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 administration
  - Overview
  - Contents
  - Schedule
  - Assignments
- Introduction to System-on-Chip design
  - Levels of abstraction
  - System design flow
  - Models of computation
  - System-level description languages
  - Computation, communication, IP

Course Administration

- Course web pages at **`http://eee.uci.edu/09f/18415/`**
  - Instructor information
  - Course description and policies
  - Objectives and outcomes
  - Contents and schedule
  - Resources and communication
  - Assignments

Introduction to SoC Design

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

System-on-Chip Design

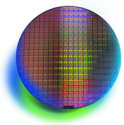
- Embedded systems are everywhere…





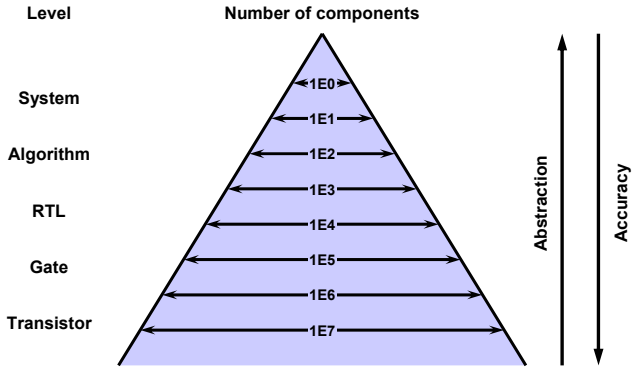


- Deep sub-micron technology enables System-on-Chip (SoC)



