ECE 298: System-on-Chip Description and Modeling Lecture 10

Rainer Dömer

doemer@uci.edu

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

Lecture 10: Overview

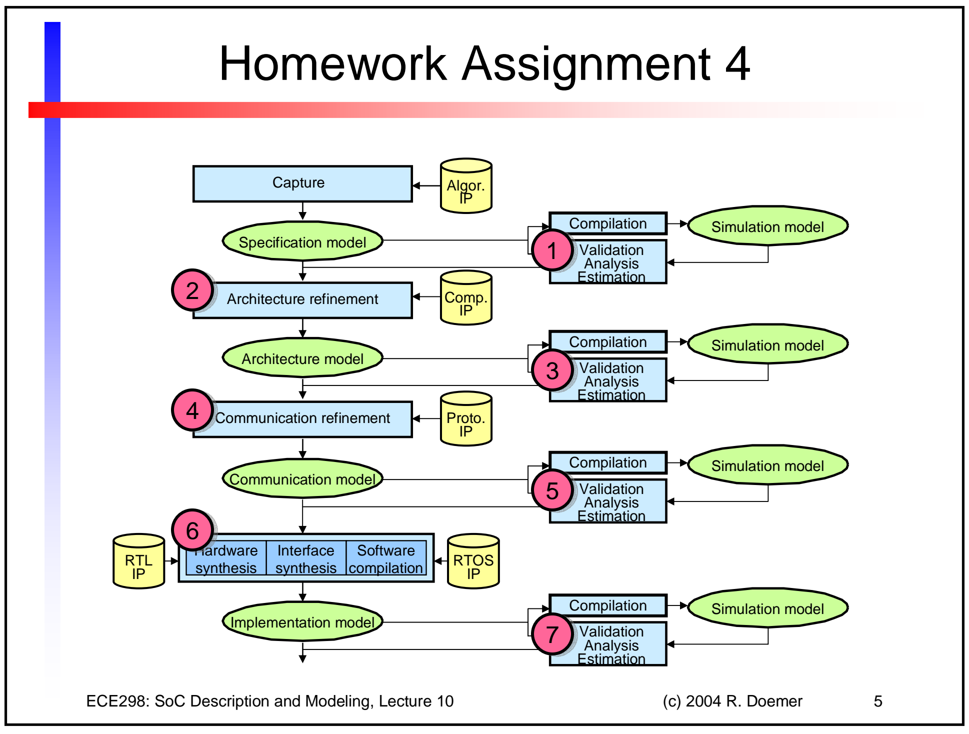
- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

Lecture 10: Overview

- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

Homework Assignment 4

- Tasks
  - Task 1: Choose design example:
    - Option 1: MPEG-Audio Decoder (from previous homework)
    - Option 2: GSM Vocoder (from SCE tutorial)
  - Task 2: Design Space Exploration with SCE
    - Use different number and type of PEs
      - SW: Motorola DSP56600, Motorola Coldfire, Toshiba TX49
      - HW: Standard custom HW block
      - Memories: Motorola Coldfire Memory, Samsung KM684002
    - Use different clock frequencies (power savings!)
    - Use different mappings (behaviors to PEs, variables to memories)
    - Use different scheduling schemes (dynamic vs. static)
    - Estimate the performance and cost for each option
  - Task 3: Design Refinement with SCE
    - Generate the “best” system architecture for the design
    - Use straightforward communication, HW, SW design
  - *Note/store all results for the final report!*



Homework Assignment 5

- Technical Report
  - Summarize the specification and implementation of a System-on-Chip design example
    - MPEG-Audio Decoder
    - GSM Vocoder (alternative)
  - Focus
    - Description and Modeling of Specification Model
    - Lessons learned in this class
  - Also
    - SoC Design Issues
    - Top-down Design flow
    - Design space exploration

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 6

Homework Assignment 5

- Technical Report, Skeleton of Contents

1. Title page
   - *Title, authors, date*
2. Introduction to SoC Design
   1. Challenges of SoC Design
      - *General SoC issues, problems and goals*
   2. SoC Specification
      - *Specification capture and modeling, goals*
   3. SoC Exploration
      - *Design flow, stages, exploration cycles*
3. Design Example
   1. Description of the design example
      - *Background, area, algorithm, features*
   2. Source
      - *Origin, C code, properties*
   3. Design considerations
      - *Design constraints, requirements, goals*
4. Specification capture
   1. Testbench
      - *Structure, conversion from C code, validation*
   2. Design Under Test
      - *Structure, hierarchy, parallelism, communication, properties, …*
5. ...

Homework Assignment 5

- Technical Report, Skeleton of Contents

5. Design space exploration
   1. System architecture
      - *Number and type of PEs, mapping*
   2. Scheduling
      - *Dynamic vs. static scheduling (if applicable)*
   3. Communication architecture
      - *Bus allocation, mapping (if applicable)*
6. Implementation
   1. Software implementation
      - *Code generation (if applicable)*
   2. Hardware implementation
      - *Behavioral and RTL synthesis (if applicable)*
   3. Cycle-accurate model
      - *Final model, properties*
7. Conclusion
   - *Concluding remarks, lessons learned*
8. References
   - *Applicable literature*

Lecture 10: Overview

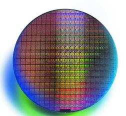
- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - ➢ Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 9

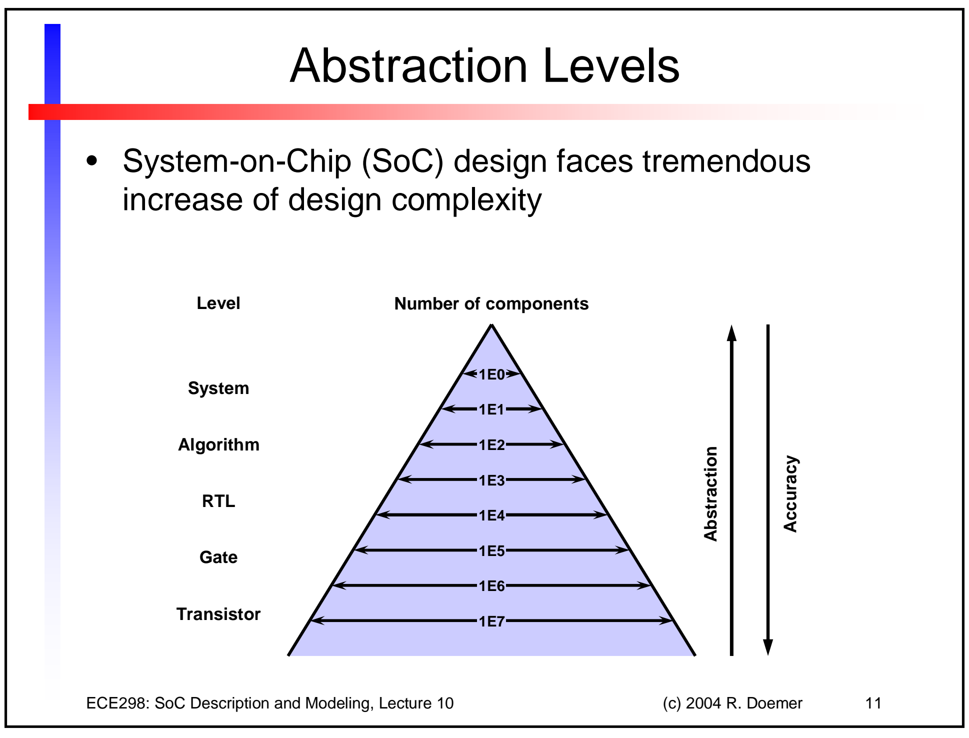
System-on-Chip Design

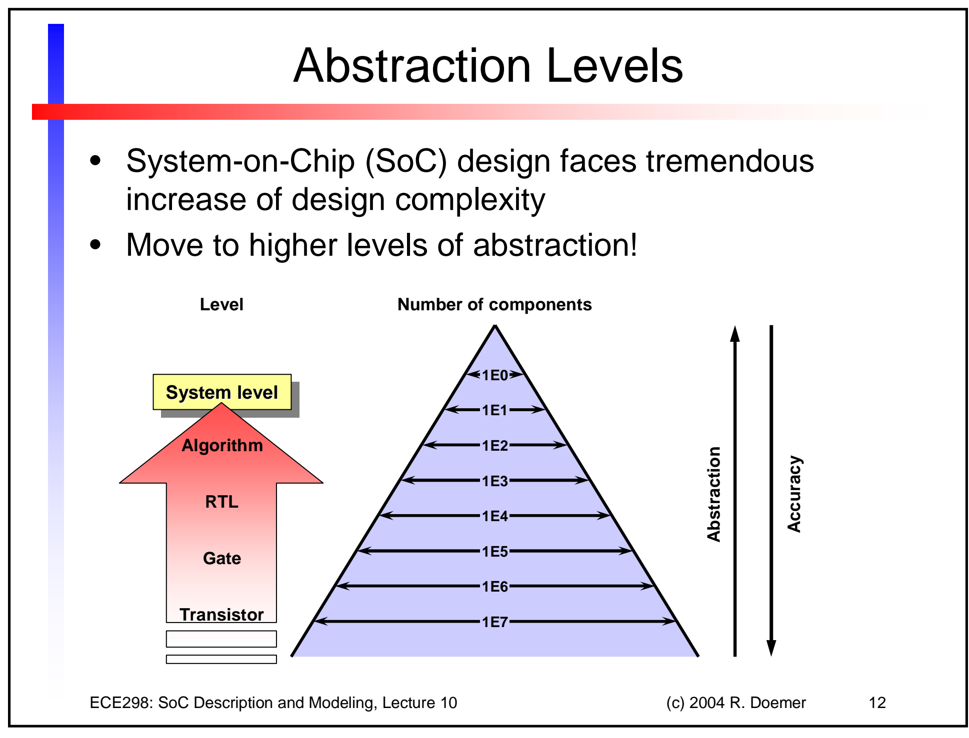
- Embedded systems are everywhere…

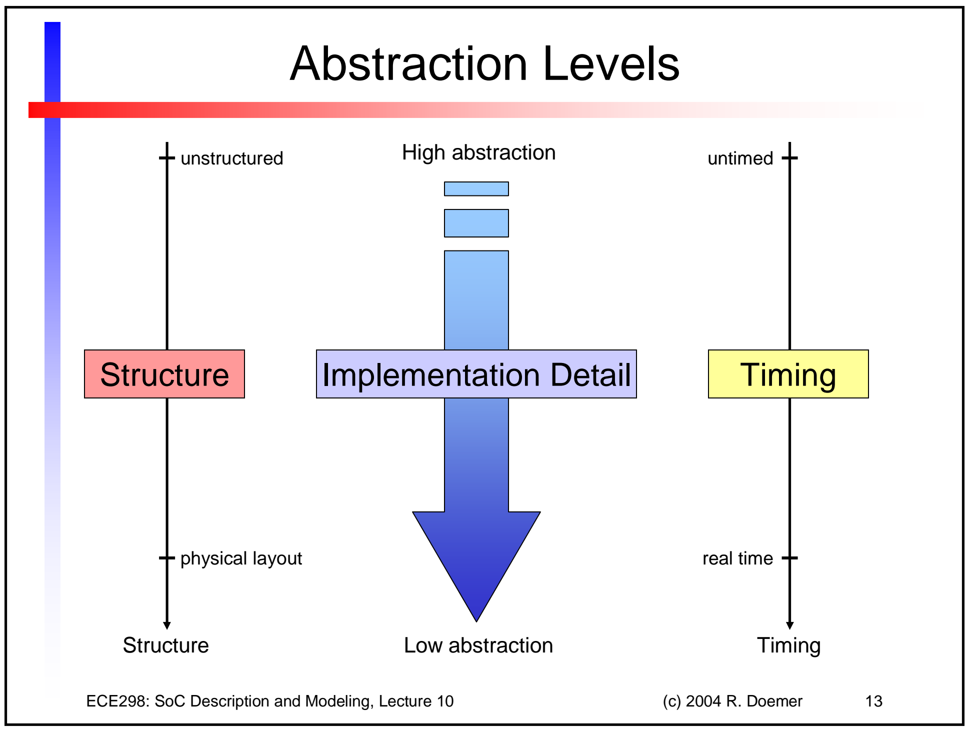
- Deep sub-micron design enables System-on-Chip (SoC)

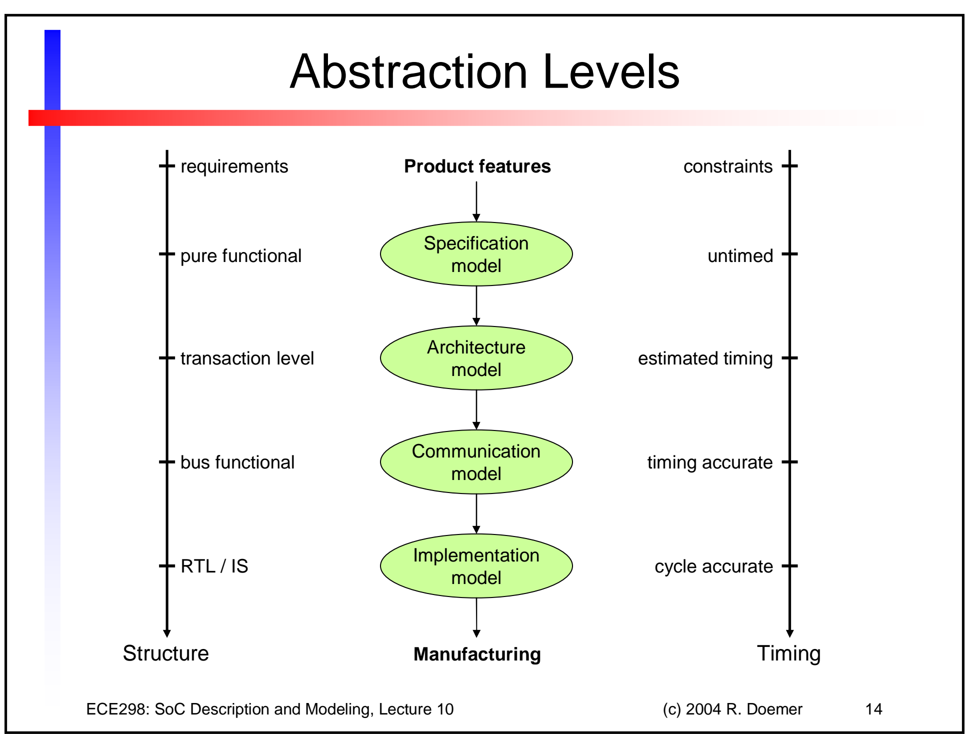


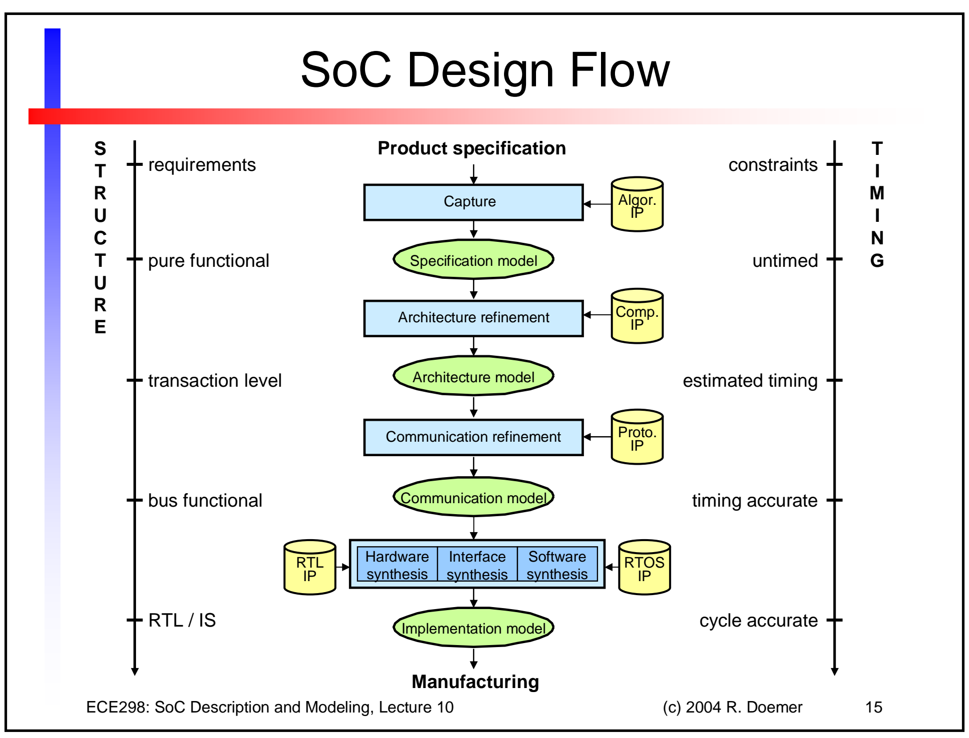
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 10

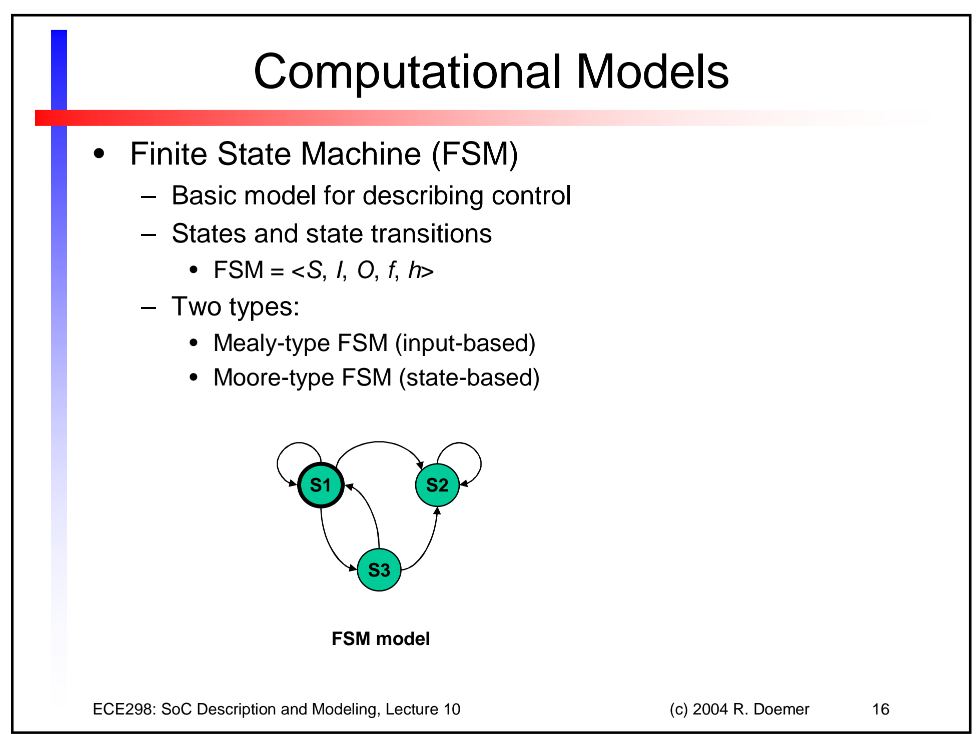

Abstraction Levels
• System-on-Chip (SoC) design faces tremendous increase of design complexity
Level
Number of components
System
Algorithm
RTL
Gate
Transistor
1E0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
Abstraction
Accuracy
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
11


Abstraction Levels
• System-on-Chip (SoC) design faces tremendous increase of design complexity
• Move to higher levels of abstraction!
Level
Number of components
System level
Algorithm
RTL
Gate
Transistor
1E0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
Abstraction
Accuracy
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
12


Abstraction Levels
unstructured
High abstraction
untimed
Structure
Implementation Detail
Timing
physical layout
real time
Structure
Low abstraction
Timing
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
13

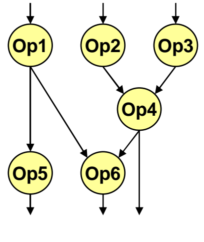

Abstraction Levels
requirements
Product features
constraints
pure functional
Specification model
untimed
transaction level
Architecture model
estimated timing
bus functional
Communication model
timing accurate
RTL / IS
Implementation model
cycle accurate
Structure
Manufacturing
Timing
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
14


SoC Design Flow
STRUCTURE
requirements
pure functional
transaction level
bus functional
RTL / IS
Product specification
Capture
Algor. IP
Specification model
Architecture refinement
Comp. IP
Architecture model
Communication refinement
Proto. IP
Communication model
RTL IP
Hardware synthesis
Interface synthesis
Software synthesis
RTOS IP
Implementation model
Manufacturing
TIMING
constraints
untimed
estimated timing
timing accurate
cycle accurate
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
15


Computational Models
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)
S1
S2
S3
FSM model
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
16

Computational Models

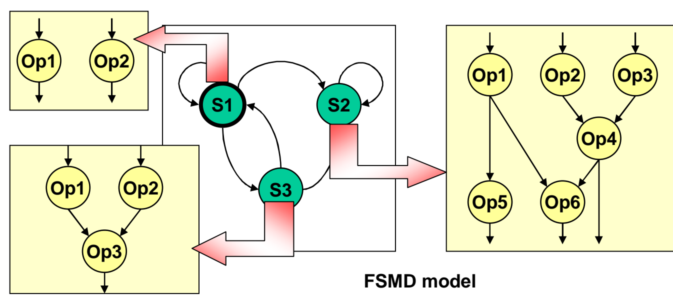
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
  - Basic model for describing computation
  - Directed graph
    - Nodes: operations
    - Arcs: dependency of operations



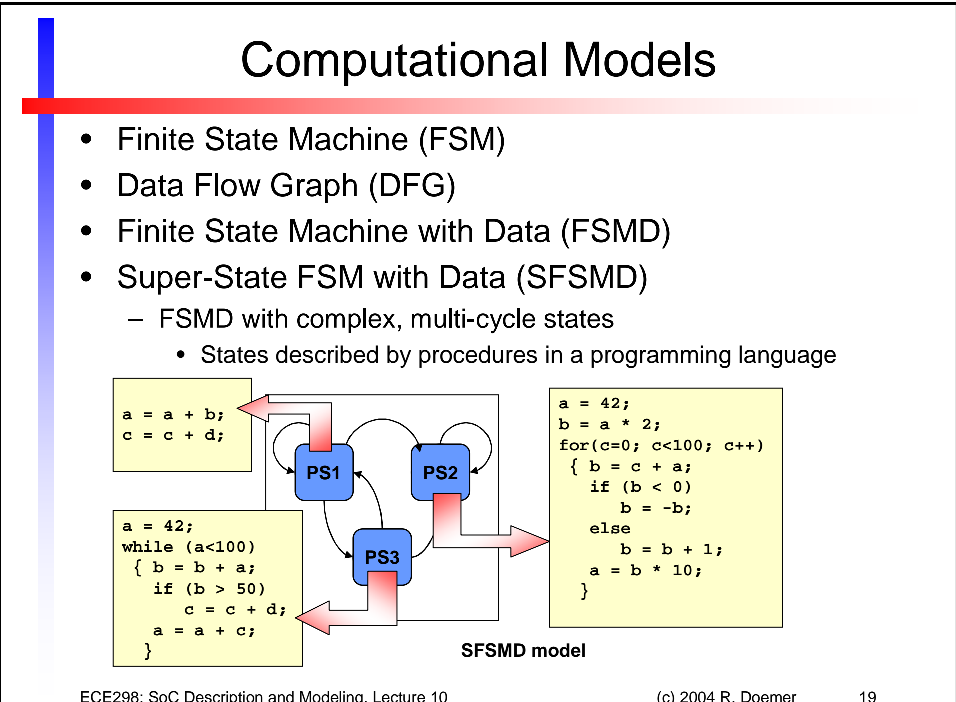
DFG model

Computational Models

- 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

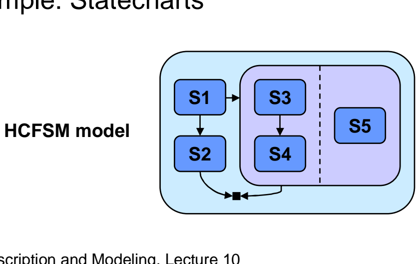


FSMD model

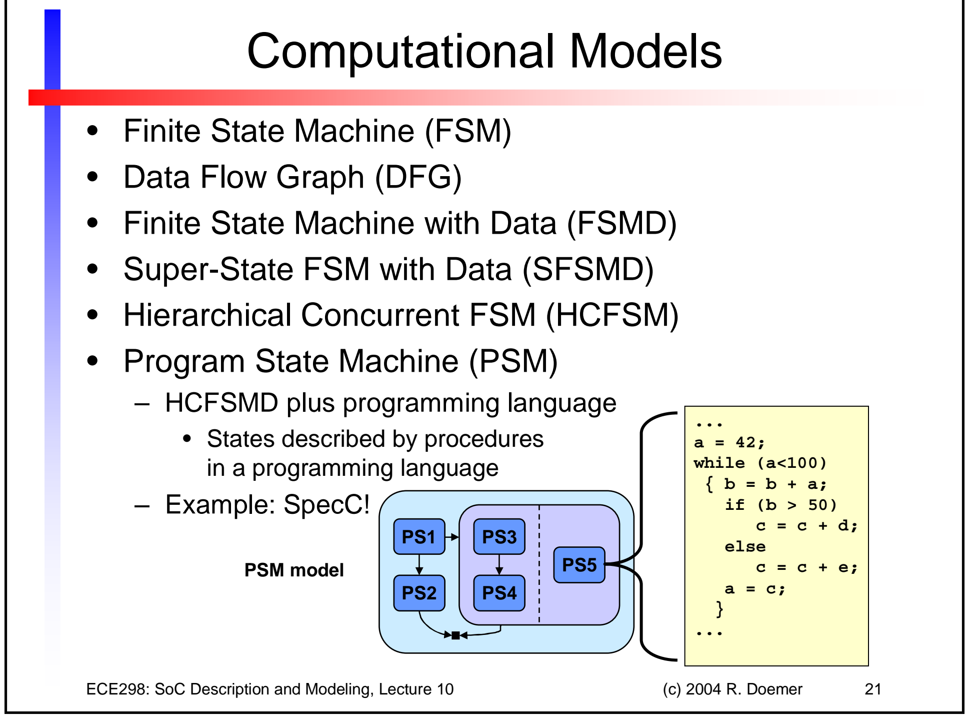


Computational Models

- 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

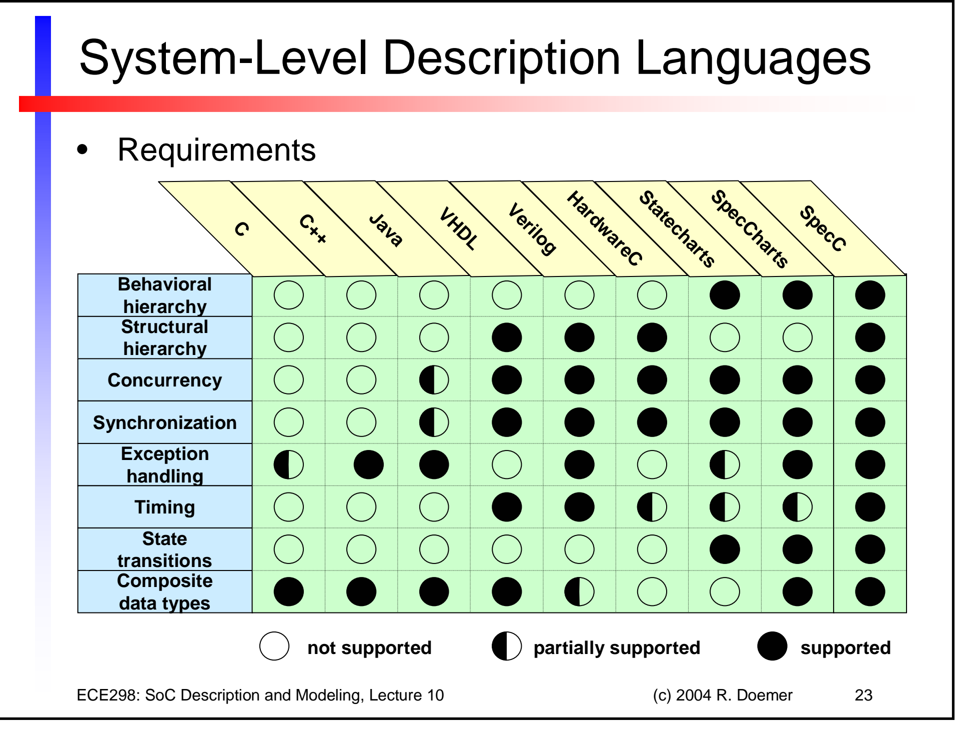


ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 20



System-Level Description Languages

- Goals
  - 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
    - Minimality
  - Simplicity



System-Level Description Languages

- Requirements

	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

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 23

System-Level Description Languages

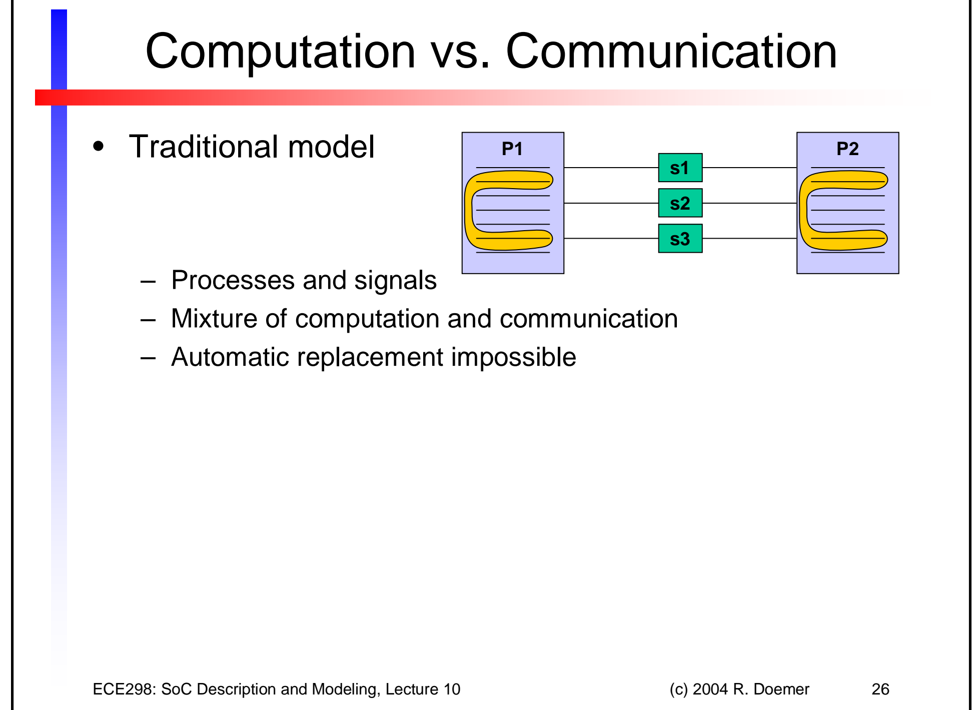
- Examples in use today
  - C/C++
    - ANSI standard programming languages, software design
    - traditionally used for system design because of practicality, availability
  - SystemC
    - C++ API and library
    - initially developed at UCI, supported by Open SystemC Initiative
  - SpecC
    - C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

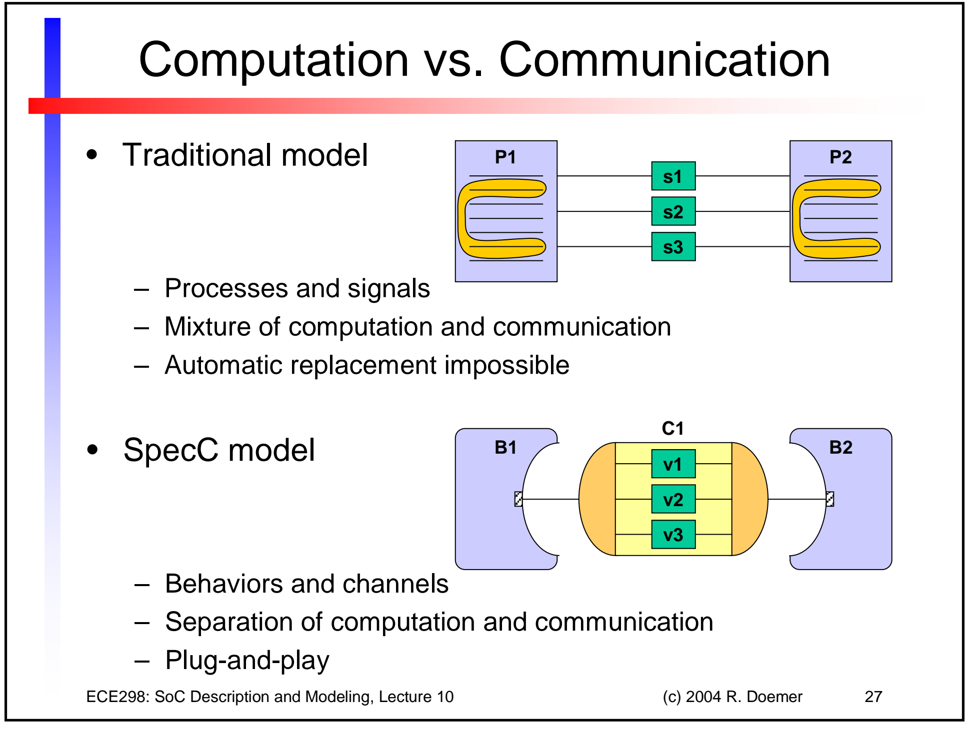
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 24

