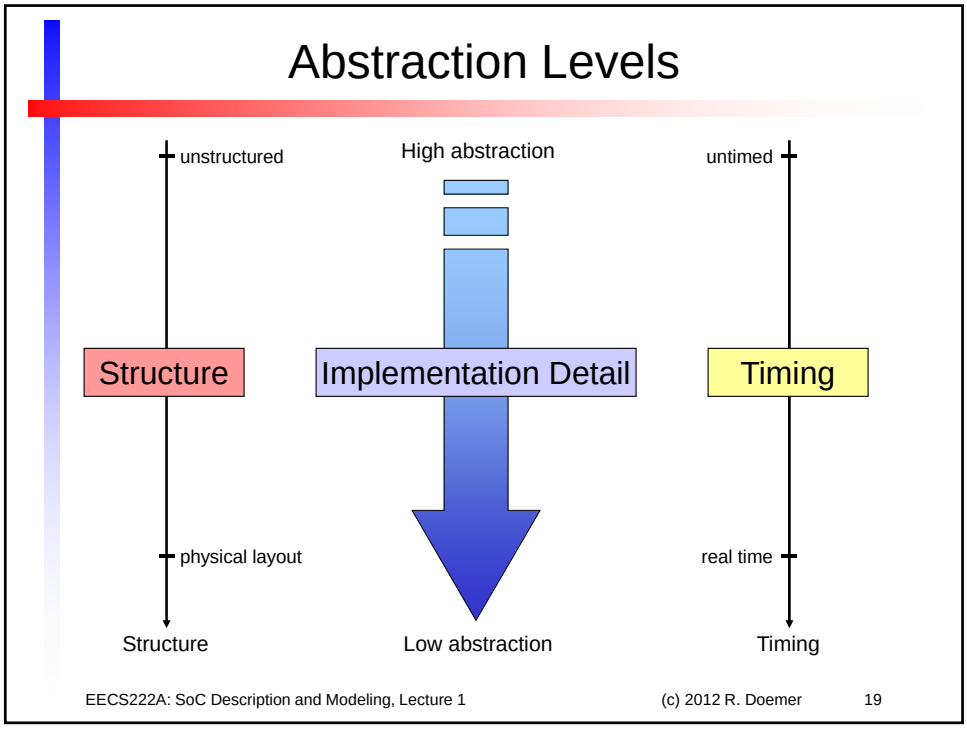
- System-on-Chip (SoC) design faces tremendous increase of design complexity