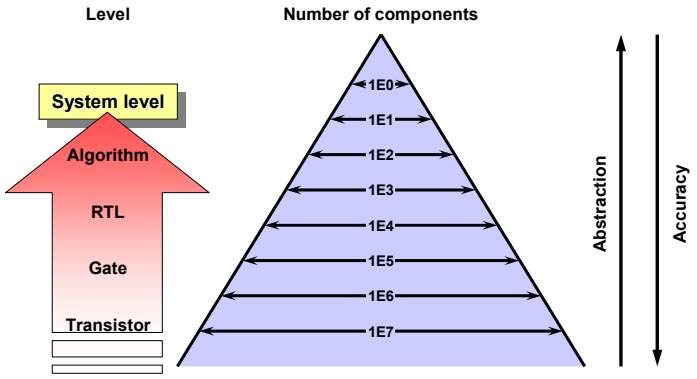
Abstraction Levels

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

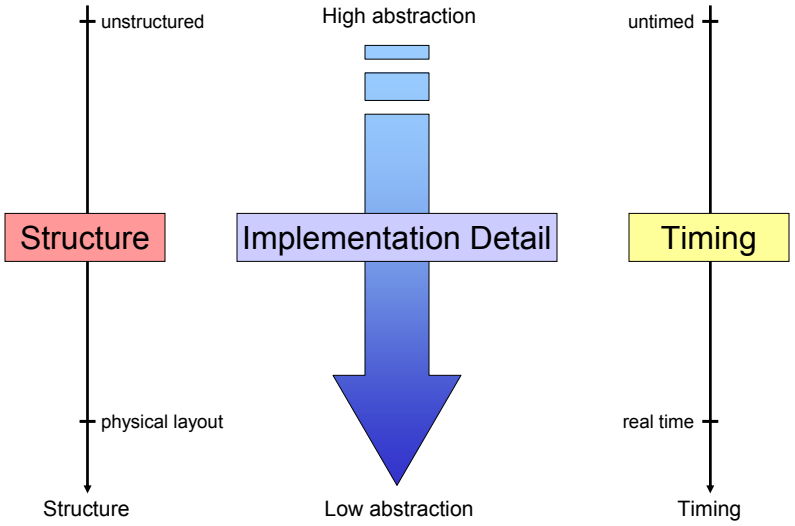


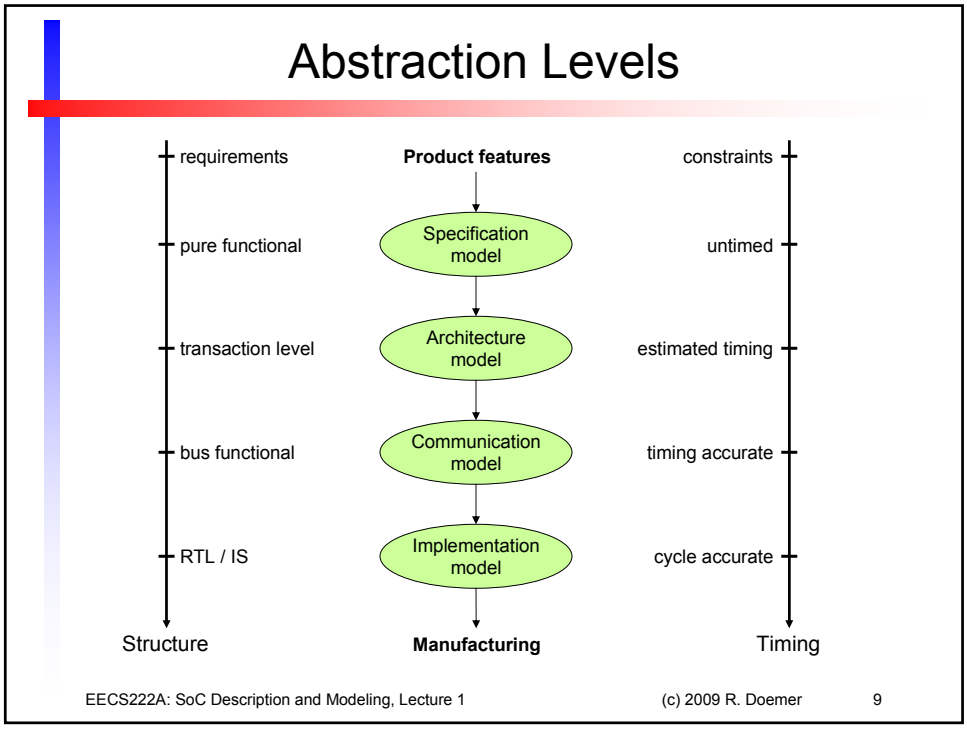
Abstraction Levels

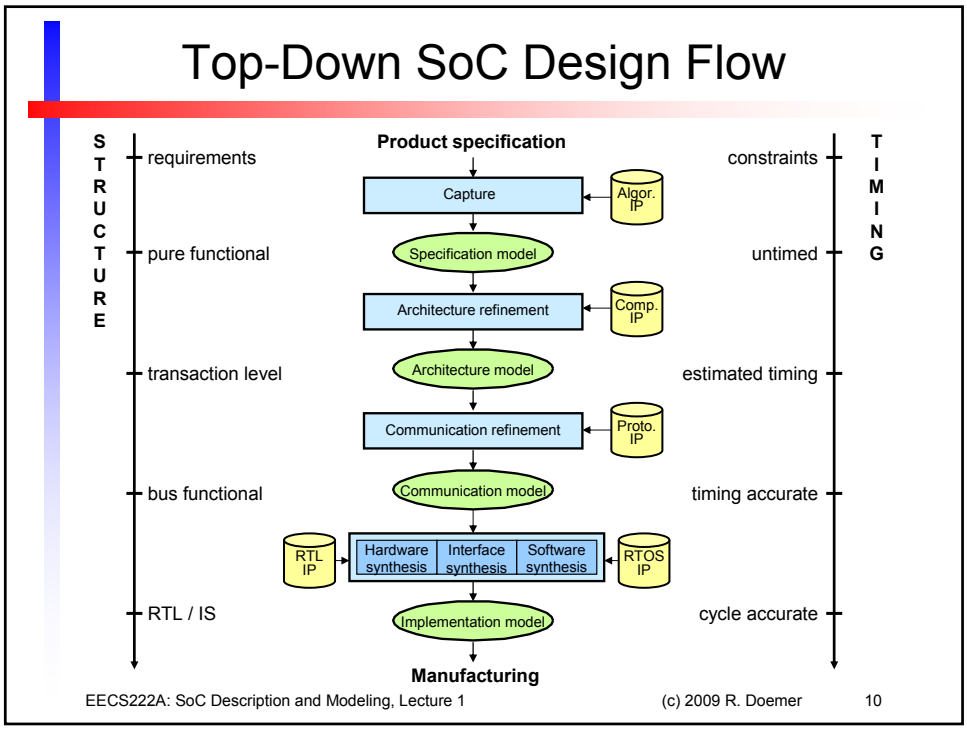
- System-on-Chip (SoC) design faces tremendous increase of design complexity
- Move to higher levels of abstraction!