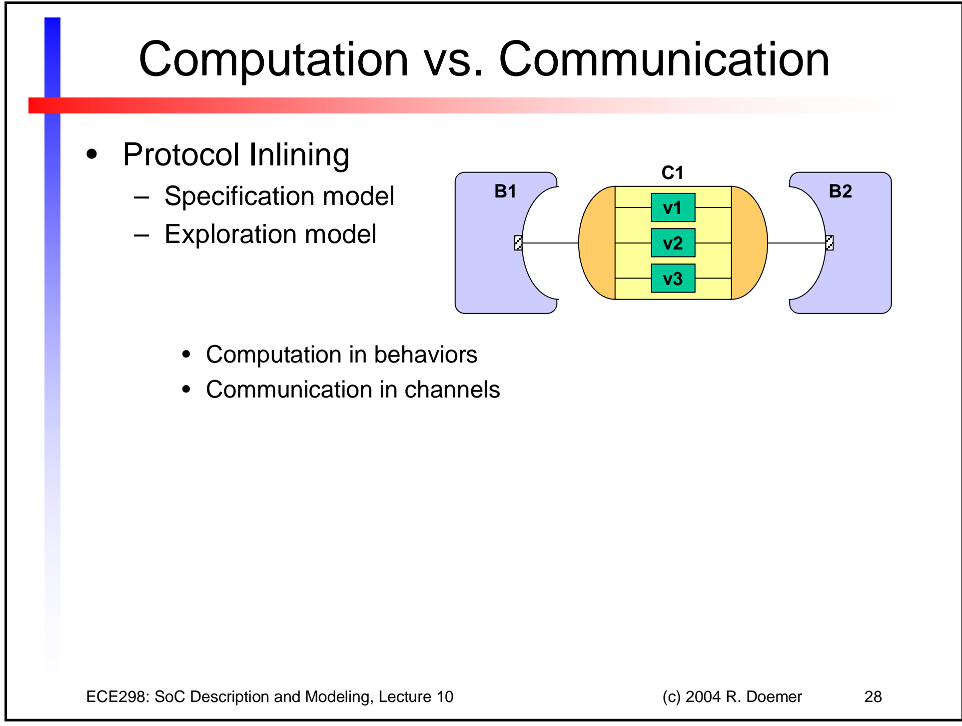
System-Level Description Languages

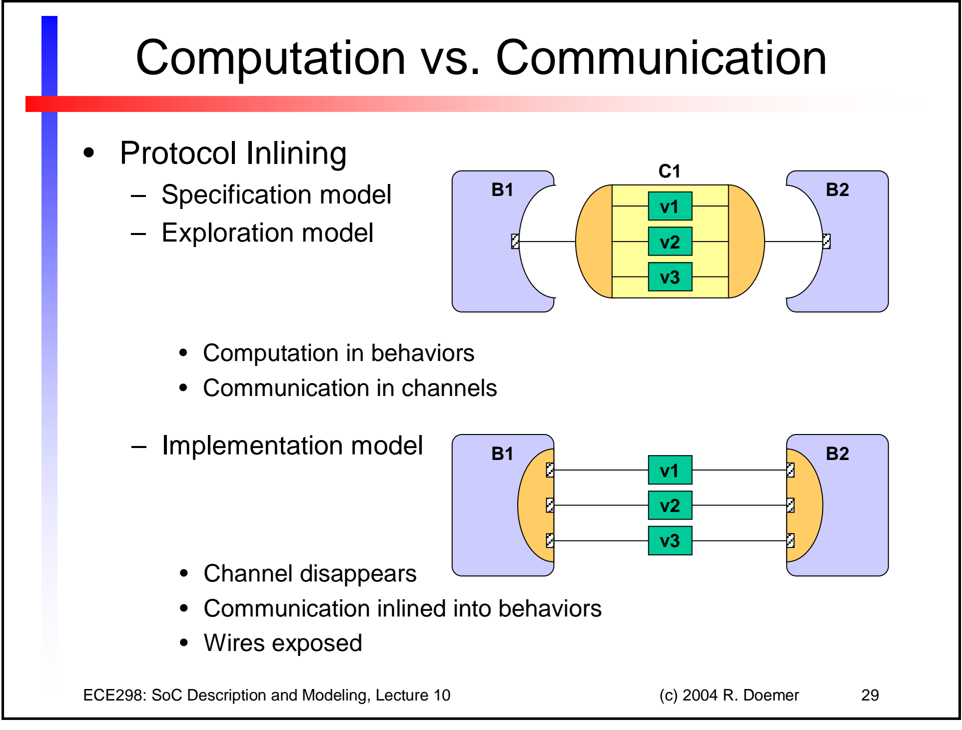
- Examples in use today
  - C/C++
    - ANSI standard programming languages, software design
    - traditionally used for system design because of practicality, availability
  - SystemC
    - C++ API and library
    - initially developed at UCI, supported by Open SystemC Initiative
  - SpecC
    - C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

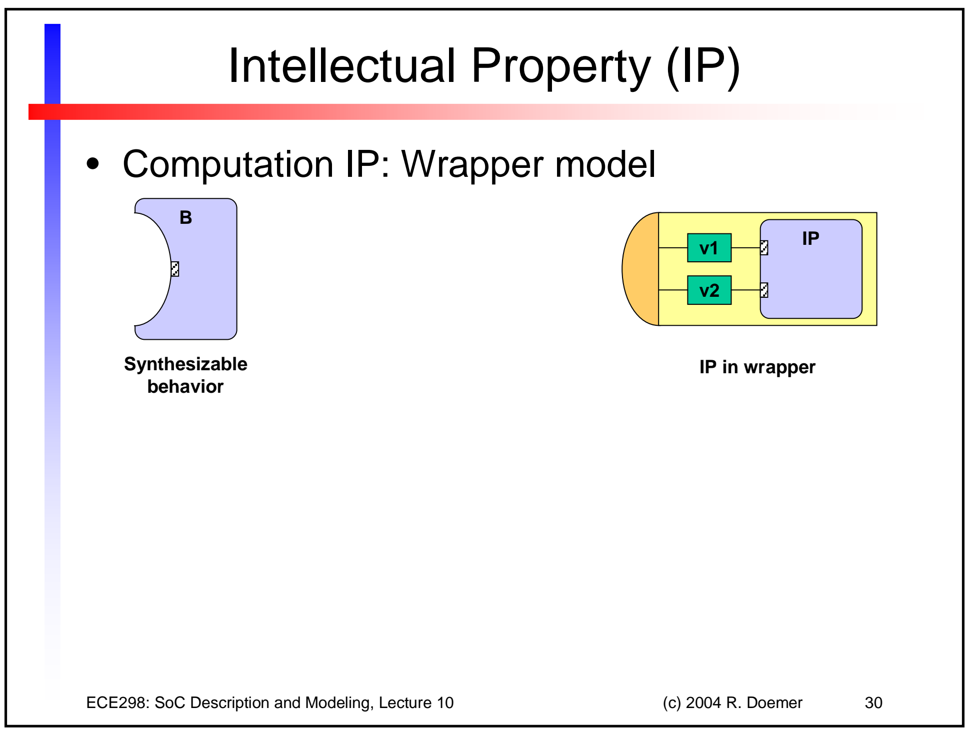
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 25

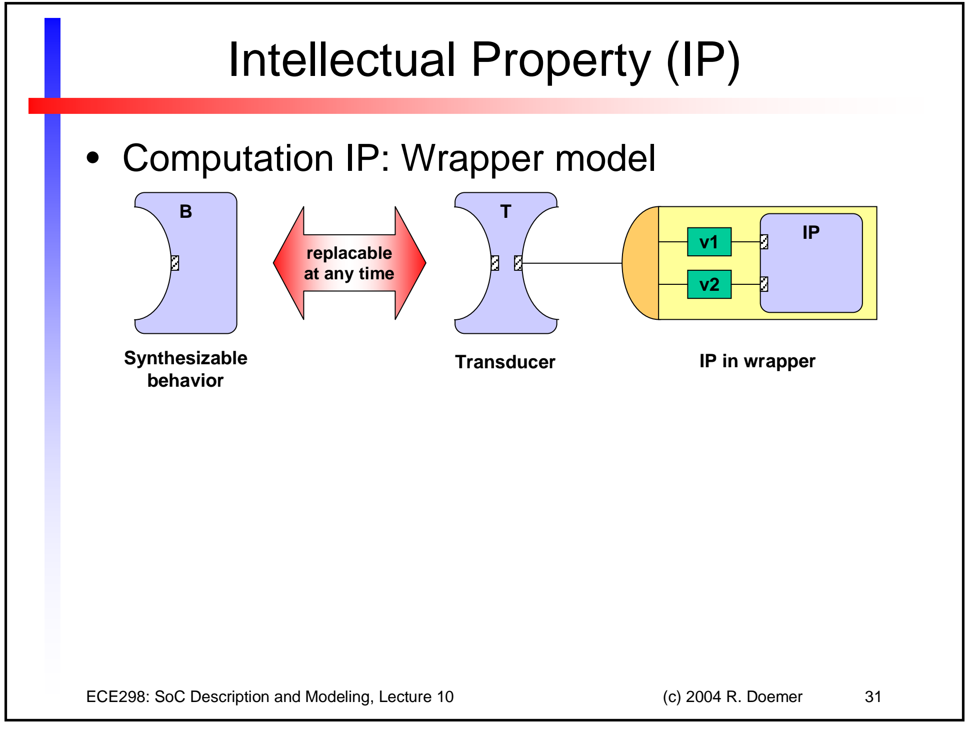


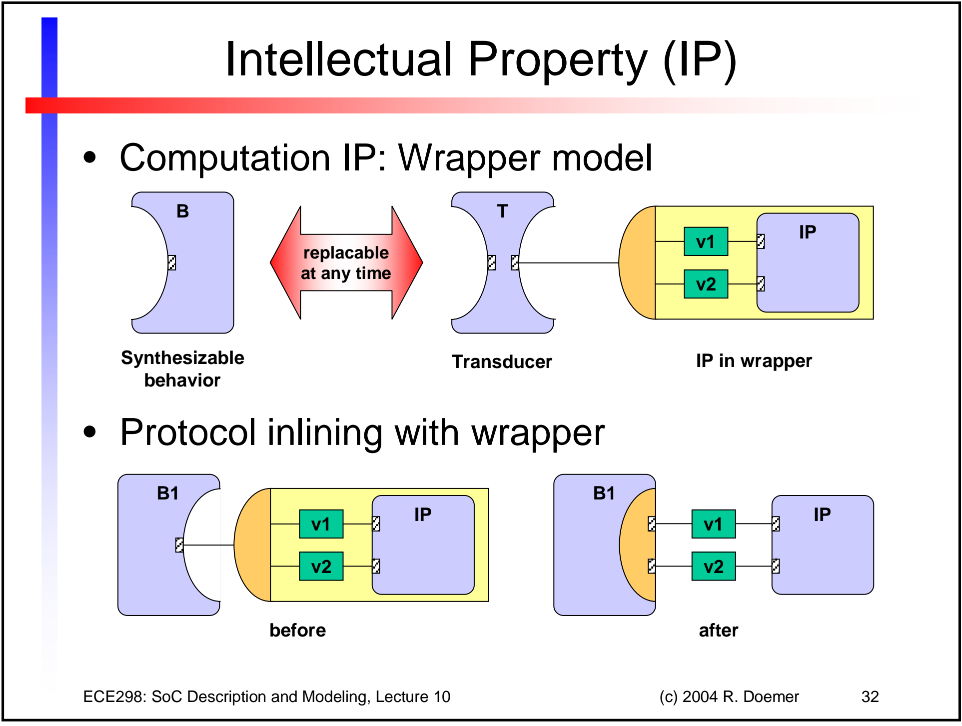

Computation vs. Communication
• Traditional model
P1
P2
s1
s2
s3
– Processes and signals
– Mixture of computation and communication
– Automatic replacement impossible
• SpecC model
B1
C1
B2
v1
v2
v3
– Behaviors and channels
– Separation of computation and communication
– Plug-and-play
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
27

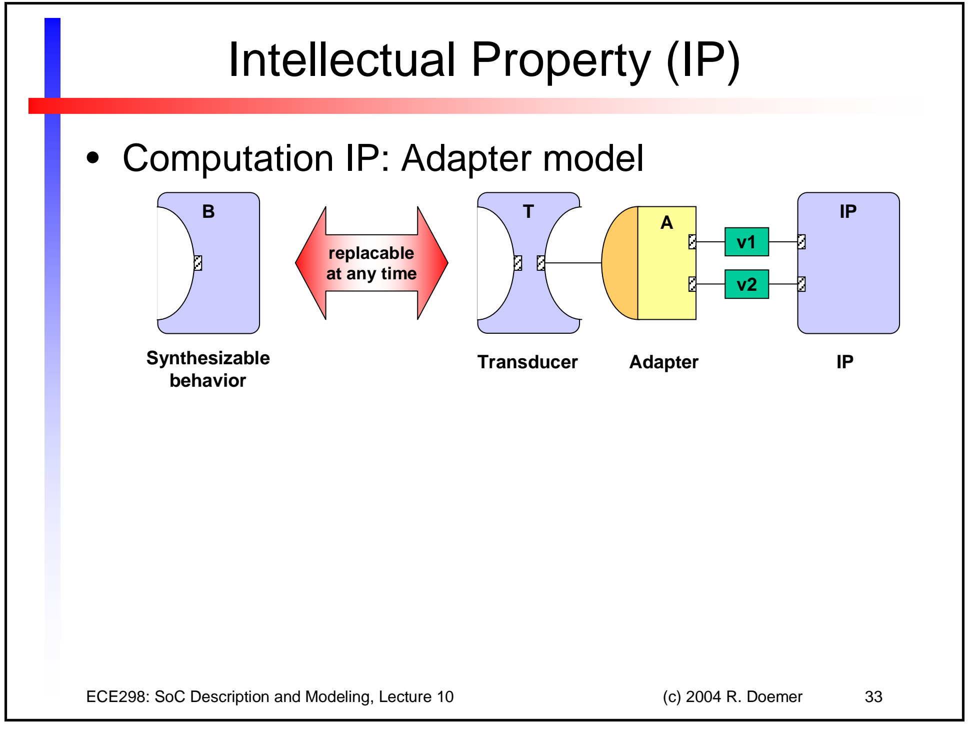

Computation vs. Communication
• Protocol Inlining
– Specification model
– Exploration model
B1
C1
B2
v1
v2
v3
• Computation in behaviors
• Communication in channels
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
28

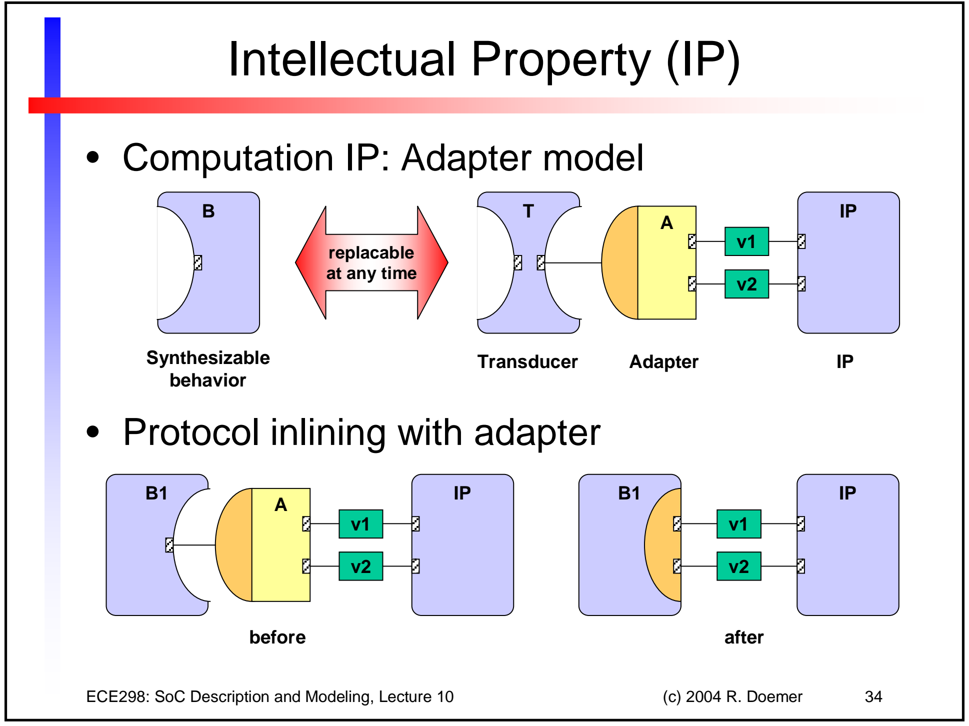

Computation vs. Communication
• Protocol Inlining
– Specification model
– Exploration model
C1
B1
B2
v1
v2
v3
• Computation in behaviors
• Communication in channels
– Implementation model
B1
B2
v1
v2
v3
• Channel disappears
• Communication inlined into behaviors
• Wires exposed
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
29

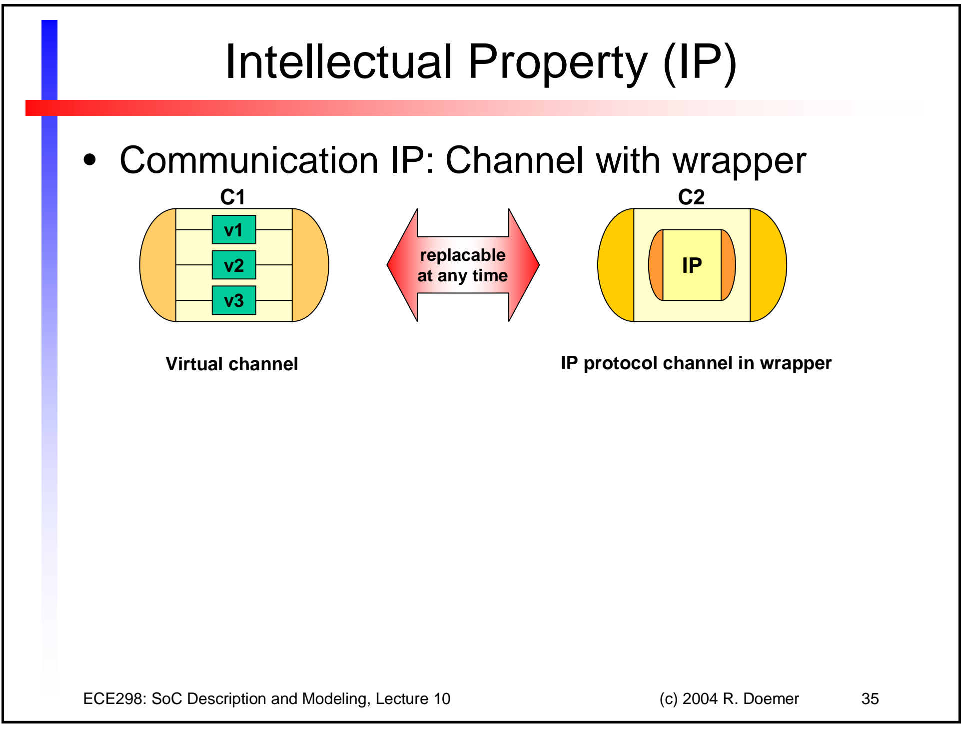

Intellectual Property (IP)
• Computation IP: Wrapper model
B
Synthesizable behavior
v1
v2
IP
IP in wrapper
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
30

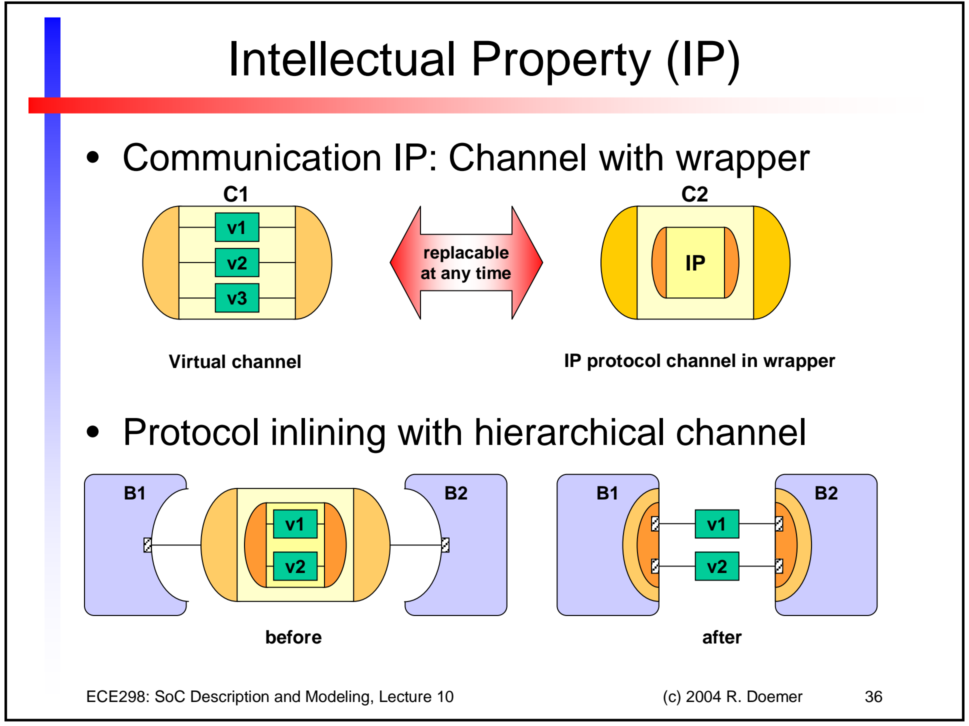

Intellectual Property (IP)
• Computation IP: Wrapper model
B
replacable at any time
T
v1
v2
IP
Synthesizable behavior
Transducer
IP in wrapper
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
31

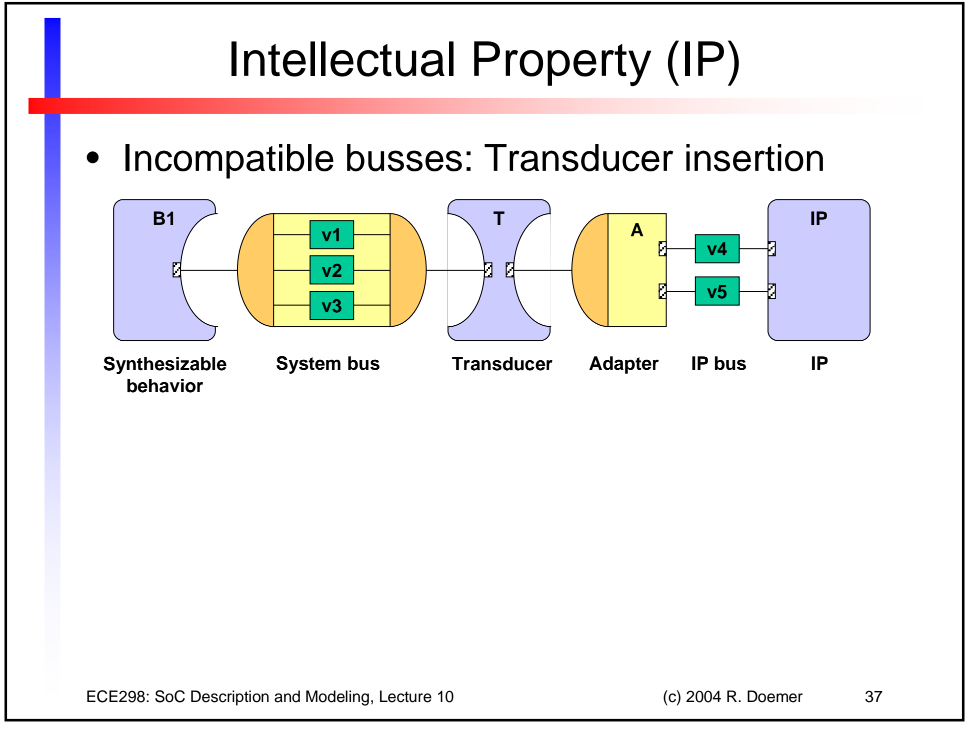

Intellectual Property (IP)
• Computation IP: Wrapper model
B
replacable at any time
T
v1
v2
IP
Synthesizable behavior
Transducer
IP in wrapper
• Protocol inlining with wrapper
B1
v1
v2
IP
before
B1
v1
v2
IP
after
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
32

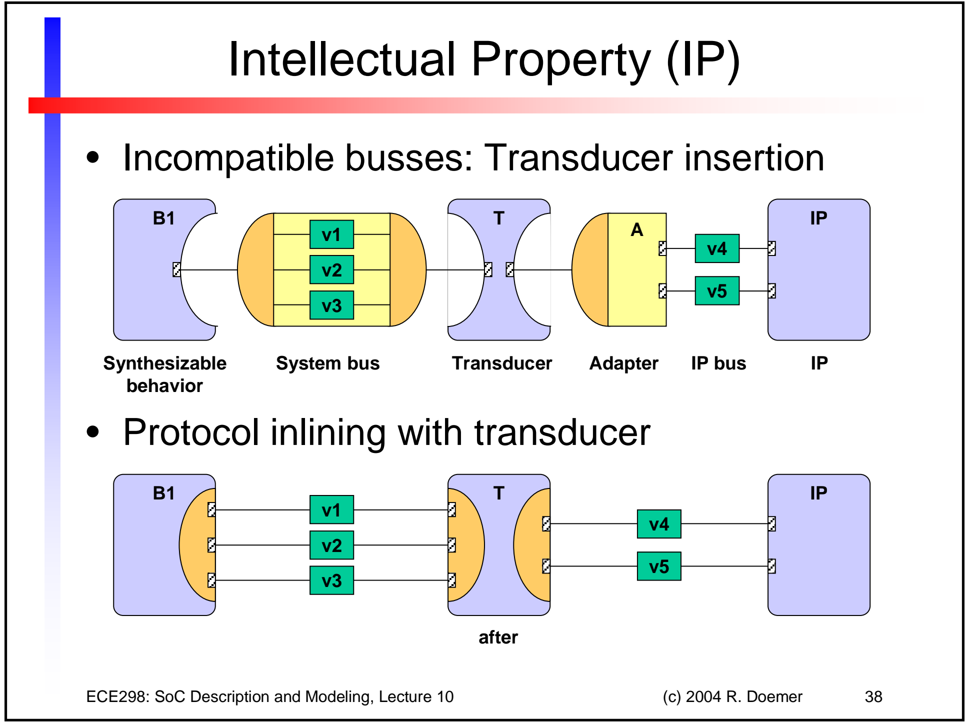

Intellectual Property (IP)
• Computation IP: Adapter model
B
replacable at any time
T
A
v1
v2
IP
Synthesizable behavior
Transducer
Adapter
IP
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
33


Intellectual Property (IP)
• Computation IP: Adapter model
B
replacable at any time
T
A
v1
v2
IP
Synthesizable behavior
Transducer
Adapter
IP
• Protocol inlining with adapter
B1
A
v1
v2
IP
before
B1
v1
v2
IP
after
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
34


Intellectual Property (IP)
• Communication IP: Channel with wrapper
C1
v1
v2
v3
replacable
at any time
C2
IP
Virtual channel
IP protocol channel in wrapper
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
35


Intellectual Property (IP)
• Communication IP: Channel with wrapper
C1
v1
v2
v3
replacable
at any time
C2
IP
Virtual channel
IP protocol channel in wrapper
• Protocol inlining with hierarchical channel
B1
v1
v2
B2
before
B1
v1
v2
B2
after
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
36


Intellectual Property (IP)
• Incompatible busses: Transducer insertion
B1
v1
v2
v3
T
A
v4
v5
IP
Synthesizable behavior
System bus
Transducer
Adapter
IP bus
IP
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
37

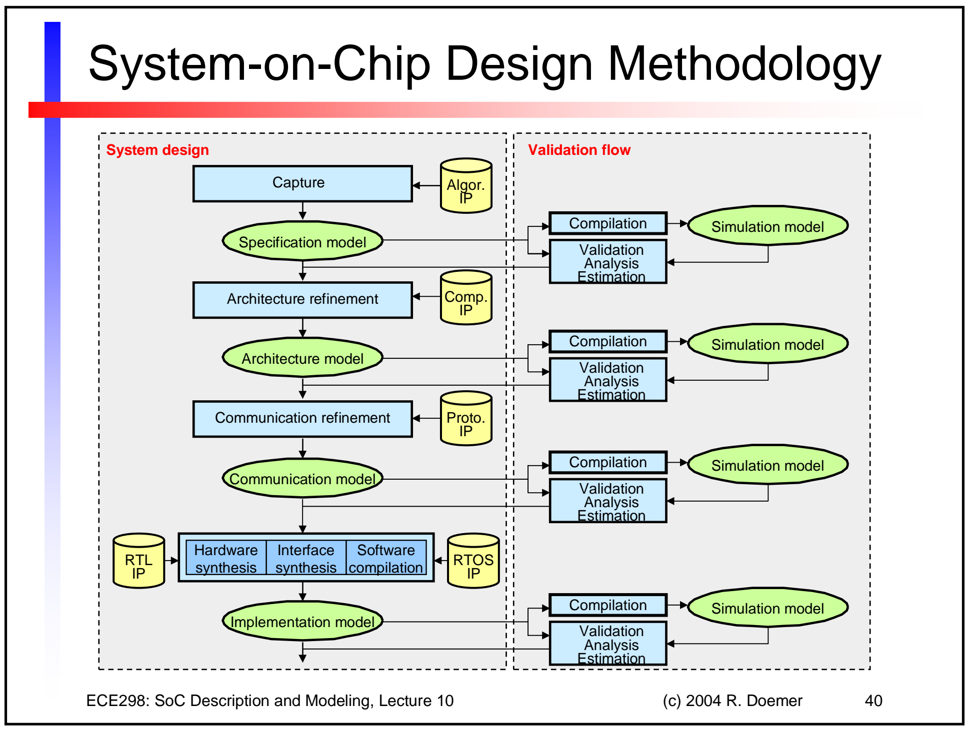

Intellectual Property (IP)
• Incompatible busses: Transducer insertion
B1
v1
v2
v3
T
A
v4
v5
IP
Synthesizable behavior
System bus
Transducer
Adapter
IP bus
IP
• Protocol inlining with transducer
B1
v1
v2
v3
T
v4
v5
IP
after
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
38

Lecture 10: Overview

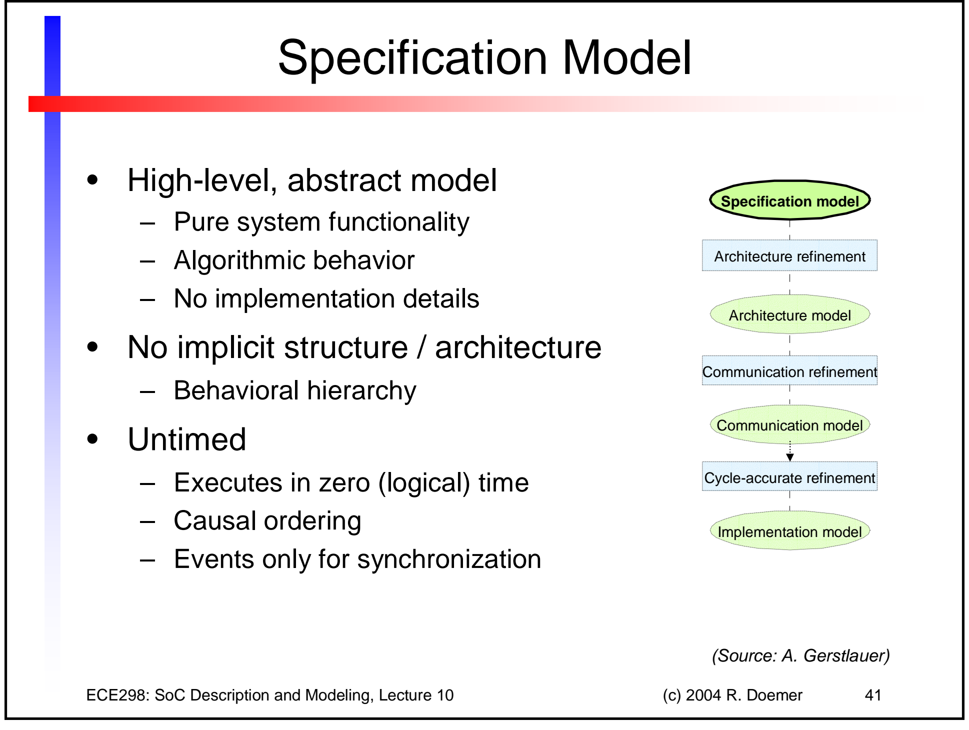
- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - ➢ SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

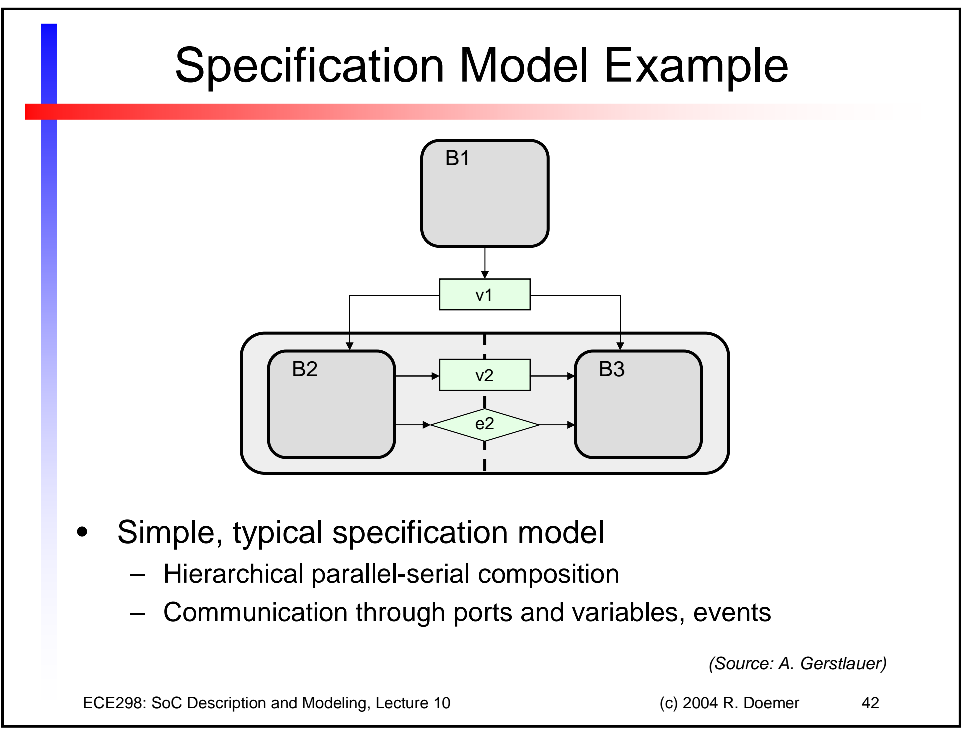
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 39

