EECS 222: Embedded System Modeling Lecture 2

Rainer Dömer

doemer@uci.edu

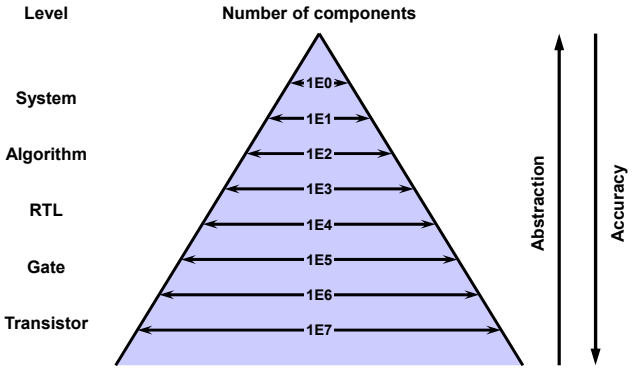
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 2: Overview

- Introduction to Embedded System Design
  - Abstraction Levels
  - Top-down system design flow
- Models of Computation
- System-Level Description Languages
- Separation of Concerns
  - Computation vs. Communication
  - Intellectual Property (IP)

Abstraction Levels

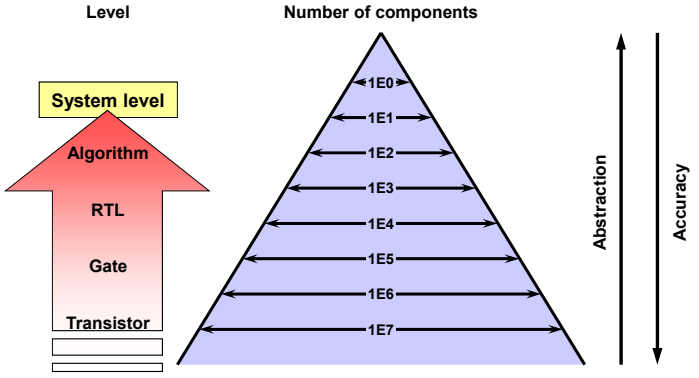
- System-on-Chip (SoC) design faces tremendous increase of design complexity



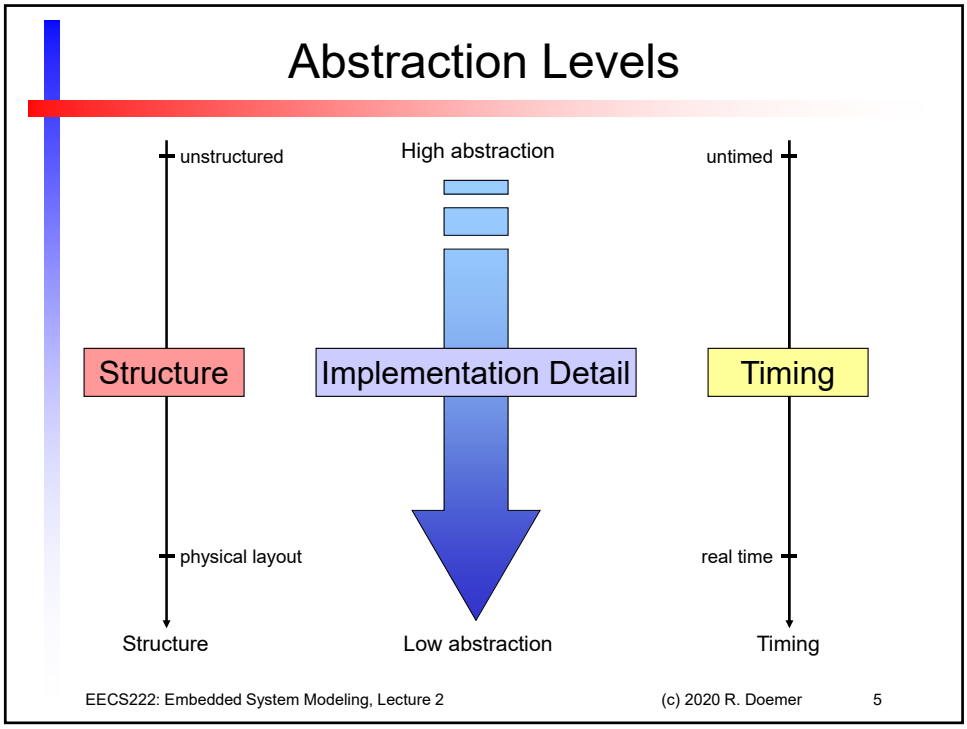
EECS222: Embedded System Modeling, Lecture 2 (c) 2020 R. Doemer 3

