SoC 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
Multi-Processor System-on-Chip (MPSoC)

System Memory | Memory Controller | Micro-Controller | ASIP | Local RAM

Controller Bus

Bridge

DSP | DSP RAM | Hardware Accelerator

DSP Bus

Shared RAM | Hardware Accelerator | Video Front End

Local Bus

Lecture 6: Outline

✓ Introduction

• System design methodology
  • Electronic system-level design (ESL/SLD)

• System-level design
  • Synthesis
  • Modeling

• Summary and conclusions

Source: C. Haubelt, Univ. of Erlangen-Nuremberg
Issues in HW Centric Design Flows

- RTL language centric
- Dysfunctional levels of abstraction
- SW Design Cycle often serial to HW Design Cycle
  - Lack of unified hardware-software representation
- Missing executable platform models early in cycle
- SW/HW integration is tough
- Simulation speed is critical
- Partitions are defined *a priori*
  - Hard to find incompatibilities across HW-SW boundary
- Lack of well-defined design flow
  - Time-to-market problems
  - Specification revision becomes difficult

The ESL Solution: One Reference Model

Algorithm & Architecture Exploration

One Reference Model

SW Development & Verification

HW Development & Verification

Courtesy: Coware, Inc. 2005
Electronic System-Level (ESL) 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

manual

(semi)automatic
Hardware-Centric Design Cycle

Task

Specification  Fixes in specification

HW design  Fixes in hardware

HW verification

SW design  Fixes in software

SW verification

Integration & verification

Time

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

New ESL Design Cycle

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification (high-level &amp; arch. models)</td>
<td></td>
</tr>
<tr>
<td>HW design Fix(es) in hardware</td>
<td></td>
</tr>
<tr>
<td>HW verification</td>
<td></td>
</tr>
<tr>
<td>SW design Fix(es) in software</td>
<td></td>
</tr>
<tr>
<td>SW verification</td>
<td></td>
</tr>
<tr>
<td>Integration &amp; verification</td>
<td></td>
</tr>
</tbody>
</table>

Find good design options here
Design Methodologies

- **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

![Diagram of Top-Down ESL Design Environment]

Cost Models

Copyright © 1995-1999 SCRA Used with Permission
Platform-Based Design (PBD)

- Models of Computation
- System Behavior
- System Platform
- Performance models: Emb. SW, Comm. and Comp. resources
- Mapping
- HW/SW Partitioning, Scheduling & Estimation
- Behavior Verification
- Architecture
- Refinement
- Flow To Implementation
- Performance Analysis and Simulation
- Synthesis & Coding

Source: UC Berkeley, EECS249

Lecture 6: Outline

- Introduction
- System design methodology
  - System-level design
    - Synthesis
    - Modeling
  - Summary and conclusions
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?

Lecture 6: Outline

- Introduction
- System design methodology
  - System-level design
    - Synthesis
    - Modeling
  - 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 methodology**
  • Set of models and transformations (design steps)
  • Modeling flow defines the design process
  ➢ From specification to implementation

➢ **Well-defined, rigorous semantics**
  • Unambiguous, explicit abstractions
  • Formal models
  ➢ Synthesis and verification

Separation of Concerns

Managing Complexity

Orthogonalizing concerns across multiple levels of abstraction

Behavior Vs. Architecture

Computation Vs. Communication

Platform-based design (PBD) Transaction-level modeling (TLM)

Source: UC Berkeley, EECS249
Computation vs. Communication

- **Separation of concerns**
  - Flexibility in modeling, IP reuse
  - Design computation & communication separately

Communication can be described in a wide range of fashions, from high-level messages, to detailed signal level handshakes without impacting the behavior description.

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

Source: Coware, Inc., 2005

---

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

![Diagram of SystemC/TLM 2.0](image)

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
    - 4 forward/backward function calls
    - Must immediately return (no `wait()`)

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
  - `tlim_fw_nonblocking_transport_if`
    ```
    tlm_sync_enum nb_transport_fw(TRANS&, PHASE&, sc_time&);
    ```
  - `tlim_bw_nonblocking_transport_if`
    ```
    tlm_sync_enum nb_transport_bw(TRANS&, PHASE&, sc_time&);
    ```

Loosely Timed (LT) Model

- **Initiator**
  - `Simulation time 0ns`
  - `call`
    ```
    b_transport(t, 0ns);
    ```
  - `Simulation time 0ns`
  - `call`
    ```
    b_transport(t, 0ns);
    ```
  - `Simulation time 30ns`
  - `call`
    ```
    b_transport(t, 0ns);
    ```
  - Return

- **Target**
  - `Simulation time 0ns`
  - `wait(30ns);`
  - `Simulation time 0ns`
  - `b_transport(t, 0ns);`
  - Return

Source: OSCI TLM-2.0
Approximately Timed (AT) Model

Initiator

Target

nb_transport(-, BEGIN_REQ, 0ns);
nb_transport(TLM_ACCEPTED, -, -);
nb_transport(-, END_REQ, 0ns);
nb_transport(TLM_ACCEPTED, -, -);
nb_transport(-, BEGIN_RESP, 0ns);
nb_transport(TLM_ACCEPTED, -, -);
nb_transport(-, END_RESP, 0ns);
nb_transport(TLM_ACCEPTED, -, -);