Abstraction Levels




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) 2009 R. Doemer
9

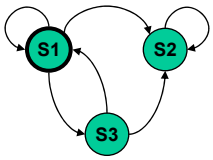

Top-Down SoC Design Flow
STRUCTURE
TIMING
requirements
Product specification
constraints
Capture
Algor. IP
pure functional
Specification model
untimed
Architecture refinement
Comp. IP
transaction level
Architecture model
estimated timing
Communication refinement
Proto. IP
bus functional
Communication model
timing accurate
RTL IP
Hardware synthesis
Interface synthesis
Software synthesis
RTOS IP
RTL / IS
Implementation model
cycle accurate
Manufacturing
EECS222A: SoC Description and Modeling, Lecture 1
(c) 2009 R. Doemer
10

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

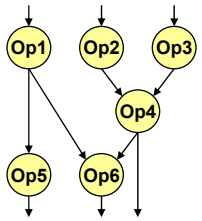
- Finite State Machine (FSM)
  - Basic model for describing control
  - States and state transitions
    - FSM = <*S*, *I*, *O*, *f*, *h*>
  - Two types:
    - Mealy-type FSM (input-based)
    - Moore-type FSM (state-based)



FSM model

Models of Computation

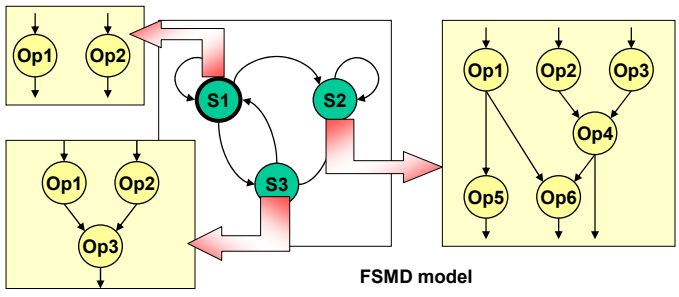
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
  - Basic model for describing computation
  - Directed graph
    - Nodes: operations
    - Arcs: dependency of operations



DFG model

Models of Computation

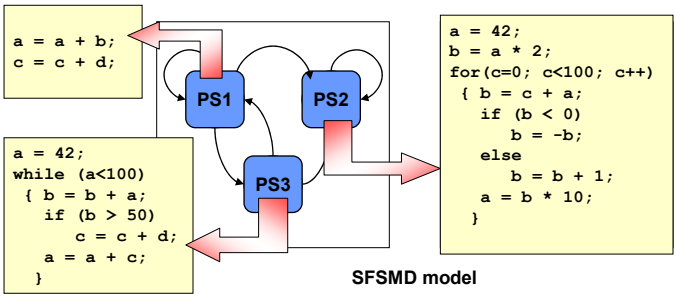
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
  - Combined model for control and computation
    - FSMD = FSM + DFG
  - Implementation: controller plus datapath



