ECPS 203
Embedded Systems Modeling and Design
Lecture 2

Rainer Dömer

doemer@uci.edu

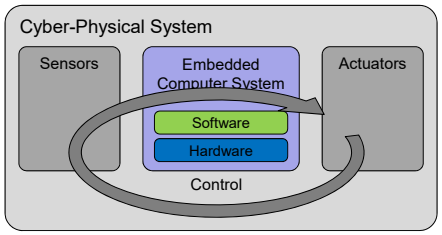
Center for Embedded and Cyber-physical Systems
University of California, Irvine

Lecture 2: Overview

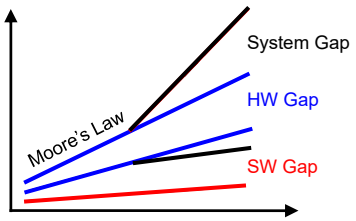
- Embedded System Design
  - Complexity challenges
  - Abstraction Levels
  - Top-down Design Flow
- Abstract Modeling of Embedded Systems
  - Models of Computation
  - System-Level Description Languages
- Separation of Concerns
  - Computation vs. Communication

Embedded System Design

- Embedded System in CPS context
  - Software
  - Hardware

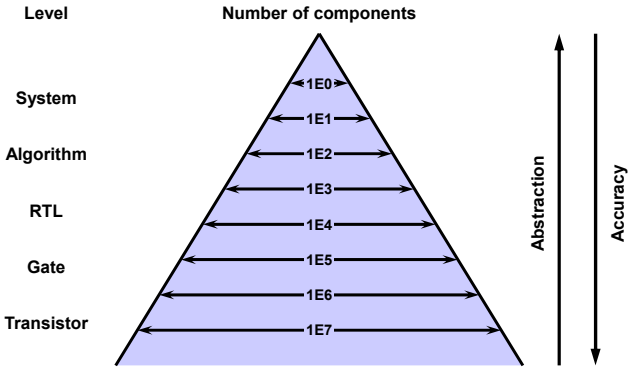


- Design Challenges
  - Hardware design gap
  - Software design gap
  - System design gap



