

EECS 222: Embedded System Modeling Lecture 2

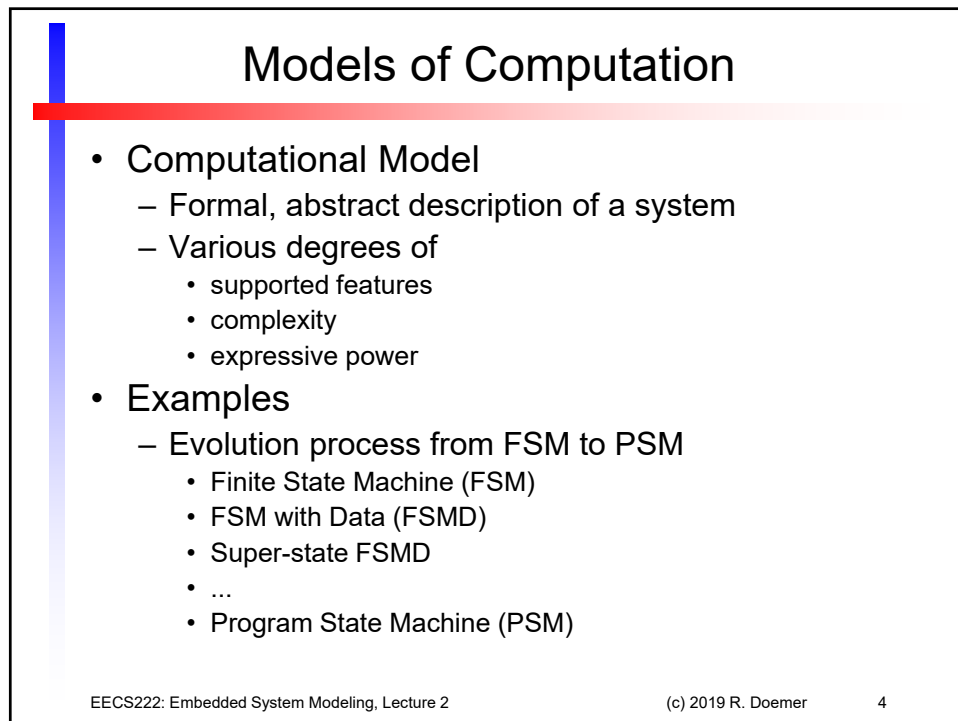
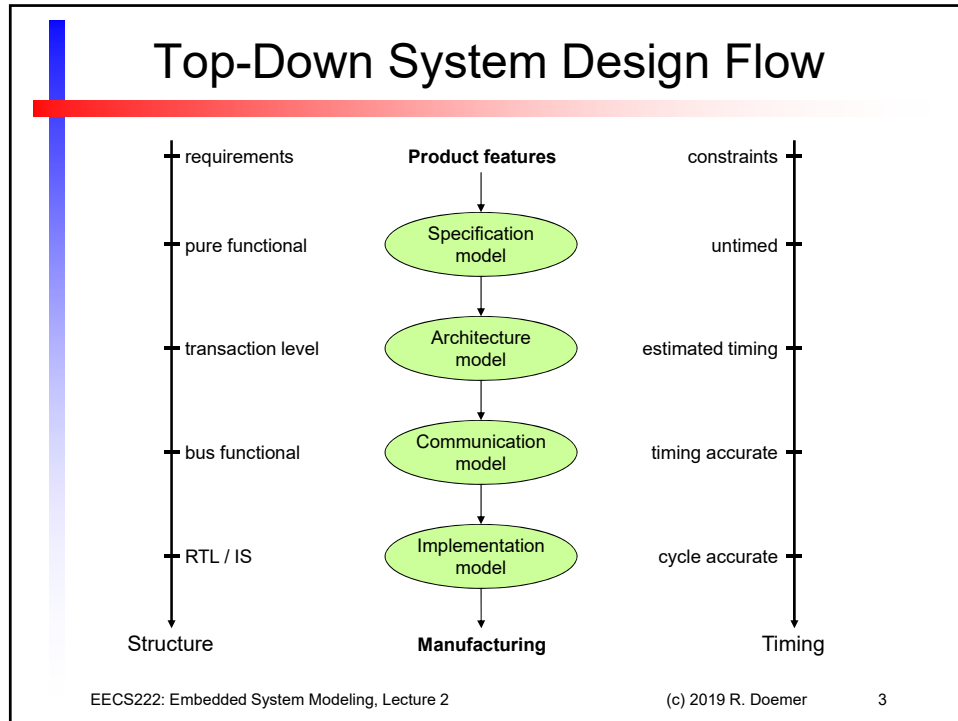
Rainer Dömer

doemer@uci.edu

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

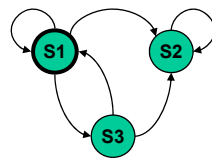
Lecture 2: Overview

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



Models of Computation

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



FSM model

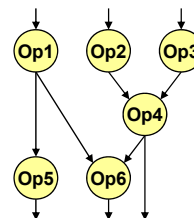
EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

