ECE 298:
System-on-Chip Description and Modeling
Lecture 7

Rainer Dömer

doemer@uci.edu

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

Lecture 7: Overview

- Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- SLDL Execution and Simulation Semantics
  - Motivation
  - System-level Language Semantics
  - Formal Execution Semantics
    - Time-interval formalism
    - Abstract State Machines
  - Simulation Semantics

Lecture 7: Overview

- ➢ Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- SLDL Execution and Simulation Semantics
  - Motivation
  - System-level Language Semantics
  - Formal Execution Semantics
    - Time-interval formalism
    - Abstract State Machines
  - Simulation Semantics

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 3

Homework Assignment 3

- Discussion
  - Task 1:
    - Complete the MPEG-Audio Specification Model such that the model ...

STATUS

Task 1a: – ... contains all necessary code for the decoder
Task 1b: – ... contains a test bench with stimulator, DUT, and monitor
Task 1c: – ... compiles successfully with the SpecC compiler
Task 1d: – ... simulates successfully

  - Task 2:
    - For the decoder (DUT), create the behavioral hierarchy necessary for a well-defined Specification Model

Task 2a: – Granularity: each function becomes a behavior
Task 2b: – Hierarchy: try to mimic the given functional hierarchy
Task 2c: – Concurrency: add explicit concurrency wherever possible
Task 2d: – Communication: use standard channels or local variables

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 4

Lecture 7: Overview

- Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- SLDL Execution and Simulation Semantics
  - Motivation
  - System-level Language Semantics
  - Formal Execution Semantics
    - Time-interval formalism
    - Abstract State Machines
  - Simulation Semantics

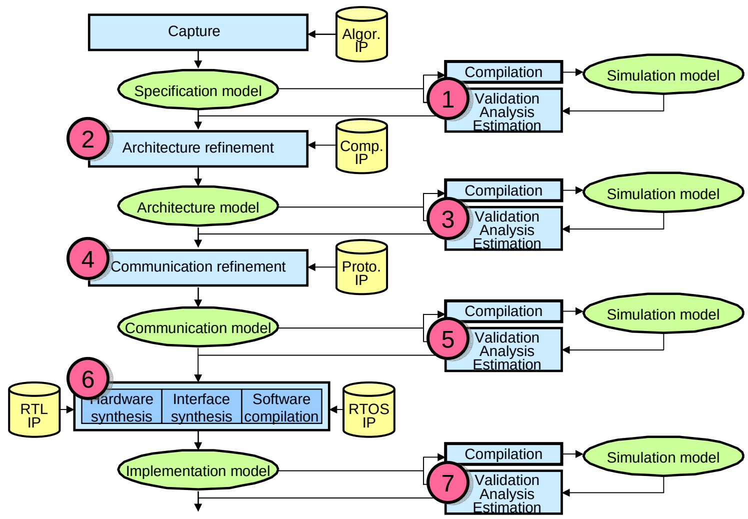
ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 5

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:
    - Explore possible system architectures with SCE
      - Use different number and type of components
      - Use different mapping
      - Use different communication
      - Estimate the performance and cost for each option
  - Task 3:
    - Generate the “best” system architecture for the design

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 6

Homework Assignment 4



ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 7

Lecture 7: Overview

- Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- ➢ SLDL Execution and Simulation Semantics
  - Motivation
  - System-level Language Semantics
  - Formal Execution Semantics
    - Time-interval formalism
    - Abstract State Machines
  - Simulation Semantics

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 8

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

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 9

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!)

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 10

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

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 11

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 7 (c) 2004 R. Doemer 12

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

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

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

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)

Lecture 7: Overview

- Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- SLDL Execution and Simulation Semantics
  - Motivation
  - ➢ System-level Language Semantics
  - Formal Execution Semantics
    - Time-interval formalism
    - Abstract State Machines
  - Simulation Semantics

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 17

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 7 (c) 2004 R. Doemer 18

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!