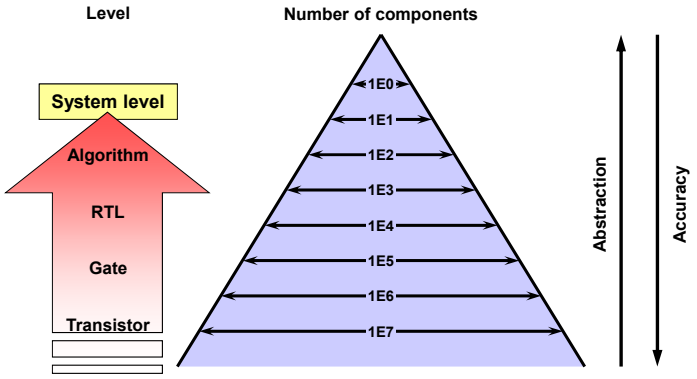
Abstraction Levels

- Embedded system design faces tremendous increase in design complexity

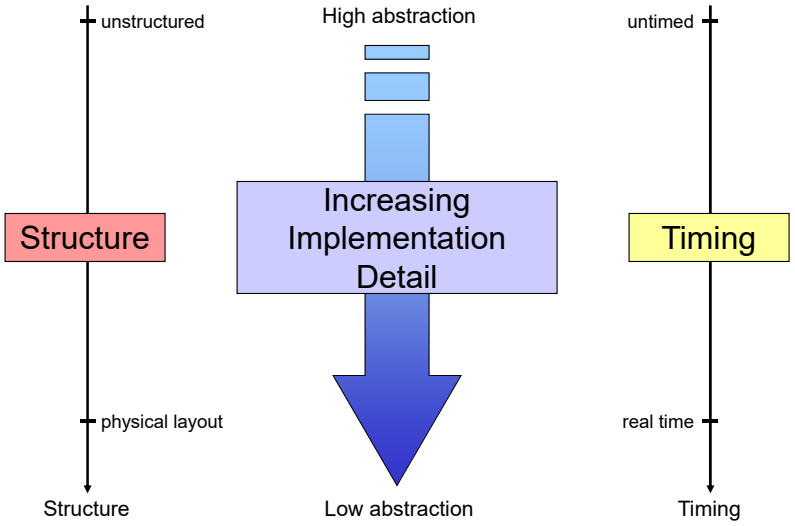


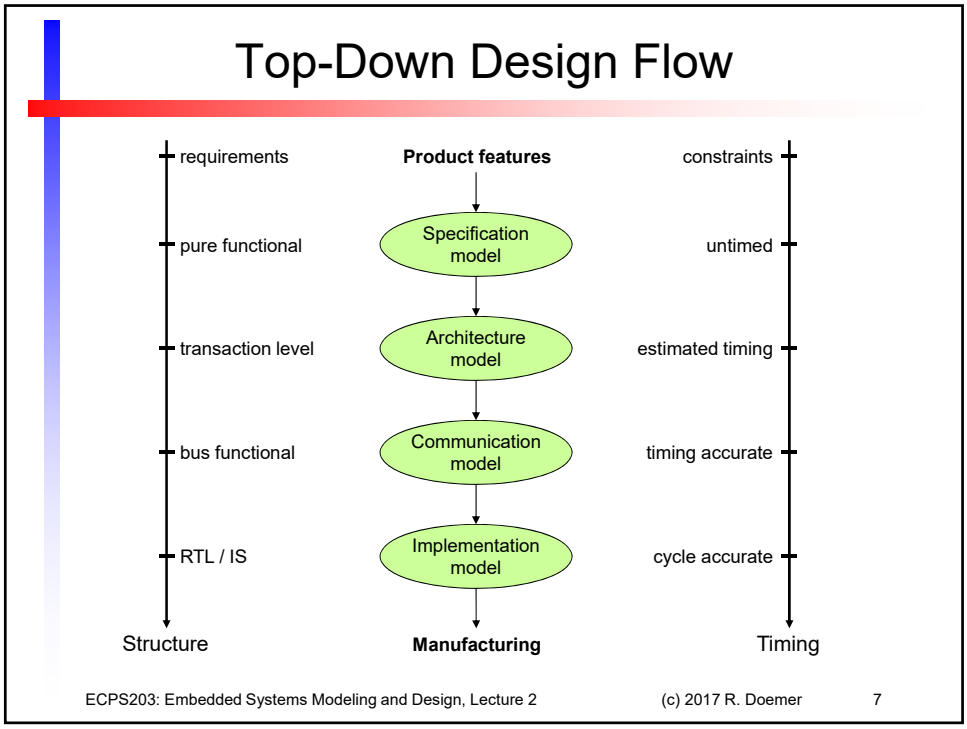
Abstraction Levels

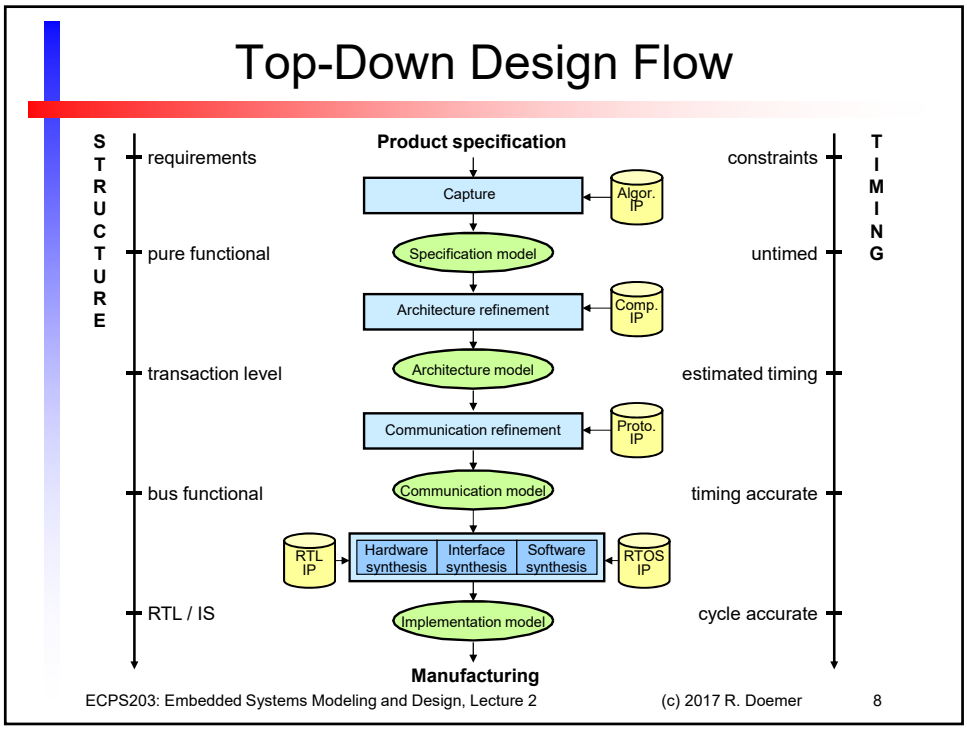
- Embedded system design faces tremendous increase of design complexity
  - Move to higher levels of abstraction!



Abstraction Levels




Top-Down Design Flow
requirements
Product features
constraints
pure functional
Specification model
untimed
transaction level
Architecture model
estimated timing
bus functional
Communication model
timing accurate
RTL / IS
Implementation model
cycle accurate
Structure
Manufacturing
Timing
ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
7

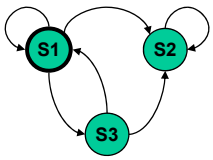

Top-Down Design Flow
STRUCTURE
TIMING
requirements
Product specification
constraints
Capture
Algor. IP
pure functional
Specification model
untimed
Architecture refinement
