

ECPS 203

Embedded Systems Modeling and Design

Lecture 2

Rainer Dömer

doemer@uci.edu

Center for Embedded and Cyber-physical Systems
University of California, Irvine



Lecture 2: Overview

- Embedded System Design
 - Complexity challenges
 - Abstraction Levels
 - Top-down Design Flow
- Abstract Modeling of Embedded Systems
 - Models of Computation
 - System-Level Description Languages
- Separation of Concerns
 - Computation vs. Communication

Embedded System Design

- Embedded System in CPS context
 - Software
 - Hardware
- Design Challenges
 - Hardware design gap
 - Software design gap
 - System design gap

The diagram shows a Cyber-Physical System (CPS) block. Inside, there are three main components: Sensors on the left, an Embedded Computer System in the center, and Actuators on the right. The Embedded Computer System is further divided into Software (green) and Hardware (blue) layers. A large grey arrow labeled 'Control' loops around the Embedded Computer System, indicating a feedback loop between the sensors and actuators through the computer system.

The graph plots four metrics over time. Moore's Law is shown as a blue line with a steep positive slope. The System Gap is a black line with a shallower positive slope. The HW Gap is a blue line with a very shallow positive slope. The SW Gap is a red line with a negative slope, indicating that the software gap is narrowing over time.

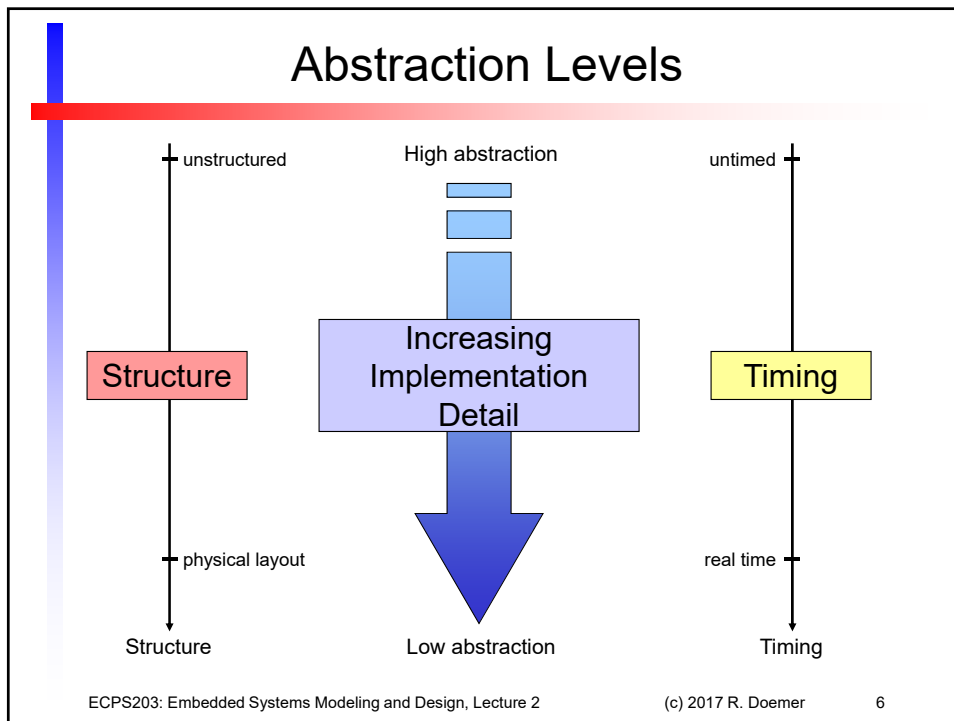
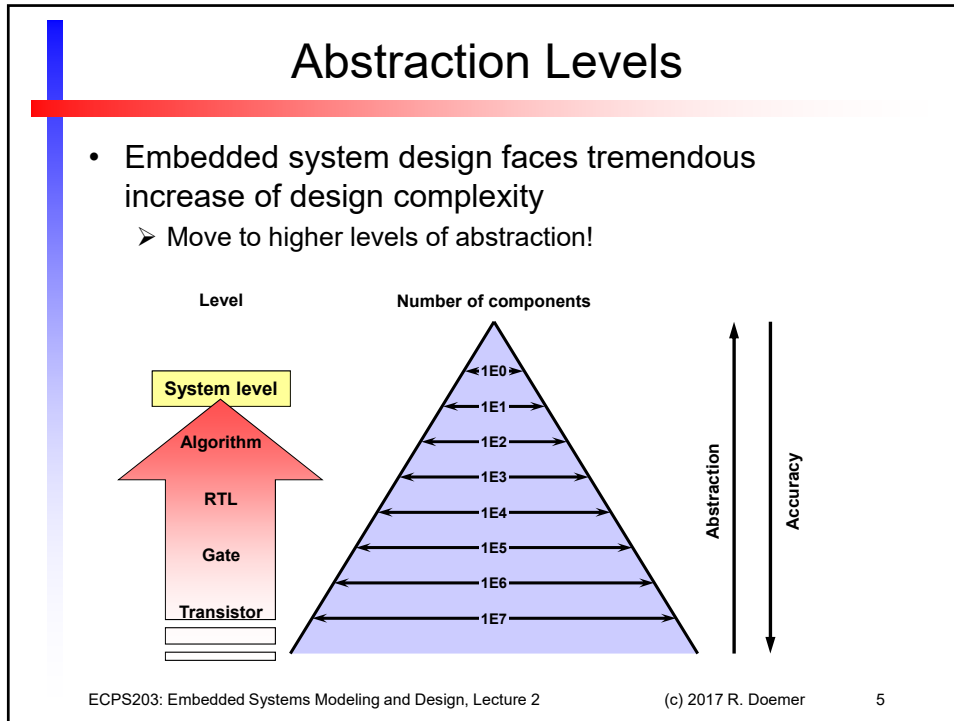
ECPS203: Embedded Systems Modeling and Design, Lecture 2 (c) 2017 R. Doemer 3