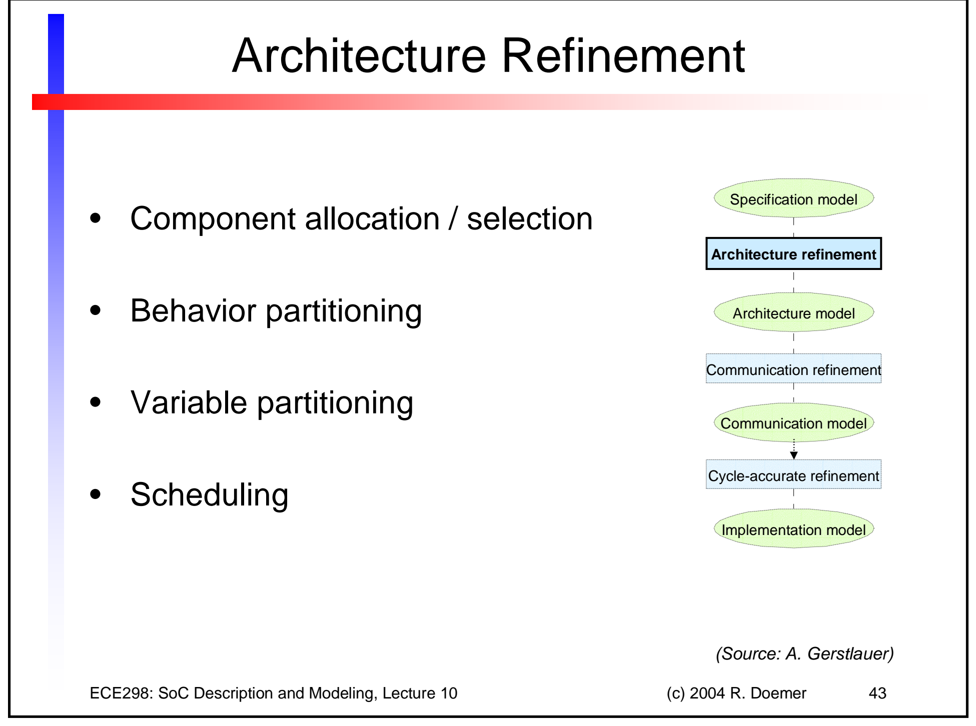
System-on-Chip Design Methodology

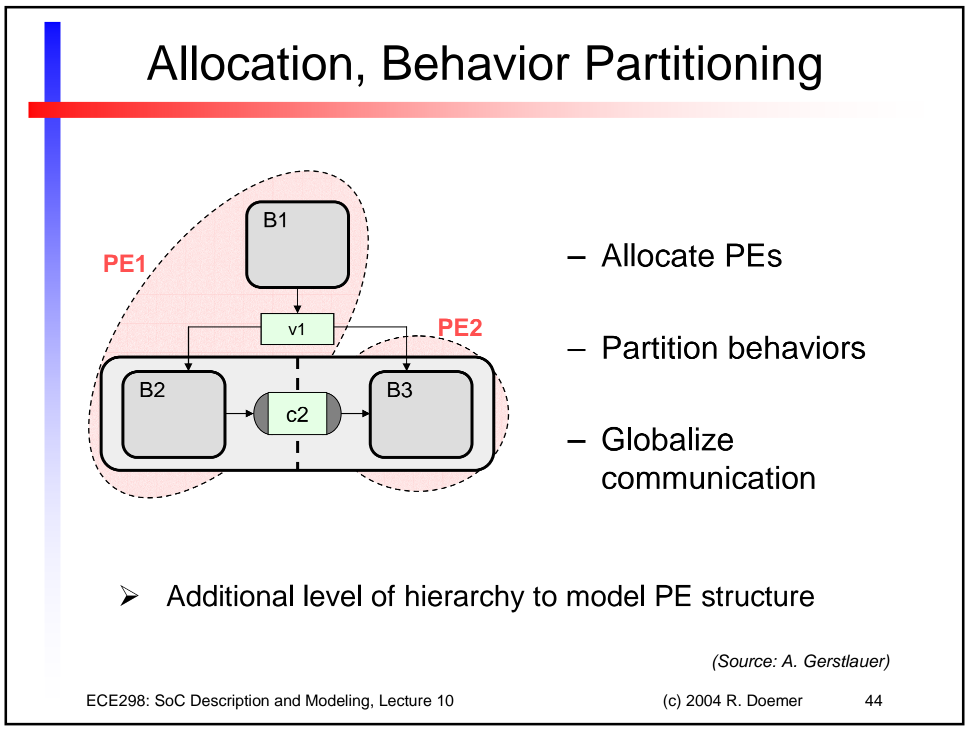


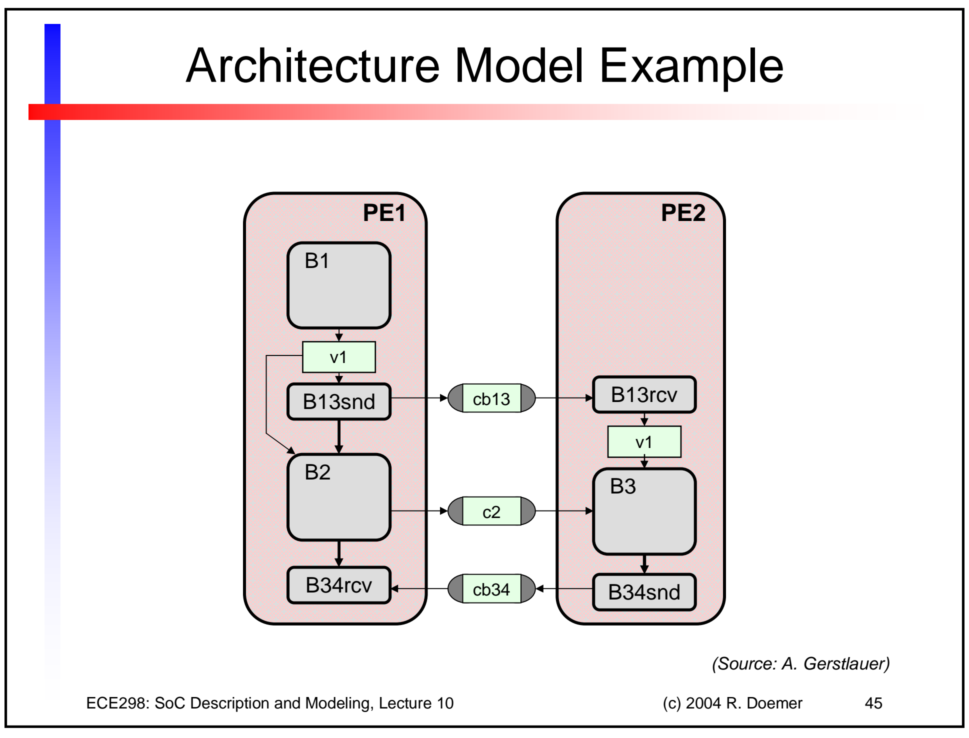
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 40

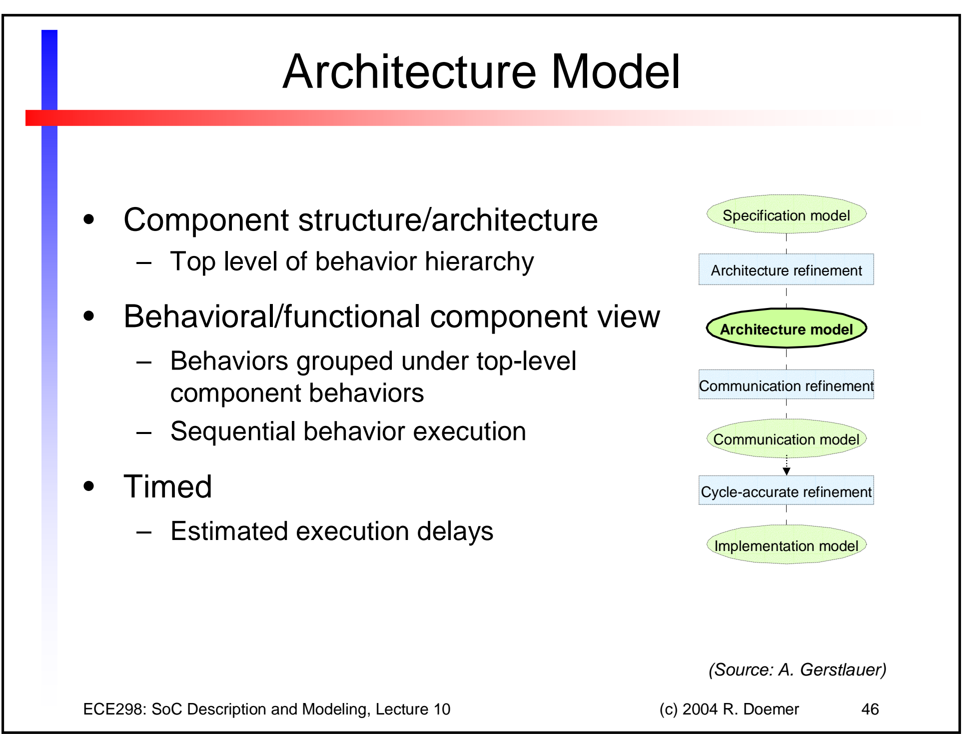

Specification Model
- High-level, abstract model
  - Pure system functionality
  - Algorithmic behavior
  - No implementation details
- No implicit structure / architecture
  - Behavioral hierarchy
- Untimed
  - Executes in zero (logical) time
  - Causal ordering
  - Events only for synchronization
Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
41

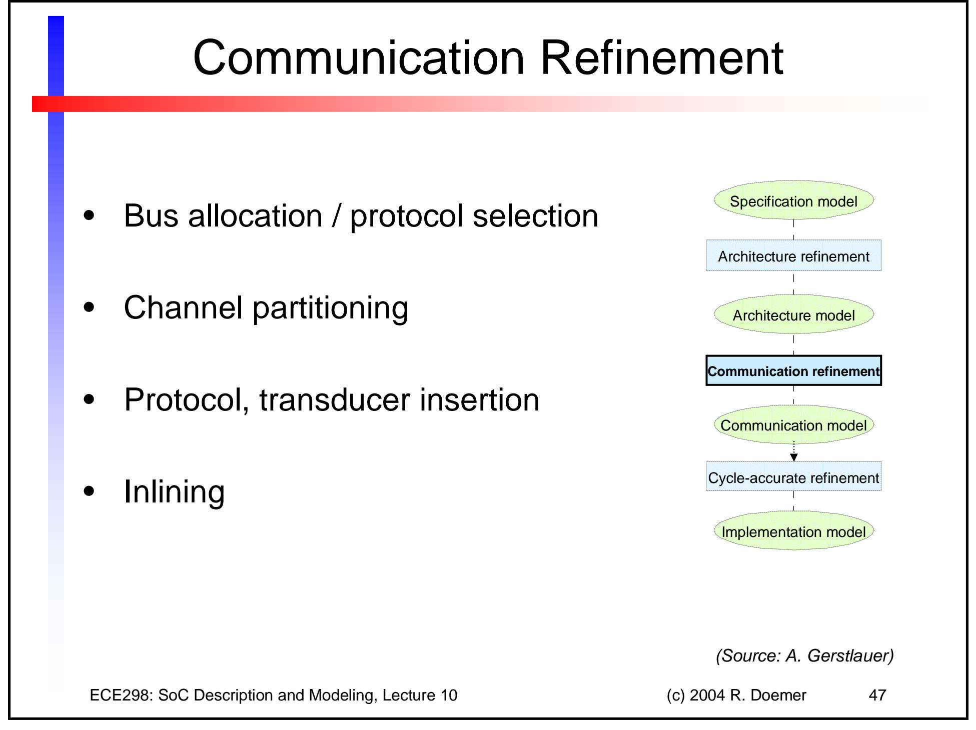

Specification Model Example
B1
v1
B2
v2
B3
e2
- Simple, typical specification model
  - Hierarchical parallel-serial composition
  - Communication through ports and variables, events
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
42

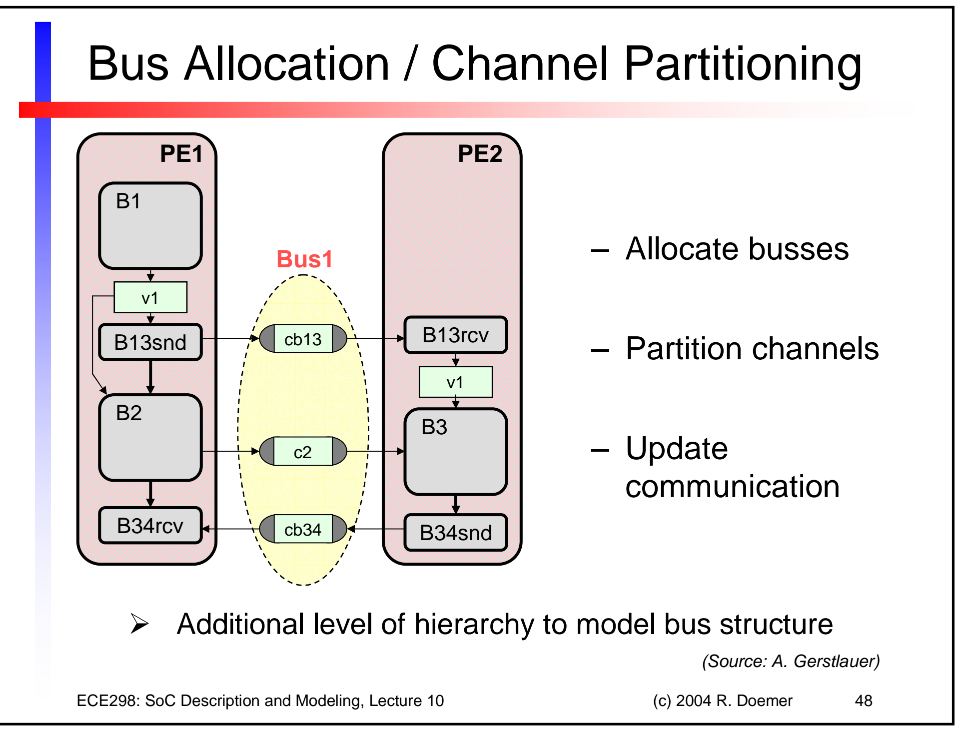

Architecture Refinement
• Component allocation / selection
• Behavior partitioning
• Variable partitioning
• Scheduling
Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
43

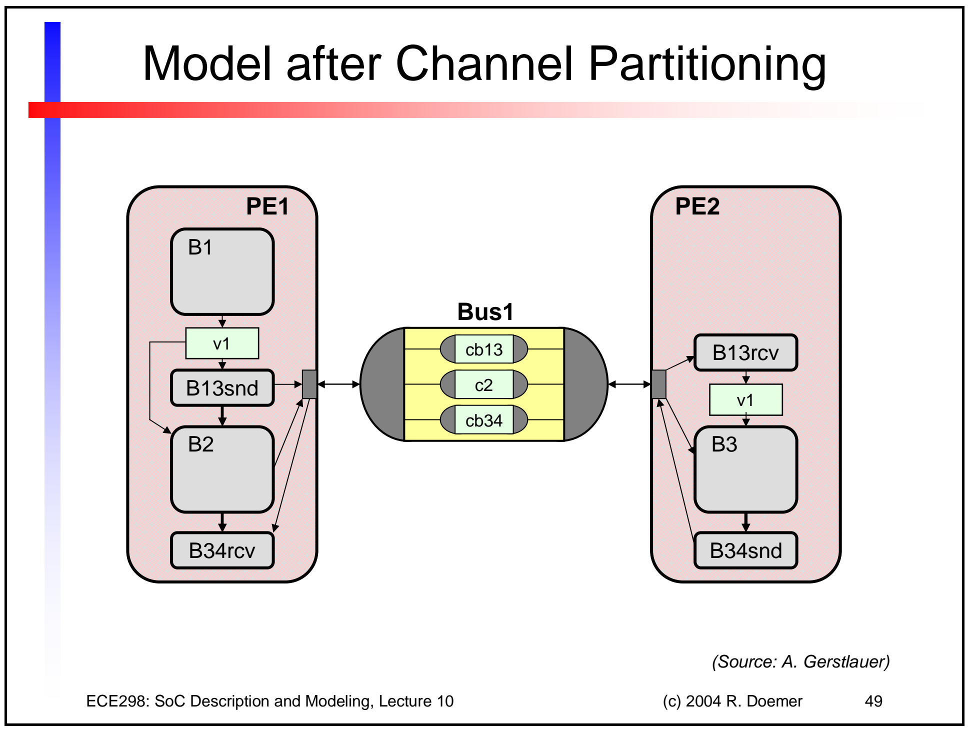

Allocation, Behavior Partitioning
PE1
B1
v1
PE2
B2
c2
B3
– Allocate PEs
– Partition behaviors
– Globalize communication
➢ Additional level of hierarchy to model PE structure
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
44

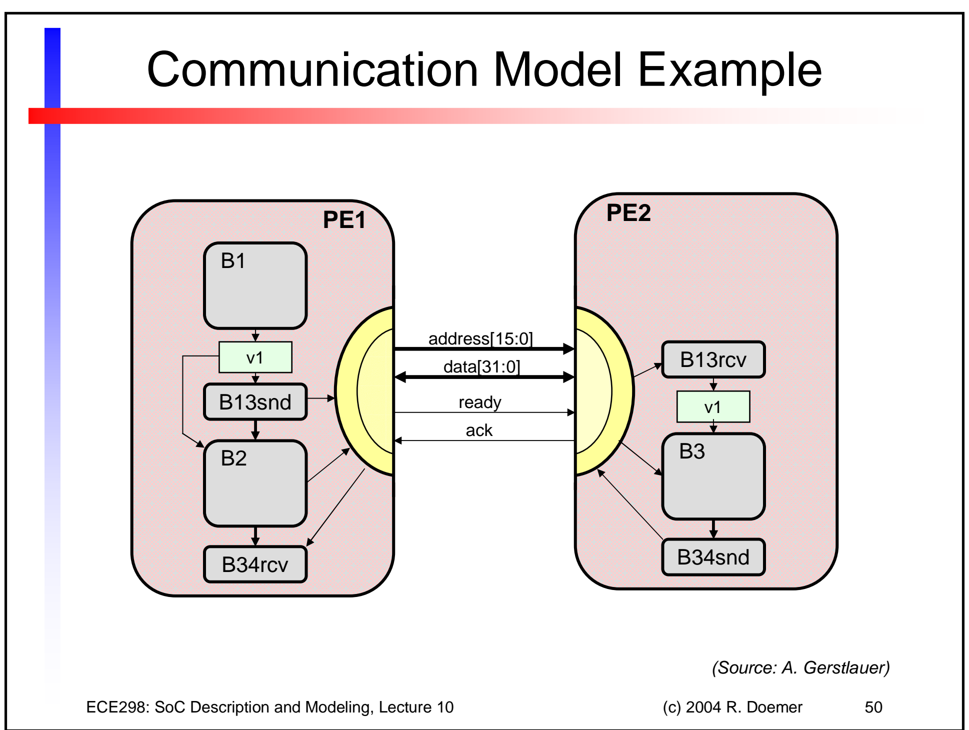

Architecture Model Example
PE1
PE2
B1
v1
B13snd
B2
B34rcv
cb13
c2
cb34
B13rcv
v1
B3
B34snd
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
45

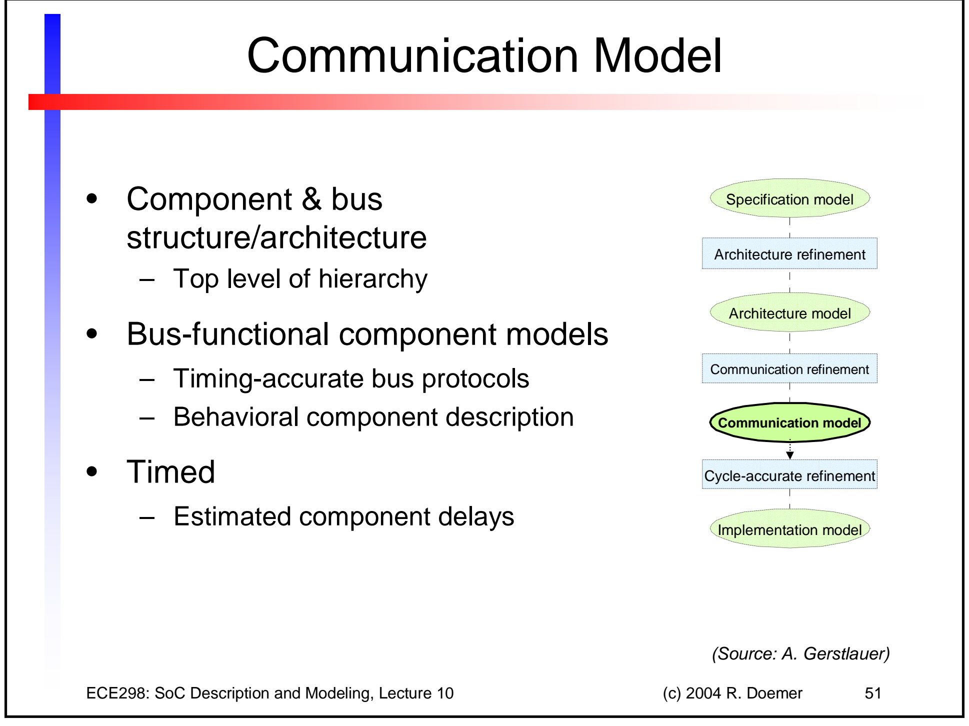

Architecture Model
• Component structure/architecture
– Top level of behavior hierarchy
• Behavioral/functional component view
– Behaviors grouped under top-level component behaviors
– Sequential behavior execution
• Timed
– Estimated execution delays
Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
46

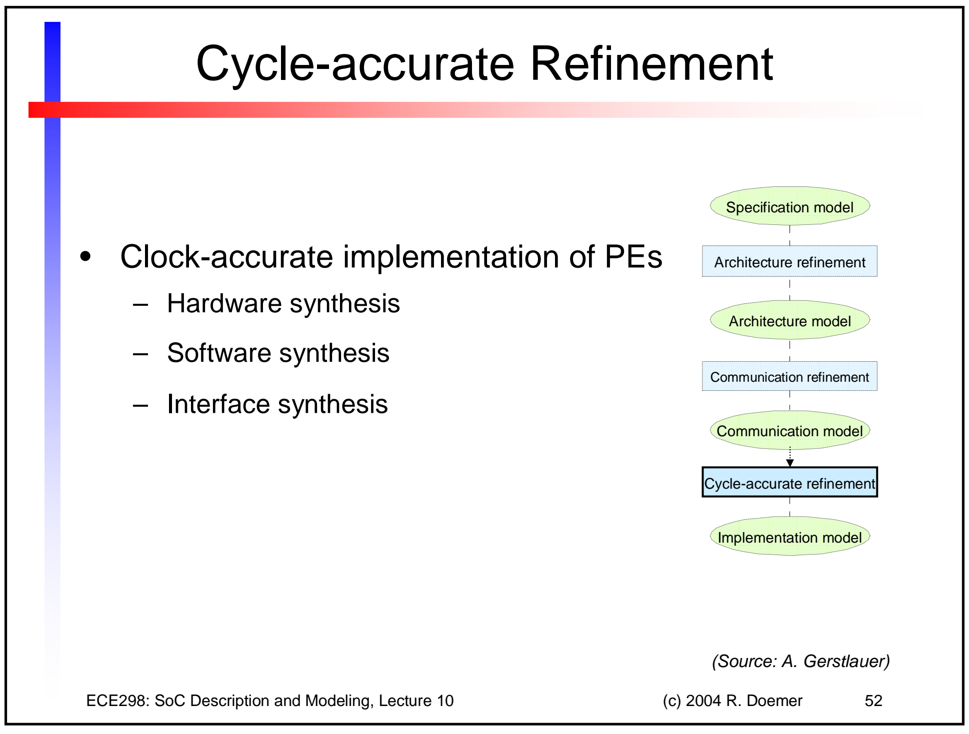

Communication Refinement
Bus allocation / protocol selection
Channel partitioning
Protocol, transducer insertion
Inlining
Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
47

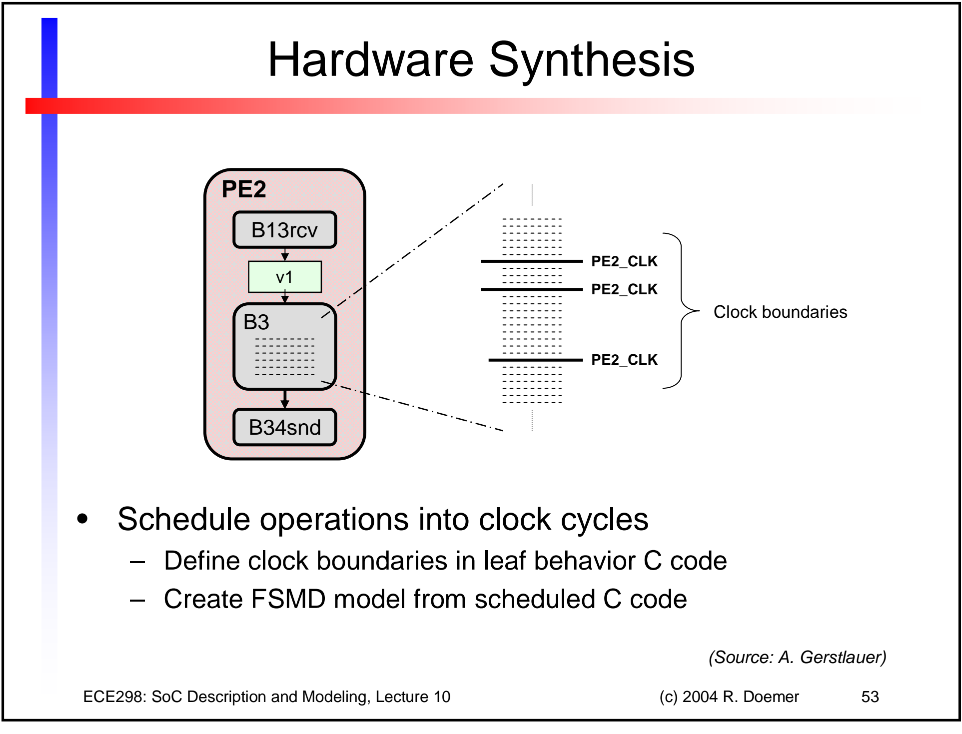

Bus Allocation / Channel Partitioning
PE1
PE2
B1
v1
B13snd
B2
B34rcv
Bus1
cb13
c2
cb34
B13rcv
v1
B3
B34snd
– Allocate busses
– Partition channels
– Update communication
Additional level of hierarchy to model bus structure
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
48


Model after Channel Partitioning
PE1
B1
v1
B13snd
B2
B34rcv
Bus1
cb13
c2
cb34
PE2
B13rcv
v1
B3
B34snd
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
49

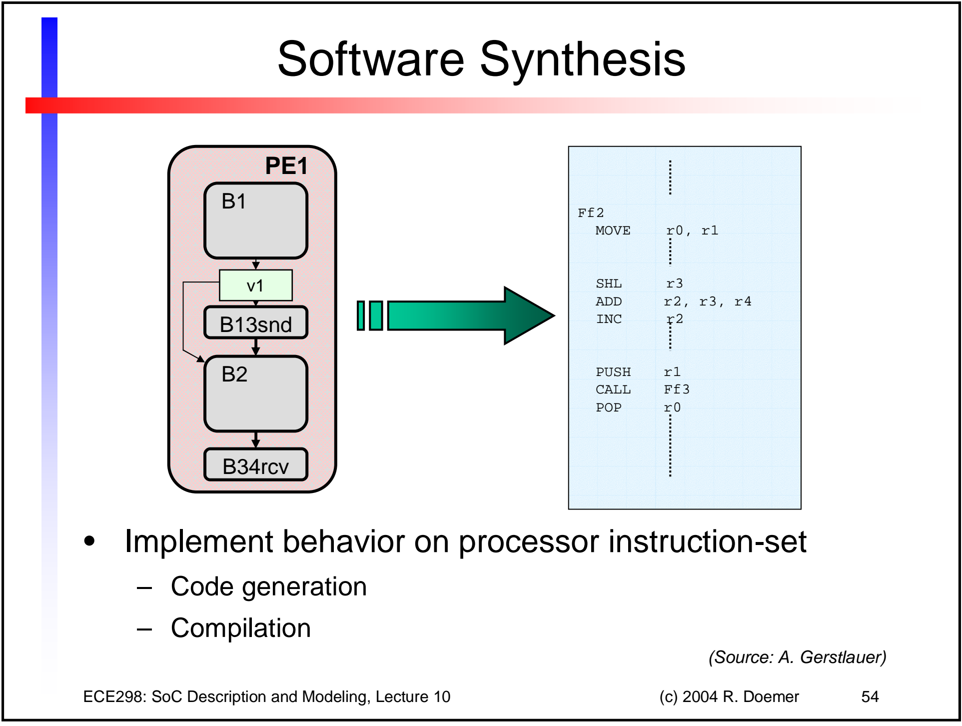

Communication Model Example
PE1
B1
v1
B13snd
B2
B34rcv
address[15:0]
data[31:0]
ready
ack
PE2
B13rcv
v1
B3
B34snd
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
50


Communication Model
Component & bus structure/architecture
Top level of hierarchy
Bus-functional component models
Timing-accurate bus protocols
Behavioral component description
Timed
Estimated component delays
Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
51

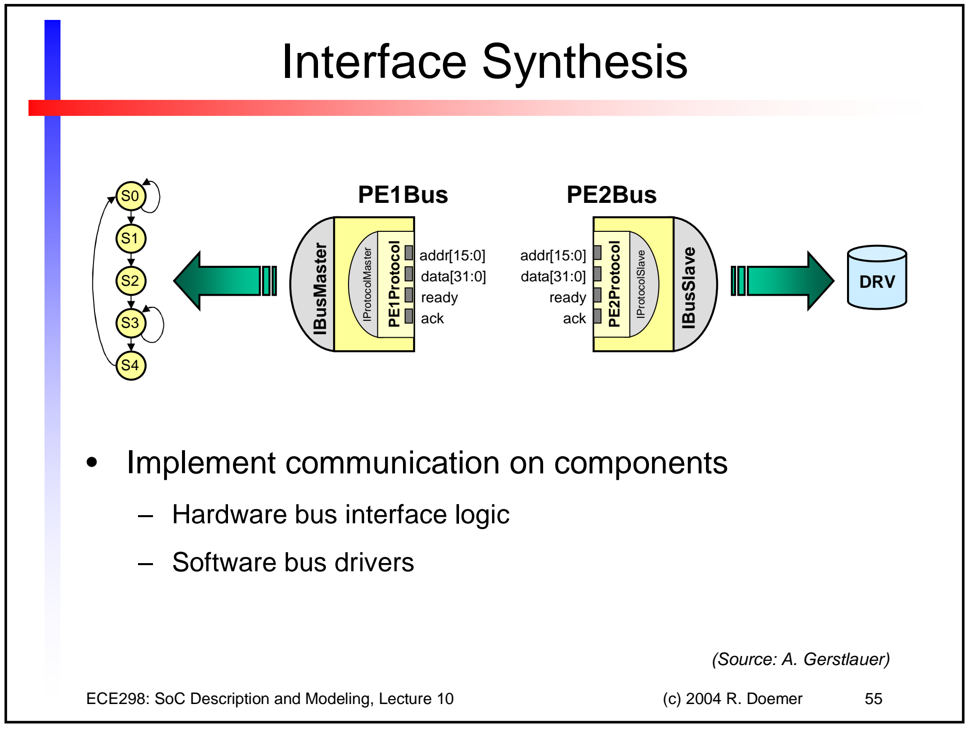

Cycle-accurate Refinement
Clock-accurate implementation of PEs
Hardware synthesis
Software synthesis
Interface synthesis
Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
52

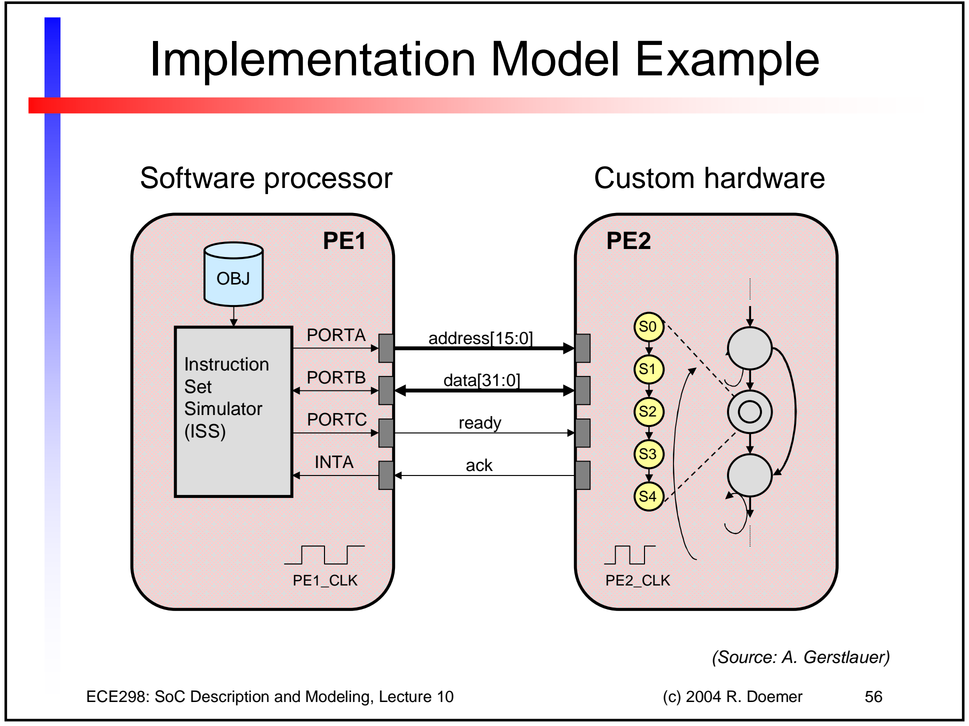


Hardware Synthesis
PE2
B13rcv
v1
B3
B34snd
PE2_CLK
PE2_CLK
PE2_CLK
Clock boundaries
• Schedule operations into clock cycles
– Define clock boundaries in leaf behavior C code
– Create FSMD model from scheduled C code
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
53



Software Synthesis
PE1
B1
v1
B13snd
B2
B34rcv
Ff2
MOVE r0, r1
SHL r3
ADD r2, r3, r4
INC r2
PUSH r1
CALL Ff3
POP r0
• Implement behavior on processor instruction-set
– Code generation
– Compilation
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
54


Interface Synthesis
PE1Bus
PE2Bus
S0
S1
S2
S3
S4
IBusMaster
IProtocolMaster
PE1Protocol
addr[15:0]
data[31:0]
ready
ack
PE2Protocol
IProtocolSlave
IBusSlave
DRV
• Implement communication on components
– Hardware bus interface logic
– Software bus drivers
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
55


Implementation Model Example
Software processor
Custom hardware
PE1
OBJ
Instruction Set Simulator (ISS)
PORTA
PORTB
PORTC
INTA
address[15:0]
data[31:0]
ready
ack
PE2
S0
S1
S2
S3
S4
PE1_CLK
PE2_CLK
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
56

Implementation Model

- Cycle-accurate system description
  - RTL description of hardware
    - Behavioral/structural FSMD view
  - Object code for processors
    - Instruction-set co-simulation
  - Clocked bus communication
    - Bus interface timing based on PE clock

Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model

(Source: A. Gerstlauer)

System-on-Chip Design Methodology

- Four levels of abstraction
  - Specification model: untimed, functional
  - Architecture model: estimated, structural
  - Communication model: timed, bus-functional
  - Implementation model: cycle-accurate, RTL/IS
- Three refinement steps
  - Architecture refinement
  - Communication refinement
  - Cycle-accurate refinement
    - HW / SW / interface synthesis
- Well-defined, formal models & transformations
  - Automatic, gradual refinement
  - Executable models, test bench re-use
  - Simple verification

(Source: A. Gerstlauer)

Lecture 10: Overview

- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - ➢ The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

The SpecC Language

- Overview
  - Foundation
  - Types
  - Structural and behavioral hierarchy
  - Concurrency
  - State transitions
  - Exception handling
  - Communication
  - Synchronization
  - Timing
  - RTL

The SpecC Language

- Foundation: ANSI-C
  - Software requirements are fully covered
  - SpecC is a true superset of ANSI-C
  - Every C program is a SpecC program
  - Leverage of large set of existing programs
  - Well-known
  - Well-established

The SpecC Language

- Foundation: ANSI-C
  - Software requirements are fully covered
  - SpecC is a true superset of ANSI-C
  - Every C program is a SpecC program
  - Leverage of large set of existing programs
  - Well-known
  - Well-established
- SpecC has extensions needed for hardware
  - Minimal, orthogonal set of concepts
  - Minimal, orthogonal set of constructs
- SpecC is a real language
  - Not just a class library

The SpecC Language

- ANSI-C
  - Program is set of functions
  - Execution starts from function `main()`

```
/* HelloWorld.c */

#include <stdio.h>

void main(void)
{
  printf("Hello World!\n");
}
```


