Lecture 4 – System Specification

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Lecture 4: Outline

• System specification
  • Essential issues
  • SpecC specification modeling guidelines
  • C-to-SpecC recoding

• Design example
  • SUSAN edge detection specification

• Models of Computation (MoCs)
  • Formal models of concurrency & communication
System Specification

- System behavior
  - Functionality
- System constraints
  - Non-functional requirements

Essential Issues in Specification

- An Example ...

Source: unknown author. Courtesy of: R. Doemer
**Specification Model**

- **High-level, abstract model**
  - Pure system functionality
  - Algorithmic behavior
  - No implementation detail

- **No implicit structure / architecture**
  - Behavioral hierarchy

- **Untimed**
  - Executes in zero (logical) time
  - Causal ordering
  - Events only for synchronization
Specification Model

- **Top-level Main behavior**
  - Test bench
    - Stimulus provides test vectors
    - Monitor observes and checks outputs
      - no restrictions in syntax and semantics (no synthesis)
  - Design under test (DUT)
    - restricted by syntax and semantic rules (synthesis!)

![Diagram of Stimulus, DUT, and Monitor](image)

Source: R. Doemer, UC Irvine

Specification Modeling Guidelines

- **“Golden” reference model**
  - First functional model in the design flow
  - All other models derived from and compared to this one
- **High abstraction level**
  - No implementation details
  - Unrestricted exploration of design space
- **Pure functional**
  - Executable for functional validation
  - No structural information
- **No timing**
  - Exception: timing constraints
- **Separation of communication and computation**
  - Channels and behaviors
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: un-timed, partial ordering

- **Communication: Channels**
  - Semantics: abstract communication, synchronization (standard channel library)
  - Dependencies: explicit data dependency, partial ordering, port connectivity

**Example rules for SoC Environment (SCE)**

- Clean behavioral hierarchy
  - hierarchical behaviors: no code other than `seq`, `par`, `pipe`, `fsm` statements
  - leaf behaviors: no SpecC code (pure ANSI-C code only)
- Clean communication
  - point-to-point communication via standard channels:
    - `c_handshake`, `c_semaphore`,
    - `c_double_handshake`, `c_queue` (typed or untyped)
  - ports of plain ANSI C type or interface type, no pointers!
  - port maps to local variables or ports only

- **Detailed rules for SoC Environment**
    by A. Gerstlauer, R. Doemer, CECS, UC Irvine, April 2005
Specification Modeling Guidelines

- **C code conversion to SpecC**
  - 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**
  - Task and data 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
  - Floating-point to fixed-point conversion

Lecture 4: Outline

- **System specification**
  - Essential issues
  - Specification modeling guidelines
  - C-to-SpecC recoding
- **Design example**
  - SUSAN edge detection specification
- **Models of Computation (MoCs)**
  - Formal models of concurrency & communication
Image Edge Detection

- **Identifying points at which the brightness changes sharply**
  - Capture important events and changes

- **Edge sources (brightness changes)**
  - Discontinuities in depth
  - Discontinuities in surface orientation, and material properties
  - Variations in scene illumination

- **Application**
  - Computer vision
    - Industrial robots, autonomous vehicles or mobile robots, medical image analysis, etc.
  - Machine vision
    - Quality assurance, sorting, material handling, robot guidance, and optical gauging

- **Smallest Unvalue Segment Assimilating Nucleus (SUSAN)**
  - Edge detection, corner detection and image noise reduction

Susan Edge Detector Specification
Lecture 4: Outline

✓ System specification
  ✓ Essential issues
  ✓ Specification modeling guidelines
  ✓ C-to-SpecC recoding

✓ Design example
  ✓ SUSAN edge detection specification

• Models of Computation (MoCs)
  • Formal models of concurrency & communication

Models of Computation (MoCs)

• Conceptual ways of describing system behavior
  • Distinguish abstract classes of behavioral modeling
    • Concurrency and time (order)
    • Computation and communication
  • Decomposition into objects and their relationship
    • Composition rules
    • Data and control flow
  • Unambiguous, formal definition and semantics
    • Analysis, synthesis, verification

➢ Formally validate functional correctness of specification
➢ Analyzability vs. expressiveness of specification models
  ➢ Fundamental tradeoffs between the two
  ➢ Implementability & predictability
Recap: Concurrency

- **Logical concurrency**
  - Partial order

- **Causality**
  - Restrictions on order

- **Fundamental issues**
  - Non-determinism
  - Deadlocks

Determinism

- **Deterministic**: same inputs always produce same results
- **Random**: probability of certain behavior
- **Non-deterministic**: undefined behavior (for some inputs)
  - Undefined execution order
    - Statement evaluation in imperative languages: $f(a++, a++)$
    - Process & thread race conditions:
      
      \[
      \begin{align*}
      x &= a; \\
      y &= b; \\
      a &= 1; \\
      b &= 2; \\
      \end{align*}
      \]

- **Can be desired or undesired**
  - How to ensure correctness?
    - Many possible behaviors, large verification space
    - Simulator will pick one behavior, not sufficient for verification
  - But: over-specification?
    - Leave freedom of implementation choice (concurrency)
Deadlocks