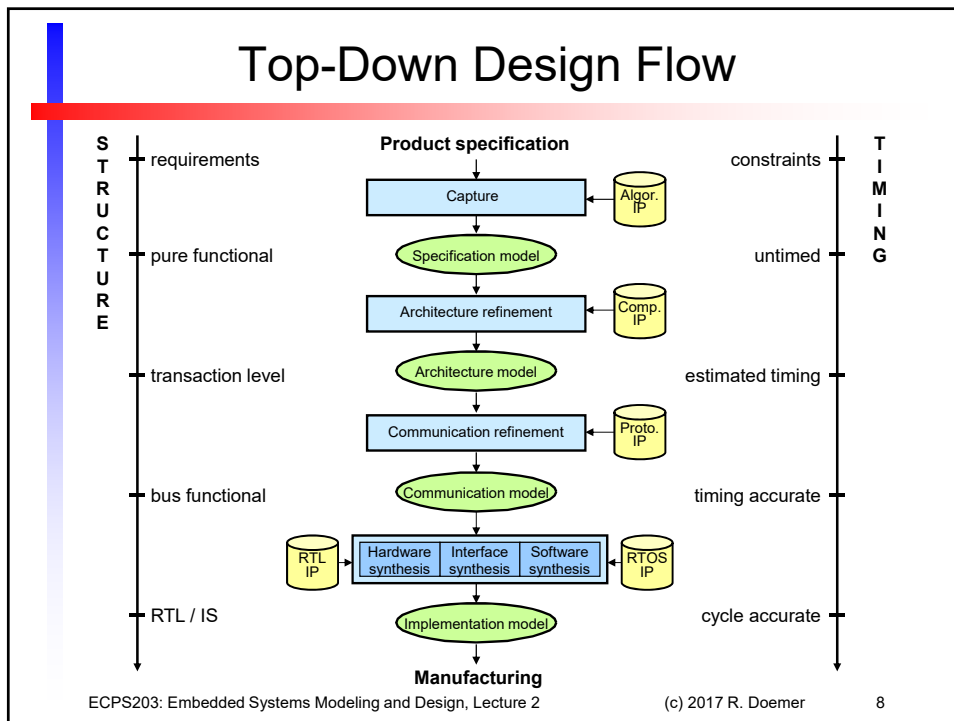
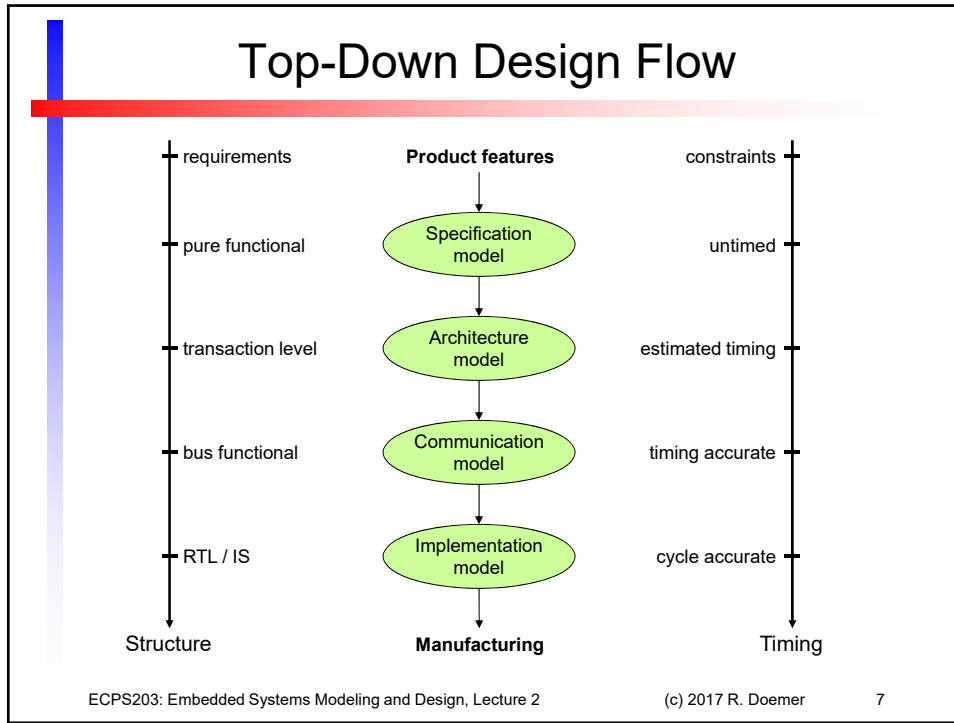
Abstraction Levels