Source: OSCI TLM-2.0

NB Return Values (tlm_sync_enum)

- **TLM_ACCEPTED**
  - Transaction, phase and timing arguments unmodified (ignored) on return
  - Target may respond later (depending on protocol)

- **TLM_UPDATED**
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine to the next state

- **TLM_COMPLETED**
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine straight to the final phase

Source: OSCI TLM-2.0
Using the Return Path

- Callee annotates delay to next transaction
  - Caller waits

<table>
<thead>
<tr>
<th>Phase</th>
<th>Initiator</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_REQ</td>
<td>Simulation time = 100ns</td>
<td>Call, - BEGIN_REQ, 0ns</td>
</tr>
<tr>
<td>END_REQ</td>
<td>Simulation time = 110ns</td>
<td>Return, TLM_UPDATED, END_REQ, 10ns</td>
</tr>
<tr>
<td>BEGIN_RESP</td>
<td>Simulation time = 150ns</td>
<td>Call</td>
</tr>
<tr>
<td>END_RESP</td>
<td>Simulation time = 155ns</td>
<td>Return</td>
</tr>
</tbody>
</table>

Source: OSCI TLM-2.0

Early Completion

- Callee annotates delay to next transaction
  - Caller waits

<table>
<thead>
<tr>
<th>Phase</th>
<th>Initiator</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_REQ</td>
<td>Simulation time = 100ns</td>
<td>Call, - BEGIN_REQ, 0ns</td>
</tr>
<tr>
<td>END_RESP</td>
<td>Simulation time = 110ns</td>
<td>Return, TLM_COMPLETED, -, 10ns</td>
</tr>
</tbody>
</table>

Source: OSCI TLM-2.0
Temporal Decoupling

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Simulation time = 100ns</em></td>
<td><em>Simulation time = 130ns</em></td>
</tr>
<tr>
<td>Local time offset</td>
<td></td>
</tr>
<tr>
<td>+5ns</td>
<td>Return</td>
</tr>
<tr>
<td><em>b_transport(t, 0ns)</em></td>
<td></td>
</tr>
<tr>
<td>+20ns</td>
<td>Call</td>
</tr>
<tr>
<td><em>b_transport(t, 0ns)</em></td>
<td>Return</td>
</tr>
<tr>
<td>+30ns</td>
<td></td>
</tr>
<tr>
<td><em>b_transport(t, 0ns)</em></td>
<td></td>
</tr>
<tr>
<td>wait(30ns)</td>
<td></td>
</tr>
</tbody>
</table>

Quantum = 25ns

Source: OSCI TLM-2.0

Direct Memory Interface (DMI)

- **Implementation approach**
  - Direct member function call of target object
    - In master context
  - Pure virtual interface classes defined interface
  - Function call encapsulates and hides all interconnect details
  - Port/export provides connection semantics (incl. restrictions)

  - **Direct memory interface**
  - **Faster (no context switch)**
  - **Debug interface**

Source: W. Ecker, Infineon

© 2014 A. Gerstlauer
Lecture 6: Outline

- Introduction
- System design methodology
- System-level design
  - Synthesis
  - Modeling
- Summary and conclusions

Virtual Platform Prototyping

Computation refinement

- Untimed
- TLM (LT/AT)
- PCAM

Communication refinement

Virtual Prototype
### 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

**Interconnect**
- Not Modeled
  - Point to point
  - Memory-mapped
  - Loosely Timed TLM
  - Approximately Timed TLM
  - Cycle-Accurate TLM (Transfer Level)

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

**Increasing Simulation Performance**
- Increasing Scope for Relative Optimization

---

### Speed vs. Accuracy

Log **S P E E D**

- 10 MIIPS
- 1 MIIPS
- 100 Kcps
- 10 Kcps
- 1 Kcps
- 100 cps

Log **A C C U R A C Y**

- CA 150 kps
- LT 3 Mcps
- PAM+RTL 15 kps

**Host-based**

**IA ISS TLM Bus**

**SystemC Executable TLM**

**Cycle Accurate -TLM**

**Pin-accurate w/RTL**

**RTL**

**Re-use for Early Software Development**

**Re-use for System-level Hardware Verification**

**ESL Architectural Design**

Source: Coware, Inc., 2005
System Design Flow Summary

Embedded System Requirements

Functional IP
C/C++ SDL SPW Simulink

Platform Function
Platform Architecture

System Integration
Performance Analysis and Platform Configuration

Communication Refinement, Integration & Synthesis

Hardware Assembly
Software Assembly

Implementation Level Verification

Synthesis / Place & Route etc.

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

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

Embedded System Requirements

Platform Function
Platform Architecture

System Integration
Performance Analysis and Platform Configuration

Communication Refinement, Integration & Synthesis

Hardware Assembly
Software Assembly

Implementation Level Verification

Synthesis / Place & Route etc.

Embedded System Requirements

Functional IP
C/C++ SDL SPW Simulink

Platform Function
Platform Architecture

System Integration
Performance Analysis and Platform Configuration

Communication Refinement, Integration & Synthesis

Hardware Assembly
Software Assembly

Implementation Level Verification

Synthesis / Place & Route etc.