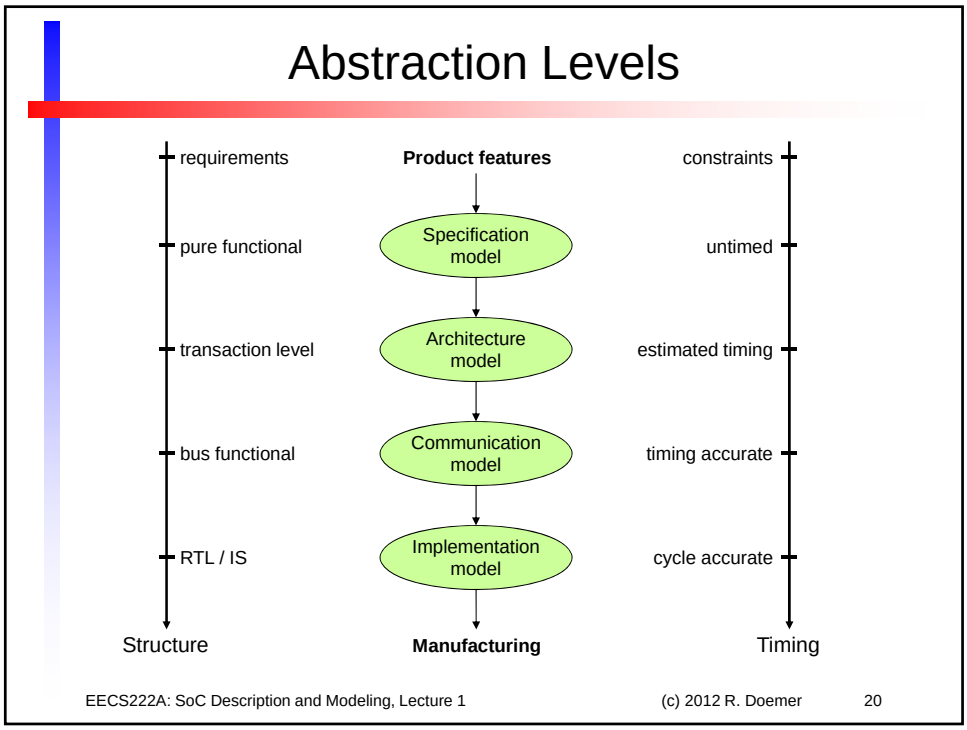


Abstraction Levels

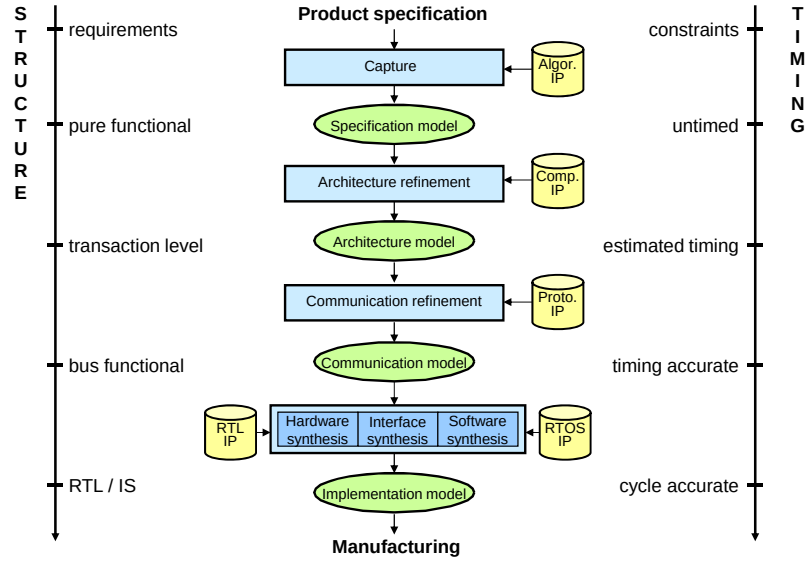
- System-on-Chip (SoC) design faces tremendous increase of design complexity
- Move to higher levels of abstraction!




Abstraction Levels
unstructured
High abstraction
untimed
Structure
Implementation Detail
Timing
physical layout
real time
Structure
Low abstraction
Timing
EECS222A: SoC Description and Modeling, Lecture 1
(c) 2012 R. Doemer
19


Abstraction Levels
requirements
Product features
constraints
pure functional
Specification model
untimed
transaction level
Architecture model
estimated timing
bus functional
Communication model
timing accurate
RTL / IS
Implementation model
cycle accurate
Structure
Manufacturing
Timing
EECS222A: SoC Description and Modeling, Lecture 1
(c) 2012 R. Doemer
20

Top-Down SoC Design Flow

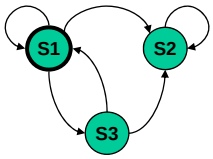


Models of Computation

- Computational Model
  - Formal, abstract description of a system
  - Various degrees of
    - supported features
    - complexity
    - expressive power
- Examples
  - Evolution process from FSM to PSM
    - Finite State Machine (FSM)
    - FSM with Data (FSMD)
    - Super-state FSMD
    - ...
    - Program State Machine (PSM)

Models of Computation

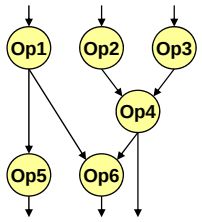
- Finite State Machine (FSM)
  - Basic model for describing control
  - States and state transitions
    - FSM = <*S*, *I*, *O*, *f*, *h*>
  - Two types:
    - Mealy-type FSM (input-based)
    - Moore-type FSM (state-based)



