EE382V-ICS: System-on-a-Chip (SoC) Design

Lecture 8 - System Design Methodology

with sources from:
Christian Haubelt, Univ. of Erlangen-Nuremberg

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

SoC Design Flow

Design Convergence and Verification Loop

Product Validation Loop
Design Convergence

- Reduced convergence time due to minimal data
- Convergence time increases due to more design data
- Reduced convergence time due to reduced solution space
- Convergence time increases due to transition phase
- Reduced convergence time due to reduced solution space
- Design Converges

Rapid Exploration
Rapid Traversal

Design Challenges

- **Complexity**
  - High degree of parallelism at various levels

- **Heterogeneity**
  - Of components
  - Of tools

- **Low-level communication mechanisms**

- **Programming model**

Source: C. Haubelt, Univ. of Erlangen-Nuremberg
Complexity Forces

- Technology churn
- Performance
- Capacity
- Availability
- Functionality
- Throughput
- Security
- Reliability
- Cost
- Compatibility
- Robustness

“The challenge over the next 20 years will not be speed or cost or performance; it will be a question of complexity.”

Bill Raduchel, Chief Strategy Officer, Sun Microsystems

Multi-Processor System-on-Chip (MPSoC)

Source: C. Haubelt, Univ. of Erlangen-Nuremberg
MPSoC Terminology

- **Multi-processor**
  - Heterogeneous, asymmetric multi-processing (AMP)
  - Distributed memory and operating system

- **Multi-core**
  - Homogeneous, symmetric multi-processing (SMP)
  - Shared memory and operating system
  - Multi-core processors in a multi-processor system

- **Many-core**
  - > 10 cores per processor...

Processor Implementation Options

Lecture 8: Outline

- Introduction
  - System design methodology
    - Electronic system-level design (ESL/SLD)
  - ESL design
    - Modeling
    - Synthesis
    - Verification
  - ESL landscape
  - Summary and conclusions

System Design

System-level design

Hardware development

Software development

Integration & Verification
Classical System Design Flow

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

Hardware-Centric Design Cycle

Task

- Specification
- HW design
- SW design
- HW verification
- SW verification
- Integr. & verification
- Fixes in specification
- HW in hardware
- Fix. in software
- Fix. in HW
- Integration & verification

Time
Hardware-Centric Design Cycle

but you want to know here

... and here

... and here

known if project is successful

Electronic System-Level (ESL) Design Flow

System requirement specification
High-level model
System-level design
Hardware design
Software development
Integration & Verification
System implementation

(semi)automatic
New ESL Design Cycle

Time

Task

Specification
(high-level & arch. models) Fixes in specification

HW design Fixes in hardware

HW verification

SW design Fixes in software

SW verification

Integration & verification

Find good design options here

Double Roof Model

Software

Hardware

Instruction

system

task

architecture

component

logic

Implementation

μArch

ISA

RTL

gate

### Design Methodologies

- **Set of models and design steps (transformations)**
  - **Top down design**
    - Starts with functional system specification
      - Application behavior
      - Models of Computation (MoC)
    - Successive refinement
      - Connect the hardware and software design teams earlier in the design cycle.
      - Allows hardware and software to be developed concurrently
    - Goes through architectural mapping
      - The hardware and software parts are either manually coded or obtained by refinement from higher model
    - Ends with HW-SW co-verification and System Integration
  - **Platform based design**
    - Starts with architecting a processing platform for a given vertical application space
      - Semiconductor, ASSP vendors
    - Enables rapid creation and verification of sophisticated SoC designs variants
    - PBD uses predictable and pre-verified firm and hard blocks
    - PBD reduces overall time-to-market
      - Shorten verification time
    - Provides higher productivity through design reuse
    - PBD allows derivative designs with added functionality
    - Allows the user to focus on the part that differentiate his design

Source: Coware, Inc., 2005

### Top-Down ESL Design Environment

- **Primarily Virtual**
  - System Def.
  - Function Design
  - HW & SW CODESIGN
  - HW DESIGN
- **Primarily Physical**
  - HW FAB
  - INTEG. & TEST

- **PROTOTYPING ENVIRONMENT**
  - HW DESIGN
  - SW DESIGN
Platform-Based Design (PBD)

