Lecture 2: Outline

- Methodologies
  - Design flows
  - Bottom-up, top-down, meet-in-the-middle, platform-based

- Modeling
  - System design flow

- System design languages
  - Goals, requirements
  - Communication and computation
Evolution of Design Flows

<table>
<thead>
<tr>
<th>Capture &amp; Simulate</th>
<th>Describe &amp; Synthesize</th>
<th>Specify, Explore &amp; Refine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specs</td>
<td>Specs</td>
<td>Executable Spec</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Algorithms</td>
<td>Algorithms</td>
</tr>
<tr>
<td>SW?</td>
<td>SW?</td>
<td>Architecture</td>
</tr>
<tr>
<td>Design</td>
<td>Design</td>
<td>SW/HW</td>
</tr>
<tr>
<td>Logic</td>
<td>Logic</td>
<td>Logic</td>
</tr>
<tr>
<td>Physical</td>
<td>Physical</td>
<td>Physical</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Manufacturing</td>
<td>Manufacturing</td>
</tr>
<tr>
<td>1960's</td>
<td>1980's</td>
<td>2000's</td>
</tr>
</tbody>
</table>

System Gap

Classical System Design Flow

System requirement specification
System architecture design
Modeling
Hardware design
Software development
Integration & Verification
System

Source: D. Gajski, UC Irvine
Classical Design Cycle

Electronic System-Level (ESL) Design Flow
New ESL Design Cycle

Task
- Specification (high-level & arch. models)
- HW design
- HW verification
- SW design
- SW verification
- Integration & verification
- Fixes in specification
- Fixes in hardware
- Fixes in software

Time

Design Flow

- **Design methodology**
  - Sequence of design models
  - Flow of transformations between models

- **Models**
  - Well-defined, rigorous semantics
  - Systematic flow from specification to implementation

- **Languages**
  - Representation of models in machine-readable form
Y-Chart

Models of Computation (MoCs)
- Specification
- Algorithm
- Boolean logic
- Transfer

Models of Structure (MoSS)
- PE_Bus
- RTL
- Gates
- Transistors

Behavior (Function)
- Processor
- Logic
- Circuit

Structure (Netlist)
- System
- Processor
- Logic
- Circuit
- Physical (Layout)

Processor Synthesis

- Software processor
  - Compilation and linking

- Hardware processor
  - High-level synthesis

Source: D. Gajski, UC Irvine
System Synthesis

- **Structure**
  - Partitioning, mapping

- **Timing**
  - Scheduling

---

Bottom-Up Methodology

- Each level generates library for the next higher level
  - Circuit: Standard cells for logic level
  - Logic: RTL components for processor level
  - Processor: Processing and communication components for system level
  - System: System platforms for different applications

- Floorplanning and layout on each level

---

Source: D. Gajski, UC Irvine
Top-down Methodology

- Functional description is converted into component netlist on each level
- Each component function is decomposed further on the next abstraction level
- Layout is given only for transistor components

Meet-in-the-Middle Methodology

- Gate netlist is hand-off
- Three levels of synthesis
  - System is synthesized with processor components
  - Processor components are synthesized with RTL library
  - RTL components are synthesized with standard cells
- Two levels of layout
  - System layout is performed with standard cells
  - Standard cells layout with transistors

Source: D. Gajski, UC Irvine
Platform-Based Design

- Meet-in-the-middle at the system level
  - System platform with standard components
  - System design reduced to mapping of specification onto pre-defined platform

Lecture 2: Outline

- Introduction
- Methodologies
  - Modeling
    - System design flow
  - System design languages
  - Design example
System Modeling

- **Basis of any design flow and design automation**
  - Inputs and outputs of design steps
    - Capability to capture complex systems
    - Precise, complete and unambiguous
  - Models at varying levels of abstraction
    - Level and granularity of implementation detail
    - Speed vs. accuracy

- **Design models as an abstraction of a design instance**
  - Representation of some aspect of reality
    - Virtual prototyping for validation through simulation or formal analysis
  - Specification for further implementation/synthesis
    - Describe desired functionality

- **Documentation & specification**
  - Abstraction to hide details that are not relevant or not yet known
  - Different parts of the model or different use cases for the same model

Abstraction Levels

```
Spatial order
  physical layout

High abstraction

Implementation Detail

Low abstraction

Timing
  real time

Temporal order

Structure

Spatial order

Implementation Detail

Temporal order

Unstructured

Untimed
```
Top-Down Design Flow

- **Product planning**
  - **Constraints**
  - **Product planning**

- **System Design**
  - **Specification**
    - **Untimed**
    - **Pure functional**
  - **Architecture**
    - **Timing accurate**
    - **Bus functional**
  - **Processor Design**
    - **Timing accurate**
    - **RTL / ISA**
  - **Implementation**
    - **Cycle accurate**
    - **Gates**
  - **Logic Design**
    - **Gate delays**

- **Structure**
- **Timing**

---

Top-Down Design Flow

- **Product planning**
  - **Constraints**

- **System Design**
  - **Specification model**
    - **Untimed**
    - **Pure functional**
  - **Timed model**
    - **Scheduled**
    - **Partitioned**
  - **Transaction-level model**
    - **Timing accurate**
    - **Bus functional**
  - **Implementation model**
    - **Cycle accurate**
    - **RTL / ISA**
  - **Logic design**