• Circular chain of 2 or more processes which each hold a shared resource that the next one is waiting for
  • Circular dependency through shared resources
  
  m1.lock();
  m2.lock();
  ...
  m2.unlock();
  m1.unlock();
  m2.lock();
  m1.lock();
  ...
  m1.unlock();
  m2.unlock();

  ➢ Prevent chain by using the same precedence
  ➢ Use timeouts (and retry), but: livelock

➢ Dependency can be created when resources are shared
  ➢ Side effects, e.g. when blocking on filled queues/buffers

Formal Model of a Design

• Most tools and designers describe the behavior of a design as a relation between a set of inputs and a set of outputs
  • This relation may be informal, even expressed in natural language
  • Such informal, ambiguous specifications may result in unnecessary redesigns…

• A formal model of a design should consist of the following components:
  • Functional specification
  • Set of properties
  • Set of performance indices
  • Set of constraints on performance indices

Source: M. Jacome, UT Austin.
Formal Model of a Design (2)

- A functional specification, given as a set of explicit or implicit relations which involve inputs, outputs and possibly internal (state) information. This fully characterizes the operation of a system.

- A set of properties that the design must satisfy. Redundant: in a properly constructed system, the functional specification satisfies these properties. Yet properties are simpler / more abstract compared to the functional specification.

- A set of performance indices that evaluate the quality of the design in terms of cost, reliability, speed, size, etc.

- A set of constraints on performance indices, specified as a set of inequalities. This bounds the cost of a system.

Properties

- A property is an assertion about the behavior, rather than a description of the behavior. It is an abstraction of the behavior along a particular axis.

- Examples:
  - Liveness property: when designing a network protocol, one may require that the design never deadlocks.
  - Fairness property: when designing a network protocol, one may require that any request will eventually be satisfied.

- The above properties do not completely specify the behavior of the protocol, they are instead properties we require the protocol to have.

- Can include other non-functional requirements:
  - Timeliness: guarantees about meeting deadlines in the worst case (real-time).

Source: M. Jacome, UT Austin.
Properties & MoCs

- **Properties can be classified in three groups:**
  1. Properties that are *inherent* to the model (i.e., that can be shown formally to hold for *all specifications* described using that model)
  2. Properties that can be verified *syntactically* for a given specification (i.e., that can be shown to hold with a simple, usually polynomial-time analysis of the specification)
  3. Properties that must be verified *semantically* for a given specification (i.e., that can be shown to hold by executing, at least implicitly, the specification for all inputs that can occur)

Model Validation

- **By construction**
  - property is inherent
- **By verification**
  - property is provable syntactically
- **By simulation**
  - check behavior for all inputs
- **By intuition**
  - property is true, I just know it is…

*better be higher in this list…*

Source: M. Jacome, UT Austin.
Model Validation Example

- **Determinate Behavior Property**: the fact that the output of a system depends only on its inputs and not on some internal, hidden choice
  - Any design described by a dataflow network is determinate, and hence this property is *inherent* (that is, need not be checked)
  - If the design is represented by a network of FSMs, determinacy can be assessed by inspection of the state transition function, and hence the property can be verified *syntactically*
  - In the discrete event models embodied in Verilog and VHDL determinacy is difficult to prove, it must be checked by *exhaustive simulation*, and thus the property requires *semantic* verification

Models of Computation (MoCs)

- A MoC is a framework in which to express what actions must be taken to complete a computation
  - Objects and their relationships

- MoCs need to
  - Be *powerful/expressive enough* for the application domain
  - Have appropriate *synthesis* and *validation* semantics

- Why different models?
  - Different models $\Rightarrow$ different properties
  - Turing complete models are too powerful!
  - Existing programming models are poor match
    - Domain-specific models

Source: M. Jacome, UT Austin.
MoCs for Reactive Systems

• Consider essential aspects of reactive systems:
  • Time/synchronization
  • Concurrency
  • Heterogeneity

• Classify models based on
  • How to specify behavior
  • How to specify communication
  • Implementability
  • Composability
  • Availability of tools for validation and synthesis

MoC Examples

• Programming models
  • Imperative [C] or declarative [Lisp, Prolog]
    ➢ Transformative not reactive, no concurrency

• Parallel programming models
  • Threads/processes, multi-tasking/-threading [any (RT)OS]
    ➢ Non-determinism, race conditions, deadlocks
    ➢ Best effort only, incomprehensible to humans/tools [Lee’06]

• Circuit and logic design
  • Finite state machines (FSMs)
    ➢ Synchronous, fine granularity of concurrency

• Timed models
  • Discrete event (DE), synchronous reactive (SR)
    ➢ Global time, simulation but not synthesis
MoCs for System Specification

- **Process-based models**
  - Kahn Process Networks (KPNs)
  - (Synchronous) Dataflow models ((S)DF)
  - …

- **State-based models**
  - Hierarchical, Concurrent State Machines (HCFSM)
  - Petri Nets
  - …

Lecture 4: Summary

- **System specification**
  - Specification modeling guidelines
    - Testbench setup
    - Hierarchy, concurrency, communication
  - Unambiguous, formal definition
    - Intended system behavior
    - Analysis and synthesis

  ➢ How to soundly capture concurrency & order?
  ➢ Models of Computation!

- **Design example**
  - SUSAN
  ➢ Specification development in Lab 1