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

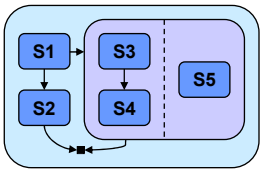


SFSMD model

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

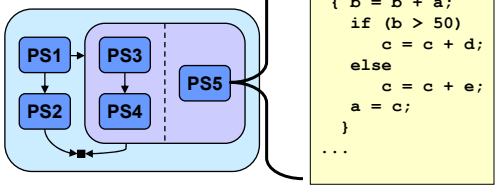
HCFSM model



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

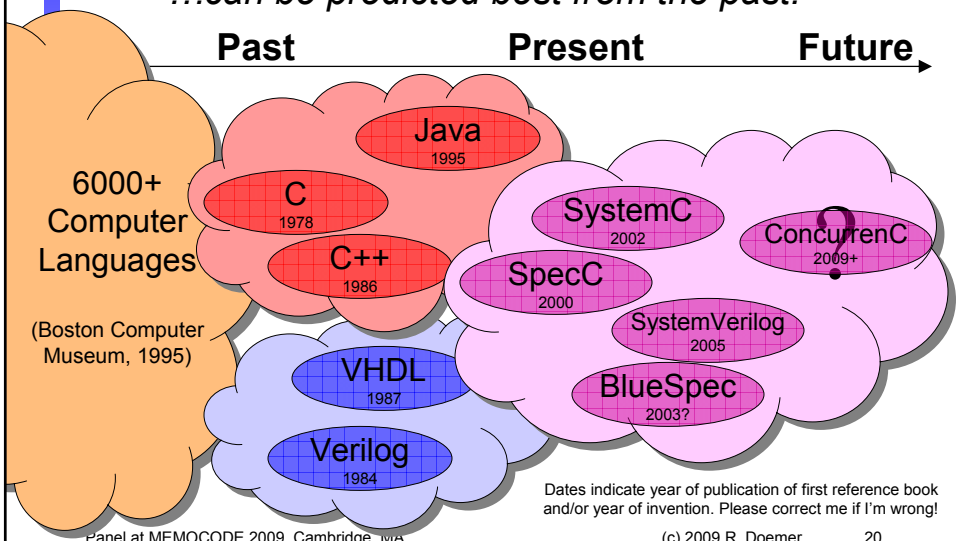
- Requirements supported by existing languages

	C	C++	Java	VHDL	Verilog	HardwareC	Statecharts	SpecCharts	SpecC
Behavioral hierarchy	○	○	○	○	○	○	●	●	●
Structural hierarchy	○	○	○	●	●	●	○	○	●
Concurrency	○	○	◐	●	●	●	●	●	●
Synchronization	○	○	◐	●	●	●	●	●	●
Exception handling	◐	●	●	○	●	○	◐	●	●
Timing	○	○	○	●	●	◐	◐	◐	●
State transitions	○	○	○	○	○	○	●	●	●
Composite data types	●	●	●	●	◐	○	○	●	●

○ not supported ◐ partially supported ● supported

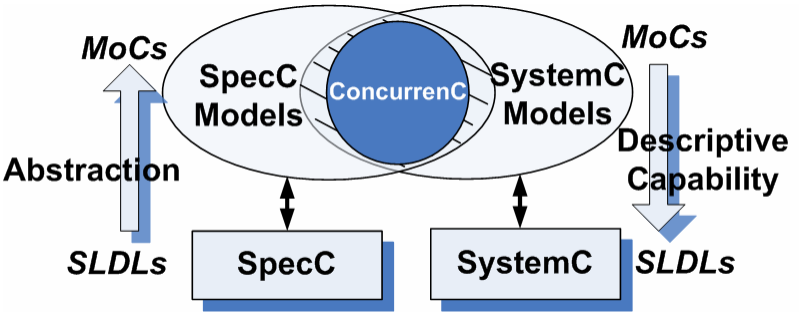
The Future of Design Languages

…can be predicted best from the past!



The Future of Design Languages

- ConcurrenC
  - Formal Model of Computation (MoC)
  - Representable in C-based SLDLS
    - SystemC
    - SpecC



Panel at MEMOCODE 2009, Cambridge, MA (c) 2009 R. Doemer 21

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.

EECS222A: SoC Description and Modeling, Lecture 1 (c) 2009 R. Doemer 22

System-Level Description Languages

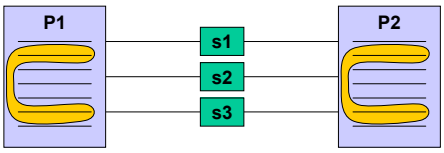
- Examples in use today and coverage in this course
  - C/C++
    - ANSI standard programming languages, software design
    - traditionally used for system design because of practicality, availability
  - ➢ SystemC
    - C++ API and library
    - initially developed at UCI, supported by Open SystemC 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