- **Structure**
- **Timing**
**Top-Down Design Flow**

- **requirements**
  - pure functional
  - transaction level
  - bus functional
  - RTL / IS

- **Product planning**
  - Capture
    - Specification model
    - Computation refinement
    - Timed model
    - Communication refinement
    - Transaction-level model
    - Hardware synthesis
    - Interface synthesis
    - Software synthesis
    - Implementation model
    - RTL
    - IP

- **constraints**
  - untimed
  - estimated timing
  - timing accurate
  - cycle accurate

- **Implementation model**
  - Software synthesis
  - Interface synthesis
  - Hardware synthesis
  - RTL
  - IP

**SpecC Design Methodology**

- **System design**
  - Capture
    - Specification model
    - Computation refinement
    - Timed model
    - Communication refinement
    - Transaction-level model
    - Hardware synthesis
    - Interface synthesis
    - Software synthesis
    - Implementation model
    - RTL
    - IP

- **Validation flow**
  - Compilation
    - Validation Analysis Estimation
    - Simulation model
  - Compilation
    - Validation Analysis Estimation
    - Simulation model
  - Compilation
    - Validation Analysis Estimation
    - Simulation model
  - Compilation
    - Validation Analysis Estimation
    - Simulation model
  - Compilation
    - Validation Analysis Estimation
    - Simulation model
  - Compilation
    - Validation Analysis Estimation
    - Simulation model
**Lecture 2: Outline**

- Introduction
- Methodologies
- Modeling
  - System design languages
    - Goals, requirements
    - Communication and computation
  - Design example

**Models vs. Languages**

- 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 \( \rightarrow \) C, C++, Java
- One language can capture variety of models
  - E.g., C++ \( \rightarrow \) sequential program model, object-oriented model, state machine model
- Certain languages better at capturing certain models

Text vs. Graphics

- Models versus languages not to be confused with text versus graphics
  - Text and graphics are just two types of languages
    - Text: letters, numbers
    - Graphics: circles, arrows (plus some letters, numbers)

Simulation vs. Synthesis

- Ambiguous semantics of languages
  - Simulatable but not synthesizable or verifiable
    - Impossible to automatically discern implicit meaning
    - Need explicit set of constructs

Source: D. Gajski, UC Irvine
Languages

- Represent a model in machine-readable form
  - Apply algorithms and tools
- Syntax defines grammar
  - Possible strings over an alphabet
  - Textual or graphical
- Semantics defines meaning
  - Mapping onto an abstract state machine model
    - Operational semantics
  - Mapping into a mathematical domain (e.g. functions)
    - Denotational semantics
- Semantic model vs. design models
  - Basic semantic models can represent many design models
    - Discrete event model for hardware and system simulation
  - Design models can be represented in different languages

Evolution of Design Languages

- Netlists
  - Structure only: components and connectivity
    - Gate-level [EDIF], system-level [SPIRIT/XML]
- Hardware description languages (HDLs)
  - Event-driven behavior: signals/wires, clocks
  - Register-transfer level (RTL): boolean logic
    - Discrete event [VHDL, Verilog]
- System-level design languages (SLDLs)
  - Software behavior: sequential functionality/programs
    - C-based, event-driven [SpecC, SystemC, SystemVerilog]
### System-Level Design Languages (SLDLs)

#### Goals

- **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
  - Minimality
- **Simplicity**

#### Requirements

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>C++</th>
<th>Java</th>
<th>VHDL</th>
<th>Verilog</th>
<th>SystemC</th>
<th>SpecCharts</th>
<th>SpecC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral hierarchy</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
<tr>
<td>Structural hierarchy</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
<tr>
<td>Concurrency</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
<tr>
<td>Synchronization</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
<tr>
<td>Exception handling</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
<tr>
<td>Timing</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
<tr>
<td>State transitions</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
<tr>
<td>Composite data types</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
<td>♦</td>
</tr>
</tbody>
</table>

- ♦ not supported
- ♦ partially supported
- ♦ supported

Source: R. Doemer, UC Irvine
### System-Level Design Languages (SLDLs)

- **C/C++**
  - ANSI standard programming languages, software design
  - Traditionally used for system design because of practicality, availability
- **SystemC**
  - C++ API and class library
  - Initially developed at UC Irvine, standard by Open SystemC Initiative (OSCI)
- **SpecC**
  - C extension
  - Developed at UC Irvine, standard by SpecC Technology Open Consortium (STOC)
- **SystemVerilog**
  - Verilog with C extensions for testbench development
- **Matlab/Simulink**
  - Specification and simulation in engineering, algorithm design
- **Unified Modeling Language (UML)**
  - Software specification, graphical, extensible (meta-modeling)
  - Modeling and Analysis of Real-time and Embedded systems (MARTE) profile
- **IP-XACT**
  - XML schema for IP component documentation, standard by SPIRIT consortium
- **Rosetta (formerly SLDL)**
  - Formal specification of constraints, requirements
- **SDL**
  - Telecommunication area, standard by ITU
- **...**

*Source: R. Doemer, UC Irvine*

### 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

*Source: R. Doemer, UC Irvine*
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

Source: R. Doemer, UC Irvine

Computation vs. Communication

• Protocol Inlining
  • Specification model
  • Exploration model
    – Computation in behaviors
    – Communication in channels

• Implementation model
  – Channel disappears
  – Communication inlined into behaviors
  – Wires exposed

Source: R. Doemer, UC Irvine