FSM model

Models of Computation

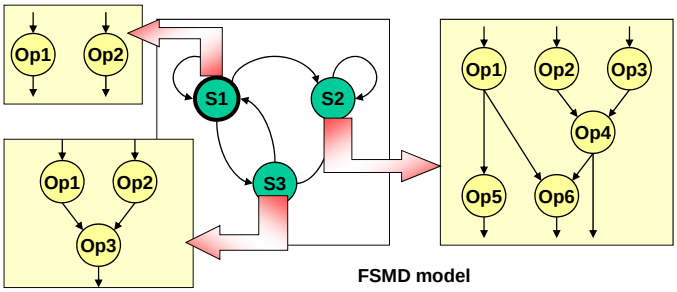
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
  - Basic model for describing computation
  - Directed graph
    - Nodes: operations
    - Arcs: dependency of operations



DFG model

Models of Computation

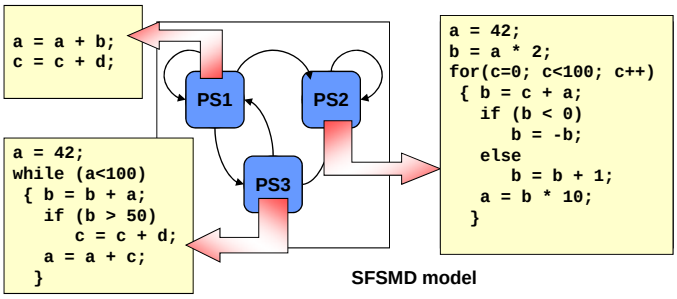
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
  - Combined model for control and computation
    - FSMD = FSM + DFG
  - Implementation: controller plus datapath



FSMD model

Models of Computation

- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
- Super-State FSM with Data (SFSMD)
  - FSMD with complex, multi-cycle states
    - States described by procedures in a programming language

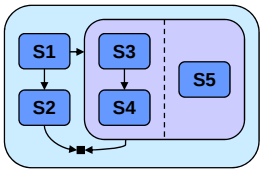


SFSMD model

Models of Computation

- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
- Super-State FSM with Data (SFSMD)
- Hierarchical Concurrent FSM (HCFSM)
  - FSM extended with hierarchy and concurrency
    - Multiple FSMs composed hierarchically and in parallel
  - Example: Statecharts

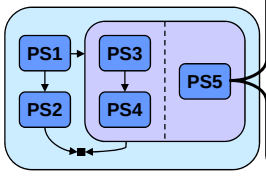
HCFSM model



Models of Computation

- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
- Super-State FSM with Data (SFSMD)
- Hierarchical Concurrent FSM (HCFSM)
- Program State Machine (PSM)
  - HCFSMD plus programming language
    - States described by procedures in a programming language
  - Example: SpecC!

PSM model



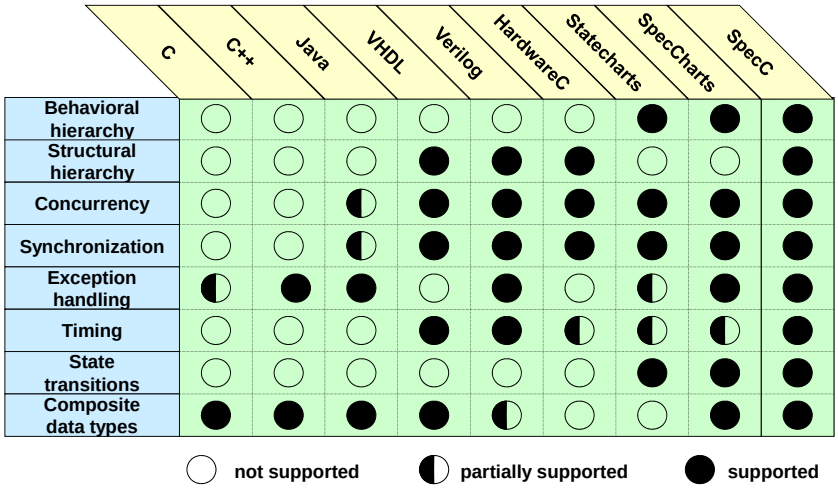

```
...
a = 42;
while (a<100)
 { b = b + a;
   if (b > 50)
      c = c + d;
   else
      c = c + e;
   a = c;
 }
...
```