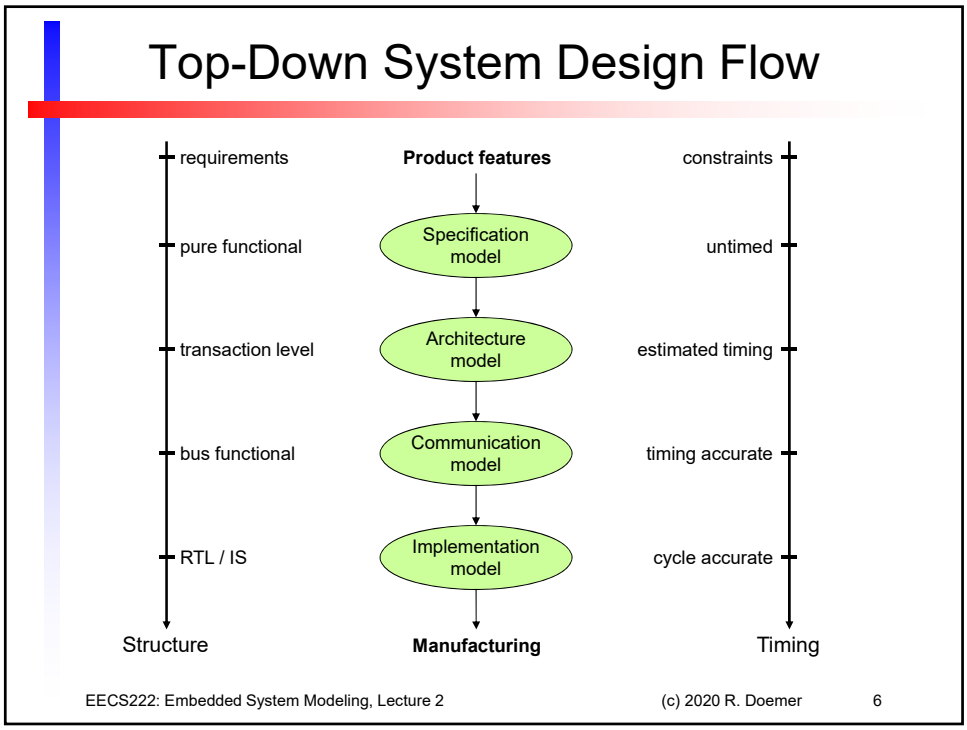
Abstraction Levels

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



EECS222: Embedded System Modeling, Lecture 2 (c) 2020 R. Doemer 4


Abstraction Levels
unstructured
High abstraction
untimed
Structure
Implementation Detail
Timing
physical layout
real time
Structure
Low abstraction
Timing
EECS222: Embedded System Modeling, Lecture 2
(c) 2020 R. Doemer
5

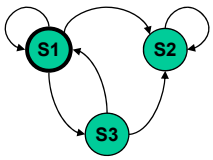

Top-Down System Design Flow
requirements
Product features
constraints
pure functional
Specification model
untimed
transaction level
Architecture model
estimated timing
bus functional
Communication model
timing accurate
RTL / IS
Implementation model
cycle accurate
Structure
Manufacturing
Timing
EECS222: Embedded System Modeling, Lecture 2
(c) 2020 R. Doemer
6

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

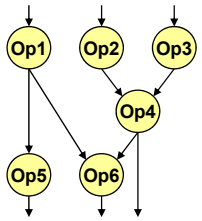
- Finite State Machine (FSM)
  - Basic model for describing control
  - States and state transitions
    - FSM = <*S*, *I*, *O*, *f*, *h*>
  - Two types:
    - Mealy-type FSM (input-based)
    - Moore-type FSM (state-based)



FSM model

Models of Computation

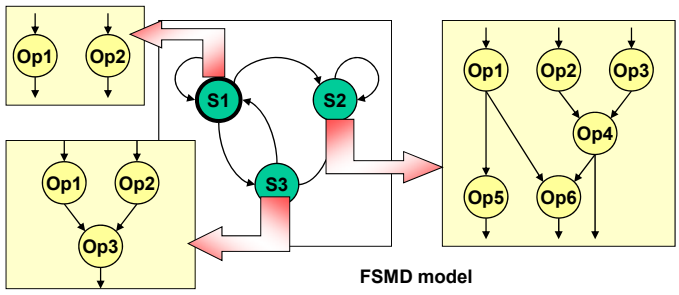
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
  - Basic model for describing computation
  - Directed graph (acyclic)
    - Nodes: operations
    - Edges: data flow, dependency of operations