System 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 [SpecC, SystemC, SystemVerilog]
Lecture 8: Outline

- Introduction
- System design flow
  - ESL design
    - Modeling
    - Synthesis
    - Verification
  - ESL landscape
- Summary and conclusions

System Modeling

- Design models as abstraction of a design instance
  - Representation for validation and analysis
  - Specification for further implementation
    - Documentation & specification

- Systematic modeling flow and methodology
  - Set of models
  - Set of design steps
    - From specification to implementation

- Well-defined, rigorous system-level semantics
  - Unambiguous, explicit abstractions, models
    - Objects and composition rules
    - Synthesis and verification
Modeling Guidelines

- A model should capture exactly the aspects required by the system, and no more.
  - There is not one model/algorithm/tool that fits all.

- Being formal is a prerequisite for algorithmic analysis.
  - Formality means having a mathematical definition (semantics) for the properties of interest.

- Being compositional is a prerequisite for scalability.
  - Compositionality is the ability of breaking a task about $A||B$ into two subtasks about $A$ and $B$, respectively.

Separation of Concerns

Managing Complexity

Orthogonalizing concerns across multiple levels of abstraction

Behavior Vs. Architecture

Computation Vs. Communication

Source: UC Berkeley, EECS249
System Design Flow

- Abstraction based on level of detail & granularity
  - Computation and communication

  **System design flow**
  - Path from model A to model F


Computation vs. Communication

- Separation of concerns
  - Flexibility in modeling
  - IP reuse

  ![Computation vs. Communication Diagram](source: Cowares, Inc., 2005)

Behavior can be described algorithmically, without the burden of the handshaking and control logic associated with bus communication.
Computation Models

- **Application model**
  - Model of Computation (MoC)
    - Process-/state-based [KPN, SDF, FSM, ...]
  - Back-annotated execution timing
    - Timing granularity (basic block level)

- **Processor model**
  - Operating system
    - Real-time multi-tasking (RTOS), drivers
  - Hardware abstraction layer (HAL)
    - Media accesses
  - Processor hardware
    - Bus I/O & interrupts

- **Instruction-set model**
  - Instruction-set or micro-architecture
    - Down to cycle-accurate behavior

Communication Models

- **Pin-Accurate Model (PAM)**
  - Redundant RTL complexity results in slow simulation
  - Each device interface must implement the bus protocol
  - Each device on the bus has a pin-accurate interface

- **Transaction-Level Model (TLM)**
  - Less code, no wires, fewer events yield faster simulation
  - Protocol is modeled as a single bus model instead of in each device
  - Each device communicates via transaction-level API
    - 100x-10,000x faster than PAM

Source: Coware, Inc., 2005
Transaction Level Modeling

The transaction level is a higher level of abstraction for communication.

For SoC, communication is often the bottleneck

TLM Details

- Abstracted communication
  - Detailed signal handshaking is reduced to series of generic events called “transactions”.
  - Blocks are interconnected via a bus model, and communicate through an API.
  - The bus model handles all the timing, and events on the bus can be used to trigger action in the peripherals.
SystemC/TLM 2.0

- Pointer to transaction object is passed from module to module using forward and backward paths
- Transactions are of generic payload type

Interconnect

Initiator/Target

Command
Address
Data
Byte enables
Response status
Extensions

Source: OSCI TLM-2.0

SystemC/TLM 2.0 Coding Styles

- Loosely-timed
  - Sufficient timing detail to boot OS and simulate multi-core systems
  - Each transaction has 2 timing points: `begin` (call) and `end` (return)

- Approximately-timed
  - Cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - Each transaction has at least 4 timing points

Source: OSCI TLM-2.0
Blocking and Non-Blocking Transports

- **Blocking transport interface**
  - Typically used with loosely-timed coding style
  - `tlm_blocking_transport_if`
    ```
    void b_transport(TRANS&, sc_time&);
    ```

- **Non-blocking transport interface**
  - Typically used with approximately-timed coding style
  - Includes transaction phases
  - `tlm_fw_nonblocking_transport_if`
    ```
    tlm_sync_enum nb_transport_fw(TRANS&, PHASE&, sc_time&);
    ```
  - `tlm_bw_nonblocking_transport_if`
    ```
    tlm_sync_enum nb_transport_bw(TRANS&, PHASE&, sc_time&);
    ```

