ECE382N.23: Embedded System Design and Modeling

Lecture 2 – System Specification

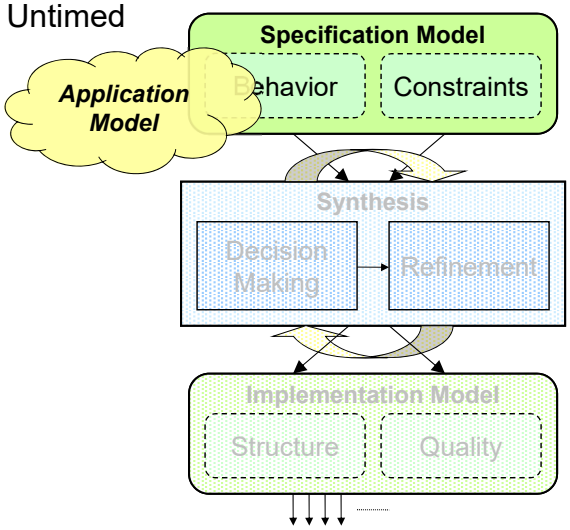
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Lecture 2: Outline

- **System specification**
  - Essential issues
  - Specification modeling guidelines
  - Formal models
- **Models of Computation (MoCs)**
  - Models for reactive systems
  - Concurrency & communication

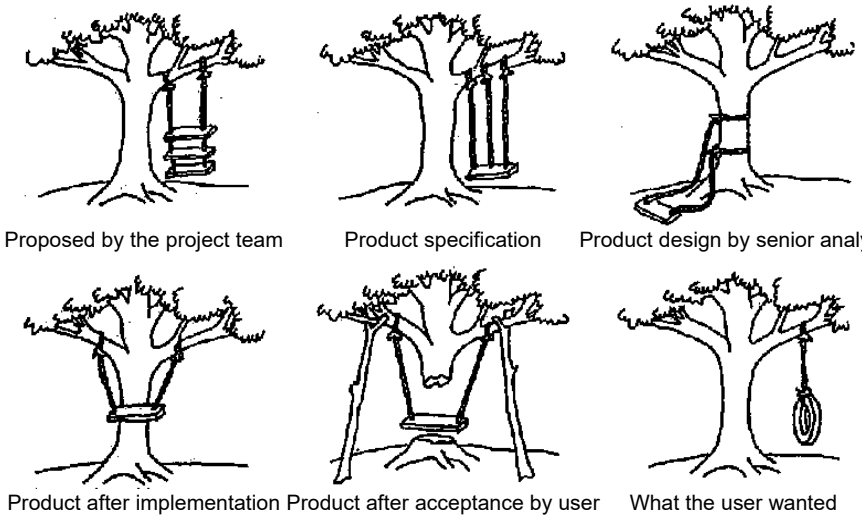
System Specification

- **Desired system behavior**
  - Pure functionality
  - Untimed
- **System constraints**
  - Non-functional requirements



Essential Issues in Specification

- **An Example ...**



Proposed by the project team | Product specification | Product design by senior analyst

Product after implementation | Product after acceptance by user | What the user wanted

Source: unknown author, Courtesy of: R. Doemer

Specification Modeling

- **Executable**
  - Simulate for functional validation
- **Pure behavior: functional, no absolute timing**
  - No structural or implementation information

➢ **"Golden" reference model**
  - First functional model in the design flow
  - All other models derived from and compared to this one

➢ **High abstraction level**
  - No premature implementation details
  - Unrestricted exploration of design space

➢ **Formal**
  - Proof of correctness

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**

  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**

  Bound the cost of a system

Source: M. Jacome, UT Austin.

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 & Models

- **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)

Source: M. Jacome, UT Austin.

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

Source: M. Jacome, UT Austin.

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 ⇒ 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:**
  - Concurrency
  - Order/synchronization
  - Heterogeneity
- **Classify models based on**
  - How to specify behavior (computation)
  - How to specify communication
  - Implementability
  - Composability
  - Availability of tools for validation and synthesis

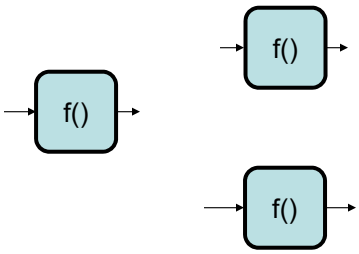
Source: M. Jacome, UT Austin.