DFG model

Models of Computation

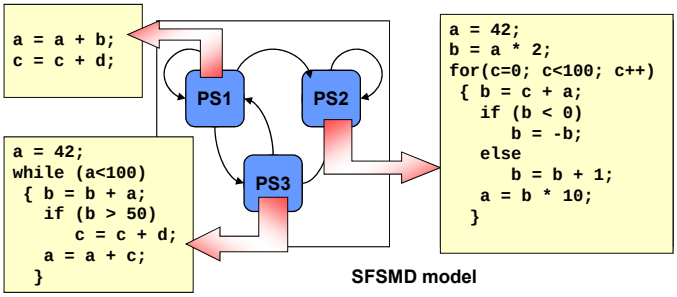
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
  - Combined model for control and computation
    - FSMD = FSM + DFG
  - Implementation: controller plus data path (RTL processor)



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

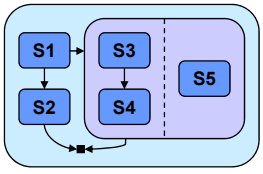


SFSMD model

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

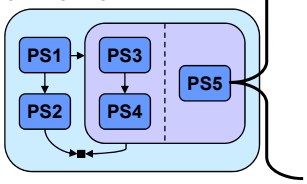
HCFSM model



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;
 }
...
```


Models of Computation

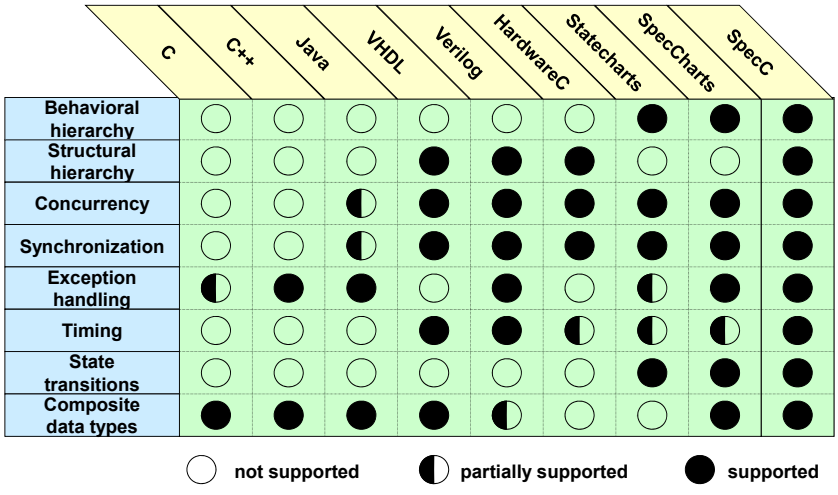
- State-based Models
  - FSM, DFG, FSMD, SFSMD, HCFSM, PSM
  - Petri Nets
  - …
- Process-based Models
  - Processes and threads
  - Kahn Process Network (KPN)
  - Synchronous Data Flow (SDF)
  - …
- Imperative Programming Models
  - C/C++, …

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



System-Level Description Languages

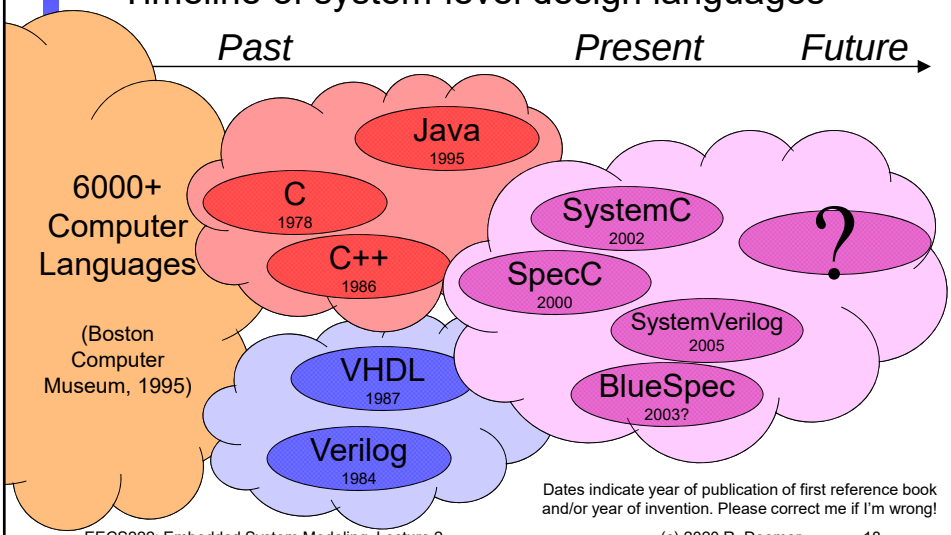
- Requirements supported by existing languages

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

○ not supported ◐ partially supported ● supported

Evolution of Design Languages

- Timeline of system-level design languages



Dates indicate year of publication of first reference book and/or year of invention. Please correct me if I'm wrong!

System-Level Description Languages

- Examples of Languages in Use Today
  - C/C++
    - ANSI standard programming languages, software design
    - Initially used for system design because of availability, practicality
  - SystemC
    - IEEE standard 1666-2011 (initially created at UCI, standardized by OSCI)
    - C++ library and application programming interface (API)
  - SpecC
    - SLDL with compiler, based on the ANSI C language standard