Source: OSCI TLM-2.0

---

**Blocking Transport**

Initiator

- Simulation time: 0ns
- `call`

Target

- Simulation time: 0ns
- `call`

- Simulation time: 30ns
- `wait(30ns)`

- Simulation time: 0ns
- `b_transport(t, 0ns);`

- Simulation time: 0ns
- `b_transport(t, 0ns);`

- Simulation time: 0ns
- `return`

Source: OSCI TLM-2.0
Non-Blocking Transport

```c
nb_transport(-, BEGIN_REQ, 0ns);
nb_transport(TLM_ACCEPTED, -, -);
nb_transport(-, END_REQ, 0ns);
```

```c
nb_transport(TLM_ACCEPTED, -, -);
```

```c
nb_transport(-, BEGIN_RESP, 0ns);
```

```c
nb_transport(TLM_ACCEPTED, -, -);
```

```c
nb_transport(-, END_RESP, 0ns);
```

```c
nb_transport(TLM_ACCEPTED, -, -);
```

Virtual Platform Prototyping

- Computation refinement
- Communication refinement

Source: OSCI TLM-2.0
Abstraction Levels

- **Functional Validation**
  - Emb. System Modeling
  - Executable spec. capture
  - Functional testing

- **Architectural Validation**
  - System Partitioning and Assembly
  - Exploration and analysis

- **Hardware Refinement**
  - RTL Design & Verification
  - Block design and unit test
  - Validation in the system

- **RTL Verification**
  - System-level Verification
  - Complete design at RTL
  - System-level testbench

**Processor**
- Host-compiled
- Instruction
- Accurate
- Cycle
- Accurate

**Interconnect**
- Not Modeled
- Point to point
- Memory-mapped
- Approximately Timed TLM
- Cycle-Accurate TLM (Transfer Level)
- RTL (DUT)
- TL (rest)
- RTL

**Peripheral**
- Untimed
- Timed Bus-Functional
- RTL

---

**Speed vs. Accuracy**

- **Host-based**
- 10MIPS

- **IA ISS TLM Bus**
- 1MIPS

- **SystemC Executable TLM**
- 100Kcps

- **Cycle Accurate -TLM**
- 10Kcps

- **Pin-accurate w/RTL**
- 1Kcps

- **RTL**
- 100cps

- **ESL Architectural Design**
- LT 3 Mcps

- **Re-use for Early Software Development**
- CA 150 kps

- **Re-use for System-level Hardware Verification**
- PAM+RTL 15 kps

---

*Source: Coware, Inc., 2005*
Lecture 8: Outline

- Introduction
- System design methodology
  - ESL design
    - Modeling
    - Synthesis
    - Verification
  - ESL landscape
  - Summary and conclusions

Design Automation

- Synthesis = Decision making + model refinement

- Successive, stepwise model refinement
- Layers of implementation detail
X-Chart

Application → Behavior → Constraints → Synthesis

Synthesis

Decision making → Refinement

Structure

Transaction Level Model

Quality numbers

Latency, Area, Throughput, etc

Platform-Based System Synthesis

Application

Platform

Optimal Mapping?

Resource Allocation

- Resource allocation, i.e., select resources from a platform for implementing the application

Process Binding

- Process mapping, i.e., bind processes onto allocated computational resources
Channel Routing

- Channel mapping, i.e., assign channels to paths over busses and address spaces

Design Space Exploration

- Design Space Exploration is an iterative process:
  - How can a single design point be evaluated?
  - How can the design space be covered during the exploration process?
Optimization Approaches

- **Exact methods**
  - Enumeration, (Integer) Linear Programs

- **Heuristics**
  - Constructive
    - Random mapping, hierarchical clustering
  - Iterative
    - Random search, simulated annealing, min-cut (Kernighan-Lin)
  - Set-based (“intelligent” randomized search)
    - Evolutionary Algorithms (EA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO)

> Exact, constructive & iterative methods are prohibitive
> - Large design space, multiple objectives, dynamic behavior

> Set-based approaches
> - Randomized, problem independent (black box), Pareto set