Comp. IP
transaction level
Architecture model
estimated timing
Communication refinement
Proto. IP
bus functional
Communication model
timing accurate
RTL IP
Hardware synthesis
Interface synthesis
Software synthesis
RTOS IP
RTL / IS
Implementation model
cycle accurate
Manufacturing
ECPS203: Embedded Systems Modeling and Design, Lecture 2
(c) 2017 R. Doemer
8

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)

Models of Computation

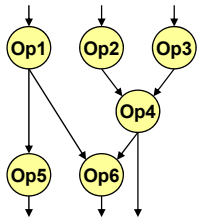
- Finite State Machine (FSM)
  - Basic model for describing control
  - States and state transitions
    - FSM = <*S*, *I*, *O*, *f*, *h*>
  - Two types:
    - Mealy-type FSM (input-based)
    - Moore-type FSM (state-based)



FSM model

Models of Computation

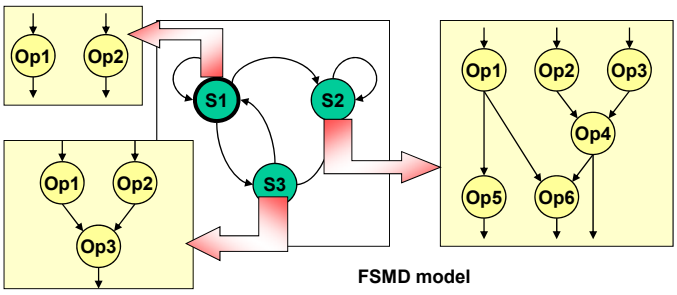
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
  - Basic model for describing computation
  - Directed graph (acyclic)
    - Nodes: operations
    - Edges: data flow, dependency of operations