    - Designed and built at UCI, promoted by SpecC Technology Open Consortium
  - Matlab
    - Algorithm design, specification and simulation in engineering
  - UML
    - Unified Modeling Language, graphical software specification and engineering
  - SystemVerilog
    - Verilog with C extensions
  - SDL
    - Telecommunication standard by ITU, used in COSMOS

System-Level Description Languages

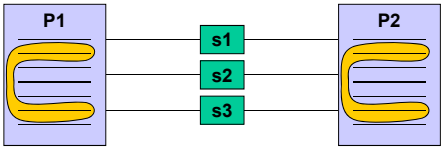
- Examples of Languages in Use Today, Course Coverage
  - C/C++
    - ANSI standard programming languages, software design
    - Initially used for system design because of availability, practicality
  - ➢ SystemC
    - IEEE standard 1666-2011 (initially created at UCI, standardized by OSCI)
    - C++ library and application programming interface (API)
  - ➢ SpecC
    - SLDL with compiler, based on the ANSI C language standard
    - Designed and built at UCI, promoted by SpecC Technology Open Consortium
  - Matlab
    - Algorithm design, specification and simulation in engineering
  - ➢ UML
    - Unified Modeling Language, graphical software specification and engineering
  - SystemVerilog
    - Verilog with C extensions
  - SDL
    - Telecommunication standard by ITU, used in COSMOS

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



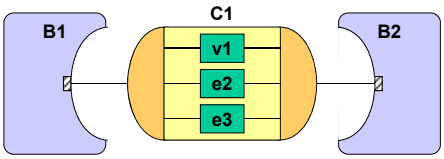
  - VHDL example:

```
entity P1 […] process […]
s1 <= '1';
s2 <= '1';
wait until s3'event and s3 = '1';
s2 <= '0';
xy = x + 2 * y;
s1 <= xy;
s2 <= '1';
wait until s3'event and s3 = '1';
s1 <= '0';
s2 <= '0';
[…]
```

  - Mixture of computation and communication
  - ➢ Automatic replacement impossible!

Computation vs. Communication

- SpecC model
  - Behaviors and channels



  - SpecC example:

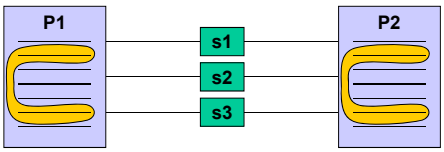
```
behavior B1 […]
{
  c.send(1);

  xy = x + 2 * y;

  c.send(xy);