The SpecC Language

- ANSI-C
  - Program is set of functions
  - Execution starts from function `main()`

```
/* HelloWorld.c */

#include <stdio.h>

void main(void)
{
  printf("Hello World!\n");
}
```


- SpecC
  - Program is set of behaviors, channels, and interfaces
  - Execution starts from behavior `Main.main()`

```
// HelloWorld.sc

#include <stdio.h>

behavior Main
{
  void main(void)
  {
    printf("Hello World!\n");
  }
};
```

The SpecC Language

- SpecC types
  - Support for all ANSI-C types
    - predefined types (**int**, **float**, **double**, …)
    - composite types (arrays, pointers)
    - user-defined types (**struct**, **union**, **enum**)
  - Boolean type: Explicit support of truth values
    - `bool b1 = true;`
    - `bool b2 = false;`
  - Bit vector type: Explicit support of bit vectors of arbitrary length
    - `bit[15:0] bv = 1111000011110000b;`
  - Event type: Support of synchronization
    - `event e;`
  - Buffered and signal types: Explicit support of RTL concepts
    - `buffered[clk] bit[32] reg;`
    - `signal bit[16] address;`

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 65

The SpecC Language

- Bit vector type
  - signed or unsigned
  - arbitrary length
  - standard operators
    - logical operations
    - arithmetic operations
    - comparison operations
    - type conversion
    - type promotion
  - concatenation operator
    - `a @ b`
  - slice operator
    - `a[l:r]`

```
typedef bit[7:0] byte;  // type definition
byte            a;
unsigned bit[16] b;

bit[31:0] BitMagic(bit[4] c, bit[32] d)
{
 bit[31:0] r;

 a = 11001100b;            // constant
 b = 1111000011110000ub; // assignment

 b[7:0] = a;               // sliced access
 b = d[31:16];

 if (b[15])                // single bit
    b[15] = 0b;            // access

 r = a @ d[11:0] @ c      // concatenation
     @ 11110000b;

 a = ~(a & 11110000);     // logical op.
 r += 42 + 3*a;           // arithmetic op.

 return r;
}
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 66

The SpecC Language

- **`buffered`** type modifier
  - Representation of storage in RTL models
    - registers
    - register files
    - memories
    - etc.
  - Update at notification of specified events
    - synchronized with explicit clock

```
event Clk1, Clk2;                // system clock
buffered[Clk1] bit[32] R1;       // register
buffered[Clk1] bit[32] R2;

buffered[CLK2] bit[16] RF[64];   // register file
buffered[CLK2] bit[ 8] M1[1024]; // memory

R1 = R2;      // swap contents of R1 and R2
R2 = R1;
wait CLK1;

RF[2] = RF[0] + RF[1];
...
wait CLK2;
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 67

The SpecC Language

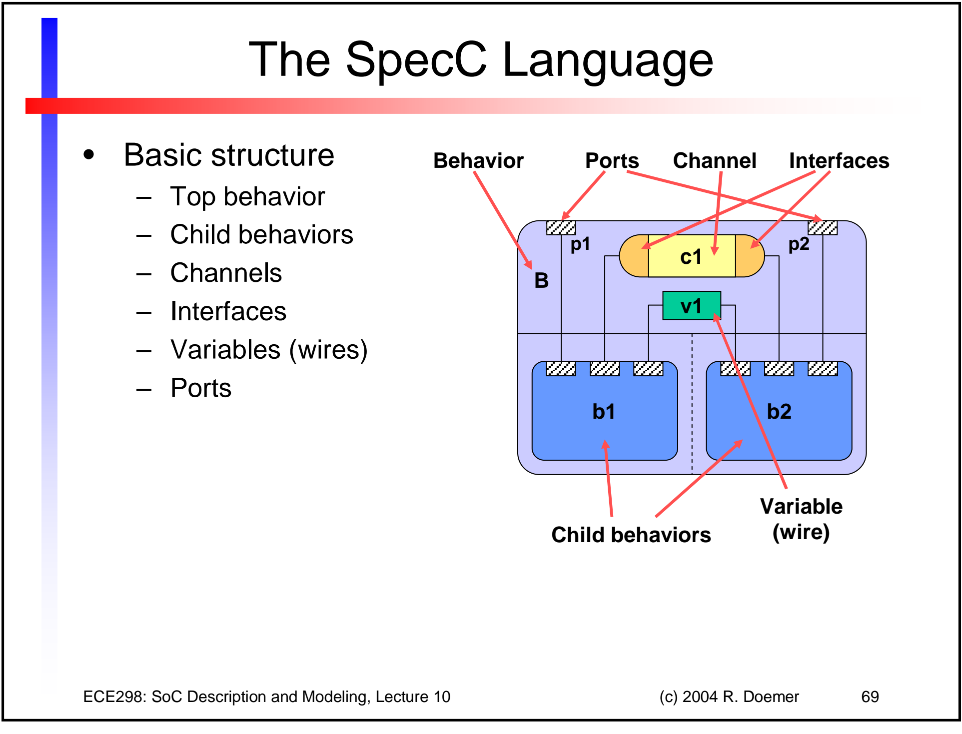
- **`signal`** type modifier
  - Representation of wires and busses in RTL models
  - Semantics as in VHDL, Verilog

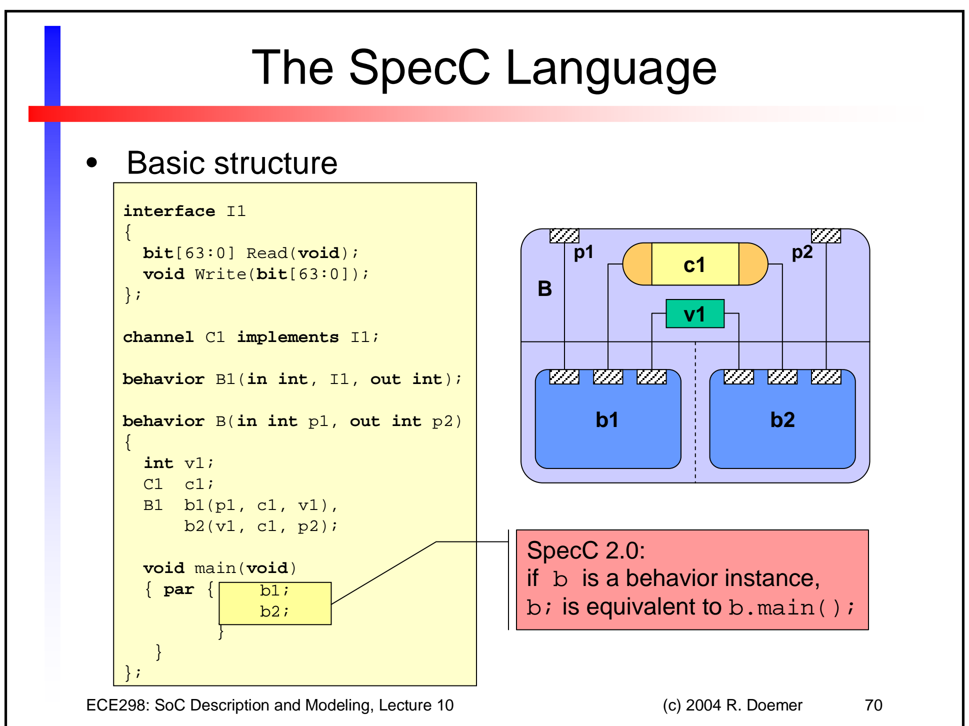
```
signal bit[31:0] addr;   // address bus
signal bit[31:0] data;   // data bus
buffered[CLK] M[1024];

wait addr;               // memory read access
data = M[addr];
...

wait addr && data;
M[addr] = data;          // memory write access
...
```

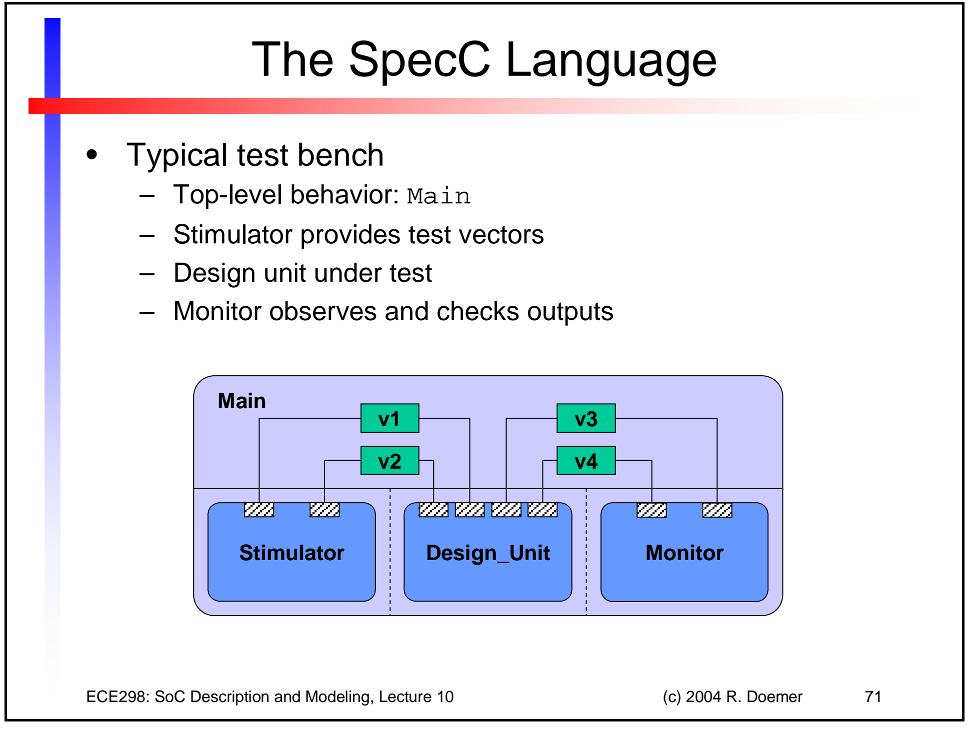
  - Implemented as buffered variables with associated event

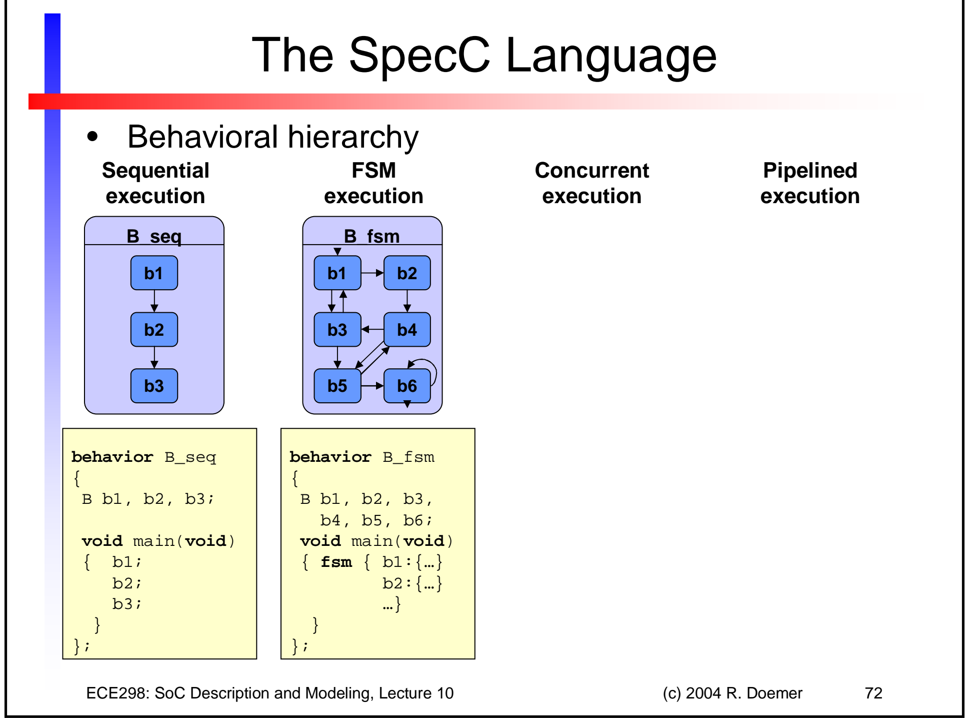
```
signal int x;     ⇔  buffered int x_v; event x_e;
x = 55;           ⇔  x_v = 55; notify x_e;
y = x + 2;        ⇔  y = x_v + 2;
wait x;           ⇔  wait x_e;
notify x;         ⇔  notify x_e;
wait (x == 5);    ⇔  while(x_v != 5) { wait x_e; }
```

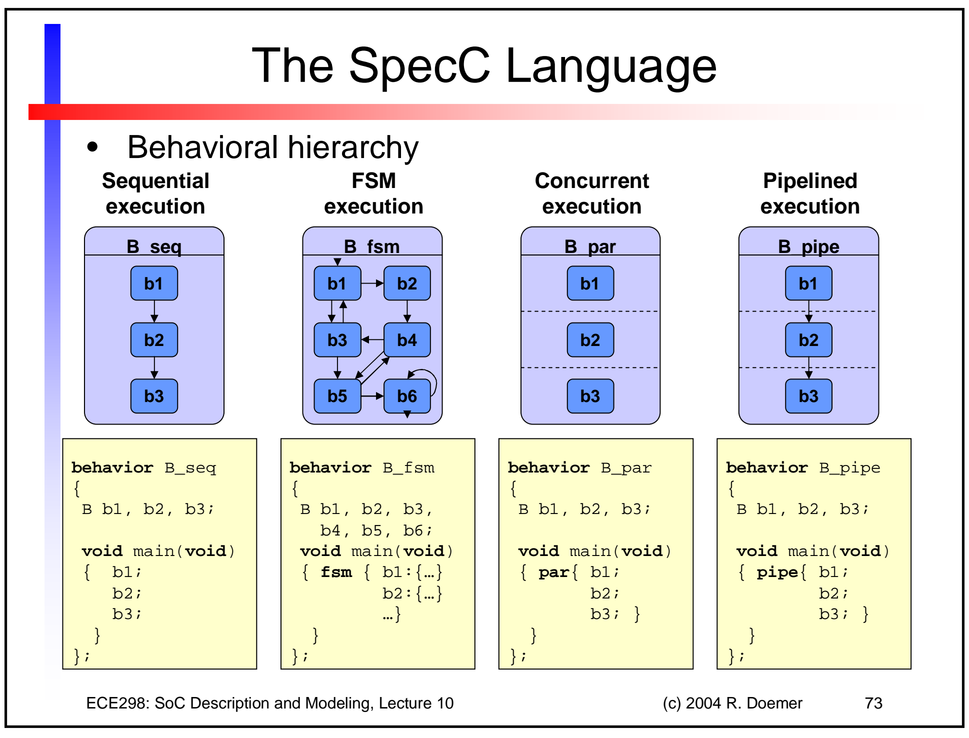
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 68


The SpecC Language
• Basic structure
– Top behavior
– Child behaviors
– Channels
– Interfaces
– Variables (wires)
– Ports
Behavior
Ports
Channel
Interfaces
p1
p2
c1
B
v1
b1
b2
Child behaviors
Variable
(wire)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
69

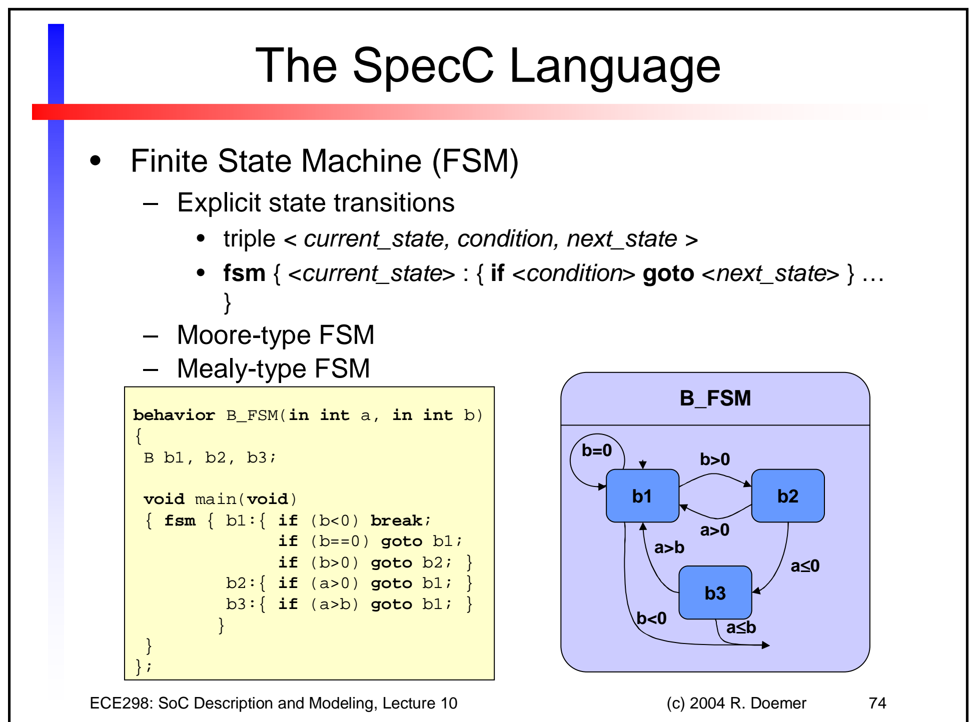

The SpecC Language
• Basic structure
interface I1
{
bit[63:0] Read(void);
void Write(bit[63:0]);
};
channel C1 implements I1;
behavior B1(in int, I1, out int);
behavior B(in int p1, out int p2)
{
int v1;
C1 c1;
B1 b1(p1, c1, v1),
b2(v1, c1, p2);
void main(void)
{ par { b1;
b2;
}
}
};
p1
p2
c1
B
v1
b1
b2
SpecC 2.0:
if b is a behavior instance,
b; is equivalent to b.main();
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
70


The SpecC Language
• Typical test bench
– Top-level behavior: Main
– Stimulator provides test vectors
– Design unit under test
– Monitor observes and checks outputs
Main
v1
v2
v3
v4
Stimulator
Design_Unit
Monitor
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
71

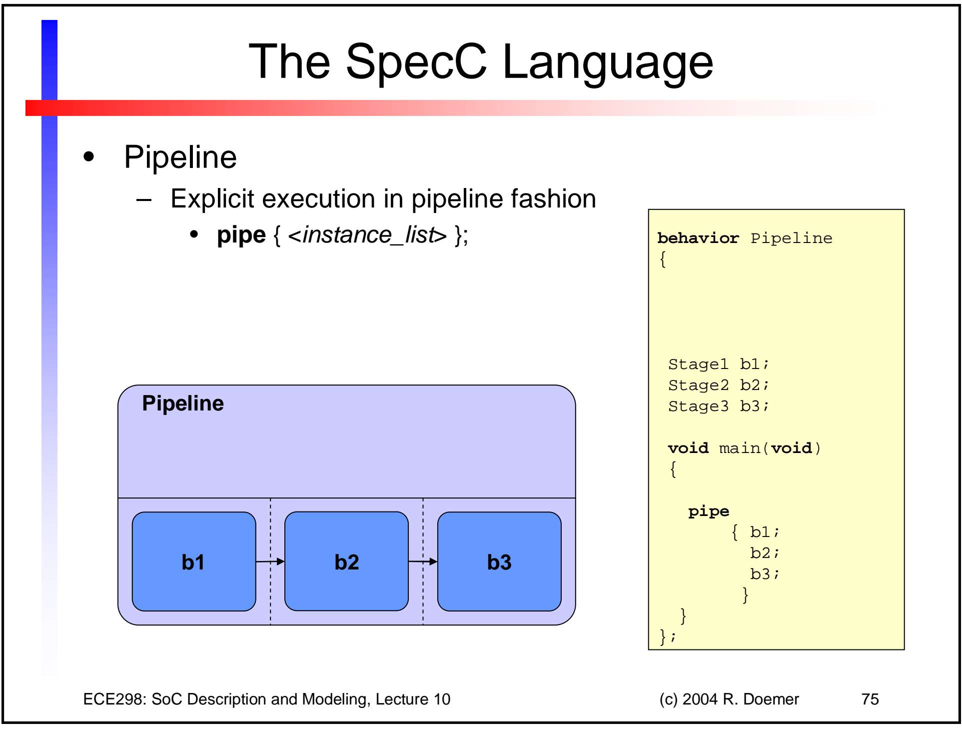

The SpecC Language
• Behavioral hierarchy
Sequential execution
FSM execution
Concurrent execution
Pipelined execution
B_seq
b1
b2
b3
B_fsm
b1
b2
b3
b4
b5
b6
behavior B_seq
{
B b1, b2, b3;
void main(void)
{ b1;
b2;
b3;
}
};
behavior B_fsm
{
B b1, b2, b3,
b4, b5, b6;
void main(void)
{ fsm { b1:{…}
b2:{…}
…}
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
72

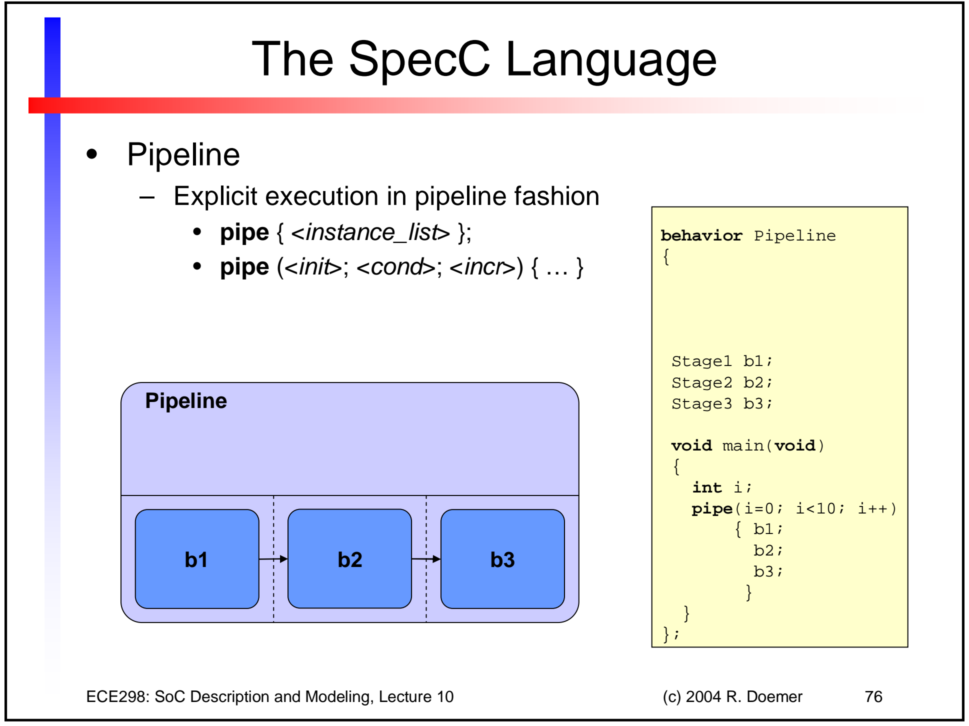


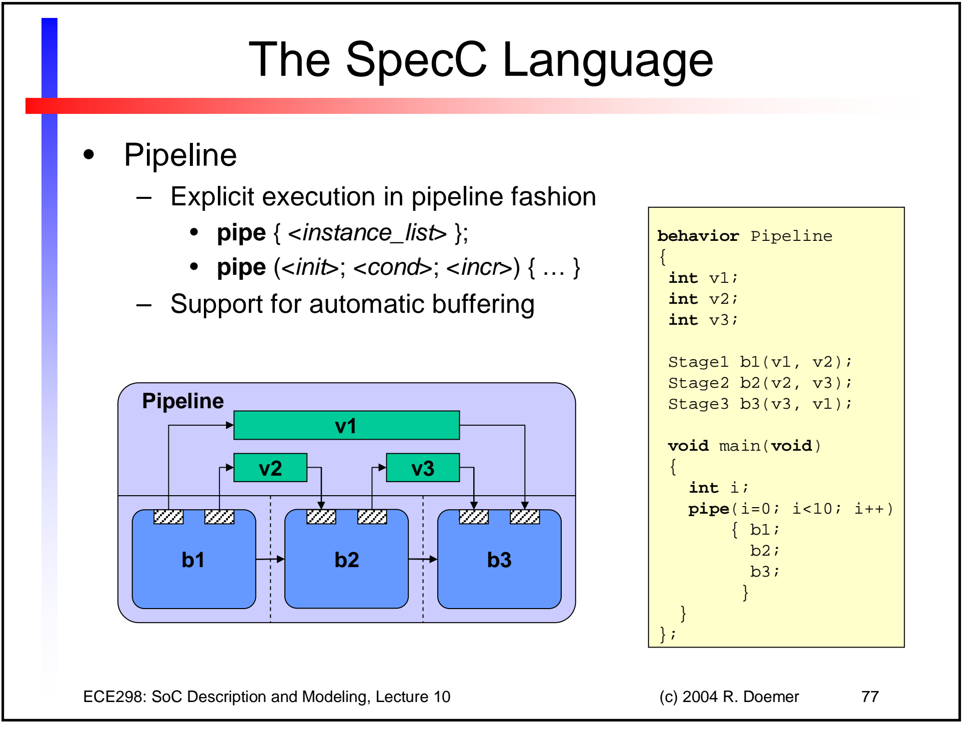
The SpecC Language
Behavioral hierarchy
Sequential execution
FSM execution
Concurrent execution
Pipelined execution
B_seq
b1
b2
b3
B_fsm
b1
b2
b3
b4
b5
b6
B_par
b1
b2
b3
B_pipe
b1
b2
b3
behavior B_seq { B b1, b2, b3; void main(void) { b1; b2; b3; } };
behavior B_fsm { B b1, b2, b3, b4, b5, b6; void main(void) { fsm { b1:{…} b2:{…} …} } };
behavior B_par { B b1, b2, b3; void main(void) { par{ b1; b2; b3; } } };
behavior B_pipe { B b1, b2, b3; void main(void) { pipe{ b1; b2; b3; } } };
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
73

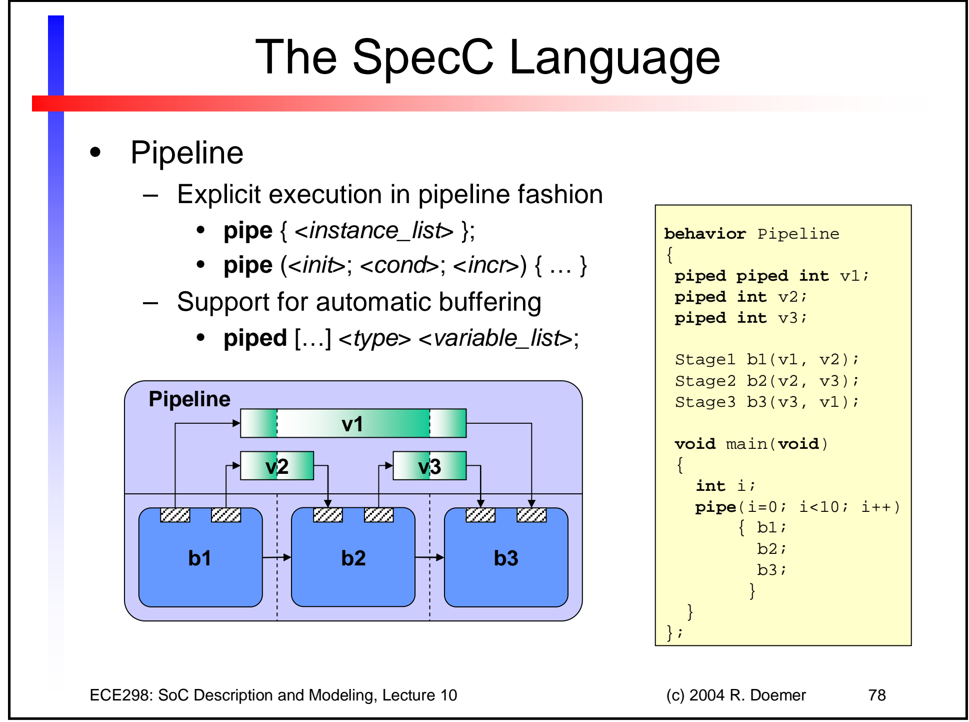


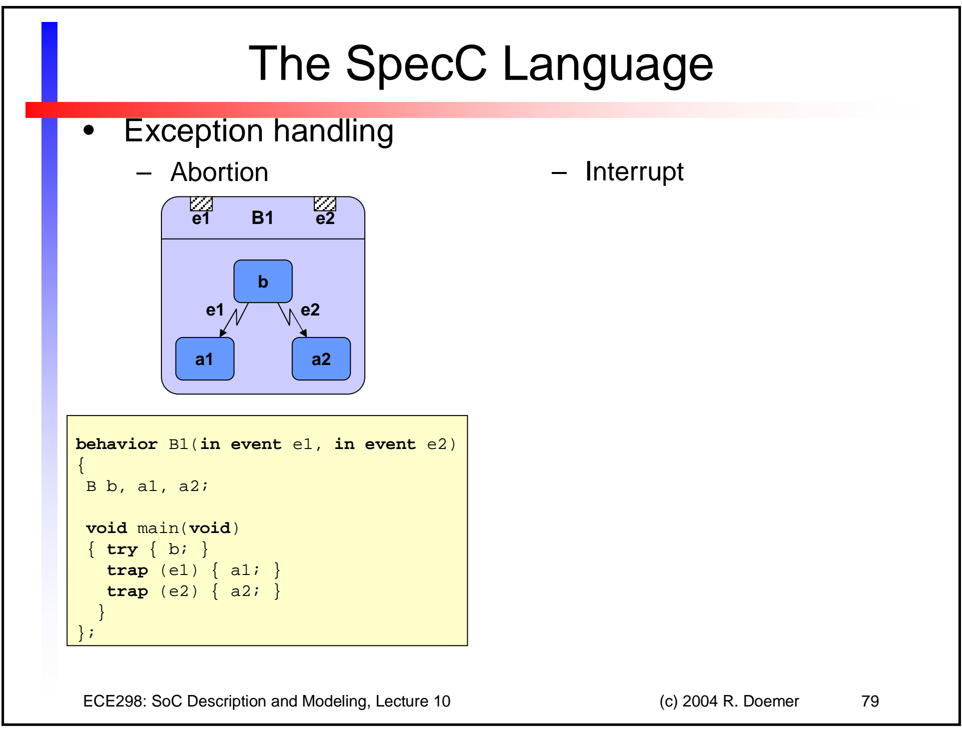
The SpecC Language
Finite State Machine (FSM)
Explicit state transitions
triple < current_state, condition, next_state >
fsm { <current_state> : { if <condition> goto <next_state> } … }
Moore-type FSM
Mealy-type FSM
behavior B_FSM(in int a, in int b) { B b1, b2, b3; void main(void) { fsm { b1:{ if (b<0) break; if (b==0) goto b1; if (b>0) goto b2; } b2:{ if (a>0) goto b1; } b3:{ if (a>b) goto b1; } } } };
B_FSM
b=0
b1
b2
b3
b>0
a>0
a>b
a≤0
b<0
a≤b
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
74

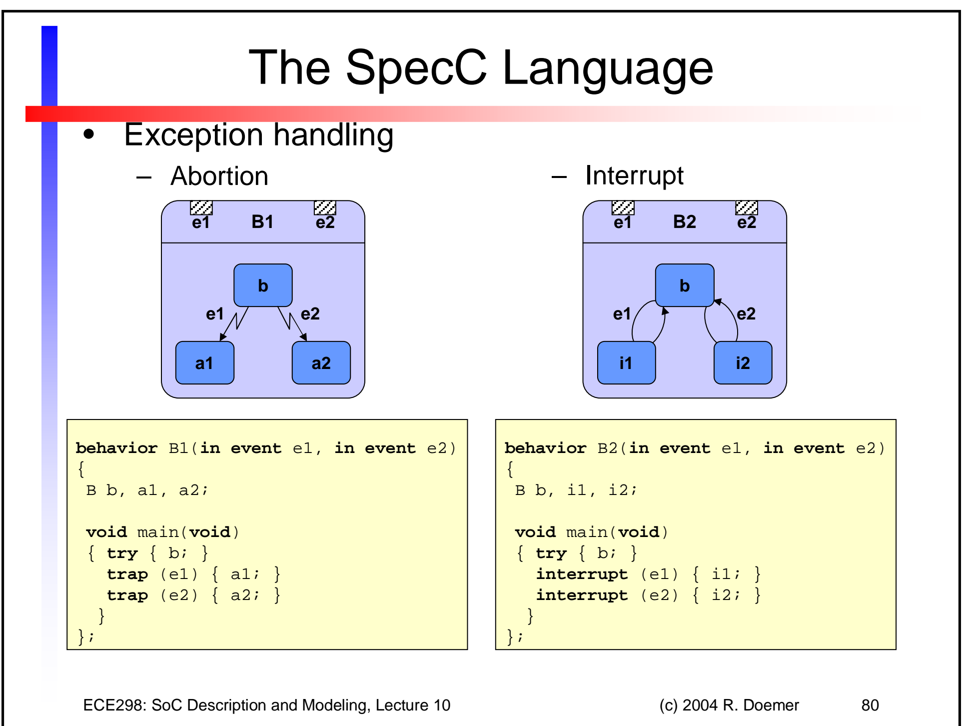

The SpecC Language
• Pipeline
– Explicit execution in pipeline fashion
• pipe { <instance_list> };
Pipeline
b1
b2
b3
behavior Pipeline
{
Stage1 b1;
Stage2 b2;
Stage3 b3;
void main(void)
{
pipe
{ b1;
b2;
b3;
}
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
75

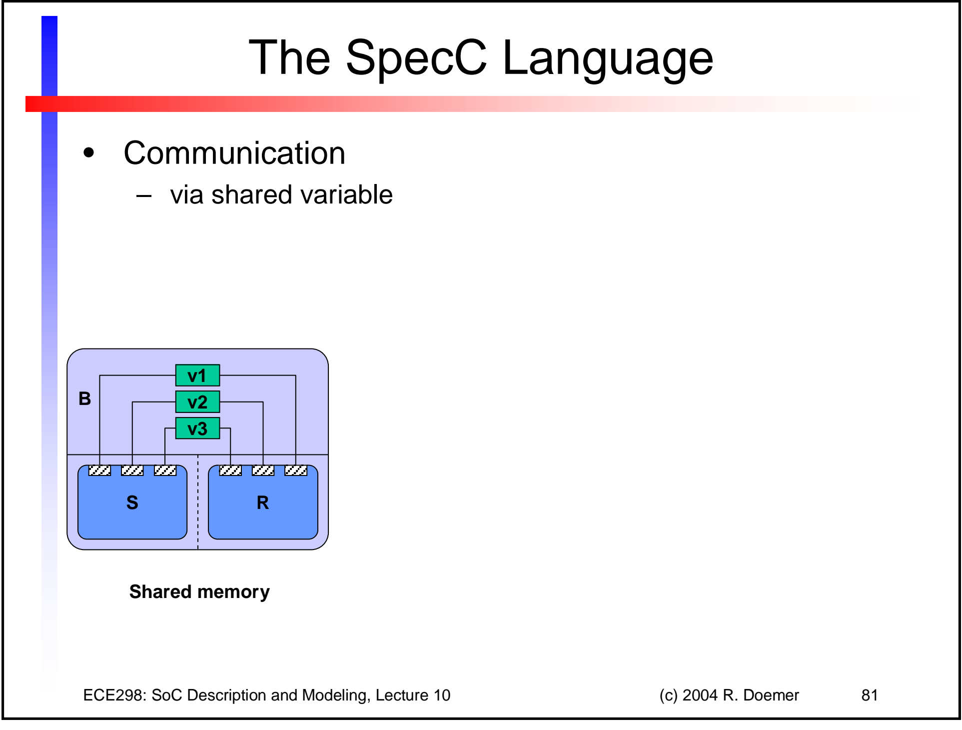

The SpecC Language
• Pipeline
– Explicit execution in pipeline fashion
• pipe { <instance_list> };
• pipe (<init>; <cond>; <incr>) { … }
Pipeline
b1
b2
b3
behavior Pipeline
{
Stage1 b1;
Stage2 b2;
Stage3 b3;
void main(void)
{
int i;
pipe(i=0; i<10; i++)
{ b1;
b2;
b3;
}
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
76