DFG model

Models of Computation

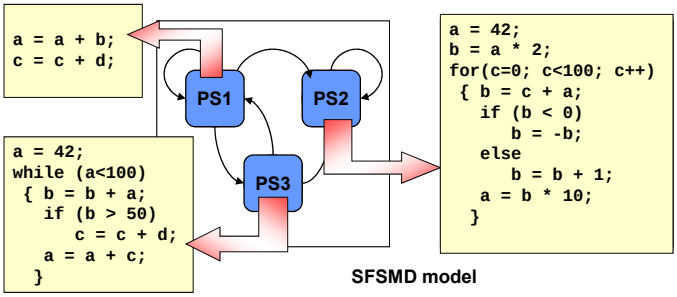
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
  - Combined model for control and computation
    - FSMD = FSM + DFG
  - Implementation: controller plus data path (RTL processor)



FSMD model

Models of Computation

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

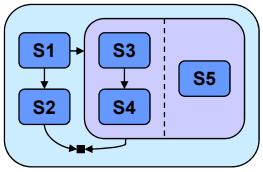


SFSMD model

Models of Computation

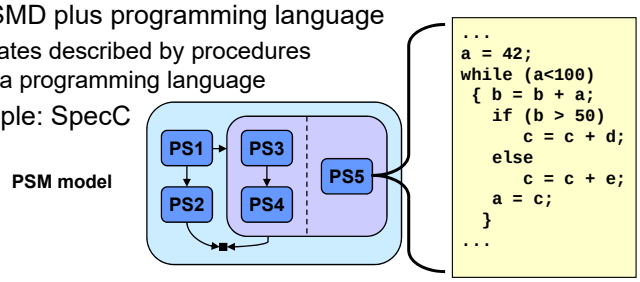
- Finite State Machine (FSM)
- Data Flow Graph (DFG)
- Finite State Machine with Data (FSMD)
- Super-State FSM with Data (SFSMD)
- Hierarchical Concurrent FSM (HCFSM)
  - FSM extended with hierarchy and concurrency
    - Multiple FSMs composed hierarchically and in parallel
  - Example: Statecharts