Evaluation Approaches

- **Dynamic simulation**
  - Profiling, ISS/RTL co-simulation
  - *Long simulation times, corner cases*

- **Static analysis**
  - Component-level estimation
    [Worst-Case Execution Time (WCET)]
  - System-level cost functions, real-time calculus
    [Modular Performance Analysis (MPA)]
  - *Inaccurate bounds, manual interference (false paths)*

> Combinations
  - Host-compiled simulation
  - Trace-driven simulation
  - *Tradeoff between accuracy and speed*
Lecture 8: Outline

- Introduction
- System design methodology
  - ESL design
    - Modeling
    - Synthesis
    - Verification
  - ESL landscape
  - Summary and conclusions

Design Verification Methods

- Simulation based methods
  - Specify input test vector, output test vector pair
  - Run simulation and compare output against expected output

- Formal Methods
  - Check equivalence of design models or parts of models
  - Check specified properties on models

- Semi-formal Methods
  - Specify inputs and outputs as symbolic expressions
  - Check simulation output against expected expression
Simulation

- Create test vectors and simulate model
  - Simulation, debugging and visualization tools
    [Synopsys VCS, Mentor ModelSim, Cadence NC-Sim]

- Inputs
  - Specification
    - Used to create interesting stimuli and monitors
  - Model of DUT
    - Typically written in HDL or C or both

- Output
  - Failed test vectors
    - Pointed out in different design representations by debugging tools

Equivalence Checking

- LEC uses boolean algebra to check for logic equivalence

- SEC uses FSMs to check for sequential equivalence
Model Checking

- Model $M$ satisfies property $P$? [Clarke, Emerson '81]
- Inputs
  - State transition system representation of $M$
  - Temporal property $P$ as formula of state properties
- Output
  - True (property holds)
  - False + counter-example (property does not hold)

Lecture 8: Outline

- Introduction
- System design methodology
- ESL design
  - ESL landscape
    - Commercial tools
    - Academic tools
  - Summary and conclusions
**Electronic System-Level (ESL) Landscape**

- **Platform library**
  - SPIRIT/VP-XACT (XML)
- **System Synthesis Front-End**
  - Academic Tools
  - MARTE (UML)
  - SystemC, CoWare, ...
  - SystemC, CoWare, ...
- **Transaction-Level Models (TLM)**
  - Software / Hardware Synthesis Back-End
  - C/C++ code
  - Software Object Code
  - Green Hills gcc, VxWorks ...
  - VaST, OVP Virtutech ...
- **Application specification**
  - Mentor Catapult, Forte, ...

**ESL Tools**

- **Electronic System-Level (ESL) terminology**
  - Often single hardware unit only
    - C-to-RTL high-level synthesis (HLS) [Mentor Catapult, Forte Cynthesizer]
- **System-level across hardware and software boundaries**
  - System-level frontend
  - Hardware and software synthesis backend
- **Commercial tools for modeling and simulation**
  - Algorithmic modeling (MoC) [UML, Matlab/Simulink, Labview]
  - Virtual system prototyping (TLM) [Coware, VaST, Virtutech]
  - Only horizontal integration across models / components
- **Academic tools for synthesis and verification**
  - MPSoC synthesis [SCE, Metropolis, SCD, PeaCE, Deadalus]
  - Vertical integration for path to implementation
Academic MPSoC Design Tools

<table>
<thead>
<tr>
<th>Approach</th>
<th>DSE</th>
<th>Comp. decision</th>
<th>Comm. decision</th>
<th>Comp. refine</th>
<th>Comm. refine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daedalus</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Koski</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Metropolis</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>PeaCE/HoPES</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>SCE</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>SystemCoDesigner</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>


System Design Flow Summary

- **Embedded System Requirements**
  - Functional IP
  - Platform Function
  - Platform Architecture
  - Architecture IP
  - System Integration
  - Performance Analysis and Platform Configuration

- **Platform Configuration**
  - at the un-clocked, timing-aware system level

- **Design Export**
  - after initial platform configuration through design refinement and communication synthesis

- **Hardware Assembly**
- **Software Assembly**
- **Implementation Level Verification**
- **Synthesis / Place & Route etc.**