Lecture 2: Outline

- ✓ **System specification**
  - ✓ Essential issues
  - ✓ Specification modeling guidelines
  - ✓ Formal models
- **Models of Computation (MoCs)**
  - Models for reactive systems
  - Concurrency & communication

Reactive System Model

- **Concurrent processes**
  - Simultaenous processing of multiple inputs and outputs
  - ➢ Block diagram as graphical representation



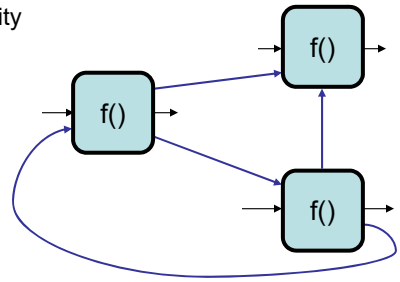
- ➢ **Execute (simulate) and synthesize**
  - Semantics of a block diagram?
  - Concurrency and time (order)?

Models of Time (Order)

- **Untimed**
  - Partial order based on causality only
    - No ordering in time, explicit dependencies only
    - ➢ Free of implementation (purely behavioral)
    - ➢ Specification & programming, Models of computation
- **Logical**
  - Discrete time, partial order
    - Discrete instants of time (time tags $t_0 < t_1 < \cdots t_k < \cdots$), nothing in between
    - Unspecified interleaving of events with same time tag
    - ➢ Freedom of implementation
    - ➢ Simulation & execution, Design languages
- **Physical**
  - Continuous time, total order
    - Physical components naturally interleaved in (very fine) time
    - ➢ Differential equations, Hybrid models

Concurrency

- **Events/actions happening "at the same time"**
  - Undefined, unspecified or unknown order
    - Implementation will determine actual interleaving
- **Communication/synchronization establishes order**
  - Partial order, causal dependencies
    - Behavior/functionality



- **Fundamental issues: communication semantics**
  - ➢ Non-determinism, causality loops
  - ➢ 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:

```
x = a;
y = b;
```

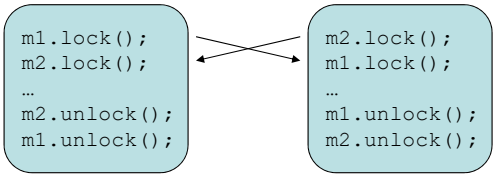
```
a = 1;
b = 2;
```

x = ?, y = ?

- ➢ **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



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

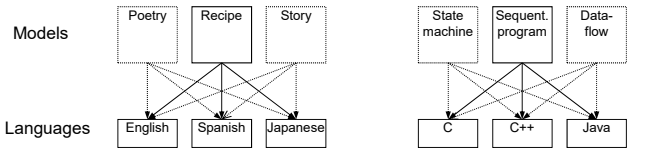
- ➢ **Dependency can be created when resources are shared**
  - ➢ Side effects, e.g. when blocking on filled queues/buffers

Models of Computation (MoCs)

- **Conceptual ways of describing system behavior**
  - Semantics of behavioral models
    - 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

Models vs. Languages



Recipes vs. English **Sequential programs vs. C**

- **Computation models describe system behavior**
  - Conceptual notion, e.g., recipe, sequential program
- **Languages capture models**
  - Concrete form, e.g., English, C
- **Variety of languages can capture one model**
  - E.g., sequential program model → C,C++, Java
- **One language can capture variety of models**
  - E.g., C++ → sequential program model, object-oriented model, state machine model
- **Certain languages better at capturing certain models**

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

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]
- **Control and logic design**
  - Finite state machines (FSMs), synchronous reactive (SR)
  - ➢ Synchronous, fine granularity of concurrency
- **Hardware description languages (HDLs)**
  - Discrete event (DE)
  - ➢ 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 2: Summary

- **System specification**
  - „Golden“ input to design flow
- **Specification modeling guidelines**
  - Hierarchy, concurrency, communication
- **Unambiguous, formal definition**
  - Intended system behavior
  - Analysis and synthesis

➢ **How to soundly capture concurrency & order?**
  - ➢ Formal Models of Computation (MoCs)