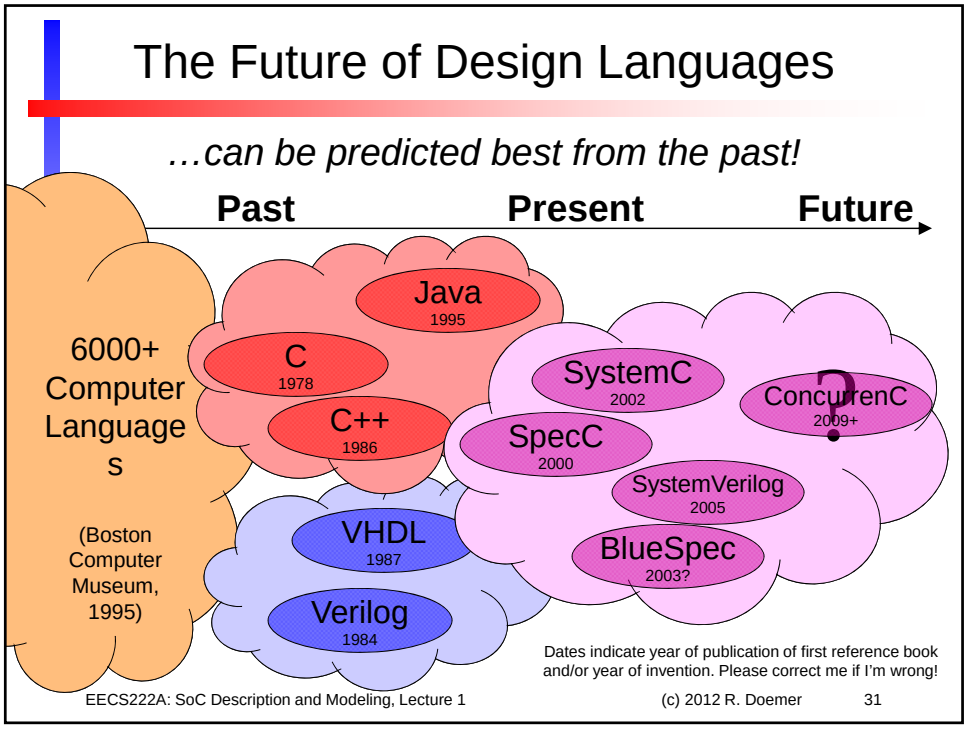
System-Level Description Languages

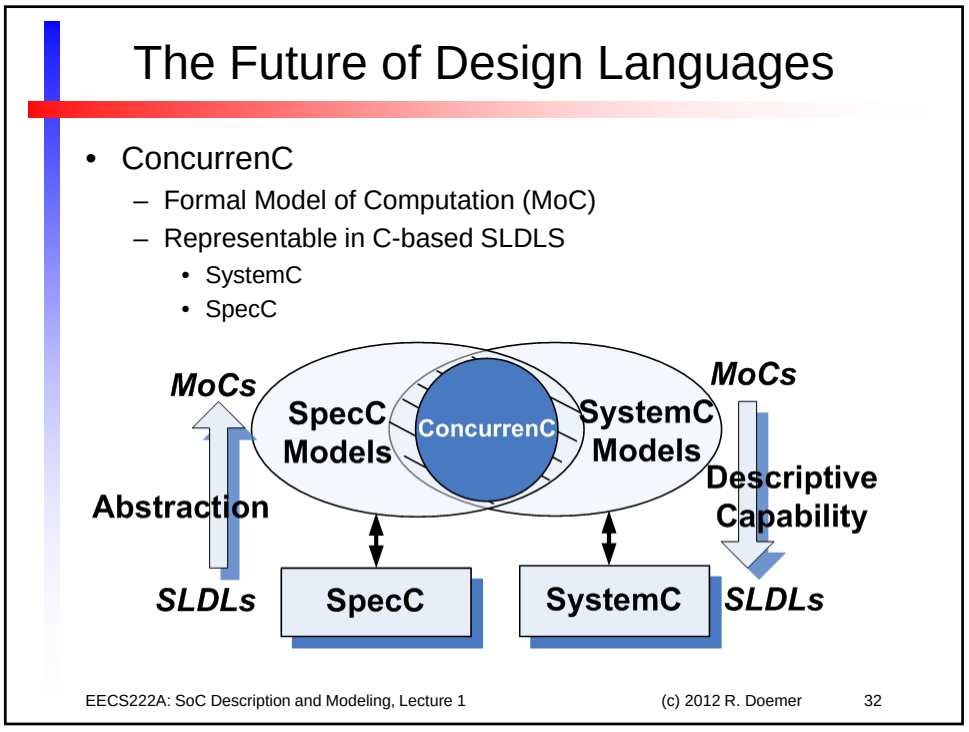
- Goals and Requirements
  - Formality
    - Formal syntax and semantics
  - Executability
    - Validation through simulation
  - Synthesizability
    - Implementation in HW and/or SW
    - Support for IP reuse
  - Modularity
    - Hierarchical composition
    - Separation of concepts
  - Completeness
    - Support for all concepts found in embedded systems
  - Orthogonality
    - Orthogonal constructs for orthogonal concepts
  - Simplicity
    - Minimality