- Embedded system design faces tremendous increase in design complexity

The diagram is a pyramid representing abstraction levels. The levels from top to bottom are: System, Algorithm, RTL, Gate, and Transistor. The number of components at each level is indicated by horizontal arrows: System (1E0), Algorithm (1E1), RTL (1E2), Gate (1E3), and Transistor (1E4). To the right of the pyramid, two vertical arrows indicate that as the abstraction level increases (moving up the pyramid), the level of abstraction increases and the level of accuracy decreases.

ECPS203: Embedded Systems Modeling and Design, Lecture 2 (c) 2017 R. Doemer 4





Abstract Modeling

- Model of Computation
 - Formal description of a system model at high abstraction level
 - Specification
 - Documentation
 - Reasoning
 - Validation
 - Synthesis
- Models for Hardware and Software design
 - State-based models of computation
 - from Finite State Machine (FSM)
 - to Program State Machine (PSM)

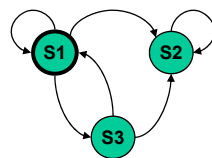
ECPS203: Embedded Systems Modeling and Design, Lecture 2

(c) 2017 R. Doemer

9

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

ECPS203: Embedded Systems Modeling and Design, Lecture 2

(c) 2017 R. Doemer

10

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

The diagram shows a Data Flow Graph (DFG) model. It consists of six nodes representing operations: Op1, Op2, Op3, Op4, Op5, and Op6. Op1, Op2, and Op3 are at the top level, each with an incoming arrow from above. Op1 has arrows pointing to Op5 and Op6. Op2 has an arrow pointing to Op4. Op3 has an arrow pointing to Op4. Op4 has an arrow pointing to Op6. Op5 and Op6 have outgoing arrows pointing downwards.

DFG model

ECPS203: Embedded Systems Modeling and Design, Lecture 2 (c) 2017 R. Doemer 11

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)

The diagram illustrates an FSMD model. It features a central state machine with three states: S1, S2, and S3, represented by green circles. S1 and S2 are at the top, and S3 is at the bottom. Transitions are shown with arrows: S1 to S2, S2 to S1, S2 to S3, and S3 to S1. There are also self-loops on S1 and S2. Three data flow graphs (DFGs) are shown in yellow boxes, each with its own set of operations (Op1-Op6). The DFGs are connected to the state machine: S1 controls the top-left DFG (Op1, Op2), S2 controls the bottom-left DFG (Op1, Op2, Op3), and S3 controls the right DFG (Op1, Op2, Op3, Op4, Op5, Op6). Red arrows indicate the control flow from the state machine to the DFGs.