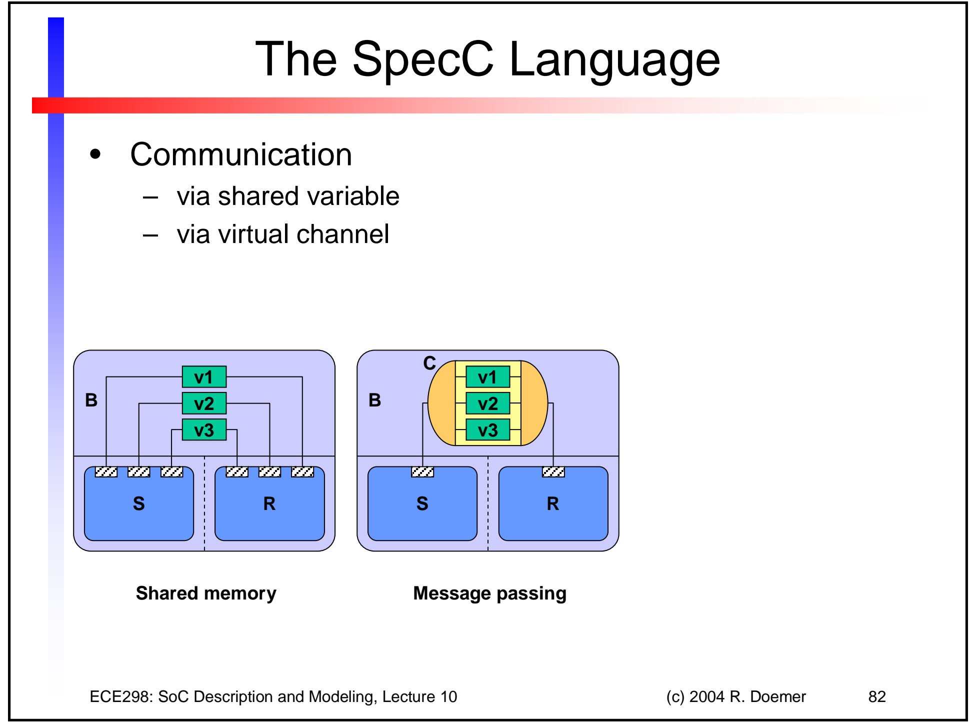

The SpecC Language
Pipeline
Explicit execution in pipeline fashion
pipe { <instance_list> };
pipe (<init>; <cond>; <incr>) { … }
Support for automatic buffering
Pipeline
v1
v2
v3
b1
b2
b3
behavior Pipeline
{
int v1;
int v2;
int v3;
Stage1 b1(v1, v2);
Stage2 b2(v2, v3);
Stage3 b3(v3, v1);
void main(void)
{
int i;
pipe(i=0; i<10; i++)
{ b1;
b2;
b3;
}
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
77

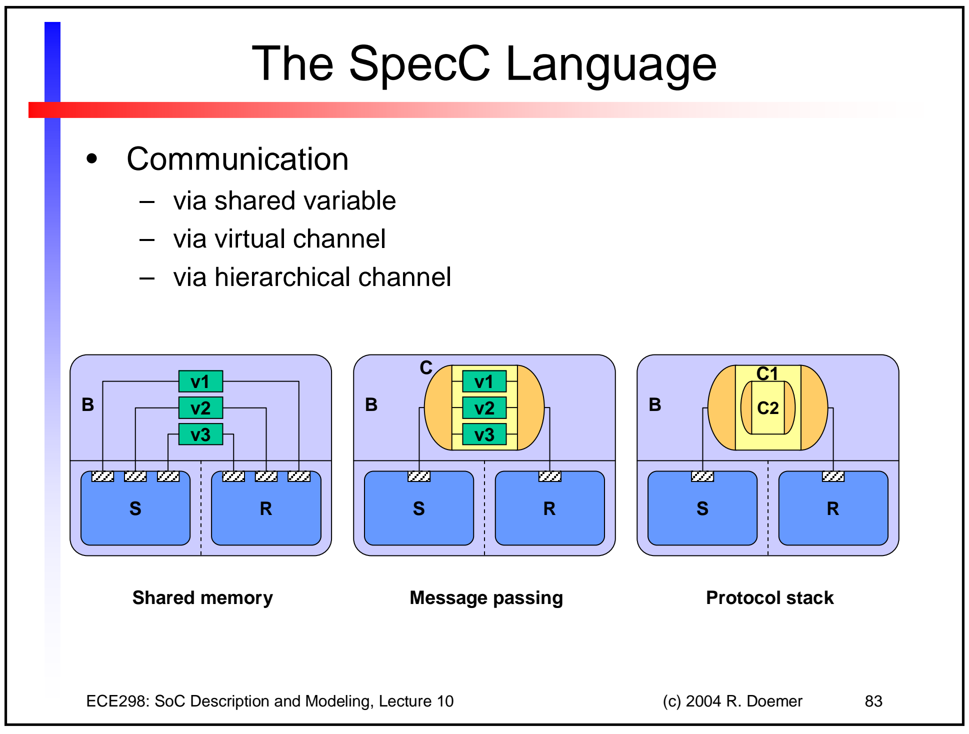

The SpecC Language
Pipeline
Explicit execution in pipeline fashion
pipe { <instance_list> };
pipe (<init>; <cond>; <incr>) { … }
Support for automatic buffering
piped […] <type> <variable_list>;
Pipeline
v1
v2
v3
b1
b2
b3
behavior Pipeline
{
piped piped int v1;
piped int v2;
piped int v3;
Stage1 b1(v1, v2);
Stage2 b2(v2, v3);
Stage3 b3(v3, v1);
void main(void)
{
int i;
pipe(i=0; i<10; i++)
{ b1;
b2;
b3;
}
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
78

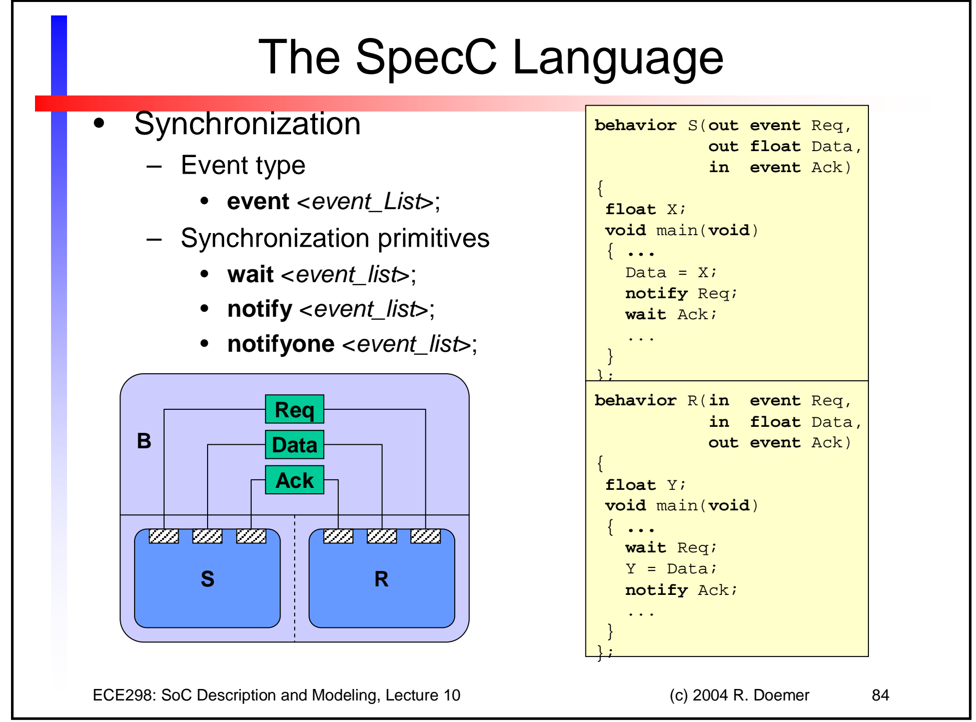

The SpecC Language
• Exception handling
– Abortion
– Interrupt
e1
B1
e2
b
e1
e2
a1
a2
behavior B1(in event e1, in event e2)
{
B b, a1, a2;
void main(void)
{ try { b; }
trap (e1) { a1; }
trap (e2) { a2; }
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
79

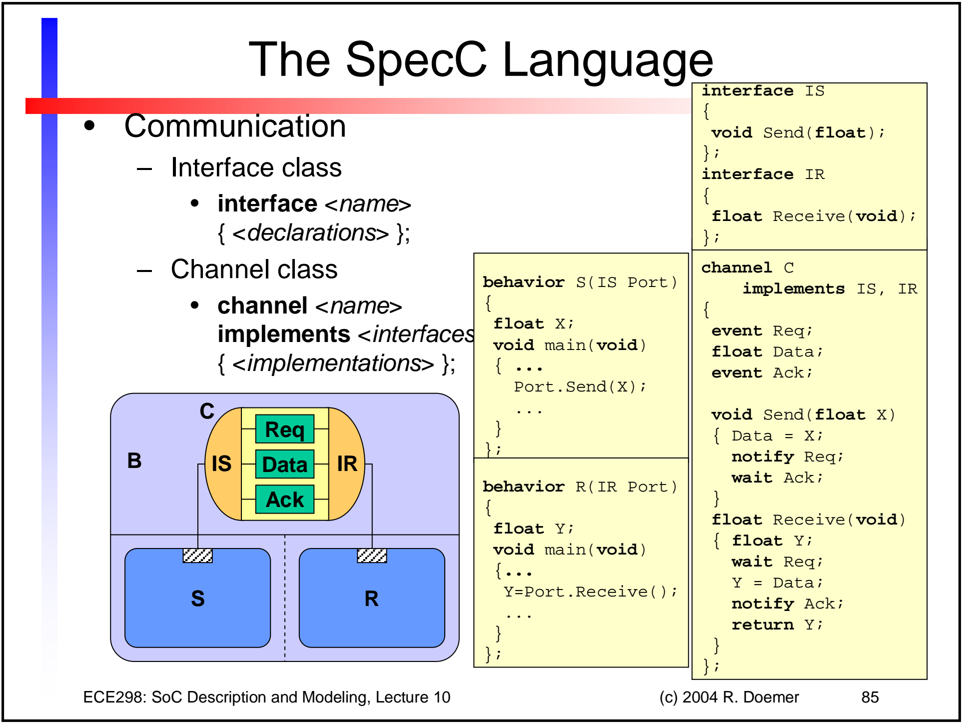

The SpecC Language
• Exception handling
– Abortion
– Interrupt
e1
B1
e2
b
e1
e2
a1
a2
e1
B2
e2
b
e1
e2
i1
i2
behavior B1(in event e1, in event e2)
{
B b, a1, a2;
void main(void)
{ try { b; }
trap (e1) { a1; }
trap (e2) { a2; }
}
};
behavior B2(in event e1, in event e2)
{
B b, i1, i2;
void main(void)
{ try { b; }
interrupt (e1) { i1; }
interrupt (e2) { i2; }
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
80


The SpecC Language
• Communication
– via shared variable
B
v1
v2
v3
S
R
Shared memory
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
81

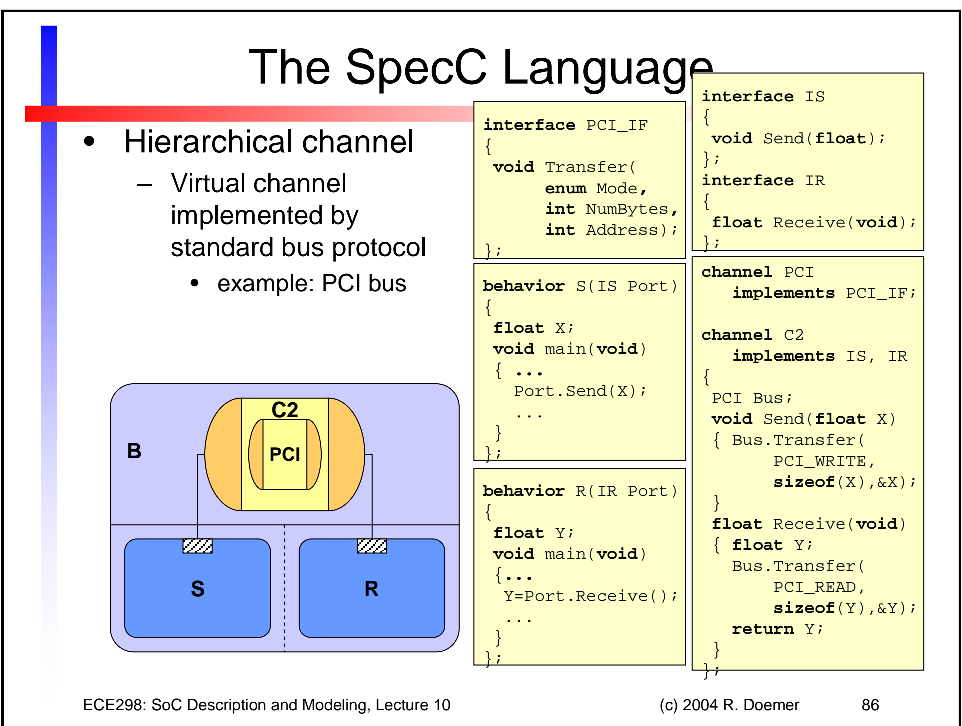

The SpecC Language
• Communication
– via shared variable
– via virtual channel
B
v1
v2
v3
S
R
Shared memory
C
B
v1
v2
v3
S
R
Message passing
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
82


The SpecC Language
• Communication
– via shared variable
– via virtual channel
– via hierarchical channel
B
v1
v2
v3
S
R
C
C1
C2
Shared memory
Message passing
Protocol stack
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
83


The SpecC Language
• Synchronization
– Event type
• event <event_List>;
– Synchronization primitives
• wait <event_list>;
• notify <event_list>;
• notifyone <event_list>;
B
Req
Data
Ack
S
R
behavior S(out event Req,
out float Data,
in event Ack)
{
float X;
void main(void)
{ ...
Data = X;
notify Req;
wait Ack;
...
}
};
behavior R(in event Req,
in float Data,
out event Ack)
{
float Y;
void main(void)
{ ...
wait Req;
Y = Data;
notify Ack;
...
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
84



The SpecC Language
• Communication
– Interface class
• interface <name> { <declarations> };
– Channel class
• channel <name> implements <interfaces> { <implementations> };
C
B
IS
Req
Data
Ack
IR
S
R
behavior S(IS Port)
{
float X;
void main(void)
{ ...
Port.Send(X);
...
}
};
behavior R(IR Port)
{
float Y;
void main(void)
{...
Y=Port.Receive();
...
}
};
interface IS
{
void Send(float);
};
interface IR
{
float Receive(void);
};
channel C
implements IS, IR
{
event Req;
float Data;
event Ack;
void Send(float X)
{ Data = X;
notify Req;
wait Ack;
}
float Receive(void)
{ float Y;
wait Req;
Y = Data;
notify Ack;
return Y;
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
85

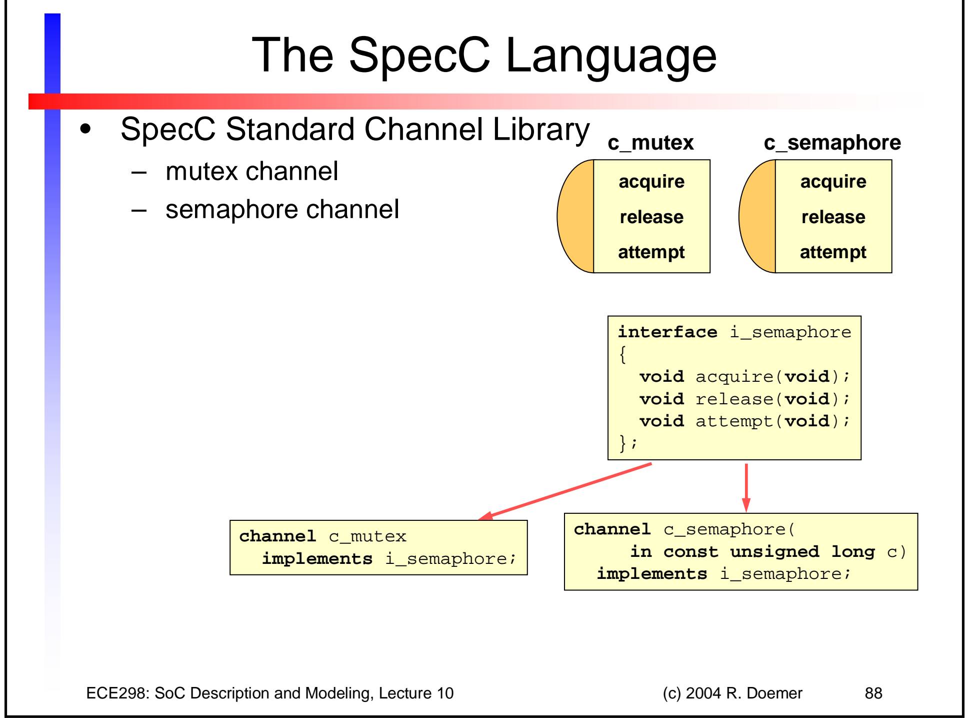


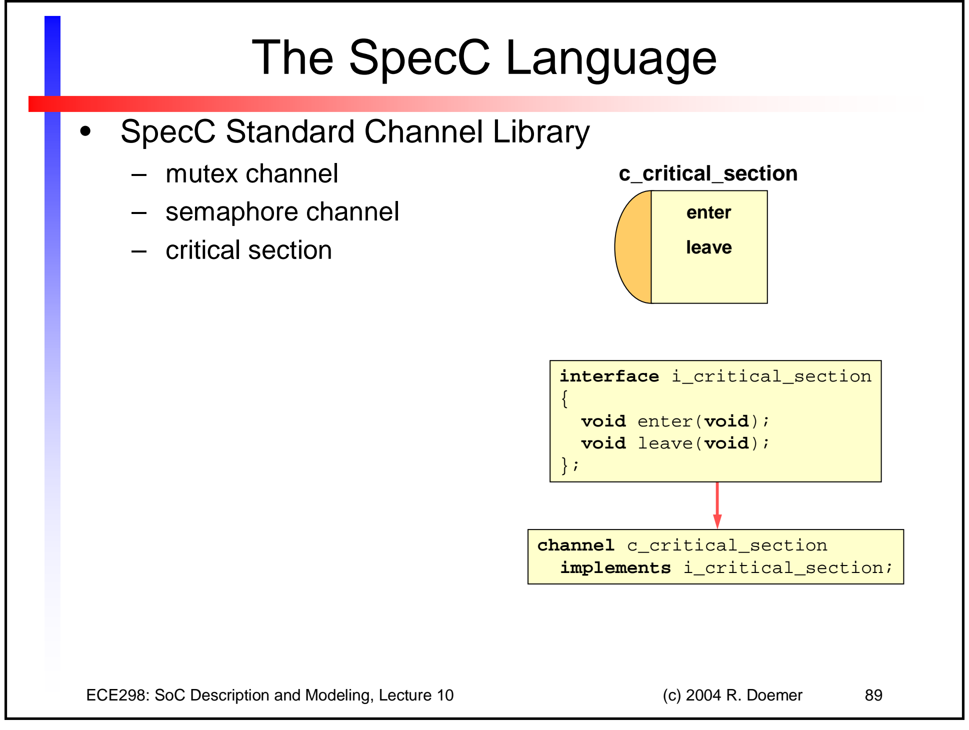
The SpecC Language
• Hierarchical channel
– Virtual channel implemented by standard bus protocol
• example: PCI bus
C2
PCI
B
S
R
interface PCI_IF
{
void Transfer(
enum Mode,
int NumBytes,
int Address);
};
behavior S(IS Port)
{
float X;
void main(void)
{ ...
Port.Send(X);
...
}
};
behavior R(IR Port)
{
float Y;
void main(void)
{...
Y=Port.Receive();
...
}
};
interface IS
{
void Send(float);
};
interface IR
{
float Receive(void);
};
channel PCI
implements PCI_IF;
channel C2
implements IS, IR
{
PCI Bus;
void Send(float X)
{ Bus.Transfer(
PCI_WRITE,
sizeof(X),&X);
}
float Receive(void)
{ float Y;
Bus.Transfer(
PCI_READ,
sizeof(Y),&Y);
return Y;
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
86

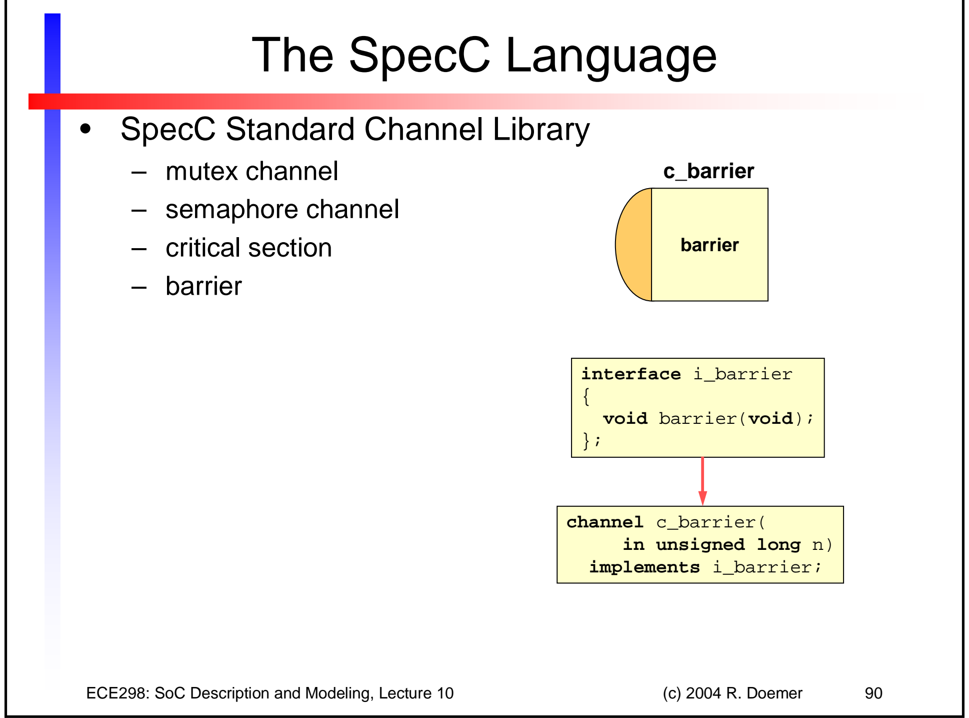
The SpecC Language

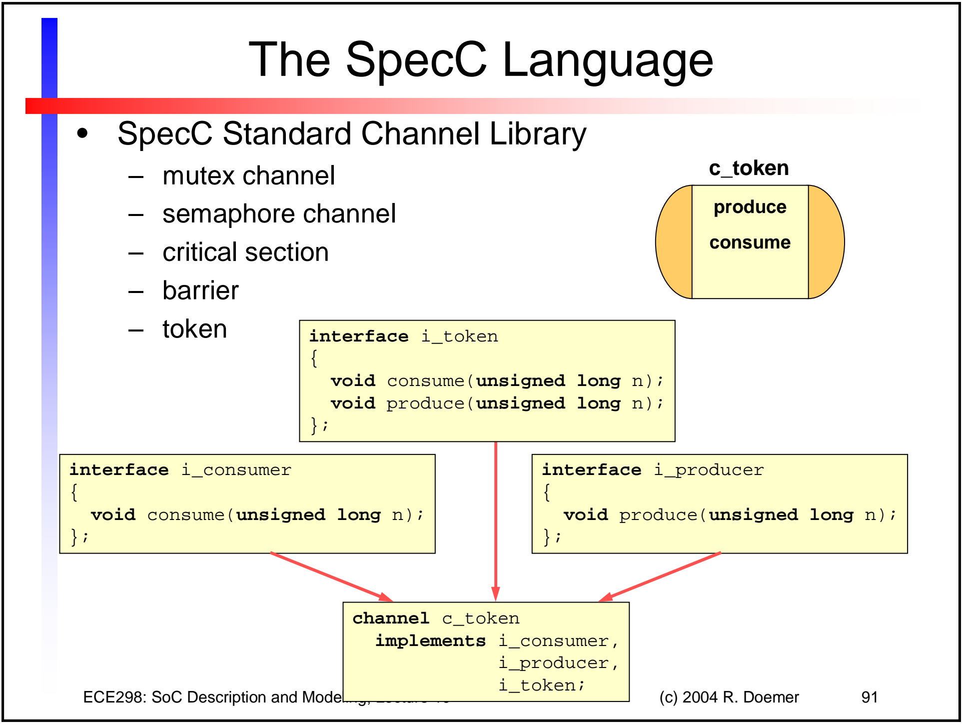
- SpecC Standard Channel Library
  - introduced with SpecC Language Version 2.0
  - includes support for
    - mutex
    - semaphore
    - critical section
    - barrier
    - token
    - queue
    - handshake
    - double handshake
    - …

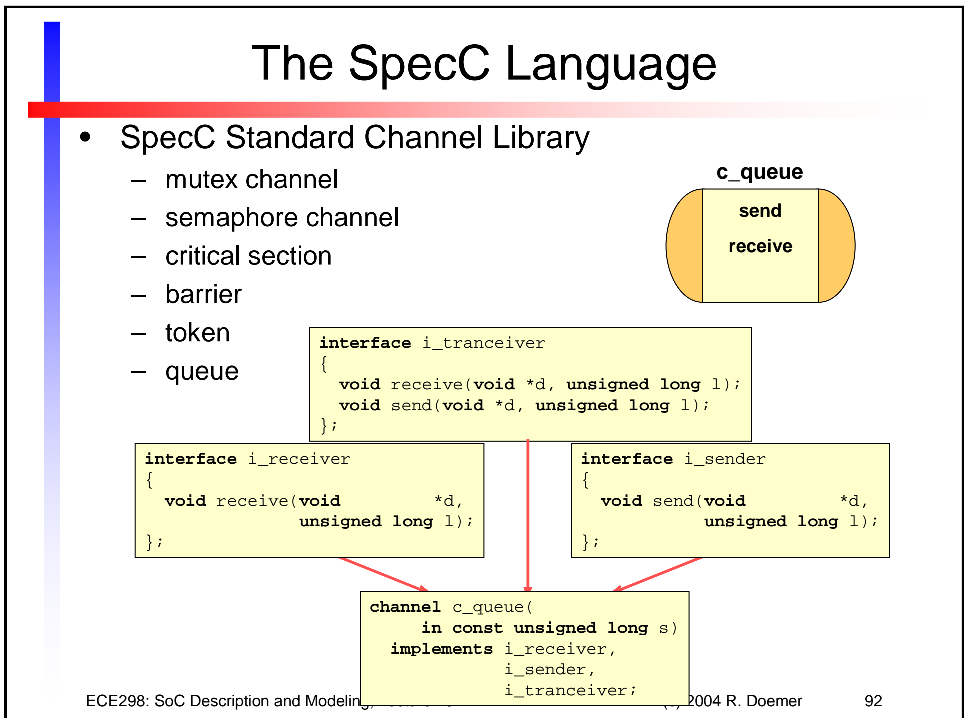
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 87

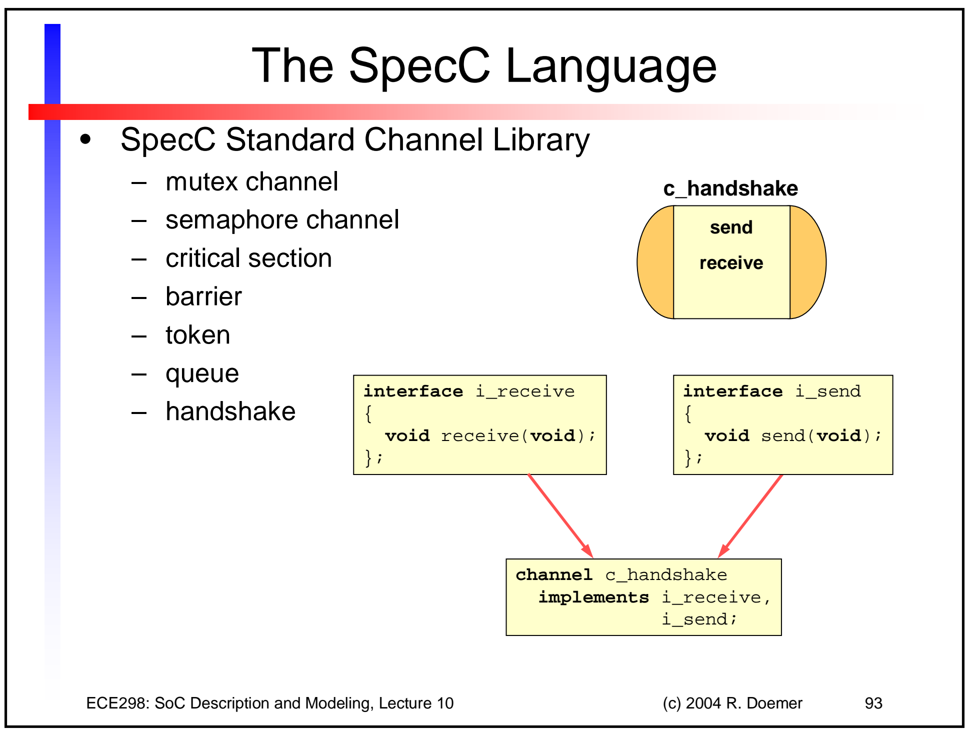


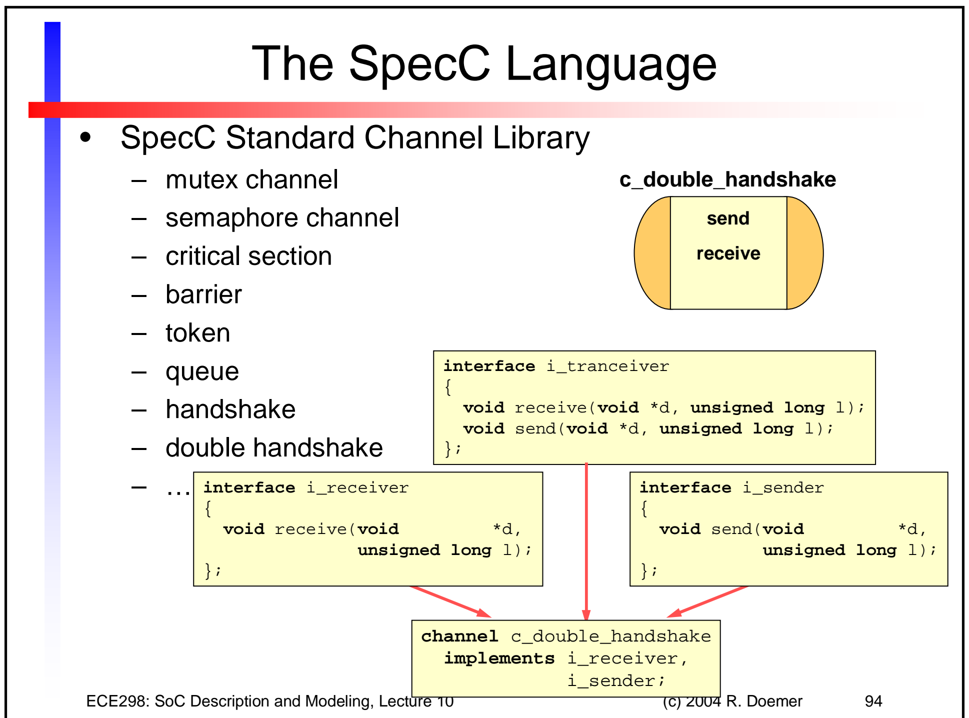

The SpecC Language
• SpecC Standard Channel Library
– mutex channel
– semaphore channel
– critical section
c_critical_section
enter
leave
interface i_critical_section
{
void enter(void);
void leave(void);
};
channel c_critical_section
implements i_critical_section;
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
89

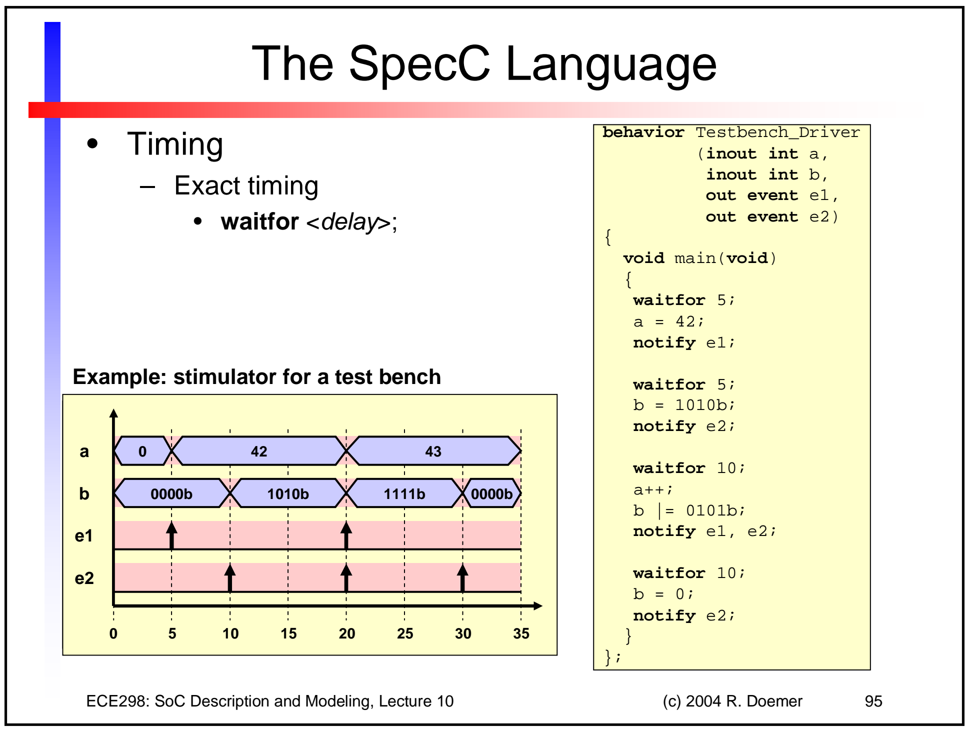

The SpecC Language
• SpecC Standard Channel Library
– mutex channel
– semaphore channel
– critical section
– barrier
c_barrier
barrier
interface i_barrier
{
void barrier(void);
};
channel c_barrier(
in unsigned long n)
implements i_barrier;
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
90