HCFSM model



Models of Computation

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

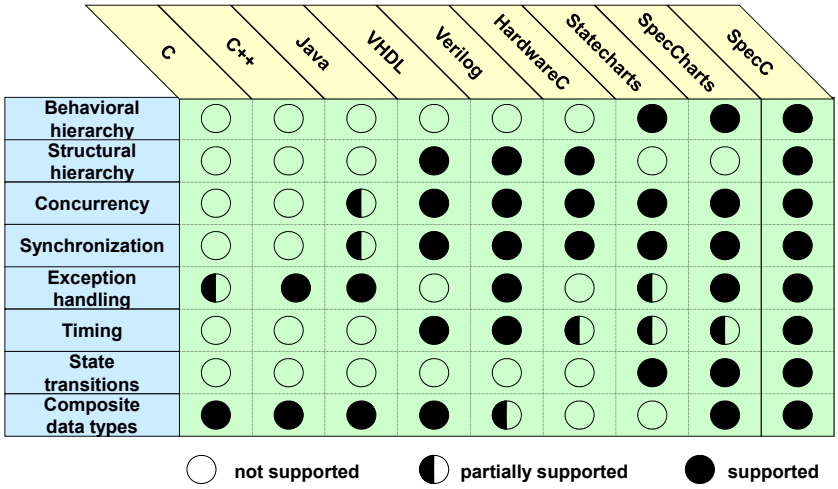


System-Level Description Languages

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

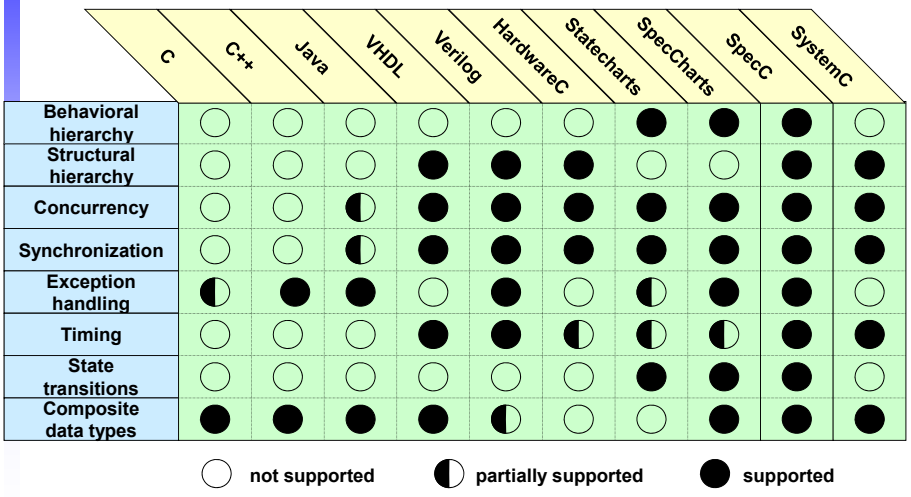
System-Level Description Languages

- Requirements supported by existing languages



System-Level Description Languages

- Requirements supported by existing languages



System-Level Description Languages

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

System-Level Description Languages

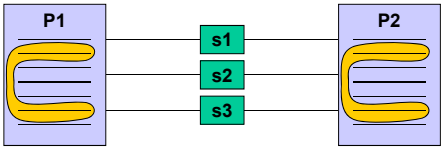
- Examples of Languages in Use Today, Course Coverage
  - C/C++
    - ANSI standard programming languages, software design
    - Initially used for system design because of availability, practicality
  - ➢ SystemC
    - IEEE standard 1666-2011 (initially created at UCI, standardized by OSCI)
    - C++ library and application programming interface (API)
  - ➢ SpecC (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

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



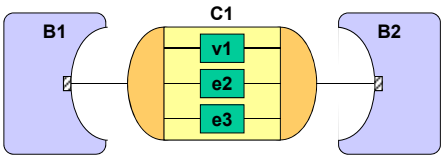
  - VHDL example:

```
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:

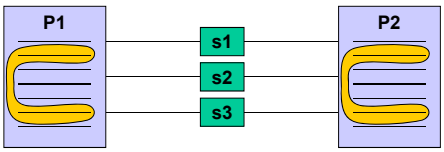
```
behavior B1 […]
{
  c.send(1);

  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!

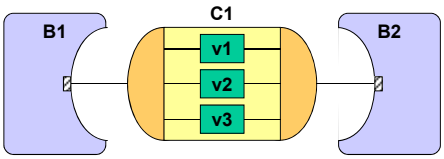
Computation vs. Communication

- Traditional model



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

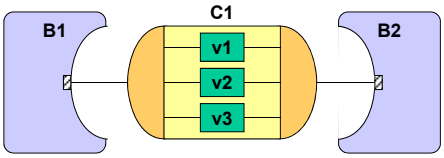
- SpecC model



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

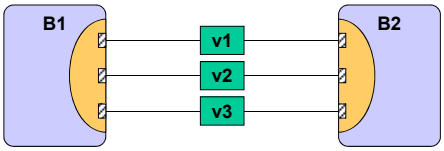
Computation vs. Communication

- System Model
  - Specification
  - Validation
  - Exploration



- Computation in behaviors
- Communication in channels

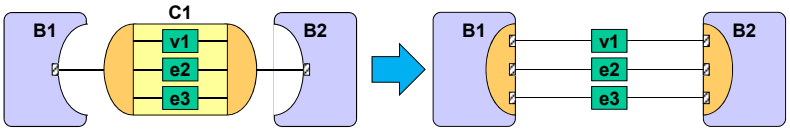
- Implementation Model
  - Synthesis
    - e.g. Verilog, VHDL, or SystemC



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

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;
  […]}
```