  v1 = 0;
  […]
}
```

```
channel C1 […]
{ send (int d)
  { v1 = d;
    notify e2;
    wait e3;
  }
  […]
}
```

  - Separation of computation and communication
  - ➢ Plug-and-play!

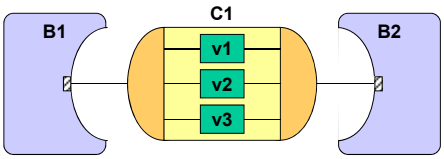
Computation vs. Communication

- Traditional model



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

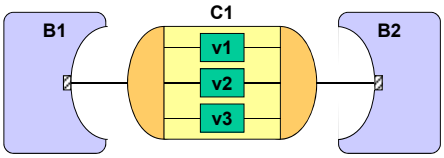
- SpecC model



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

Computation vs. Communication

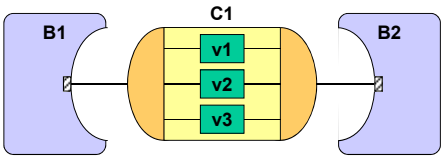
- Protocol Inlining
  - Specification model
  - Exploration model



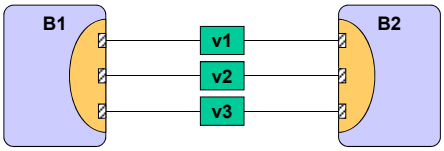
  - Computation in behaviors
  - Communication in channels

Computation vs. Communication

- Protocol Inlining
  - Specification model
  - Exploration model



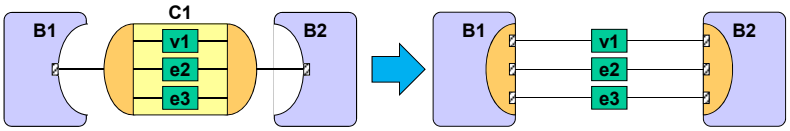
  - Computation in behaviors
  - Communication in channels
  - Implementation model



  - Channel disappears
  - Communication inlined into behaviors
  - Wires exposed

Computation vs. Communication

- Protocol Inlining



– SpecC example:

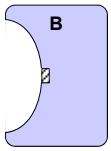
```
behavior B1 […]
{
  c.send(1);

  xy = x + 2 * y;

  c.send(xy);