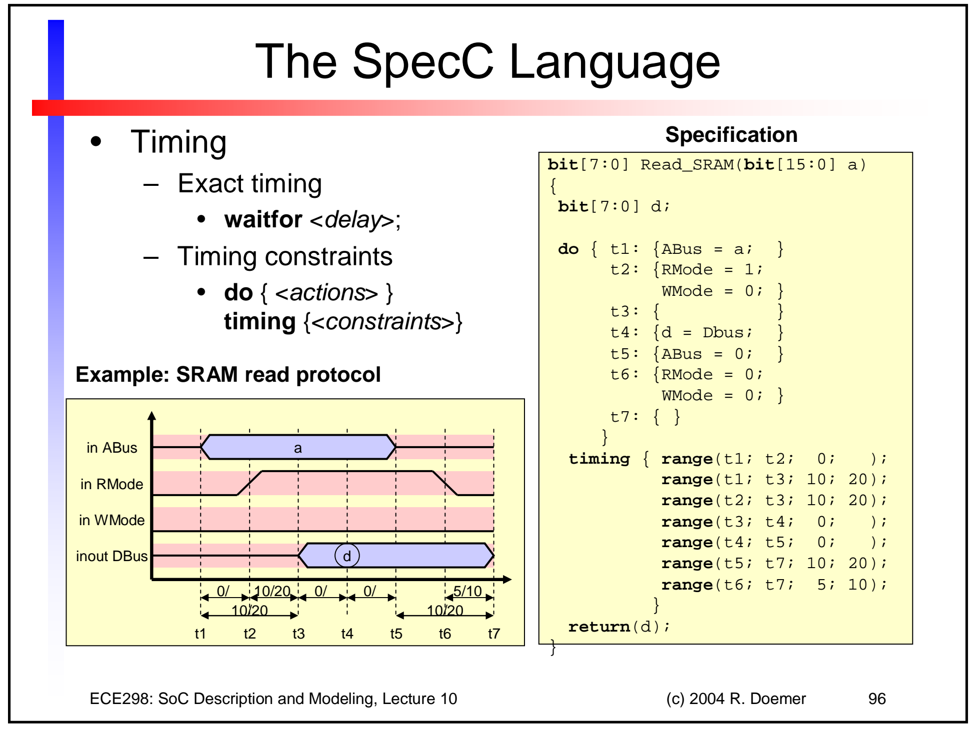

The SpecC Language
• SpecC Standard Channel Library
– mutex channel
– semaphore channel
– critical section
– barrier
– token
c_token
produce
consume
interface i_token
{
void consume(unsigned long n);
void produce(unsigned long n);
};
interface i_consumer
{
void consume(unsigned long n);
};
interface i_producer
{
void produce(unsigned long n);
};
channel c_token
implements i_consumer,
i_producer,
i_token;
(c) 2004 R. Doemer
91

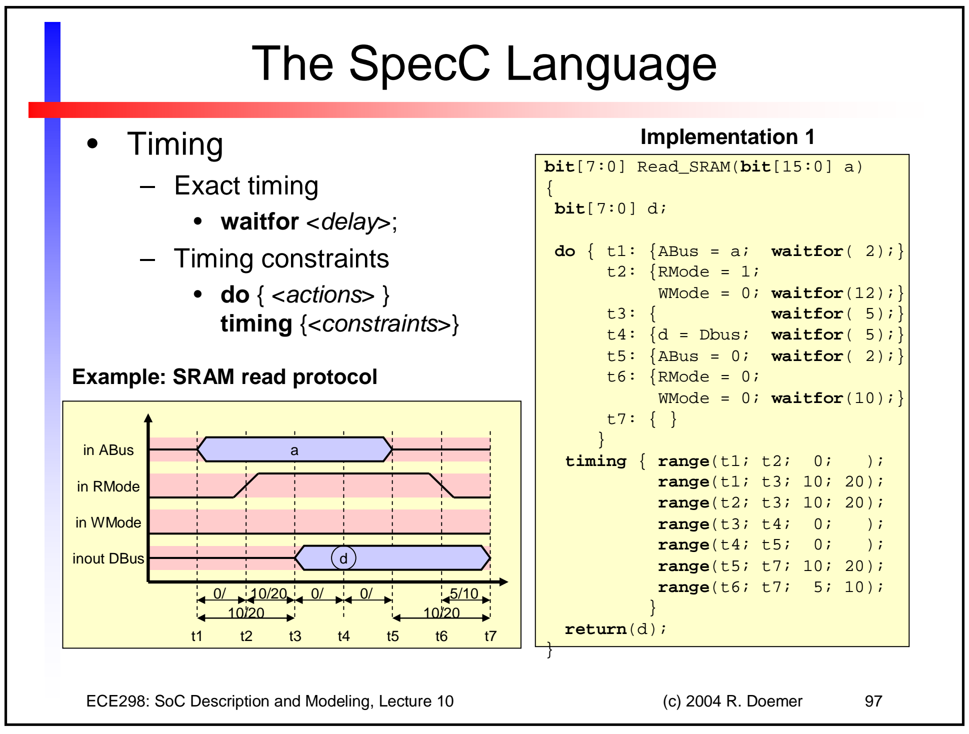

The SpecC Language
• SpecC Standard Channel Library
– mutex channel
– semaphore channel
– critical section
– barrier
– token
– queue
c_queue
send
receive
interface i_tranceiver
{
void receive(void *d, unsigned long l);
void send(void *d, unsigned long l);
};
interface i_receiver
{
void receive(void *d,
unsigned long l);
};
interface i_sender
{
void send(void *d,
unsigned long l);
};
channel c_queue(
in const unsigned long s)
implements i_receiver,
i_sender,
i_tranceiver;
92

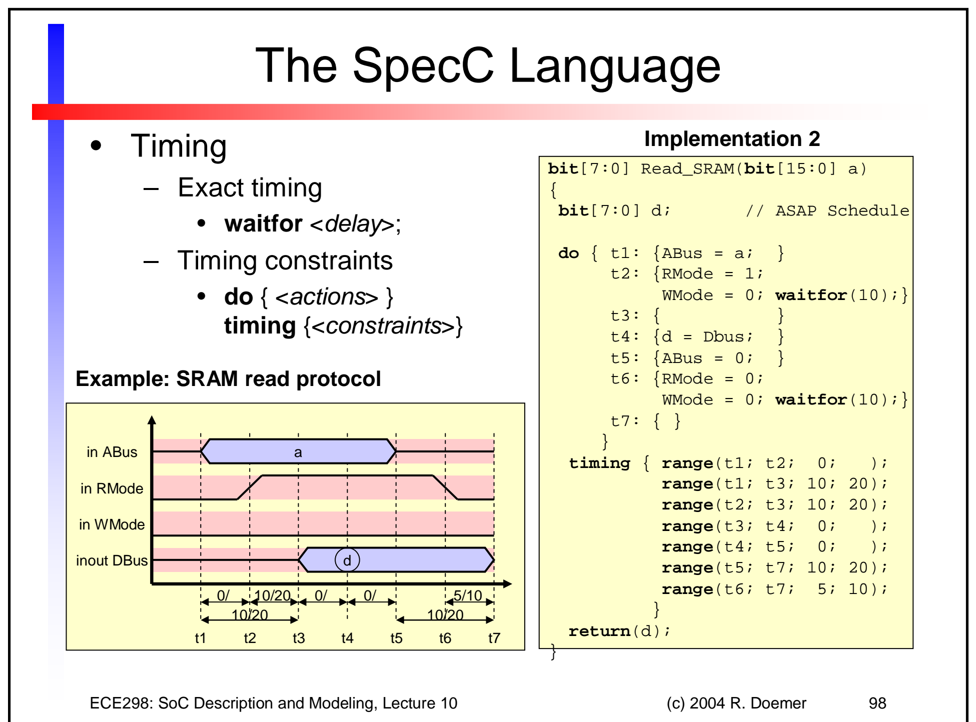

The SpecC Language
• SpecC Standard Channel Library
– mutex channel
– semaphore channel
– critical section
– barrier
– token
– queue
– handshake
c_handshake
send
receive
interface i_receive
{
void receive(void);
};
interface i_send
{
void send(void);
};
channel c_handshake
implements i_receive,
i_send;
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
93


The SpecC Language
• SpecC Standard Channel Library
– mutex channel
– semaphore channel
– critical section
– barrier
– token
– queue
– handshake
– double handshake
– …
c_double_handshake
send
receive
interface i_tranceiver
{
void receive(void *d, unsigned long l);
void send(void *d, unsigned long l);
};
interface i_receiver
{
void receive(void *d,
unsigned long l);
};
interface i_sender
{
void send(void *d,
unsigned long l);
};
channel c_double_handshake
implements i_receiver,
i_sender;
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
94


The SpecC Language
• Timing
– Exact timing
• waitfor <delay>;
Example: stimulator for a test bench
a
b
e1
e2
0
42
43
0000b
1010b
1111b
0000b
0
5
10
15
20
25
30
35
behavior Testbench_Driver
(inout int a,
inout int b,
out event e1,
out event e2)
{
void main(void)
{
waitfor 5;
a = 42;
notify e1;
waitfor 5;
b = 1010b;
notify e2;
waitfor 10;
a++;
b |= 0101b;
notify e1, e2;
waitfor 10;
b = 0;
notify e2;
}
};
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
95


The SpecC Language
• Timing
– Exact timing
• waitfor <delay>;
– Timing constraints
• do { <actions> }
timing {<constraints>}
Example: SRAM read protocol
in ABus
in RMode
in WMode
inout DBus
a
d
0/
10/20
0/
0/
5/10
10/20
10/20
t1
t2
t3
t4
t5
t6
t7
Specification
bit[7:0] Read_SRAM(bit[15:0] a)
{
bit[7:0] d;
do { t1: {ABus = a; }
t2: {RMode = 1;
WMode = 0; }
t3: { }
t4: {d = Dbus; }
t5: {ABus = 0; }
t6: {RMode = 0;
WMode = 0; }
t7: { }
}
timing { range(t1; t2; 0;);
range(t1; t3; 10; 20);
range(t2; t3; 10; 20);
range(t3; t4; 0;);
range(t4; t5; 0;);
range(t5; t7; 10; 20);
range(t6; t7; 5; 10);
}
return(d);
}
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
96


The SpecC Language
• Timing
– Exact timing
• waitfor <delay>;
– Timing constraints
• do { <actions> }
timing {<constraints>}
Example: SRAM read protocol
in ABus
in RMode
in WMode
inout DBus
a
d
0/
10/20
0/
0/
5/10
10/20
10/20
t1
t2
t3
t4
t5
t6
t7
Implementation 1
bit[7:0] Read_SRAM(bit[15:0] a)
{
bit[7:0] d;
do { t1: {ABus = a; waitfor(2);}
t2: {RMode = 1;
WMode = 0; waitfor(12);}
t3: { waitfor(5);}
t4: {d = Dbus; waitfor(5);}
t5: {ABus = 0; waitfor(2);}
t6: {RMode = 0;
WMode = 0; waitfor(10);}
t7: { }
}
timing { range(t1; t2; 0;);
range(t1; t3; 10; 20);
range(t2; t3; 10; 20);
range(t3; t4; 0;);
range(t4; t5; 0;);
range(t5; t7; 10; 20);
range(t6; t7; 5; 10);
}
return(d);
}
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
97


The SpecC Language
• Timing
– Exact timing
• waitfor <delay>;
– Timing constraints
• do { <actions> }
timing {<constraints>}
Example: SRAM read protocol
in ABus
in RMode
in WMode
inout DBus
a
d
0/
10/20
0/
0/
5/10
10/20
10/20
t1
t2
t3
t4
t5
t6
t7
Implementation 2
bit[7:0] Read_SRAM(bit[15:0] a)
{
bit[7:0] d; // ASAP Schedule
do { t1: {ABus = a; }
t2: {RMode = 1;
WMode = 0; waitfor(10);}
t3: { }
t4: {d = Dbus; }
t5: {ABus = 0; }
t6: {RMode = 0;
WMode = 0; waitfor(10);}
t7: { }
}
timing { range(t1; t2; 0;);
range(t1; t3; 10; 20);
range(t2; t3; 10; 20);
range(t3; t4; 0;);
range(t4; t5; 0;);
range(t5; t7; 10; 20);
range(t6; t7; 5; 10);
}
return(d);
}
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
98

The SpecC Language

- RTL Modeling
  - Accellera RTL Semantics Standard
    - Style 1: *unmapped*
      - `a = b * c`
    - Style 2: *storage mapped*
      - `R1 = R1 * RF2[4]`
    - Style 3: *function mapped*
      - `R1 = ALU1(MULT, R1, RF2[4])`
    - Style 4: *connection mapped*
      - `Bus1=R1; Bus2=RF2[4]; Bus3=ALU1(MULT, Bus1, Bus2)`
    - Style 5: *exposed control*
      - `ALU_CTRL = 011001; RF2_CTRL = 010; …`

Source: http://www.eda.org/alc-cwg/cwg-open.pdf

  - Types specific for RTL:
    - **`buffered`** type modifier: Storage
    - **`signal`** type modifier: Communication
  - Control flow specific for RTL:
    - **`fsmd`** construct: Explicit finite state machine with datapath

The SpecC Language

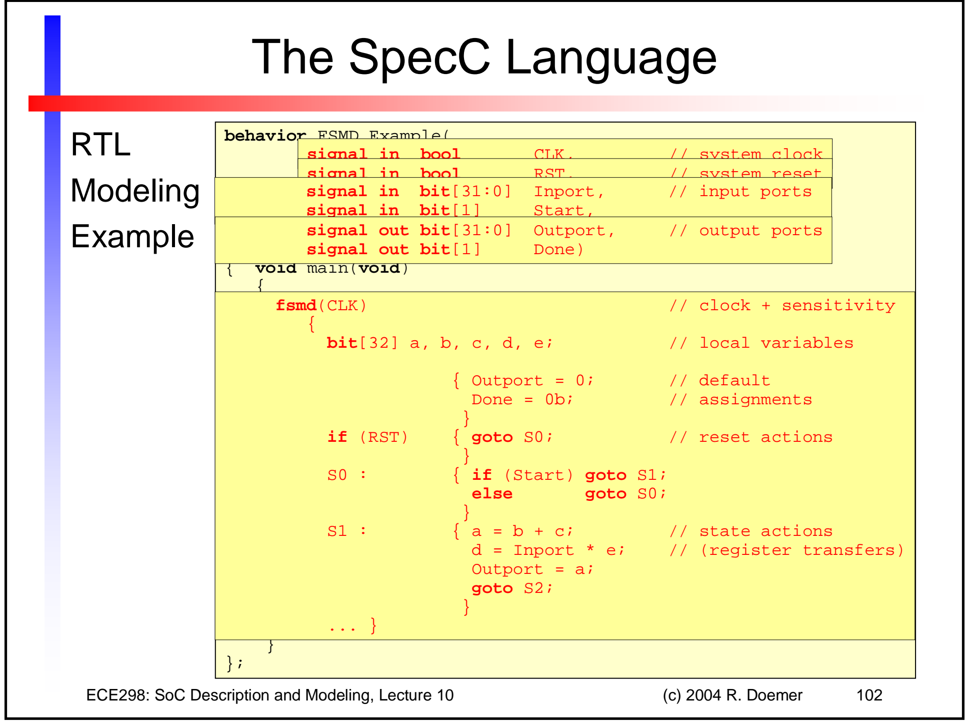
- RTL Modeling
  - **`fsmd`** construct
    - Similar to **`fsm`** construct, but specifically for RTL
    - Explicit states and state transitions
    - State actions represent well-defined register transfers
      - limited to conditional/unconditional assignments and function calls
      - general loops, exceptions, synchronization, timing are not allowed
    - Explicit clock specifier
      - event list (external clock)
      - time delay (internal clock)
    - Explicit sensitivity list
      - needed for Mealy machine support
    - Explicit reset state
      - synchronous reset
      - asynchronous reset

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool        CLK,         // system clock
        signal in  bool        RST,         // system reset
        signal in  bit[31:0]   Inport,      // input ports
        signal in  bit[1]      Start,
        signal out bit[31:0]   Outport,     // output ports
        signal out bit[1]      Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
          {
            bit[32] a, b, c, d, e;          // local variables

                        { Outport = 0;      // default
                          Done = 0b;        // assignments
                        }
            if (RST)    { goto S0;          // reset actions
                        }
            S0 :        { if (Start) goto S1;
                          else       goto S0;
                        }
            S1 :        { a = b + c;        // state actions
                          d = Inport * e;   // (register transfers)
                          Outport = a;
                          goto S2;
                        }
            ... }
    }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 101



The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool        CLK,         // system clock
        signal in  bool        RST,         // system reset
        signal in  bit[31:0]   Inport,      // input ports
        signal in  bit[1]      Start,
        signal out bit[31:0]   Outport,     // output ports
        signal out bit[1]      Done)
{  void main(void)
   {
    fsmd(CLK; Inport, Start)                // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                       { Outport = 0;       // default
                         Done = 0b;         // assignments
                       }
          if (RST)     { goto S0;           // reset actions
                       }
          S0 :         { if (Start) goto S1;
                         else       goto S0;
                       }
          S1 :         { a = b + c;         // state actions
                         d = Inport * e;    // (register transfers)
                         Outport = a;
                         goto S2;
                       }
          ... }
    }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 103

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool        CLK,         // system clock
        signal in  bool        RST,         // system reset
        signal in  bit[31:0]   Inport,      // input ports
        signal in  bit[1]      Start,
        signal out bit[31:0]   Outport,     // output ports
        signal out bit[1]      Done)
{  void main(void)
   {
    fsmd(CLK; RST)                          // asynchronous reset
        {
          bit[32] a, b, c, d, e;            // local variables

                       { Outport = 0;       // default
                         Done = 0b;         // assignments
                       }
          if (RST)     { goto S0;           // reset actions
                       }
          S0 :         { if (Start) goto S1;
                         else       goto S0;
                       }
          S1 :         { a = b + c;         // state actions
                         d = Inport * e;    // (register transfers)
                         Outport = a;
                         goto S2;
                       }
          ... }
    }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 104

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{   void main(void)
    {
      fsmd(CLK)                             // clock + sensitivity
      {
        bit[32] a, b, c, d, e;              // local variables

                    { Outport = 0;          // default
                      Done = 0b;            // assignments
                    }
        if (RST)    { goto S0;              // reset actions
                    }
        S0 :        { if (Start) goto S1;
                      else       goto S0;
                    }
        S1 :        { a = b + c;            // state actions
                      d = Inport * e;       // (register transfers)
                      Outport = a;
                      goto S2;
                    }
        ... }
    }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 105

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{   void main(void)
    {
      fsmd(CLK)                             // clock + sensitivity
      {
        bit[32] a, b, c, d, e;              // local variables

                    { Outport = 0;          // default
                      Done = 0b;            // assignments
                    }
        if (RST)    { goto S0;              // reset actions
                    }
        S0 :        { if (Start) goto S1;
                      else       goto S0;
                    }
        S1 :        { a = b + c;            // state actions
                      d = Inport * e;       // (register transfers)
                      Outport = a;
                      goto S2;
                    }
        ... }
    }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 106

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{   void main(void)
    {
      fsmd(CLK)                             // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                        { Outport = 0;      // default
                          Done = 0b;        // assignments
                        }
          if (RST)      { goto S0;          // reset actions
                        }
          S0 :          { if (Start) goto S1;
                          else       goto S0;
                        }
          S1 :          { a = b + c;        // state actions
                          d = Inport * e;   // (register transfers)
                          Outport = a;
                          goto S2;
                        }
          ... }
    }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 107

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{   void main(void)
    {
      fsmd(CLK)                             // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // unmapped variables

                        { Outport = 0;      // default
                          Done = 0b;        // assignments
                        }
          if (RST)      { goto S0;          // reset actions
                        }
          S0 :          { if (Start) goto S1;
                          else       goto S0;
                        }
          S1 :          { a = b + c;        // Accellera style 1
                          d = Inport * e;   // (unmapped)
                          Outport = a;
                          goto S2;
                        }
          ... }
    }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 108

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,         // system clock
        signal in  bool       RST,         // system reset
        signal in  bit[31:0]  Inport,      // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,     // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                             // clock + sensitivity
        {
         buffered[CLK] bit[32] RF[4];      // register file

                       { Outport = 0;      // default
                         Done = 0b;        // assignments
                       }
          if (RST)     { goto S0;          // reset actions
                       }
          S0 :         { if (Start) goto S1;
                         else       goto S0;
                       }
          S1 :         { RF[0]=RF[1]+RF[2]; // Accellera style 2
                         RF[3]=Inport*RF[4];// (storage mapped)
                         Outport = RF[0];
                         goto S2;
                       }
          ... }
   }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 109

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,         // system clock
        signal in  bool       RST,         // system reset
        signal in  bit[31:0]  Inport,      // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,     // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                             // clock + sensitivity
        {
         buffered[CLK] bit[32] RF[4];      // register file

                       { Outport = 0;      // default
                         Done = 0b;        // assignments
                       }
          if (RST)     { goto S0;          // reset actions
                       }
          S0 :         { if (Start) goto S1;
                         else       goto S0;
                       }
          S1 :         { RF[0] =            // Accellera style 3
                          ADD0(RF[1],RF[2]);// (function mapped)
                         RF[3] =
                          MUL0(Inport,RF[4]);
                         Outport = RF[0];
                         goto S2;
   }                   }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 110

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
        buffered[CLK] bit[32] RF[4];        // register file
        bit[32] BUS0, BUS1, BUS2;           // busses
                       { Outport = 0;       // default
                         Done = 0b;         // assignments
                       }
          if (RST)     { goto S0;           // reset actions
                       }
          S0 :         { if (Start) goto S1;
                         else       goto S0;
                       }
          S1 :         { BUS0 = RF[1];      // Accellera style 4
                         BUS1 = RF[2];      // (connection mapped)
                         BUS3 = ADD0(BUS0,BUS1);
                         RF[0]= BUS3;
                         ...
                         goto S2;
     }                 }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 111

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
        signal bit[5:0] RF_CTRL;            // control wires
        signal bit[1:0] ADD0_CTRL, MUL0_CTRL;
                       { Outport = 0;       // default
                         Done = 0b;         // assignments
                       }
          if (RST)     { goto S0;           // reset actions
                       }
          S0 :         { if (Start) goto S1;
                         else       goto S0;
                       }
          S1 :         { RF_CTRL = 011000b; // Accellera style 5
                         ADD0_CTRL = 01b;   // (exposed control)
                         MUL0_CTRL = 11b;
                         ...

                         goto S2;
     }                 }
};
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 112

The SpecC Language

- Library support
  - Import of precompiled SpecC code
    - **import** <*component_name*>;
  - Automatic handling of multiple inclusion
    - no need to use **#ifdef** - **#endif** around included files
  - Visible to the compiler/synthesizer
    - not inline-expanded by preprocessor
    - simplifies reuse of IP components

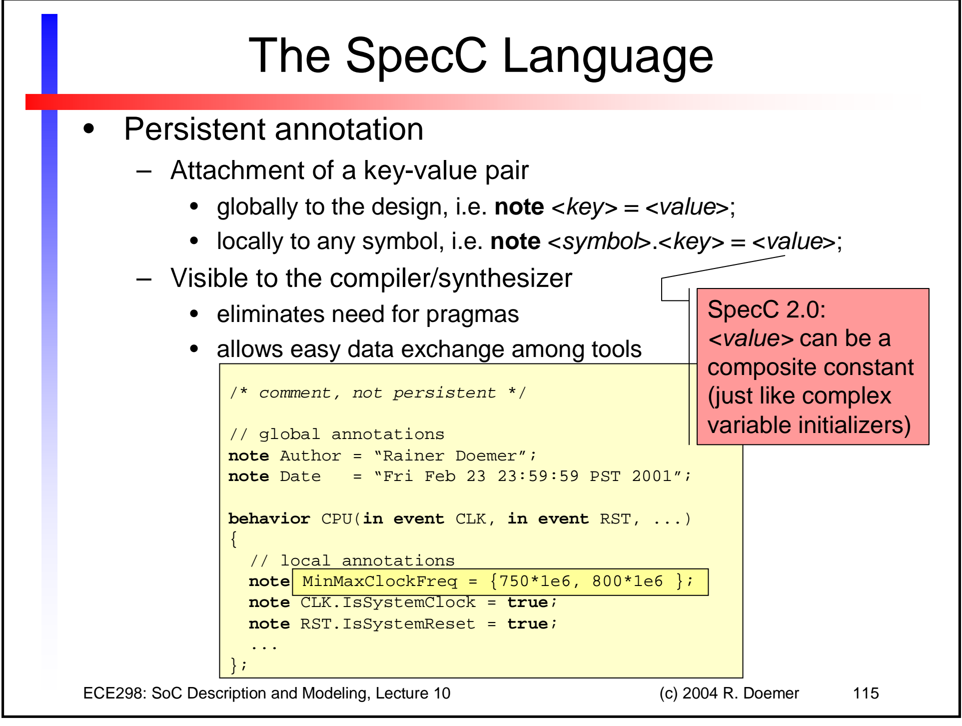
```
// MyDesign.sc

#include <stdio.h>
#include <stdlib.h>

import "Interfaces/I1";
import "Channels/PCI_Bus";
import "Components/MPEG-2";

...
```

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 113

The SpecC Language

- Persistent annotation
  - Attachment of a key-value pair
    - globally to the design, i.e. **note** <*key*> = <*value*>;
    - locally to any symbol, i.e. **note** <*symbol*>.<*key*> = <*value*>;
  - Visible to the compiler/synthesizer
    - eliminates need for pragmas
    - allows easy data exchange among tools

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 114



Summary

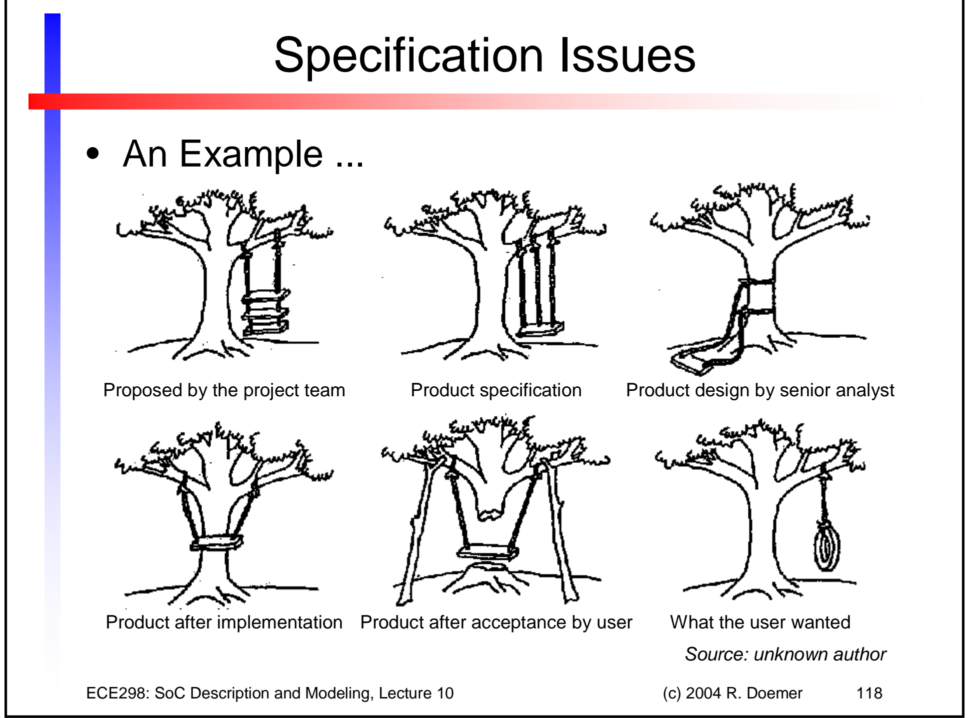
- SpecC model
  - PSM model of computation
  - Separation of communication and computation
  - Hierarchical network of behaviors and channels
- SpecC language
  - True superset of ANSI-C
    - ANSI-C plus extensions for HW-design
  - Support of all concepts needed in system design
    - Structural and behavioral hierarchy
    - Concurrency
    - State transitions
    - Communication
    - Synchronization
    - Exception handling
    - Timing
    - RTL

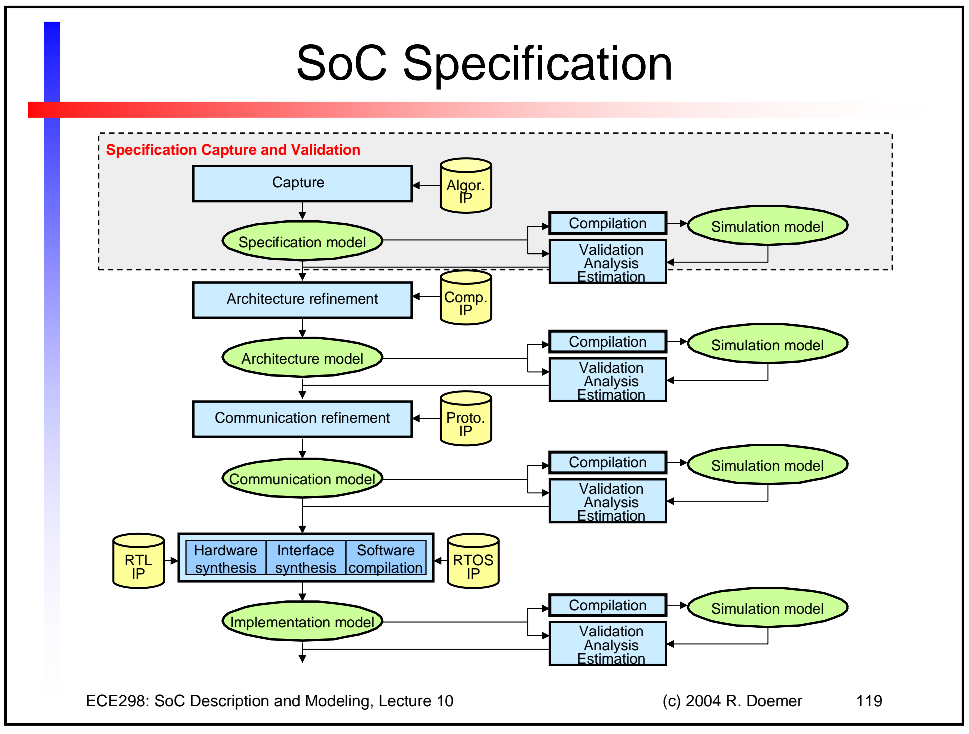
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 116

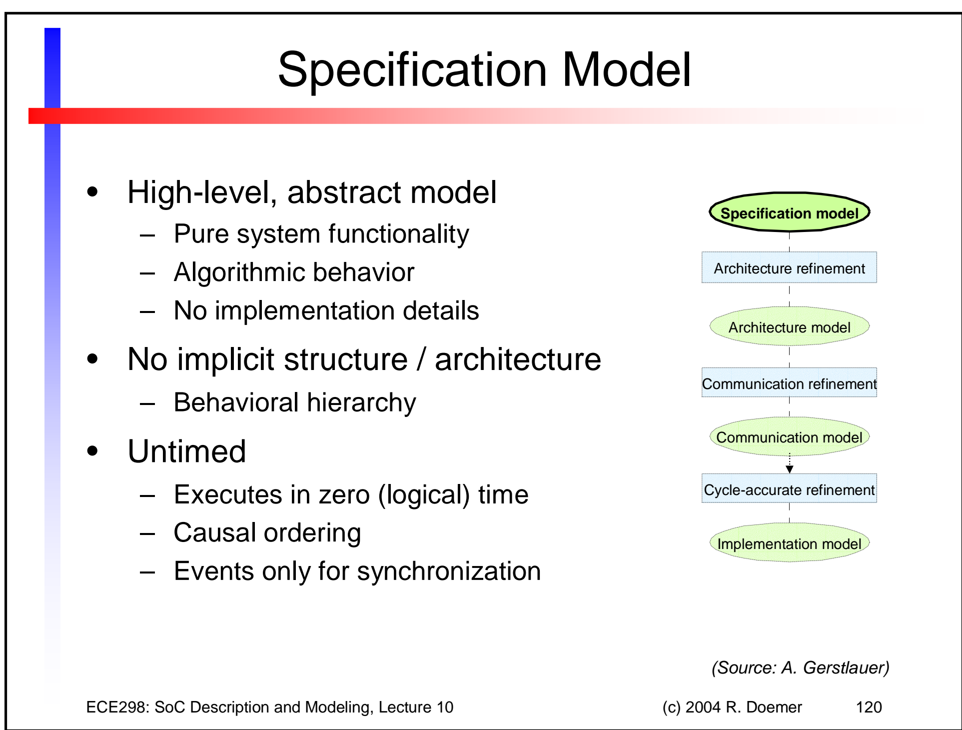
Lecture 10: Overview

- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - ➢ SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 117




SoC Specification
Specification Capture and Validation
Capture
Algor. IP
Specification model
Compilation
Simulation model
Validation Analysis Estimation
Architecture refinement
Comp. IP
Architecture model
Compilation
Simulation model
Validation Analysis Estimation
Communication refinement
Proto. IP
Communication model
Compilation
Simulation model
Validation Analysis Estimation
RTL IP
Hardware synthesis
Interface synthesis
Software compilation
RTOS IP
Implementation model
Compilation
Simulation model
Validation Analysis Estimation
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
119


Specification Model
• High-level, abstract model
– Pure system functionality
– Algorithmic behavior
– No implementation details
• No implicit structure / architecture
– Behavioral hierarchy
• Untimed
– Executes in zero (logical) time
– Causal ordering
– Events only for synchronization
Specification model
Architecture refinement
Architecture model
Communication refinement
Communication model
Cycle-accurate refinement
Implementation model
(Source: A. Gerstlauer)
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
120

Specification Language

- Specification model
  - PSM model of computation
  - Separation of communication and computation
  - Hierarchical network of behaviors and channels
- SpecC language
  - True superset of ANSI-C
    - ANSI-C plus extensions for HW-design
  - Support of all concepts needed in system design
    - Structural and behavioral hierarchy
    - Concurrency
    - State transitions
    - Communication
    - Synchronization
    - Exception handling
    - Timing
    - RTL

Specification Modeling Guidelines

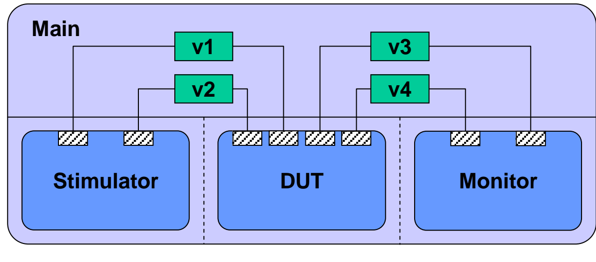
- Specification model
  - First executable system model in the design flow
    - "Golden Model"
    - all other models will be derived from this one
  - High abstraction level
  - Separation of communication and computation
    - channels and behaviors
  - No implementation details
    - unrestricted exploration of design space
  - Pure functional
    - no structural information
  - No timing
    - exception: timing constraints

Specification Modeling Guidelines

- Computation: Behaviors
  - Hierarchy: explicit concurrency, state transitions, ...
  - Granularity: leaf behaviors = smallest indivisible units
  - Encapsulation: localization, explicit dependencies
  - Concurrency: explicitly specified (par, pipe, fsm, seq, …)
  - Time: untimed, partial ordering
- Communication: Channels
  - Semantics: abstract communication, synchronization (standard channel library)
  - Dependencies: explicit data dependency, partial ordering, port connectivity

Specification Modeling Guidelines

- Modeling rules
  - Syntax and semantics
    - SpecC language, Version 2.0
  - Test bench (Stimulator, Monitor)
    - no restrictions in syntax and semantics (no synthesis)
  - DUT (Design under test)
    - restricted by syntax and semantic rules (for synthesis!)