Lecture 7: Overview

- Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- SLDL Execution and Simulation Semantics
  - Motivation
  - System-level Language Semantics
  - ➢ Formal Execution Semantics
    - Time-interval formalism
    - Abstract State Machines
  - Simulation Semantics

Formal Execution Semantics

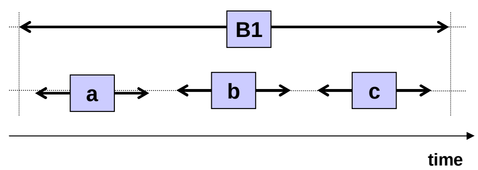
- Two examples of semantics definition:
  1) Time-interval formalism
     - formal definition of timed execution semantics
     - sequentiality, concurrency, synchronization
     - allows reasoning over execution order, dependencies
  2) Abstract State Machines
     - complete execution semantics of 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

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

Tstart(B1) <= Tstart(a) < Tend(a) <=
Tstart(b) < Tend(b) <=
Tstart(c) < Tend(c) <= Tend(B1)

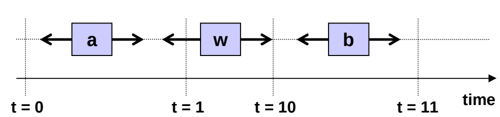


Formal Execution Semantics (1)

- 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



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;}
  }
};
```

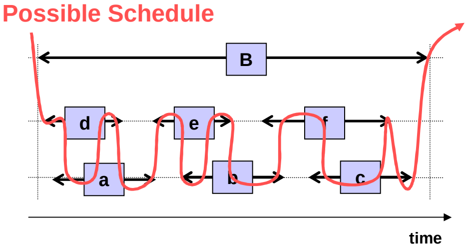
```
behavior B1
{ void main(void)
  { a; b; c; }
};
```

```
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



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;}
  }
};
```

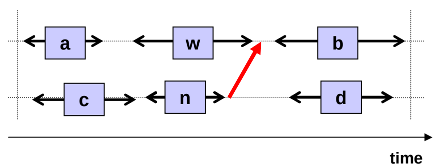
```
behavior B1
{ void main(void)
  { a; wait e;   b; }
};
```

```
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)



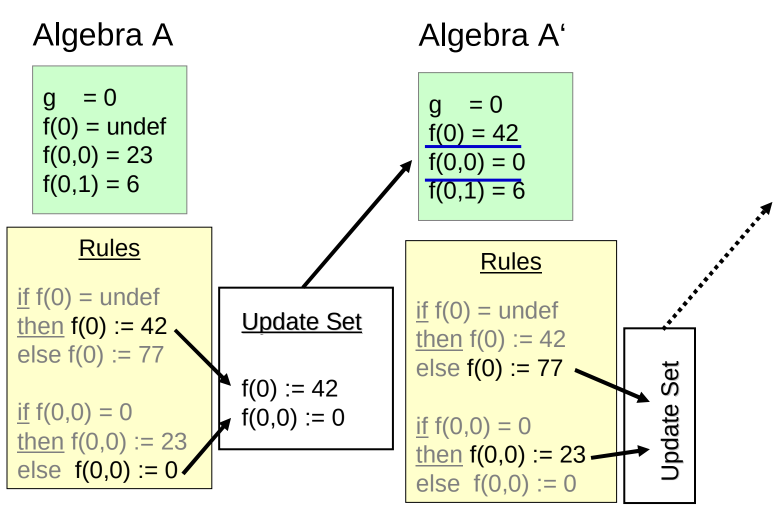
Lecture 7: Overview

- Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- SLDL Execution and Simulation Semantics
  - Motivation
  - System-level Language Semantics
  - Formal Execution Semantics
    - Time-interval formalism
    - ➢ Abstract State Machines
  - Simulation Semantics

Formal Execution Semantics (2)