System-Level Description Languages

- Requirements supported by existing languages




The Future of Design Languages
…can be predicted best from the past!
Past
Present
Future
6000+ Computer Languages
(Boston Computer Museum, 1995)
Java 1995
C 1978
C++ 1986
VHDL 1987
Verilog 1984
SystemC 2002
SpecC 2000
SystemVerilog 2005
BlueSpec 2003?
ConcurrenC 2009+
?
Dates indicate year of publication of first reference book and/or year of invention. Please correct me if I'm wrong!
EECS222A: SoC Description and Modeling, Lecture 1
(c) 2012 R. Doemer
31


The Future of Design Languages
• ConcurrenC
– Formal Model of Computation (MoC)
– Representable in C-based SLDLS
• SystemC
• SpecC
MoCs
SpecC Models
ConcurrenC
SystemC Models
MoCs
Abstraction
Descriptive Capability
SLDLs
SpecC
SystemC
SLDLs
EECS222A: SoC Description and Modeling, Lecture 1
(c) 2012 R. Doemer
32

System-Level Description Languages

- Examples in use today
  - C/C++
    - ANSI standard programming languages, software design
    - traditionally used for system design because of practicality, availability
  - SystemC
    - C++ API and library
    - initially developed at UCI, supported by Open SystemC Initiative
  - SpecC
    - C extension
    - developed at UCI, supported by SpecC Technology Open Consortium
  - SystemVerilog
    - Verilog with C extensions
  - Matlab
    - specification and simulation in engineering, algorithm design
  - UML
    - unified modeling language, software specification, graphical
  - SDL
    - telecommunication area, standard by ITU, used in COSMOS
  - SLDL
    - formal specification of requirements, not executable
  - etc.

System-Level Description Languages

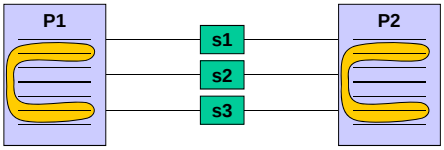
- Examples in use today and coverage in this course
  - C/C++
    - ANSI standard programming languages, software design
    - traditionally used for system design because of practicality, availability
  - ➢ SystemC
    - C++ API and library
    - initially developed at UCI, supported by Open SystemC Initiative
  - ➢ SpecC
    - C extension
    - developed at UCI, supported by SpecC Technology Open Consortium
  - SystemVerilog
    - Verilog with C extensions
  - Matlab
    - specification and simulation in engineering, algorithm design
  - ➢ UML
    - unified modeling language, software specification, graphical
  - SDL
    - telecommunication area, standard by ITU, used in COSMOS
  - SLDL
    - formal specification of requirements, not executable
  - etc.

Separation of Concerns

- Fundamental Principle in Modeling of Systems
- Clear *separation of concerns*
  - address separate issues independently
- System-Level Description Language (SLDL)
  - orthogonal concepts
  - orthogonal constructs
- System-level Modeling
  - Computation
    - encapsulated in modules / behaviors
  - Communication
    - encapsulated in channels