Specification Modeling Guidelines

- Example rules for SpecC environment
  - Clean behavioral hierarchy
    - hierarchical behaviors:
      no code other than par, pipe, seq, fsm, try-trap, ... statements
    - leaf behaviors:
      no child behavior calls (basically pure ANSI-C code)
  - Clean communication
    - point-to-point communication via standard channels
    - ports of plain type or interface type, no pointers!
    - port maps to local variables or ports only
- Detailed rules for SpecC Environment
  - CECS Technical Report 03-21:
    "*System-on-Chip Specification Style Guide*"
    by A. Gerstlauer, K. Ramineni, R. Doemer, D. Gajski
  - http://www.ics.uci.edu/~doemer/publications/CECS_TR_03_21.pdf

Specification Modeling Guidelines

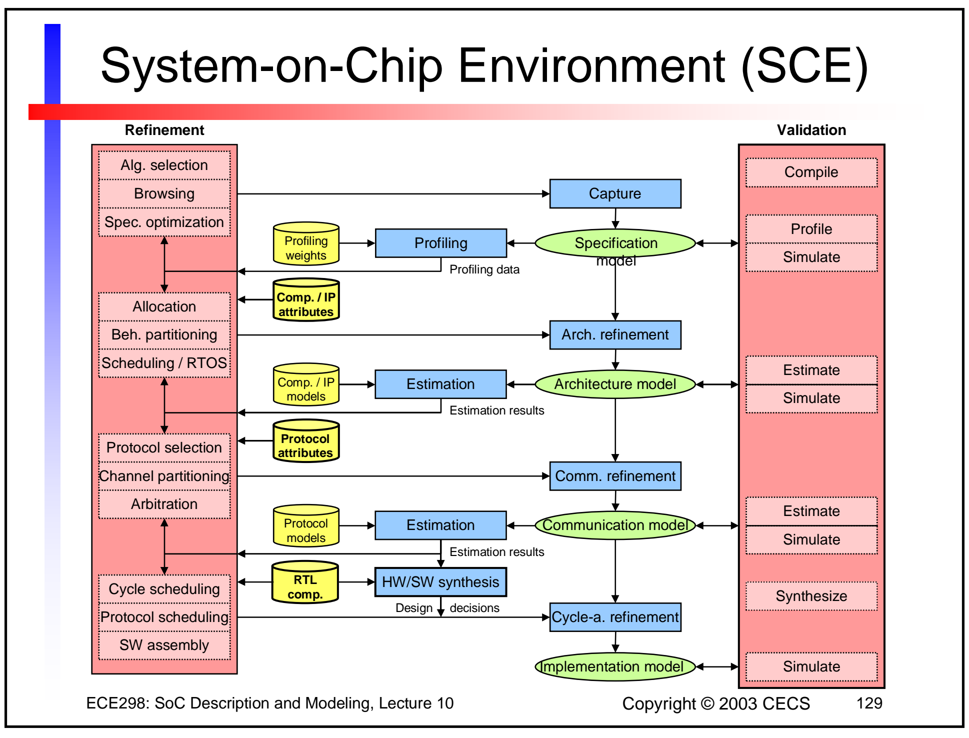
- C code conversion
  - Functions become behaviors or channels
  - Functional hierarchy becomes behavioral hierarchy
    - clean behavioral hierarchy required
    - if-then-else structure becomes FSM
    - while/for/do loops become FSM
  - Explicitly specify potential parallelism
  - Explicitly specify communication
    - avoid global variables
    - use local variables and ports (signals, wires)
    - use standard channels
  - Data types
    - avoid pointers, use arrays instead
    - use explicit SpecC data types if suitable

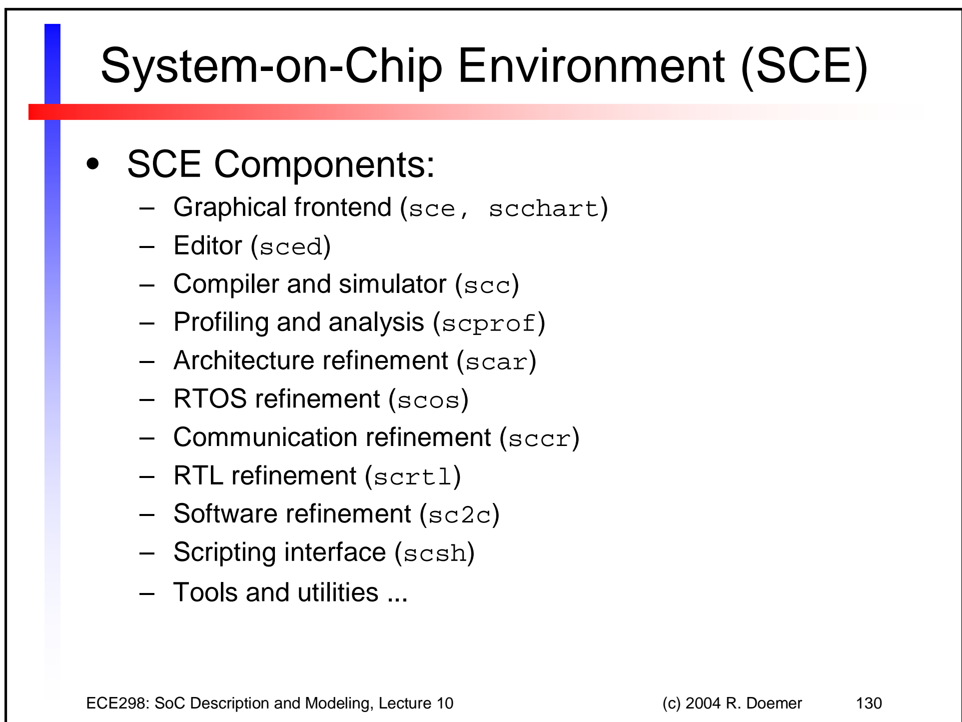
Specification Example

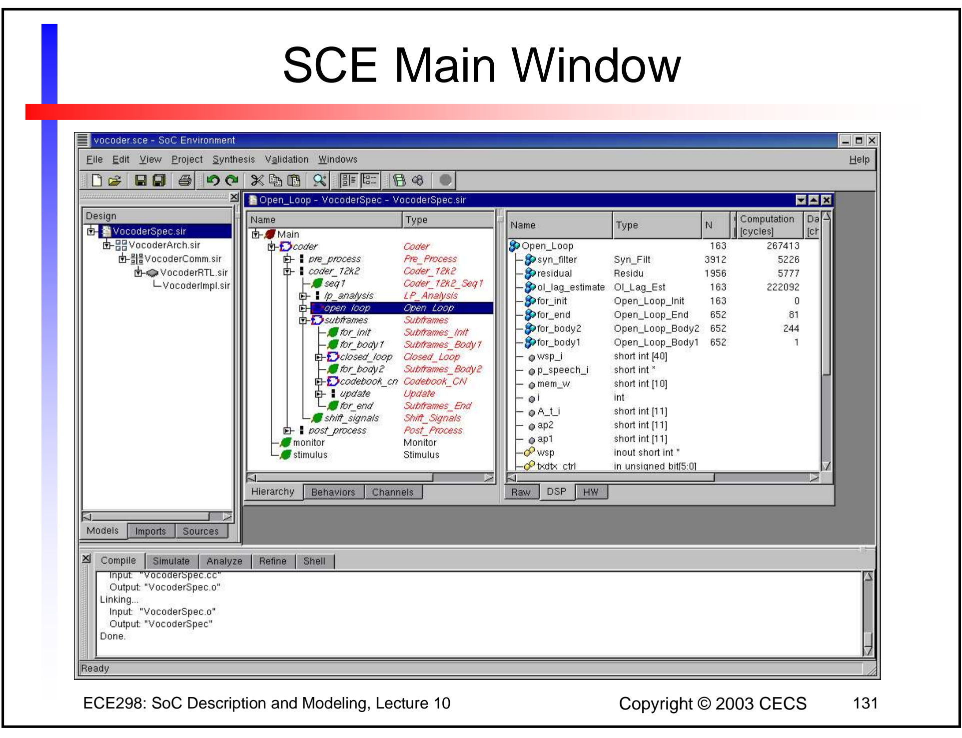
- Design example: GSM Vocoder
  - Enhanced full-rate voice codec
  - GSM standard for mobile telephony (GSM 06.10)
  - Lossy voice encoding/decoding
    - Incoming speech samples @ 104 kbit/s
    - Encoded bit stream @ 12.2 kbit/s
    - Frames of 4 x 40 = 160 samples (4 x 5ms = 20ms of speech)
  - Real-time constraint:
    - max. 20ms per speech frame
      (max. total of 3.26s for sample speech file)
  - SpecC specification model
    - 29 hierarchical behaviors (9 par, 10 seq, 10 fsm)
    - 73 leaf behaviors
    - 9139 formatted lines of SpecC code
      (~13000 lines of original C code, including comments)

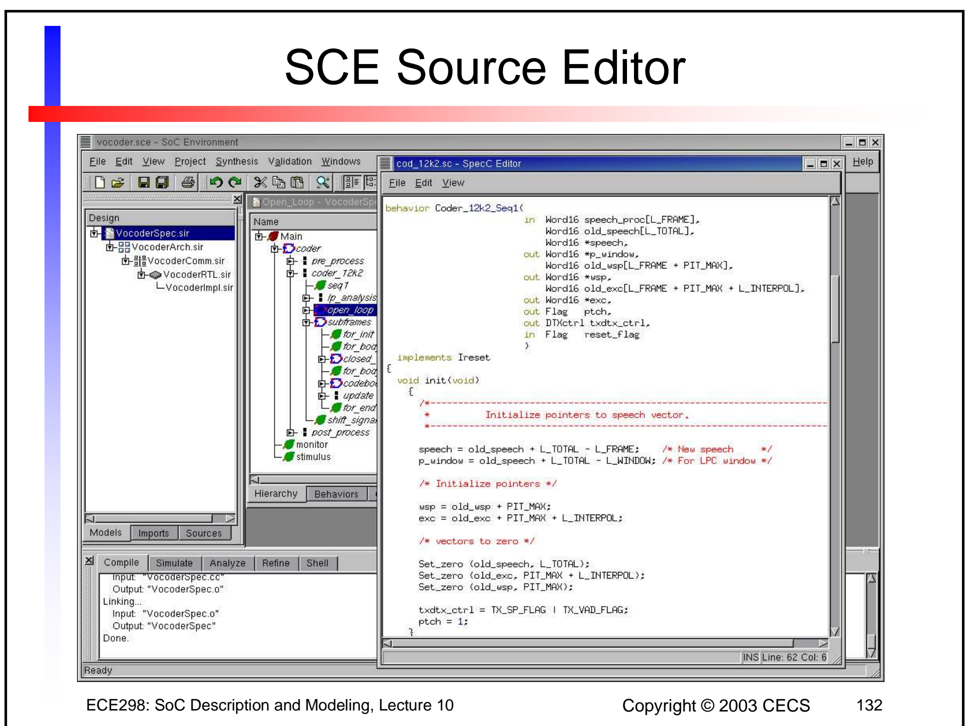
Lecture 10: Overview

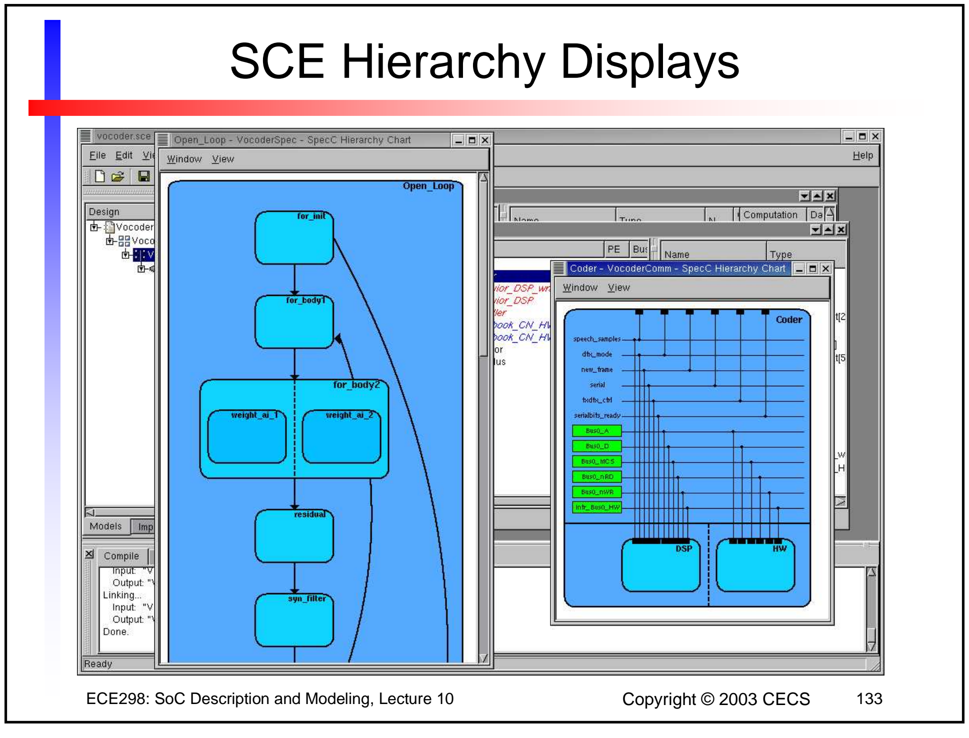
- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - ➢ SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

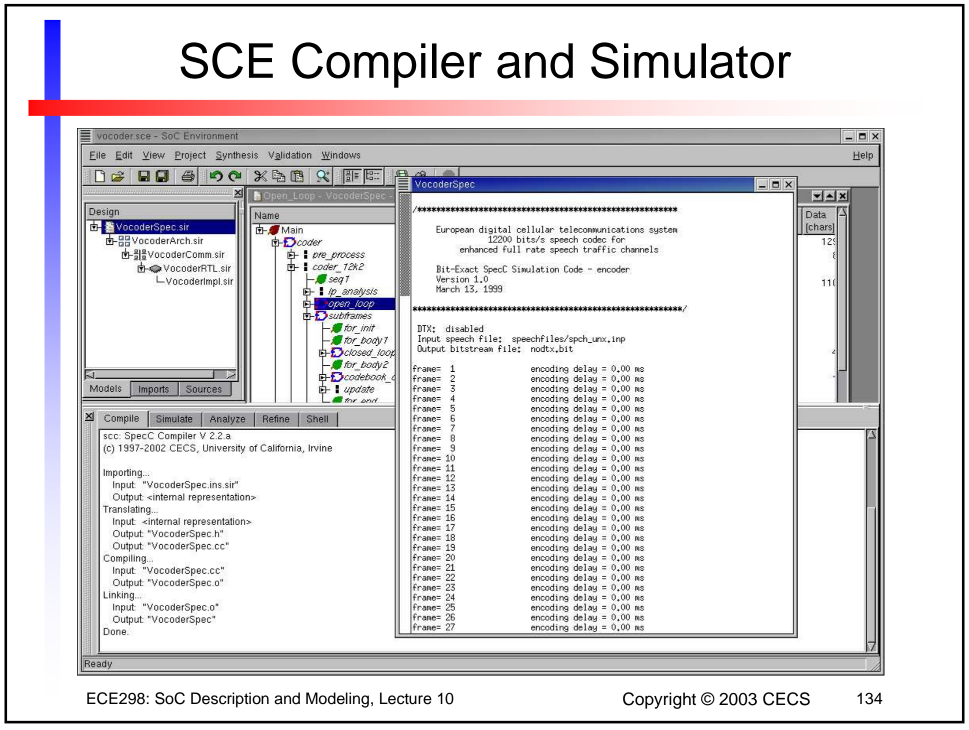

System-on-Chip Environment (SCE)
Refinement
Validation
Alg. selection
Browsing
Spec. optimization
Allocation
Beh. partitioning
Scheduling / RTOS
Protocol selection
Channel partitioning
Arbitration
Cycle scheduling
Protocol scheduling
SW assembly
Capture
Profiling weights
Profiling
Specification model
Profiling data
Comp. / IP attributes
Arch. refinement
Comp. / IP models
Estimation
Architecture model
Estimation results
Protocol attributes
Comm. refinement
Protocol models
Estimation
Communication model
Estimation results
RTL comp.
HW/SW synthesis
Design decisions
Cycle-a. refinement
Implementation model
Compile
Profile
Simulate
Estimate
Simulate
Estimate
Simulate
Synthesize
Simulate
ECE298: SoC Description and Modeling, Lecture 10
Copyright © 2003 CECS
129


System-on-Chip Environment (SCE)
• SCE Components:
– Graphical frontend (sce, scchart)
– Editor (sced)
– Compiler and simulator (scc)
– Profiling and analysis (scprof)
– Architecture refinement (scar)
– RTOS refinement (scos)
– Communication refinement (sccr)
– RTL refinement (scrtl)
– Software refinement (sc2c)
– Scripting interface (scsh)
– Tools and utilities ...
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
130

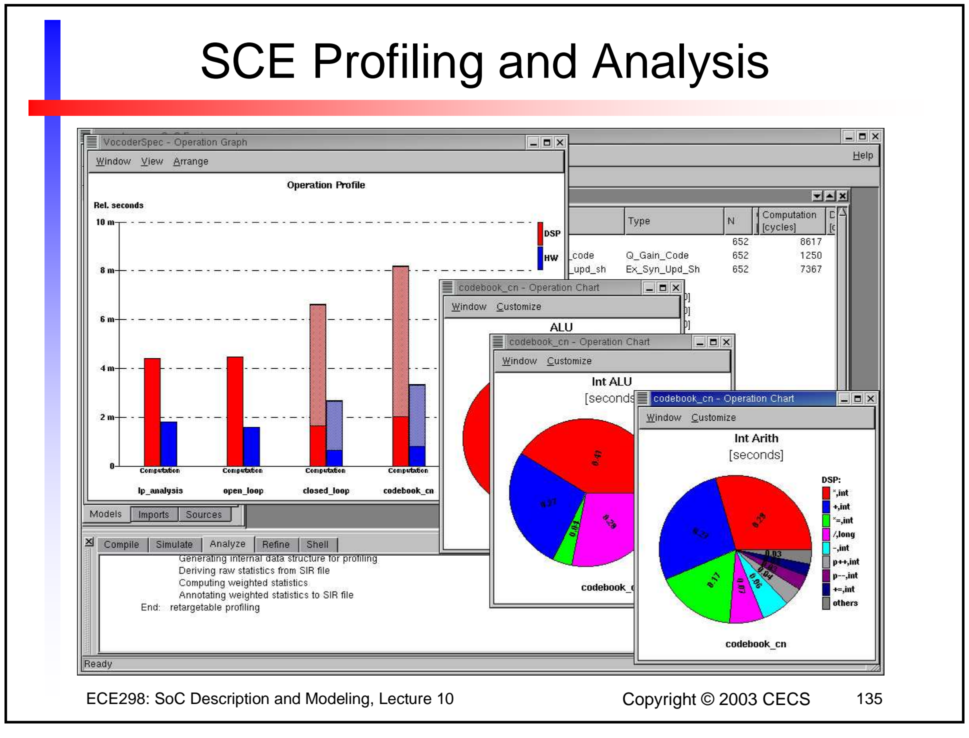

SCE Main Window
ECE298: SoC Description and Modeling, Lecture 10
Copyright © 2003 CECS
131


SCE Source Editor
ECE298: SoC Description and Modeling, Lecture 10
Copyright © 2003 CECS
132


SCE Hierarchy Displays
Open_Loop - VocoderSpec - SpecC Hierarchy Chart
Coder - VocoderComm - SpecC Hierarchy Chart
ECE298: SoC Description and Modeling, Lecture 10
Copyright © 2003 CECS
133


SCE Compiler and Simulator
vocoder.sce - SoC Environment
VocoderSpec
European digital cellular telecommunications system
12200 bits/s speech codec for
enhanced full rate speech traffic channels
Bit-Exact SpecC Simulation Code - encoder
Version 1.0
March 13, 1999
scc: SpecC Compiler V 2.2.a
(c) 1997-2002 CECS, University of California, Irvine
ECE298: SoC Description and Modeling, Lecture 10
Copyright © 2003 CECS
134

SCE Profiling and Analysis

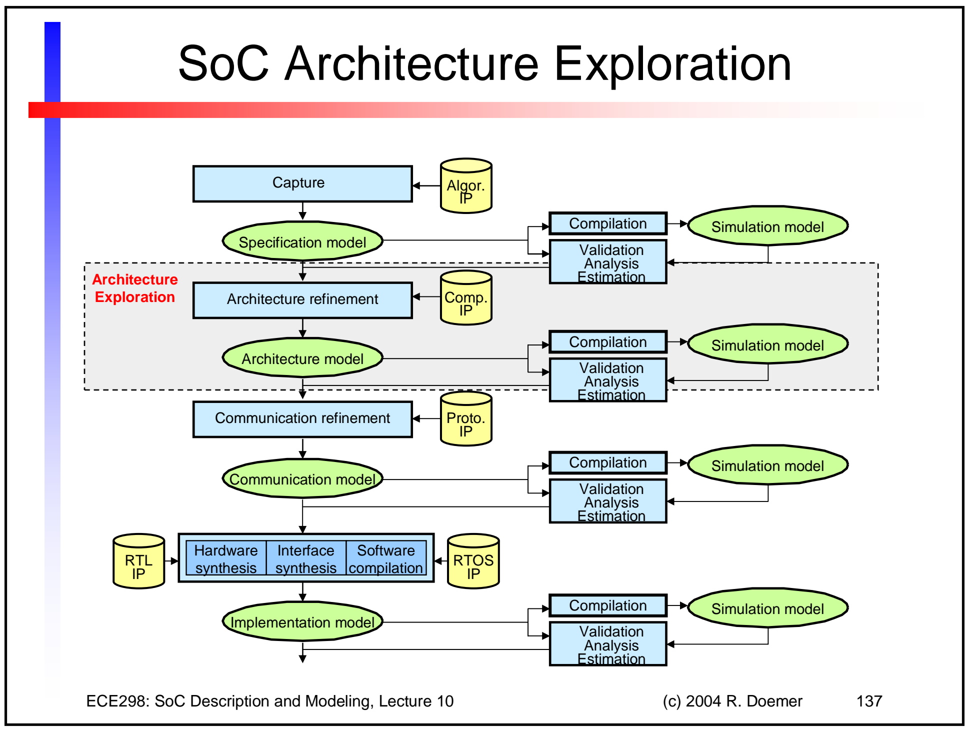


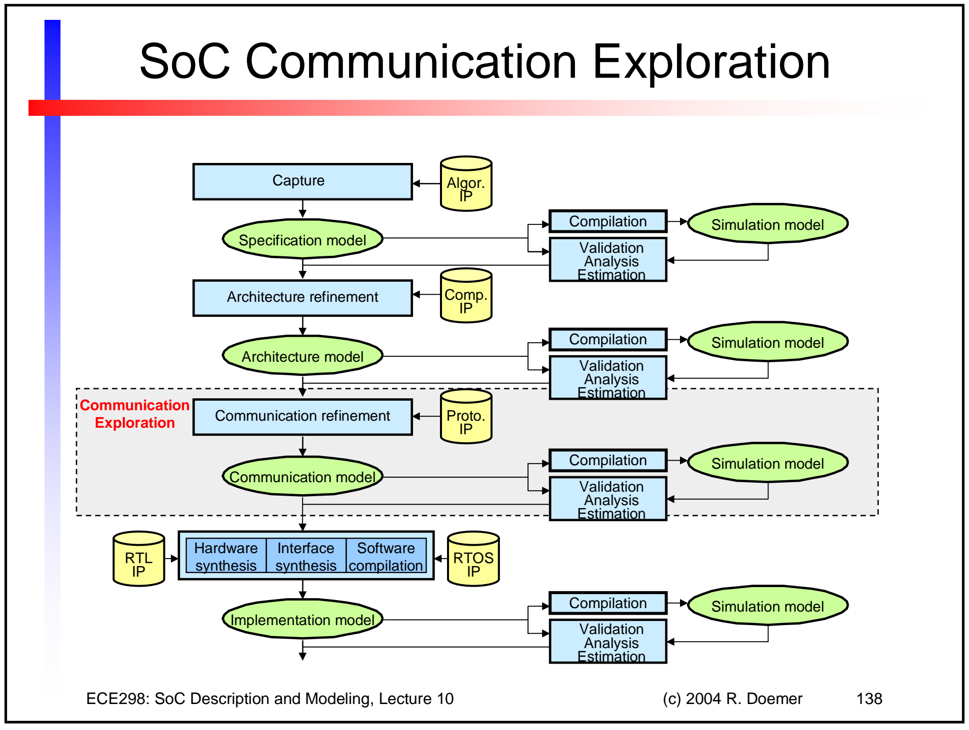
ECE298: SoC Description and Modeling, Lecture 10 Copyright © 2003 CECS 135

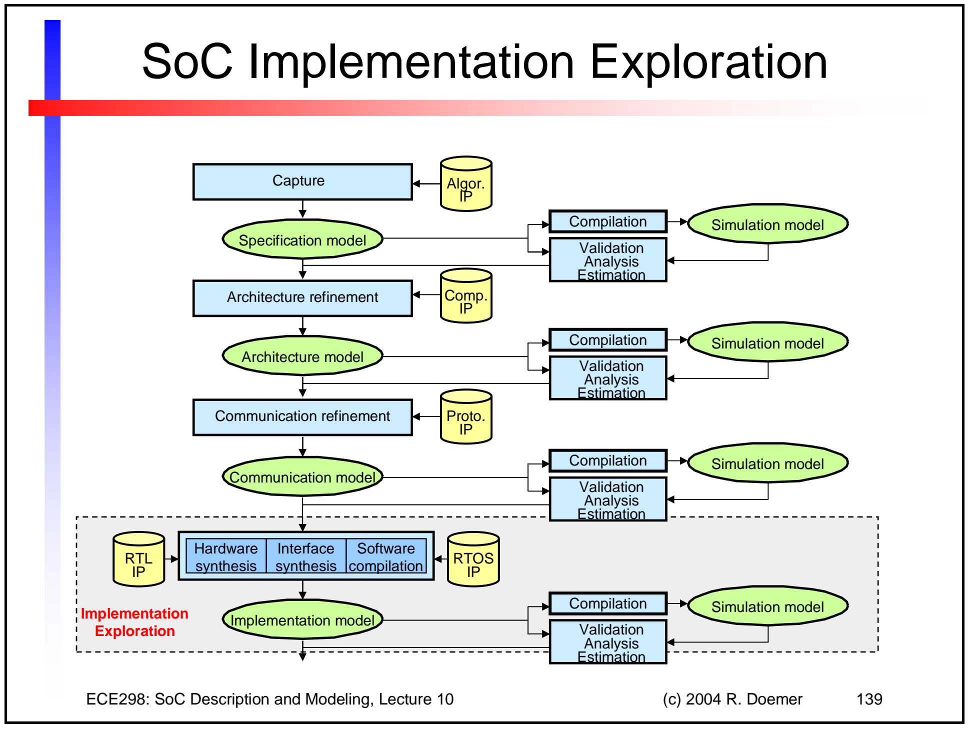
Lecture 10: Overview

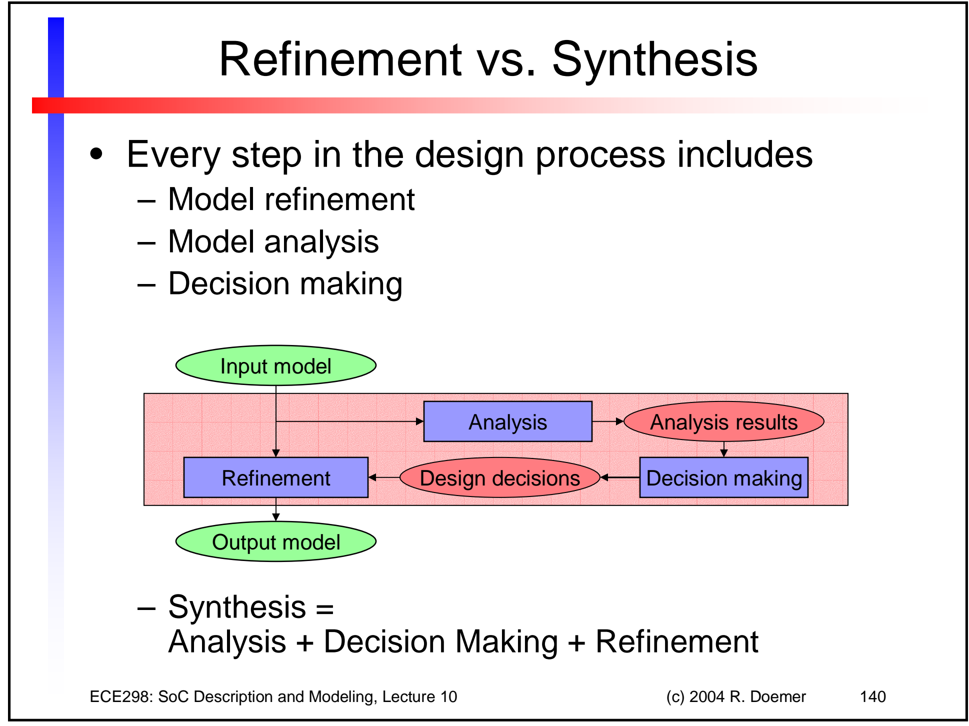
- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - ➢ SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

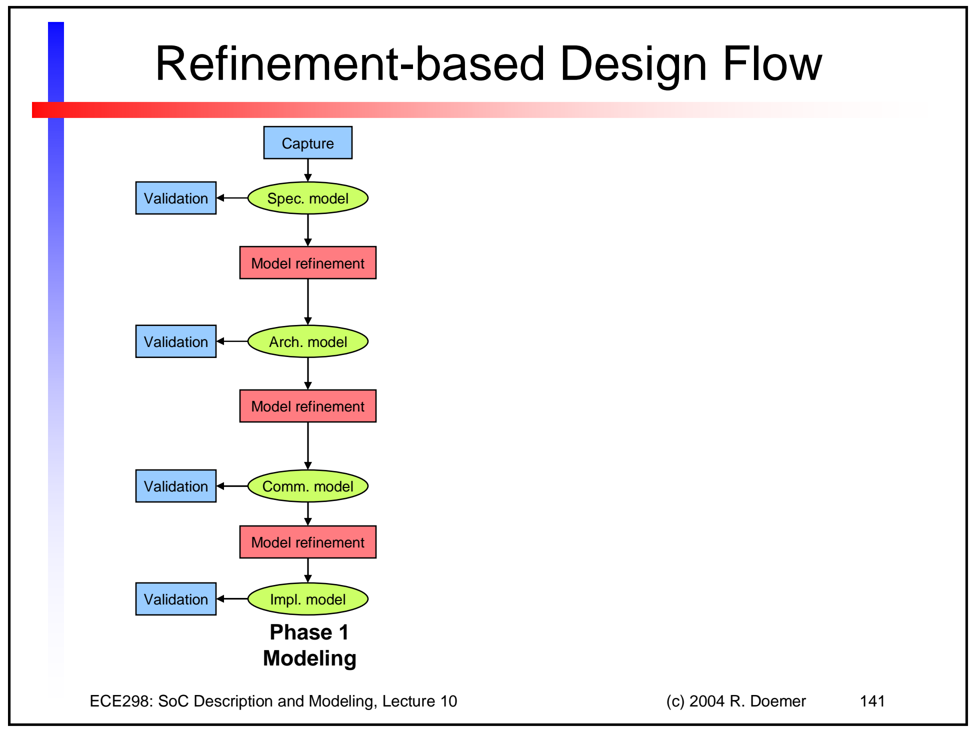
ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 136

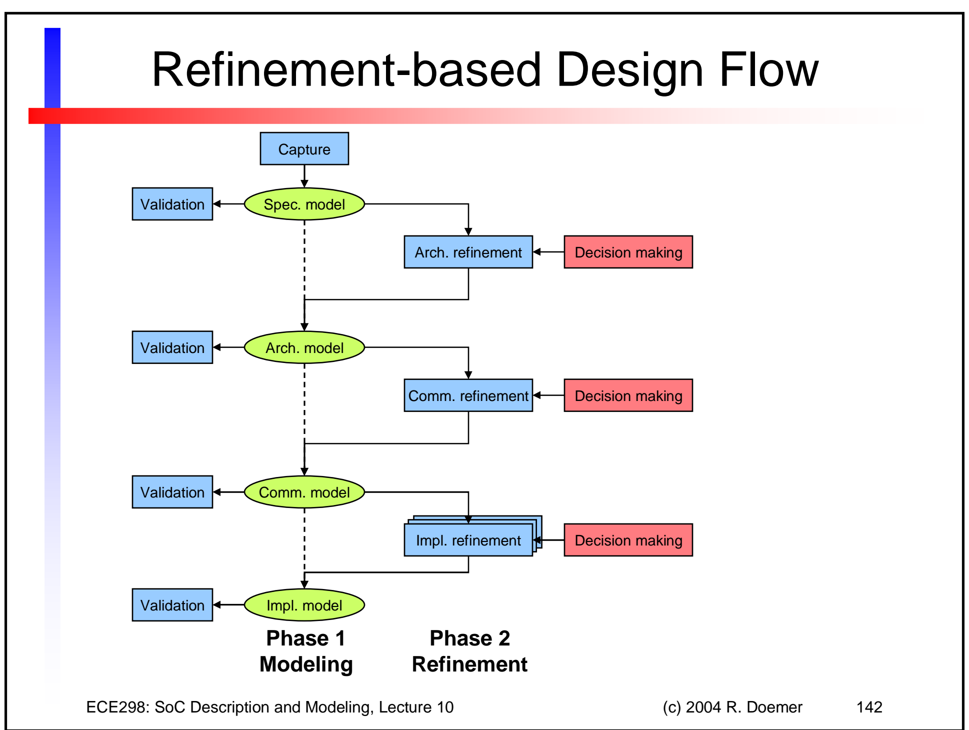

SoC Architecture Exploration
Capture
Algor. IP
Specification model
Compilation
Simulation model
Validation Analysis Estimation
Architecture Exploration
Architecture refinement
Comp. IP
Architecture model
Communication refinement
Proto. IP
Communication model
Hardware synthesis
Interface synthesis
Software compilation
RTL IP
RTOS IP
Implementation model
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
137