5

Models of Computation

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



DFG model

EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

6

Models of Computation

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

FSMD model

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 7

Models of Computation

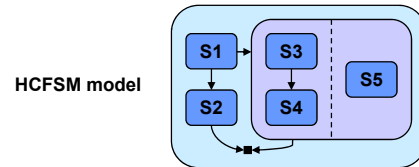
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
- Super-State FSM with Data (SFSMD)
 - FSMD with complex, multi-cycle states
 - States described by procedures in a programming language

SFSMD model

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 8

Models of Computation

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



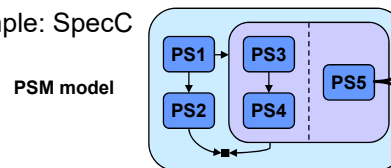
EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

9

Models of Computation

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



```

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

```

EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

10

Models of Computation

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

EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

11

System-Level Description Languages

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

EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

12

System-Level Description Languages

- Requirements supported by existing languages

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

○ not supported ◐ partially supported ● supported

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 13

System-Level Description Languages

- Requirements supported by existing languages

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

○ not supported ◐ partially supported ● supported

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 14

System-Level Description Languages

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

EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

17

Separation of Concerns

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

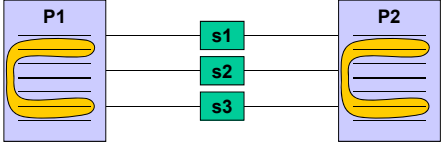
EECS222: Embedded System Modeling, Lecture 2

(c) 2019 R. Doemer

18

Computation vs. Communication

- Traditional model
 - Processes and signals



- VHDL example:

```

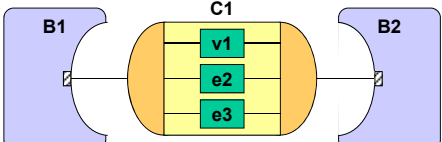
entity P1 [...] process [...]
s1 <= '1';
s2 <= '1';
wait until s3'event and s3 = '1';
s2 <= '0';
xy = x + 2 * y;
s1 <= xy;
s2 <= '1';
wait until s3'event and s3 = '1';
s1 <= '0';
s2 <= '0';
[...]
```

- Mixture of computation and communication
- Automatic replacement impossible!

EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
19

Computation vs. Communication

- SpecC model
 - Behaviors and channels



- SpecC example:

```

behavior B1 [...]
{
c.send(1);
xy = x + 2 * y;
c.send(xy);
v1 = 0;
[...]
```

```

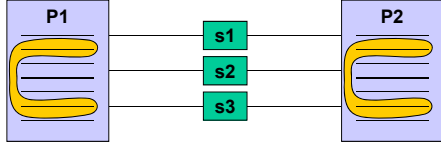
channel C1 [...]
{
send (int d)
{
v1 = d;
notify e2;
wait e3;
}
[...]
```

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

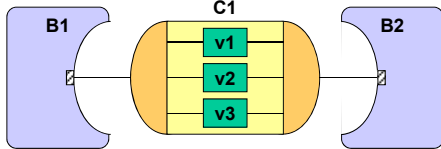
EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
20

Computation vs. Communication

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



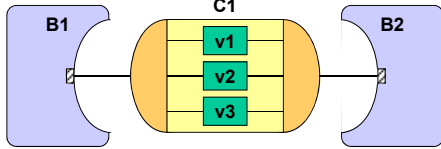
- SpecC model
 - Behaviors and channels
 - Separation of computation and communication
 - Plug-and-play



EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 21

Computation vs. Communication

- Protocol Inlining
 - Specification model
 - Exploration model

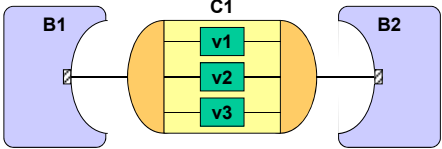


- Computation in behaviors
- Communication in channels

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 22

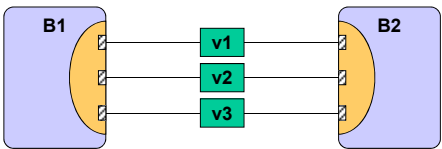
Computation vs. Communication

- Protocol Inlining
 - Specification model
 - Exploration model



- Computation in behaviors
- Communication in channels

- Implementation model

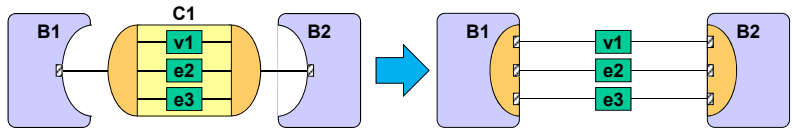


- Channel disappears
- Communication inlined into behaviors
- Wires exposed

EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
23

Computation vs. Communication

- Protocol Inlining



– SpecC example:

```
behavior B1 [...]
{
  c.send(1);

  xy = x + 2 * y;


  c.send(xy);

  v1 = 0;
  [...]}

```

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

```



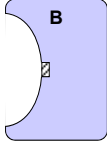
```
behavior B1 [...]
{
  v1 = 1;
  notify e2;
  wait e3;
  xy = x + 2 * y;
  v1 = xy;
  notify e2;
  wait e3;
  v1 = 0;
  [...]}

```

EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
24

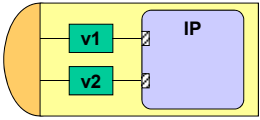
Intellectual Property (IP)

- Computation IP: Wrapper model



B

Synthesizable
behavior

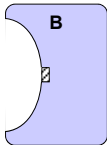


IP in wrapper

EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
25

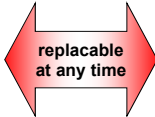
Intellectual Property (IP)

- Computation IP: Wrapper model

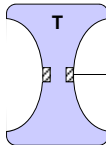


B

Synthesizable
behavior

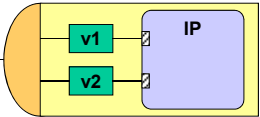


replacable
at any time



T

Transducer



IP in wrapper

EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
26

Intellectual Property (IP)

- Computation IP: Wrapper model

Synthesizable behavior Transducer IP in wrapper

- Protocol inlining with wrapper

before after

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 27

Intellectual Property (IP)

- Computation IP: Adapter model

Synthesizable behavior Transducer Adapter IP

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 28

Intellectual Property (IP)

- Computation IP: Adapter model

- Protocol inlining with adapter

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 29

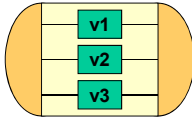
Intellectual Property (IP)

- Communication IP: Channel with wrapper

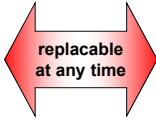
EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 30

Intellectual Property (IP)

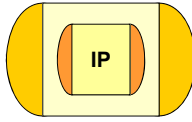
- Communication IP: Channel with wrapper



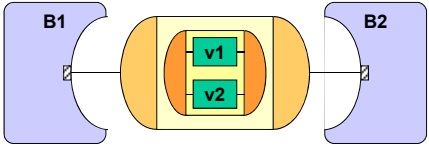
C1
Virtual channel



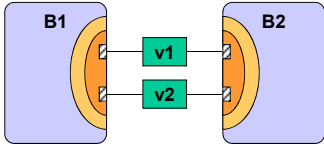
replacable
at any time



C2
IP protocol channel in wrapper
- Protocol inlining with hierarchical channel



before

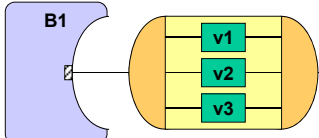


after

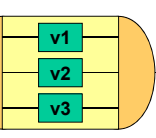
EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
31

Intellectual Property (IP)

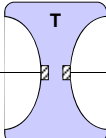
- Incompatible busses: Transducer insertion



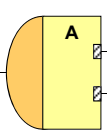
B1
Synthesizable
behavior



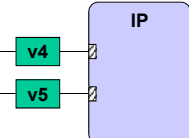
System bus



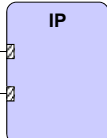
T
Transducer



A
Adapter



v4
v5
IP bus



IP

EECS222: Embedded System Modeling, Lecture 2
(c) 2019 R. Doemer
32

Intellectual Property (IP)

- Incompatible busses: Transducer insertion

Synthesizable behavior System bus Transducer Adapter IP bus IP
- Protocol inlining with transducer

after

EECS222: Embedded System Modeling, Lecture 2 (c) 2019 R. Doemer 33