Computation vs. Communication

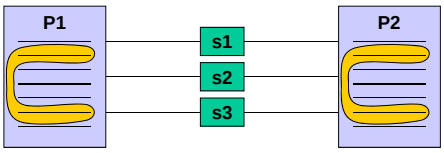
- Traditional model



  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible

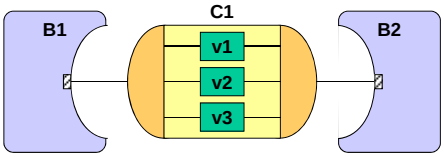
Computation vs. Communication

- Traditional model



  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible

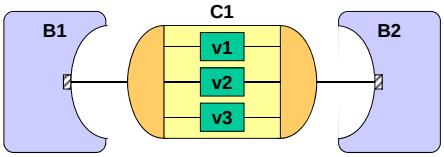
- SpecC model



  - Behaviors and channels
  - Separation of computation and communication
  - Plug-and-play

Computation vs. Communication

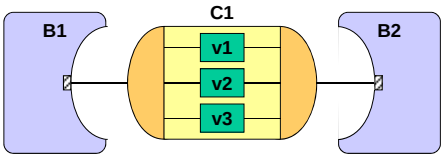
- Protocol Inlining
  - Specification model
  - Exploration model



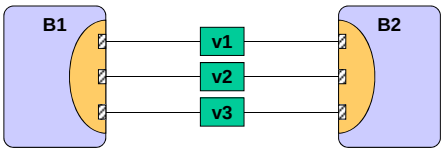
  - Computation in behaviors
  - Communication in channels

Computation vs. Communication

- Protocol Inlining
  - Specification model
  - Exploration model



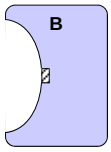
    - Computation in behaviors
    - Communication in channels
  - Implementation model



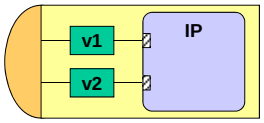
    - Channel disappears
    - Communication inlined into behaviors
    - Wires exposed

Intellectual Property (IP)

- Computation IP: Wrapper model



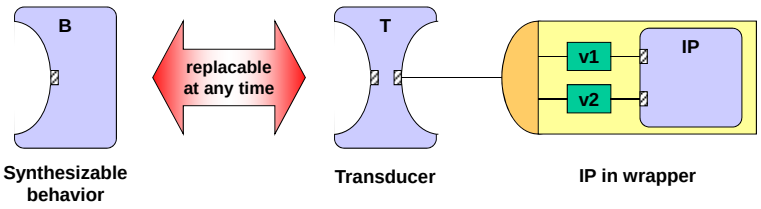
Synthesizable behavior



IP in wrapper

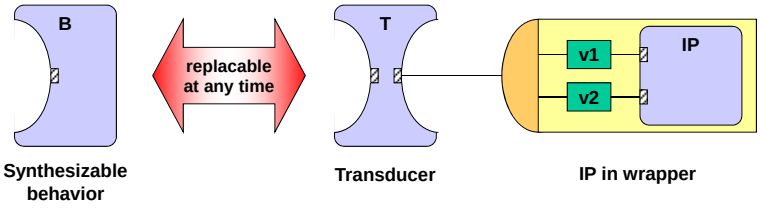
Intellectual Property (IP)

- Computation IP: Wrapper model

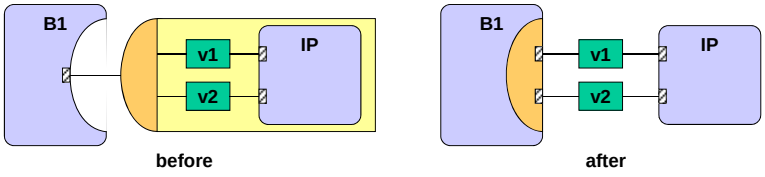


Intellectual Property (IP)