FSMD model

ECPS203: Embedded Systems Modeling and Design, Lecture 2 (c) 2017 R. Doemer 12

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

```
a = a + b;
c = c + d;
```

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

SFSMD model

```
a = 42;
b = a * 2;
for (c=0; c<100; c++)
{
  b = c + a;
  if (b < 0)
    b = -b;
  else
    b = b + 1;
  a = b * 10;
}
```

ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
13

Models of Computation

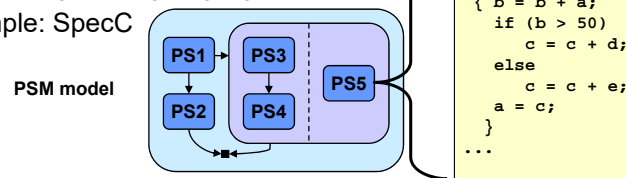
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
- Super-State FSM with Data (SFSMD)
- Hierarchical Concurrent FSM (HCFSM)
 - FSM extended with hierarchy and concurrency
 - Multiple FSMs composed hierarchically and in parallel
 - Example: Statecharts

HCFSM model

ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
14

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



ECPS203: Embedded Systems Modeling and Design, Lecture 2

(c) 2017 R. Doemer

15

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

ECPS203: Embedded Systems Modeling and Design, Lecture 2

(c) 2017 R. Doemer

16

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

ECPS203: Embedded Systems Modeling and Design, Lecture 2 (c) 2017 R. Doemer 17

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

ECPS203: Embedded Systems Modeling and Design, Lecture 2 (c) 2017 R. Doemer 18

System-Level Description Languages

- Examples of Languages in Use Today

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

ECPS203: Embedded Systems Modeling and Design, Lecture 2

(c) 2017 R. Doemer

19

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 (concepts!)**
 - 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

ECPS203: Embedded Systems Modeling and Design, Lecture 2

(c) 2017 R. Doemer

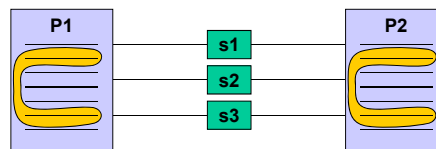
20

Separation of Concerns

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

Computation vs. Communication

- Traditional model
 - Processes and signals



- VHDL example:

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

- Mixture of computation and communication
 - Automatic replacement impossible!

Computation vs. Communication

- SpecC model
 - Behaviors and channels
 - SpecC example:


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

  xy = x + 2 * y;

  c.send(xy);

  v1 = 0;
  [...]
}
```

```
channel C1 [...]
{
  send (int d)
  { v1 = d;
    notify e2;
    wait e3;
  }
  [...]
}
```
- Clear separation of computation and communication
 - Plug-and-play!

ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
23

Computation vs. Communication

- Traditional model

P1

P2

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

B1

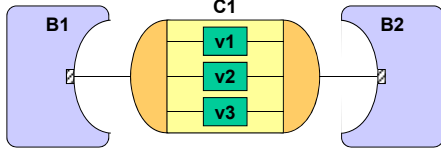
B2

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

ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
24

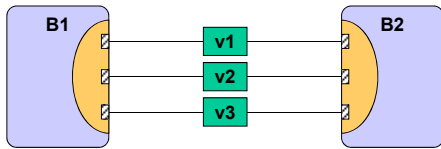
Computation vs. Communication

- System Model
 - Specification
 - Validation
 - Exploration



- Computation in behaviors
- Communication in channels

- Implementation Model
 - Synthesis
 - e.g. Verilog, VHDL, or SystemC

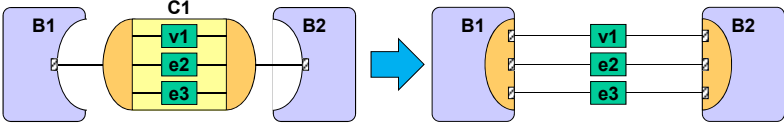


- Channel disappears, signals get exposed
- Communication protocol is *inlined* into behaviors

ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
25

Computation vs. Communication

- Communication Protocol Inlining



– SystemC example:

```
SC_MODULE(M1)
{
  [...]
  c.send(1);

  xy = x + 2 * y;

  c.send(xy);

  v1 = 0;
  [...]
}
```

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

```
SC_MODULE(M1)
{
  [...]
  v1 = 1;
  e2.notify();
  wait(e3);
  xy = x + 2 * y;
  v1 = xy;
  e2.notify();
  wait(e3);
  v1 = 0;
  [...]
}
```

ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
26