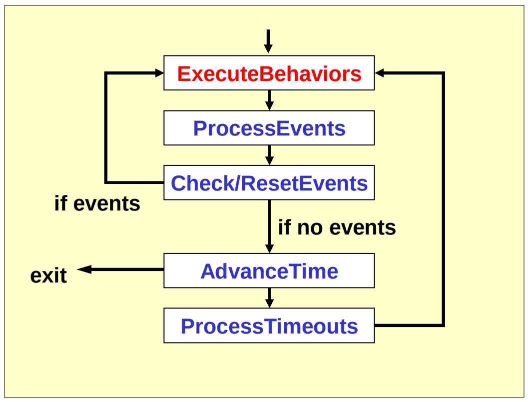
- Abstract State Machine (ASM)
  - aka. Evolving Algebras (Y. Gurevich, 1987)
  - ASM semantics already exist for
    - Prolog, Concurrent Prolog
    - C, C++, Java
    - VHDL, VHDL-AMS, SystemC
  - ASM semantics for SpecC published at ISSS'02
- ASM components
  - Sequence of algebras (functions over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

Abstract State Machine (ASM)



ASM: SpecC Kernel Semantics

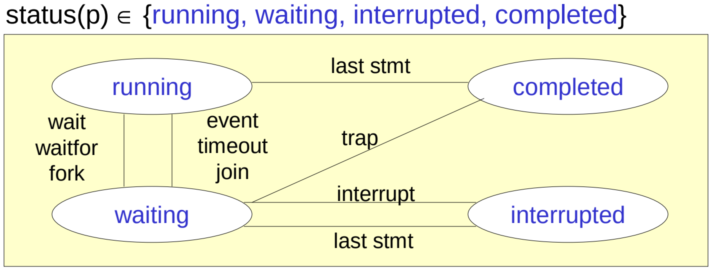
- Phase 1: at least one BEHAVIOR is running
- Phase 2: all BEHAVIORs are not running



ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 31

ASM: SpecC Behavior Semantics

p ∈ BEHAVIOR:
status(p) ∈ {running, waiting, interrupted, completed}



- 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 7 (c) 2004 R. Doemer 32

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 7 (c) 2004 R. Doemer 33

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 7 (c) 2004 R. Doemer 34

Lecture 7: Overview

- Homework Assignment 3
  - Discussion
- Homework Assignment 4
  - Design Exploration and Refinement
- SLDL Execution and Simulation Semantics
  - Motivation
  - System-level Language Semantics
  - Formal Execution Semantics
    - Time-interval formalism
    - Abstract State Machines
  - ➢ Simulation Semantics

ECE298: SoC Description and Modeling, Lecture 7 (c) 2004 R. Doemer 35

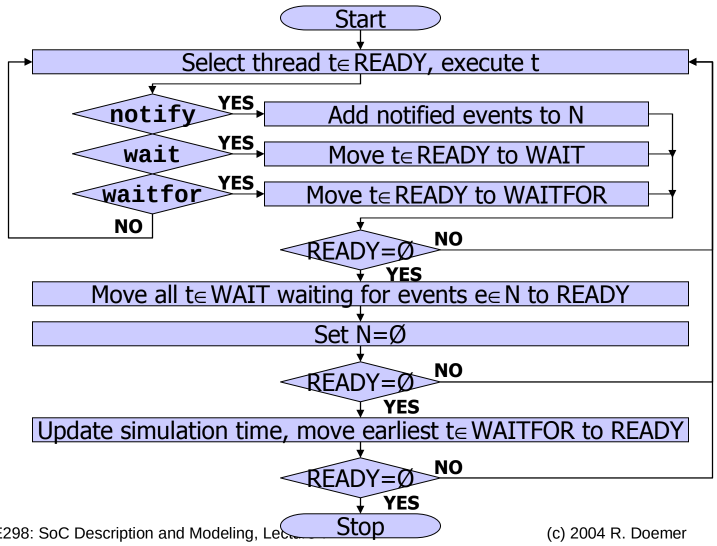
Simulation Semantics

- 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 7 (c) 2004 R. Doemer 36

Simulation Semantics

- Abstract Simulation Algorithm for SpecC



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