- Computation IP: Wrapper model

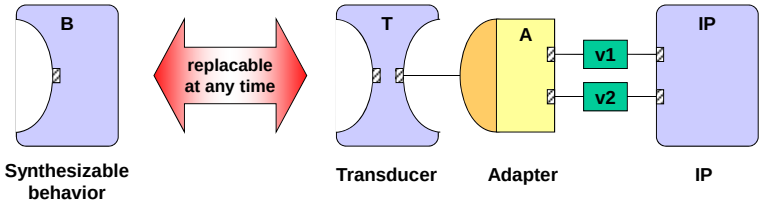


- Protocol inlining with wrapper



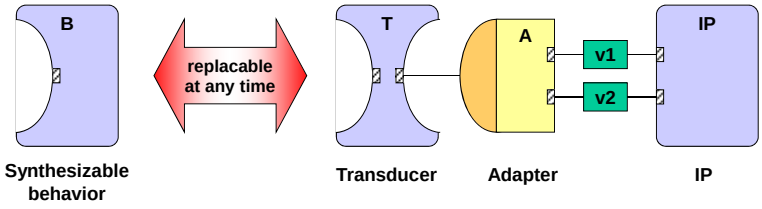
Intellectual Property (IP)

- Computation IP: Adapter model

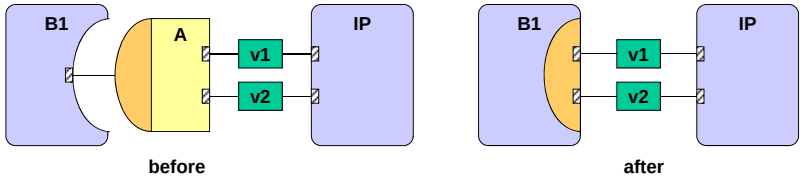


Intellectual Property (IP)

- Computation IP: Adapter model

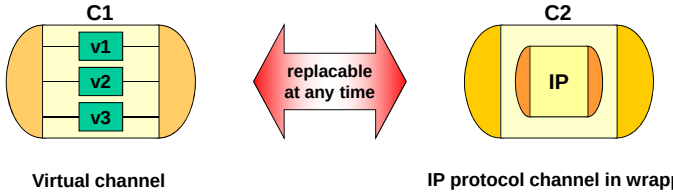


- Protocol inlining with adapter



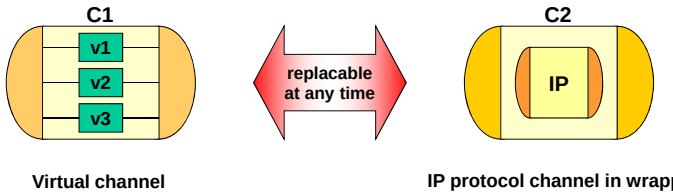
Intellectual Property (IP)

- Communication IP: Channel with wrapper

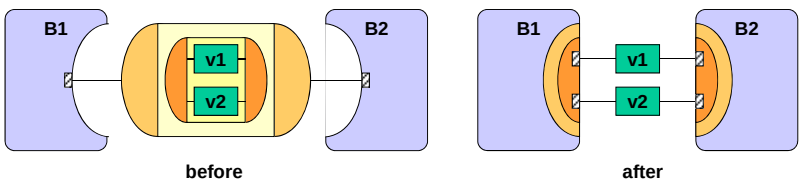


Intellectual Property (IP)

- Communication IP: Channel with wrapper

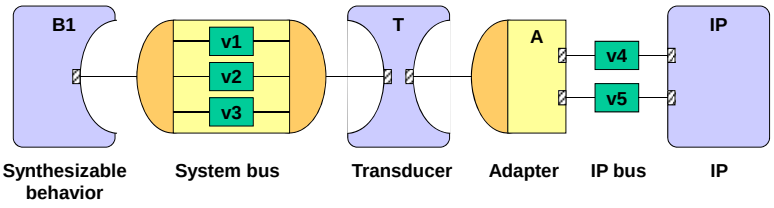


- Protocol inlining with hierarchical channel



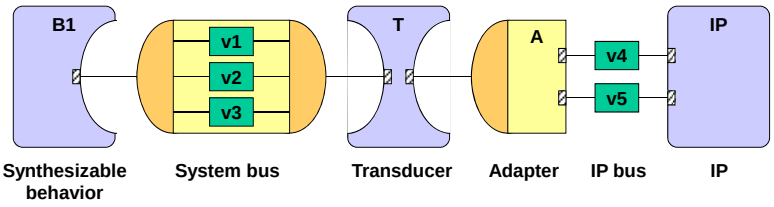
Intellectual Property (IP)

- Incompatible busses: Transducer insertion



Intellectual Property (IP)

- Incompatible busses: Transducer insertion



- Protocol inlining with transducer