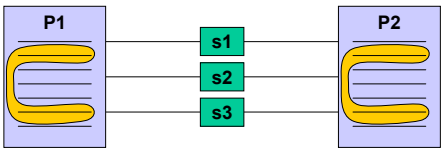
- Traditional model



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

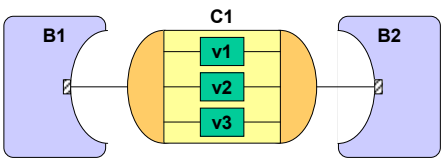
Computation vs. Communication

- Traditional model



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

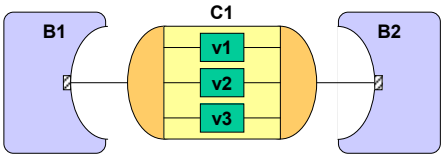
- SpecC model



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

Computation vs. Communication

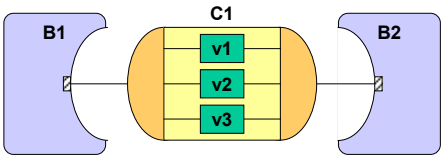
- Protocol Inlining
  - Specification model
  - Exploration model



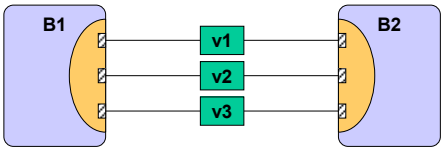
  - Computation in behaviors
  - Communication in channels

Computation vs. Communication

- Protocol Inlining
  - Specification model
  - Exploration model



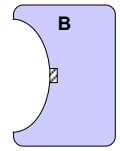
  - Computation in behaviors
  - Communication in channels
  - Implementation model



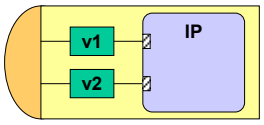
  - Channel disappears
  - Communication inlined into behaviors
  - Wires exposed

Intellectual Property (IP)

- Computation IP: Wrapper model



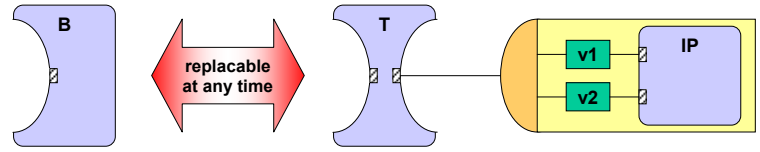
Synthesizable behavior



IP in wrapper

Intellectual Property (IP)

- Computation IP: Wrapper model



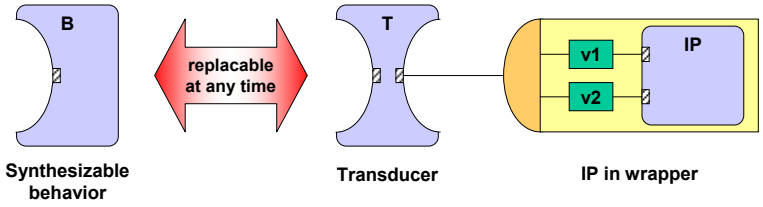
Synthesizable behavior

Transducer

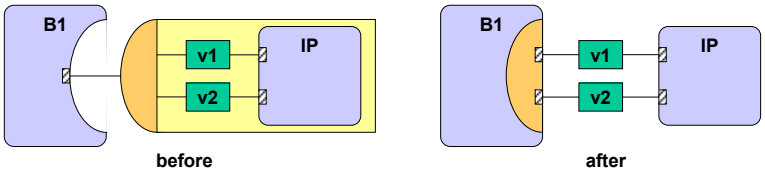
IP in wrapper

Intellectual Property (IP)