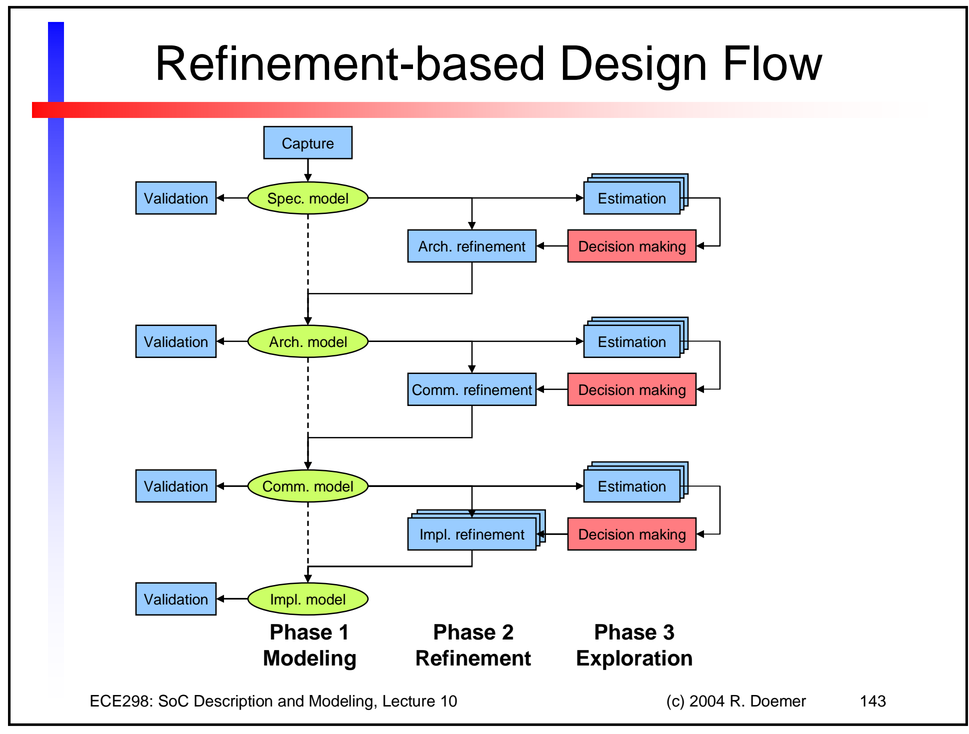

SoC Communication Exploration
Capture
Algor. IP
Specification model
Compilation
Simulation model
Validation Analysis Estimation
Architecture refinement
Comp. IP
Architecture model
Communication Exploration
Communication refinement
Proto. IP
Communication model
Hardware synthesis
Interface synthesis
Software compilation
RTL IP
RTOS IP
Implementation model
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
138

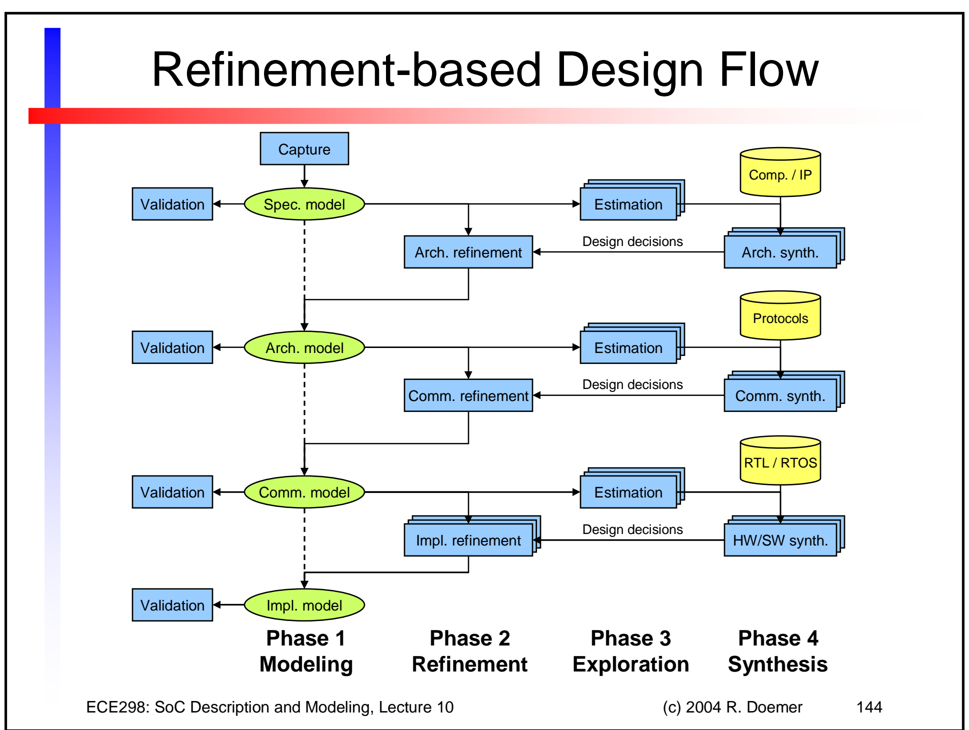

SoC Implementation Exploration
Capture
Algor. IP
Specification model
Compilation
Simulation model
Validation Analysis Estimation
Architecture refinement
Comp. IP
Architecture model
Compilation
Simulation model
Validation Analysis Estimation
Communication refinement
Proto. IP
Communication model
Compilation
Simulation model
Validation Analysis Estimation
RTL IP
Hardware synthesis
Interface synthesis
Software compilation
RTOS IP
Implementation Exploration
Implementation model
Compilation
Simulation model
Validation Analysis Estimation
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
139


Refinement vs. Synthesis
• Every step in the design process includes
– Model refinement
– Model analysis
– Decision making
Input model
Analysis
Analysis results
Refinement
Design decisions
Decision making
Output model
– Synthesis =
Analysis + Decision Making + Refinement
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
140


Refinement-based Design Flow
Capture
Validation
Spec. model
Model refinement
Validation
Arch. model
Model refinement
Validation
Comm. model
Model refinement
Validation
Impl. model
Phase 1
Modeling
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
141


Refinement-based Design Flow
Capture
Validation
Spec. model
Arch. refinement
Decision making
Validation
Arch. model
Comm. refinement
Decision making
Validation
Comm. model
Impl. refinement
Decision making
Validation
Impl. model
Phase 1
Modeling
Phase 2
Refinement
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
142


Refinement-based Design Flow
Capture
Validation
Spec. model
Estimation
Arch. refinement
Decision making
Validation
Arch. model
Estimation
Comm. refinement
Decision making
Validation
Comm. model
Estimation
Impl. refinement
Decision making
Validation
Impl. model
Phase 1
Modeling
Phase 2
Refinement
Phase 3
Exploration
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
143


Refinement-based Design Flow
Capture
Comp. / IP
Validation
Spec. model
Estimation
Arch. refinement
Design decisions
Arch. synth.
Protocols
Validation
Arch. model
Estimation
Comm. refinement
Design decisions
Comm. synth.
RTL / RTOS
Validation
Comm. model
Estimation
Impl. refinement
Design decisions
HW/SW synth.
Validation
Impl. model
Phase 1
Modeling
Phase 2
Refinement
Phase 3
Exploration
Phase 4
Synthesis
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
144

Refinement Decisions

- Step 1: Architecture Refinement
  - Allocation of Processing Elements (PE)
    - Type and number of processors
    - Type and number of custom hardware blocks
    - Type and number of system memories
  - Mapping to PEs
    - Map each behavior to a PE
    - Map each channel to a PE
    - Map each variable to a PE
  - Result:
    System architecture of concurrent PEs
    with abstract communication in channels

Refinement Decisions

- Step 2: Scheduling Refinement
  - For each PE, serialize the execution of behaviors to a single thread of control
  - Option (a): Static scheduling
    - For each set of concurrent behaviors, determine fixed order of execution
  - Option (b): Dynamic scheduling by RTOS
    - Choose scheduling policy, i.e. Round-robin or priority-based
    - For each set of concurrent behaviors, determine scheduling priority
  - Result:
    System model with abstract RTOS scheduler inserted in each PE

Refinement Decisions

- Step 3: Communication Refinement
  - Allocation of system busses
    - Type and number of system busses
    - Type of bus protocol for each bus (if applicable)
    - Number of transducers (if applicable)
    - System connectivity
  - Mapping of channels to busses
    - Map each communication channel to a system bus (or multiple busses, if applicable)
  - Result:
    Bus-functional model of the system

Refinement Decisions

- Step 4: Hardware Refinement (for HW PE)
  - Allocation of RTL components
    - Type and number of functional units
    - Type and number of storage units
    - Type and number of interconnecting busses
  - Scheduling
    - Basic blocks assigned to super-states
    - Operations assigned to clock cycles
  - Binding
    - Bind functional operations to functional units
    - Bind variables to storage units
    - Bind transfers to busses
  - Result:
    Clock-cycle accurate model of each HW PE
  - Output: Synthesizable Verilog description

Refinement Decisions

- Step 5: Software Refinement (for SW PE)
  - C code generation
    - For selected target processor
    - For selected target RTOS
  - Compilation to Instruction Set
    - for Instruction Set Simulation (ISS)
  - Assembly
  - Result:
    Clock-cycle accurate model of each SW PE
  - Output: downloadable object code

SoC Exploration and Refinement

- Example:
  - GSM Vocoder design in SCE
- Online demonstration ...

Lecture 10: Overview

- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - ➢ SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - UML and other SLDL

Execution and Simulation Semantics

- Motivational Example 1
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    b1; b2;
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

Execution and Simulation Semantics

- Motivational Example 2
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: The program is non-deterministic!
    (x may be 5, or 6, or any other value!)

Execution and Simulation Semantics

- Motivational Example 3
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    waitfor 10;
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 5

Execution and Simulation Semantics

- Motivational Example 4
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    waitfor 10;
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    waitfor 10;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: The program is non-deterministic!
    (x may be 5, or 6, or any other value!)

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 155

Execution and Simulation Semantics

- Motivational Example 5
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    x = 5;
    notify e;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 156

Execution and Simulation Semantics

- Motivational Example 6
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    notify e;
    x = 5;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 157

Execution and Simulation Semantics

- Motivational Example 7
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    waitfor 10;
    x = 5;
    notify e;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 158

Execution and Simulation Semantics

- Motivational Example 8
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    x = 5;
    notify e;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    waitfor 10;
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: B never terminates!
    (the event is lost)

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 159

System-level Language Semantics

- Concepts found in Embedded Systems
  - Behavioral and structural hierarchy
  - Concurrency
  - Synchronization and communication
  - Exception handling
  - Timing
  - State transitions
- System-level language must support these concepts
- Language semantics needed to define the *meaning*
  - Semantics of execution (modeling, simulation, synthesis)
  - Deterministic vs. non-deterministic behavior
  - Preemptive vs. non-preemptive concurrency
  - Atomic operations

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 160

System-level Language Semantics

- Language semantics are needed for
  - System designer (understanding)
  - Tools
    - Validation (compilation, simulation)
    - Formal verification (equivalence, property checking)
    - Synthesis
  - Documentation and standardization
- Objective:
  - Clearly define the execution semantics of the language
- Requirements and goals:
  - completeness
  - precision (no ambiguities)
  - abstraction (no implementation details)
  - formality (enable formal reasoning)
  - simplicity (easy understanding)

System-level Language Semantics

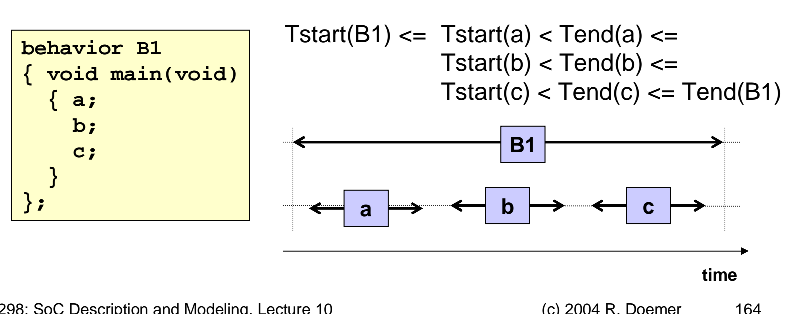
- Example: SpecC language
- Documentation
  - Language Reference Manual (LRM)
  - ⇒ set of rules written in English, thus not formal
  - Abstract simulation algorithm
  - ⇒ set of valid implementations, but incomplete, not formal
- Reference implementation
  - SpecC Reference Compiler and Simulator
  - ⇒ only one instance of a valid implementation
  - Compliance test bench
  - ⇒ set of specific test cases, thus incomplete
- Formal execution semantics are needed!

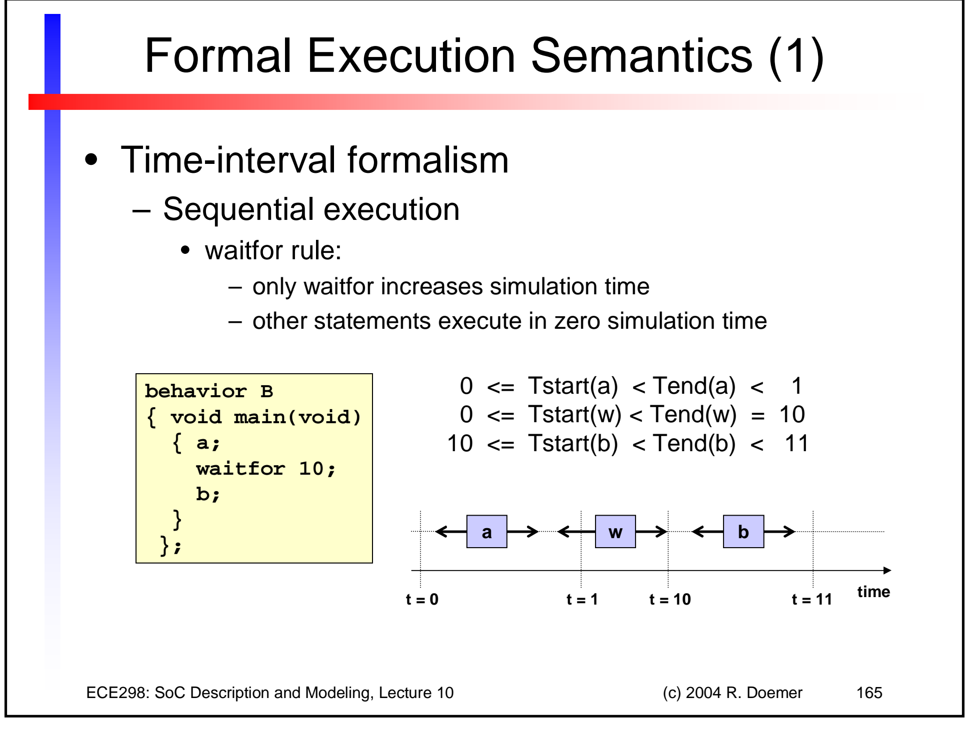
Formal Execution Semantics

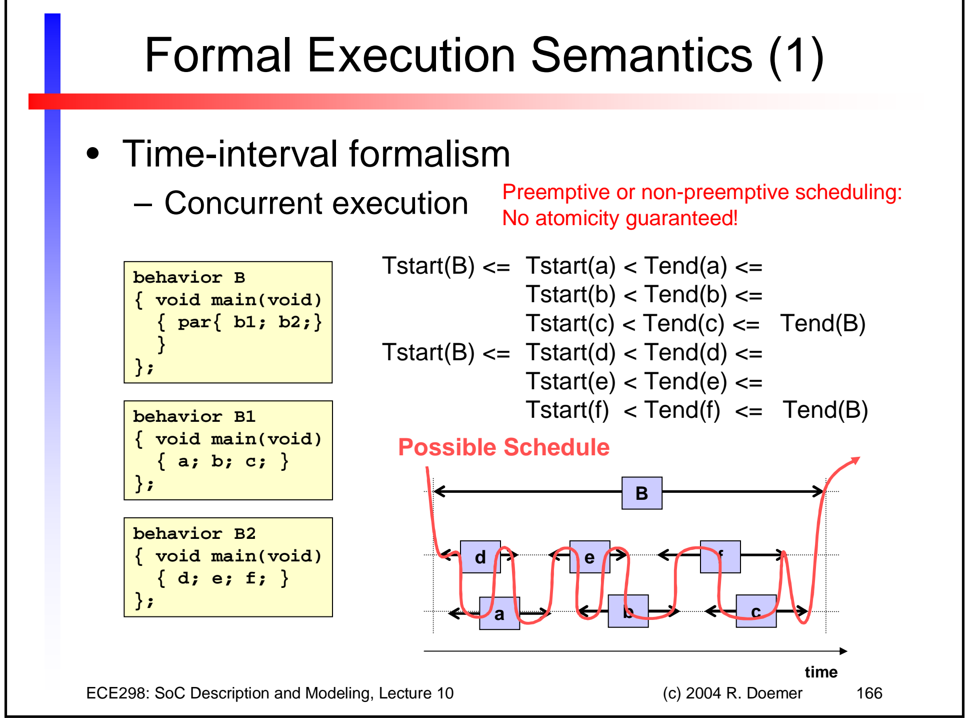
- Two examples of semantics definition:
  1) Time-interval formalism
     - formal definition of timed execution semantics
     - sequentiality, concurrency, synchronization
     - allows reasoning over execution order, dependencies
  2) Abstract State Machines
     - complete execution semantics of SpecC V1.0
       - wait, notify, notifyone, par, pipe, traps, interrupts
       - operational semantics (no data types!)
     - influence on the definition of SpecC V2.0
     - straightforward extension for SpecC V2.0
     - comparable to ASM specifications of SystemC and VHDL 93

Formal Execution Semantics (1)

- Time-interval formalism
  - Definition of execution semantics of SpecC 2.0
    - sequential execution
    - concurrent execution (semantics of **par**)
    - synchronization (semantics of **notify**, **wait**)
  - Sequential execution




Formal Execution Semantics (1)
• Time-interval formalism
– Sequential execution
• waitfor rule:
– only waitfor increases simulation time
– other statements execute in zero simulation time
behavior B
{ void main(void)
{ a;
waitfor 10;
b;
}
};
0 <= Tstart(a) < Tend(a) < 1
0 <= Tstart(w) < Tend(w) = 10
10 <= Tstart(b) < Tend(b) < 11
a
w
b
t = 0
t = 1
t = 10
t = 11
time
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
165


Formal Execution Semantics (1)
• Time-interval formalism
– Concurrent execution
Preemptive or non-preemptive scheduling:
No atomicity guaranteed!
behavior B
{ void main(void)
{ par{ b1; b2;}
}
};
behavior B1
{ void main(void)
{ a; b; c; }
};
behavior B2
{ void main(void)
{ d; e; f; }
};
Tstart(B) <= Tstart(a) < Tend(a) <=
Tstart(b) < Tend(b) <=
Tstart(c) < Tend(c) <= Tend(B)
Tstart(B) <= Tstart(d) < Tend(d) <=
Tstart(e) < Tend(e) <=
Tstart(f) < Tend(f) <= Tend(B)
Possible Schedule
B
d
e
f
a
b
c
time
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
166

Formal Execution Semantics (1)

- Atomicity
  - Since there is no atomicity guaranteed, a safe mechanism for mutual exclusion is necessary
  - SpecC 2.0:
    - A mutex is implicitly contained in each channel instance
    - Each channel method implicitly acquires the mutex when it starts execution and releases the mutex again when it finishes
    - An acquired mutex is also released at **`wait`** and **`waitfor`** statements and will be re-acquired before execution resumes
  - This easily enables safe communication without unnecessary restrictions to the implementation!

Formal Execution Semantics (1)

- Time-interval formalism
  - Synchronization

```
behavior B
{ void main(void)
  { par{ b1; b2;}
  }
};
```

```
behavior B1
{ void main(void)
  { a; wait e;   b; }
};
```

```
behavior B2
{ void main(void)
  { c; notify e; d; }
};
```

Tstart(B) <= Tstart(a) < Tend(a) <=
Tstart(w) < Tend(w) <=
Tstart(b) < Tend(b) <= Tend(B)

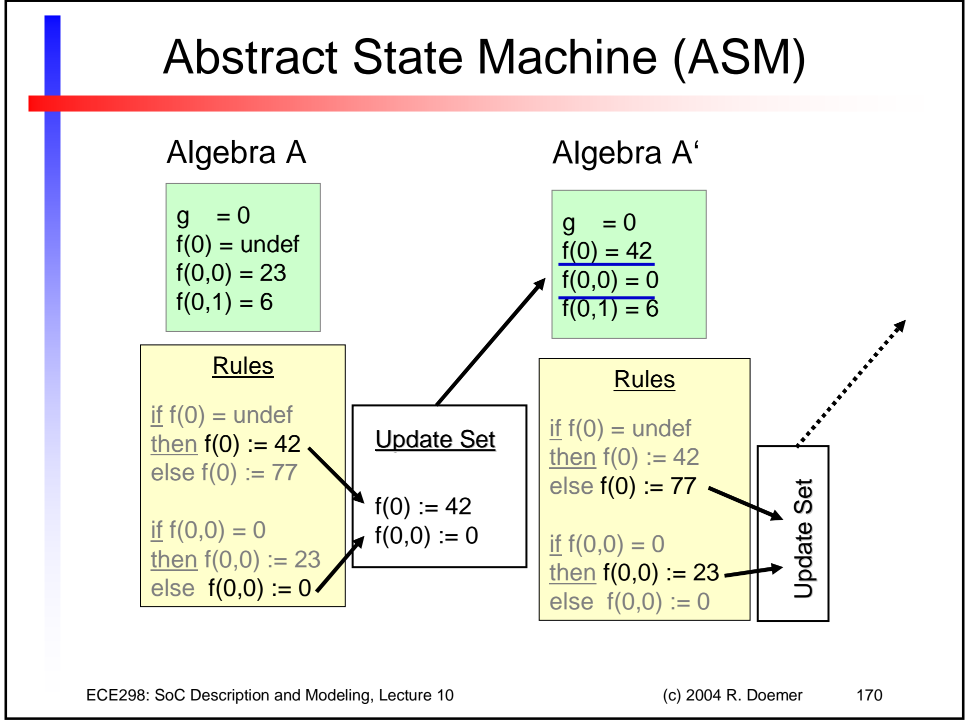
Tstart(B) <= Tstart(c) < Tend(c) <=
Tstart(n) < Tend(n) <=
Tstart(d) < Tend(d) <= Tend(B)

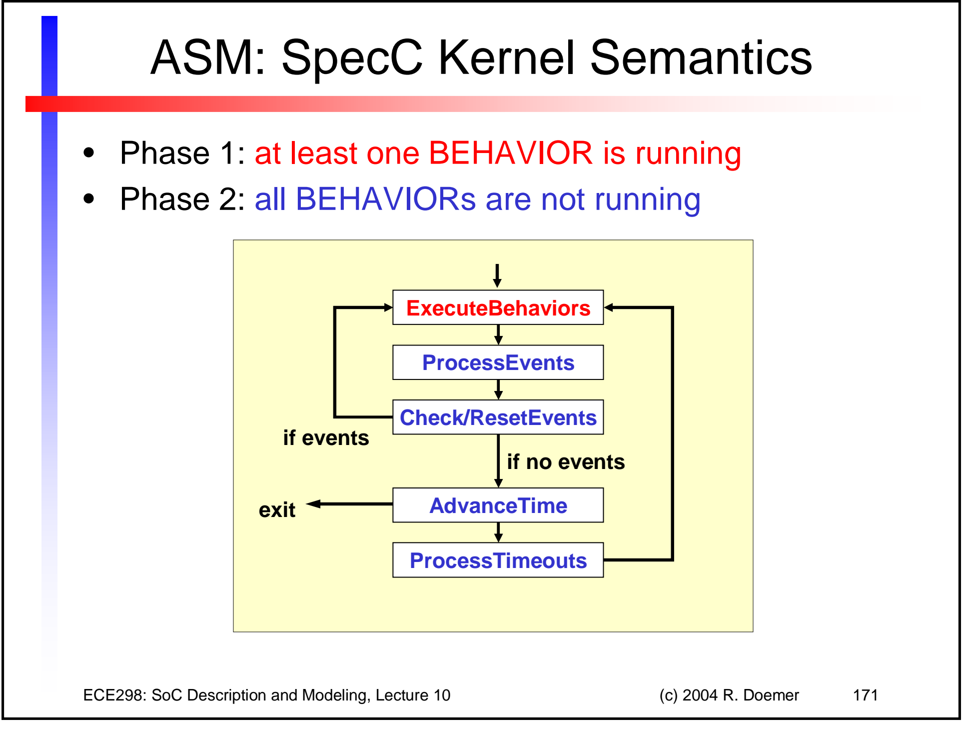
Tend(w) >= Tend(n)

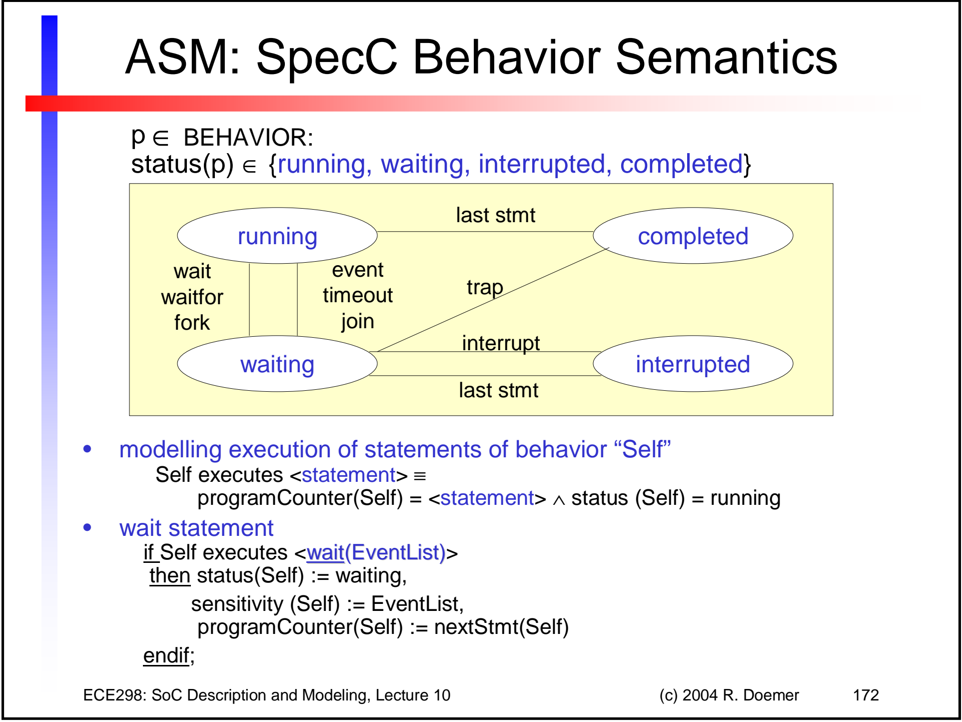
Formal Execution Semantics (2)

- Abstract State Machine (ASM)
  - aka. Evolving Algebras (Y. Gurevich, 1987)
  - ASM semantics already exist for
    - Prolog, Concurrent Prolog
    - C, C++, Java
    - VHDL, VHDL-AMS, SystemC
  - ASM semantics for SpecC published at ISSS'02
- ASM components
  - Sequence of algebras (functions over domains): *states*
  - Rules define updates of functions: *state transitions*

Abstract State Machine (ASM)




ASM: SpecC Kernel Semantics
• Phase 1: at least one BEHAVIOR is running
• Phase 2: all BEHAVIORs are not running
ExecuteBehaviors
ProcessEvents
Check/ResetEvents
if events
if no events
exit
AdvanceTime
ProcessTimeouts
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
171


ASM: SpecC Behavior Semantics
p ∈ BEHAVIOR:
status(p) ∈ {running, waiting, interrupted, completed}
running
last stmt
completed
wait
waitfor
fork
event
timeout
join
trap
interrupt
waiting
interrupted
last stmt
• modelling execution of statements of behavior "Self"
Self executes <statement> ≡
programCounter(Self) = <statement> ∧ status (Self) = running
• wait statement
if Self executes <wait(EventList)>
then status(Self) := waiting,
sensitivity (Self) := EventList,
programCounter(Self) := nextStmt(Self)
endif;
ECE298: SoC Description and Modeling, Lecture 10
(c) 2004 R. Doemer
172

ASM: SpecC Statement Semantics

- modelling execution of statements of behavior "Self"
  Self executes <statement> ≡
  programCounter(Self) = <statement> ∧ status (Self) = running
- wait statement
  if Self executes <wait(EventList)>
  then status(Self) := waiting,
  sensitivity (Self) := EventList,
  programCounter(Self) := nextStmt(Self)
  endif;
- notify statement
  if Self executes <notify(EventList)>
  then ∀ e ∈ EventList: notified(e) := true,
  programCounter(Self) := nextStmt(Self)
  endif;
- The simulation kernel sets each behavior to
  status(b):= running if ∃e: notified(e) = true ∧ e ∈ sensitivity (b)

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 173

ASM: SpecC Summary

- Formal Semantics of SpecC Execution
  - complete execution semantics of SpecC V1.0 by ASMs
    - wait, notify, notifyone, par, pipe, traps, interrupts
    - operational semantics (no data types!)
  - can be easily extended to V2.0
  - influenced the definition of SpecC V2.0
  - SpecC ASM specification is comparable to other ASM specifications
    - SystemC
    - VHDL 93

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 174

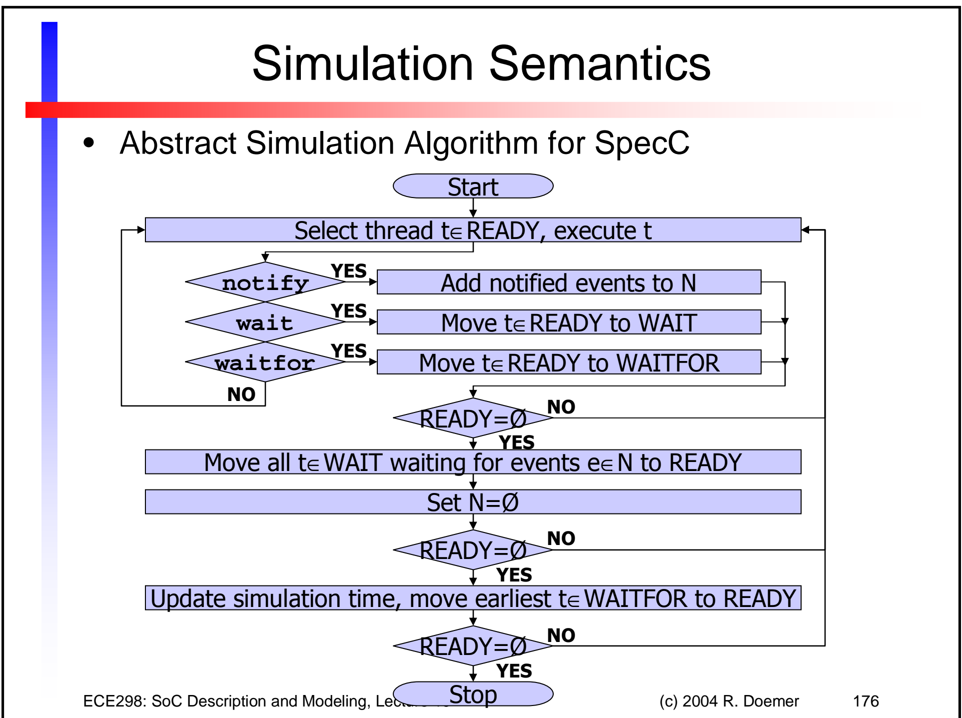
Simulation Semantics

- Abstract Simulation Algorithm for SpecC
  - available in LRM (appendix), good for understanding
  - ⇒ set of valid implementations
  - ⇒ possibly incomplete!
- Definitions:
  - At any time, each thread t is in one of the following sets:
    - **READY**: set of threads ready to execute (initially root thread)
    - **WAIT**: set of threads suspended by `wait` (initially Ø)
    - **WAITFOR**: set of threads suspended by `waitfor` (initially Ø)
  - Notified events are stored in a set **N**
    - `notify e1` adds event `e1` to **N**
    - `wait e1` will wakeup when `e1` is in **N**
    - Consumption of event e means event e is taken out of **N**
    - Expiration of notified events means **N** is set to Ø

ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 175

Simulation Semantics

- Abstract Simulation Algorithm for SpecC



ECE298: SoC Description and Modeling, Lecture 10 (c) 2004 R. Doemer 176

Simulation Semantics

- Abstract Simulation Algorithm for SpecC
  - clearly specifies the simulation semantics
  - is one valid implementation of the semantics
  - other valid implementations may exist as well

Lecture 10: Overview

- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - ➢ Modeling with SystemC SLDL
  - UML and other SLDL

SystemC Overview

- Goals
  - Common C++ Modeling Platform
    - System Level Design
    - HW/SW Codesign
    - RTL
  - Seamless Co-Simulation of HW and SW
  - IP Reuse
  - Free licensing, Open Source
  - De-facto Standard
- Open SystemC Initiative (OSCI)
  - Consortium of many EDA companies
    - Synopsys, Cadence, CoWare, Frontier, ...
  - Open Community (very large!)

SystemC Overview

- Language
  - C++ class library (layered SW architecture)
  - Hierarchy of Modules connected by Ports
  - Communication via Interfaces and Channels
  - Discrete-Event Simulation
- Methodology
  - Untimed Model
  - Transaction-level Model
  - Bus-functional Model
  - Cycle-accurate Model

Introduction to SystemC

- Presentation by Stuart Swan, Cadence, 2002
  - Goals and Requirements
  - History and Organization
  - Versions, Contents, Coverage
  - Language Architecture
  - Modeling, Models of Computation, Examples
  - Communication Refinement
  - Outlook

SystemC 2.0 Tutorial

- Presentation by Thorsten Groetker, Synopsys, 2001
  - Motivation
  - Models of Computation
  - Model of Time
  - Communication, Interfaces and Channels
  - Platform Modeling
  - Transaction-level Model, Examples
  - Benefits
  - Summary

Lecture 10: Overview

- Homework Assignments 4 and 5
  - Status, Discussion
- Course Review
  - Introduction to SoC Design
  - SoC Design Methodology
  - The SpecC Language
  - SoC Specification Modeling
  - SoC Environment
  - SoC Exploration and Refinement
  - SLDL Execution and Simulation Semantics
  - Modeling with SystemC SLDL
  - ➢ UML and other SLDL

UML and other SLDL

- Invited Talk by Dr. Wolfgang Mueller, C-LAB
  - Short Biography
    - 1996 Ph.D., University of Paderborn, Germany
    - Member of C-LAB, a joint R&D institute of
      - University of Paderborn, Germany
      - Siemens Business Services
    - Head of groups for
      - User Interfaces
      - Human-Computer Interaction
      - Visual Interactive Systems
    - Served in various program committees (e.g. DATE, FDL)
    - Authored more than 100 publications
    - Research interests
      - System design methodologies
      - System description languages
      - System integration technologies