  v1 = 0;
  […]
}
```

```
channel C1 […]
{ send (int d)
  { v1 = d;
    notify e2;
    wait e3;
  }
  […]
}
```

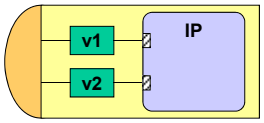
```
behavior B1 […]
{ v1 = 1;
  notify e2;
  wait e3;
  xy = x + 2 * y;
  v1 = xy;
  notify e2;
  wait e3;
  v1 = 0;
  […]
}
```


Intellectual Property (IP)

- Computation IP: Wrapper model



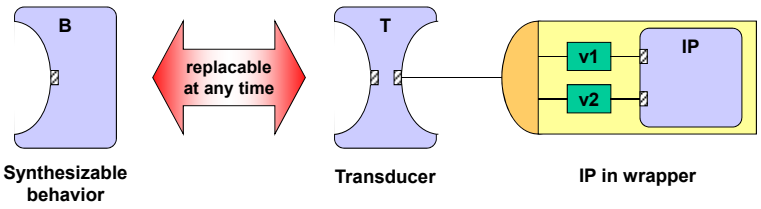
Synthesizable behavior



IP in wrapper

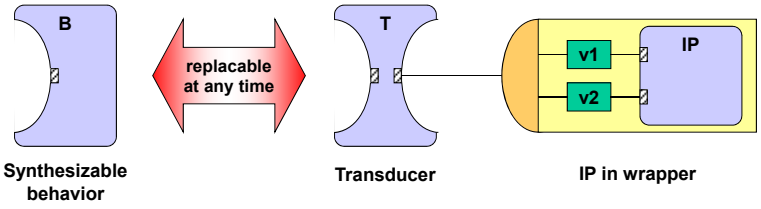
Intellectual Property (IP)

- Computation IP: Wrapper model

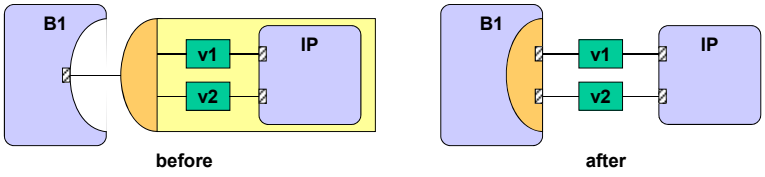


Intellectual Property (IP)

- Computation IP: Wrapper model

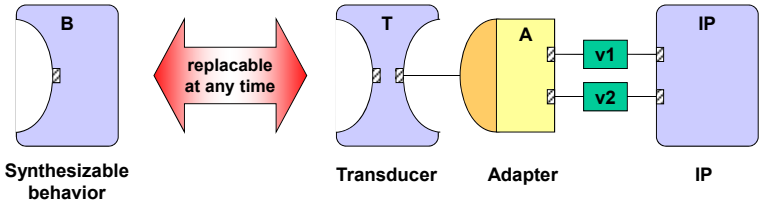


- Protocol inlining with wrapper



Intellectual Property (IP)

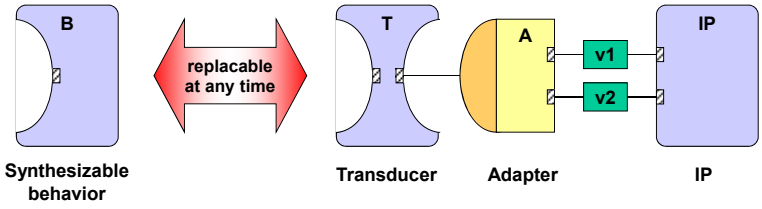
- Computation IP: Adapter model



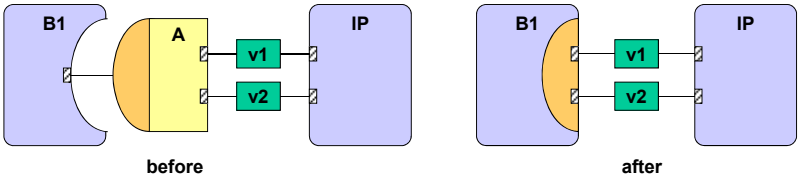
EECS222: Embedded System Modeling, Lecture 2 (c) 2020 R. Doemer 31

Intellectual Property (IP)

- Computation IP: Adapter model



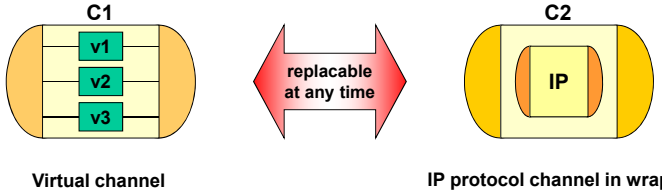
- Protocol inlining with adapter



EECS222: Embedded System Modeling, Lecture 2 (c) 2020 R. Doemer 32

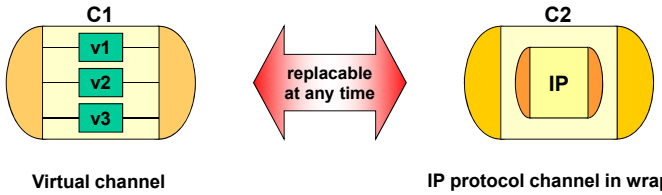
Intellectual Property (IP)

- Communication IP: Channel with wrapper

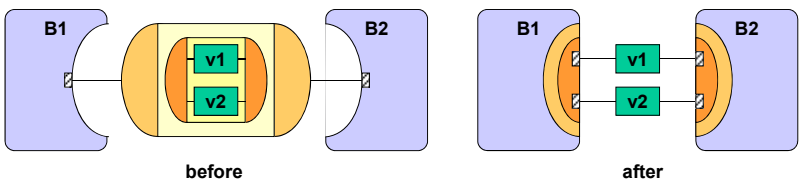


Intellectual Property (IP)

- Communication IP: Channel with wrapper

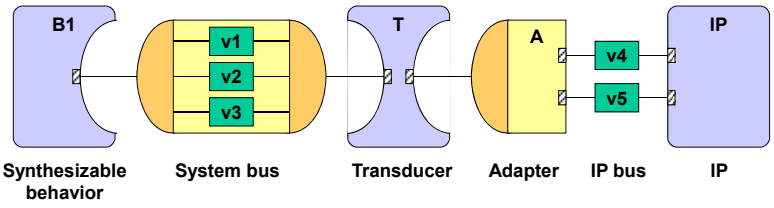


- Protocol inlining with hierarchical channel



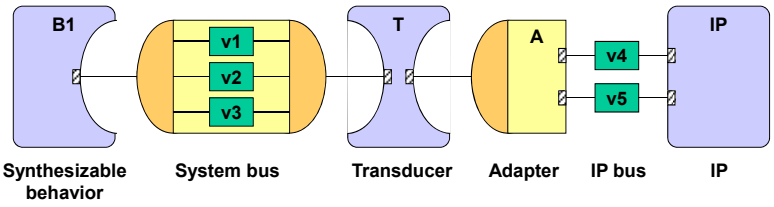
Intellectual Property (IP)

- Incompatible busses: Transducer insertion



Intellectual Property (IP)

- Incompatible busses: Transducer insertion



- Protocol inlining with transducer