- Computation IP: Wrapper model

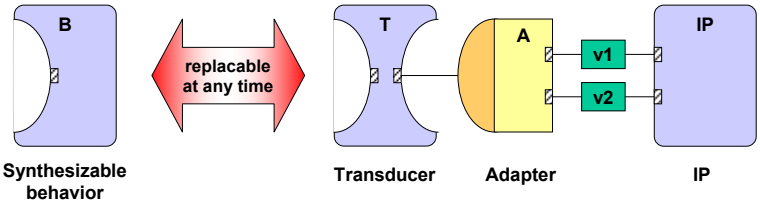


- Protocol inlining with wrapper



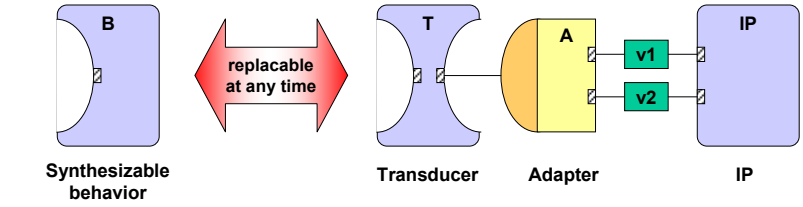
Intellectual Property (IP)

- Computation IP: Adapter model

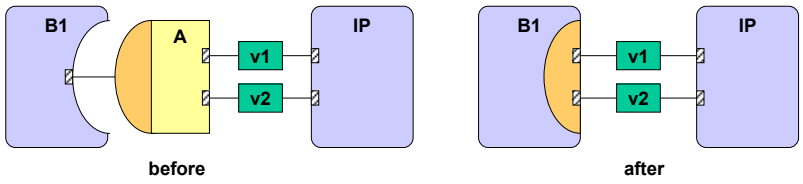


Intellectual Property (IP)

- Computation IP: Adapter model

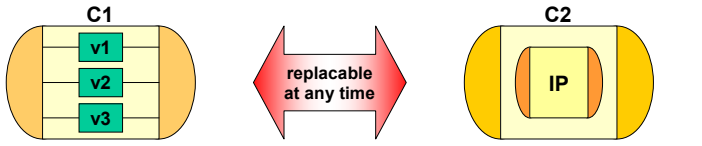


- Protocol inlining with adapter



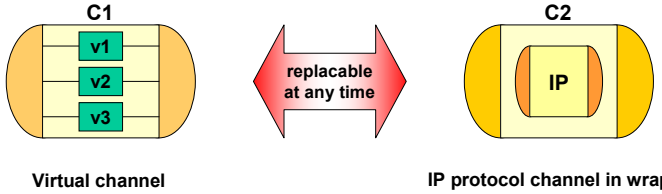
Intellectual Property (IP)

- Communication IP: Channel with wrapper

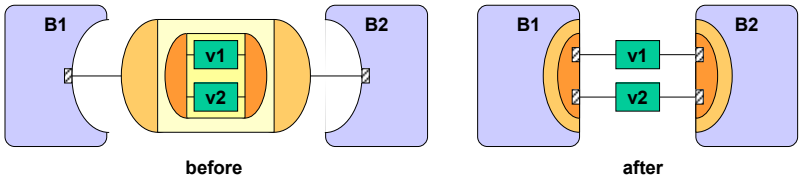


Intellectual Property (IP)

- Communication IP: Channel with wrapper

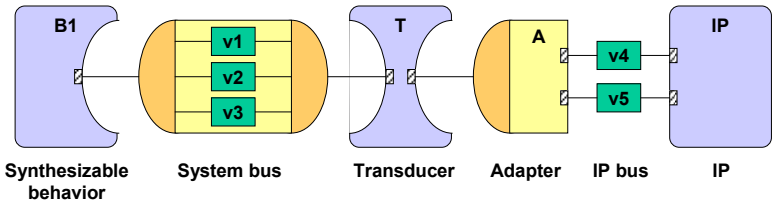


- Protocol inlining with hierarchical channel



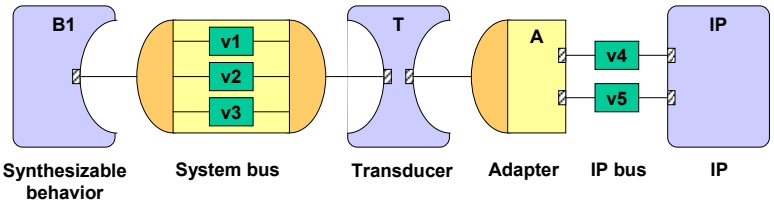
Intellectual Property (IP)

- Incompatible busses: Transducer insertion



Intellectual Property (IP)

- Incompatible busses: Transducer insertion



- Protocol inlining with transducer