System Level Design with IBM PowerPC Models
A view of system level design
The System-Level Challenges

- **Verification escapes cost design success**
  - “There is a 45% chance of committing a logical/functional error when designing an IC/ASIC”
  
Source: Collett International Research 2003 IC/ASIC Design Closure Study

- Can no longer verify large systems at detail level

- **Functionality convergence challenges validation**
  - Multipurpose devices - video phone, MP3 etc
  - Functional validation and test is very complex

- **Multiple IP blocks bring interface problems**
  - Integration, interconnect and interoperability issues

- **Software dominance of most designs**
  - Need early executable hardware specification

- **Desire to have physical and power information earlier in the development cycle**
“Traditional” Simplified Flow

Specification(s)

Software

Hardware

Limited ability to address complexity, assess full range of tradeoffs

Weak link between specification and implementation
There is a Need for Something New

Paper Specification

- More abstract system model
- Faster to create and simulate
- More “what-if’s

System Model

Consistent verification from concept to implementation

- Application
- RTOS
- BSP (drivers)

HDL - RTL
- Design
- Debug
- Verification

Software Hardware
An Evolution of the “Traditional” Flow

- Paper Specification
- System High Level Model Executable Specification
- Virtual Prototype
- Hardware High Level Model
- Co-Verification

Requirements follow-up
Consistent Verification

Software
- Application
- RTOS
- BSP (drivers)

Hardware
- HDL - RTL
  - Design
  - Debug
  - Verification

Software Hardware

Consistent Verification
System Level Tasks and Stages

- Explore the feasibility of requirements
- Partition HW and SW - Define the architecture
- Create a first prototype of the HW
- Implement the hardware at register level
- Finalize the specification
- Create a verification infrastructure

System Level Design

Uncommitted Systems

Hardware Committed

Functional Requirements

Gates

Uncommitted Systems

Hardware Committed
Advantages of System Level Design

- Functional behavior validated early in the process speeds the flow to working silicon
- Fast execution allows architectural exploration to find the best design alternatives and encourages more testing to eliminate functional bugs
- The system level model serves as a rapid, updateable early prototype to support concurrent software development
Modeling Abstraction / Languages

- Behavioral Specification
- Architectural Exploration
- Transaction-level Verification
- Cycle Accurate Verification
- RTL Verification
xtUML for Initial System Modeling

Use functional requirements specification as input
Build platform-independent executable models of
the Application & Test Suite

Generate 100% C/C++
Code from the model

Model Verifier

Future: xtUML to TLM Compiler could build
Software + Hardware platform for performance
analysis & architecture validation

Reference external models
such as C/C++ algorithms
suitable for Catapult C
Algorithmic C Synthesis

Use TLM Performance Analysis to Refine Block Constraints

Constraints

C Algorithm

Algorithmic C Synthesis

TLM Models

Generate TLM models (un-timed, timed & CA)

TLM Simulation Environment

RTL

Synthesize RTL & Generate testbench infrastructure

Links to Simulation Environments

Use ANSI C++: Focus on the Functional Intent
Explore the design space to find the optimum micro-architecture
Transaction Level Modeling & Simulation

Assemble Structural System Design from existing IP and new user-created models
Validate and analyze system function and performance

Use Interface Generation Tools to simplify new TLM creation

Generate additional TLMs from xtUML or C++ algorithms

TLM simulation speed 100k -> 2M IPS

IP Library

Processor cores for software execution

OSCI TLM and SPIRIT standards increase IP library availability

Simulation Environment common between TLM and RTL

Performance Metrics

Power Metrics

Virtual Prototype for SW

Host/Target Code

TLM System Assembly

TLM Simulation Environment

Manual Design / Interface Automation
With the Introduction of Synthesis, Automation and Re-Use
...The Value of High Level Modeling is Made Accessible
To Provide a Complete System-level Design Process

Behavioral / Functional Modeling

Capture design as executable specification
Validate functionality

Full System

Generate S/W Code (& TLMs)
Structure design
Validate architecture
Analyze performance

Application S/W Validation

Compile & Link

Application Testing
S/W <-> H/W integration testing is done at TLM & RTL

Software

Generate S/W Code
(& TLMs)

Micro-Architecture Implementation

Hardware

Refine block timing & power
Synthesize RTL

Application S/W
Validation

Compile & Link

Applications Testing

S/W <-> H/W integration testing is done at TLM & RTL

Full System

Performance analysis includes timing & power
Both s/w and h/w dependent effects

Architectural Design
The Vision of System-Level Design

- Architectural Analysis
  - With functional, performance and power models

- Early Software Development
  - With rapid adaptable virtual prototypes

- Higher Quality Design
  - With progressive refinement to implementation

- Better Verification
  - With Transaction Based methodologies

- Continuous refinement
  - Bi-directional flows
Key Productivity Directions

- **Model system or function at the highest appropriate level**
  - xtUML for system behavior
  - TLMs for architecture
  - C++ for algorithms

- **Automation & Re-Use**
  - xtUML to embedded C code
  - C++ algorithm to SystemC TLM
  - Automated SystemC TLM interface generation
  - xtUML to TLM
  - TLM Model Interoperability
    - Catapult C, Perspecta, Questa, virtual prototypes
    - C++ algorithm to RTL
    - Embedded PowerPC aware platform
A View of System Level Design
Technologies & Methodologies
Conclusion
Transaction Level Modeling

- This is a methodology, also known as TLM, that defines new abstraction levels above the register.

- It is itself made of several stages, which gradually abstract from hardware implementation constraints but still with a structured view of the design.

- Its goal is to reduce the number of events and the amount of data that has to be treated during simulation.

- This modeling method is built as a set of interfaces that define how models communicates.
Architectural Synthesis

RTL Flow vs. Catapult C Flow

- Safer design flow
- Shorter time to RTL
- More efficient methodology
- Design optimized to system requirements through incremental refinement
**Catapult C Synthesis - Algorithm to RTL**

- **Develop Algorithms using ANSI C++**
  - No proprietary extension
  - Focus on the functional intent

- **Synthesize with Catapult C**
  - Explore the design space
  - Find the optimal architecture

- **Generate High Speed Models**
  - Verilog, VHDL, SystemC
  - Accelerate system level verification

- **Generate Target Optimized RTL**
  - Faster and better than hand-coded
  - For ASIC, FPGA or FPGA prototyping of ASICs

- **Automatically Verify the RTL**
  - Generation of testbench infrastructure
  - Seamlessly reuse original C++ test vectors
System-Level Design - Perspecta

- Modeling components
  - Processor & IP libraries
  - Model integration tools
- System Architecture
  - Assemble and modify design
- Performance analysis
  - Throughput, bandwidth
- Design validation
  - Functional and performance goals
- HW/ SW co-design
  - Full system integration
- Verification
  - Hardware & software functional test
Power Analysis in Perspecta

- Provides a mechanism to dynamically record and display power consumption across the whole system
- Suitable for comparison of different architectural options
  - Includes software dependent effects
- Accuracy is driven by power characterization data available to the modeler
  - Can be ‘data book’ level or better
  - Only limited by level of the TLM in question
### Component Power Modeling

- **Within the functional model**
  - Associate known power profile with each distinct model state
  - Use an API to set the new mode dependent state
  - Allocate energy value for particular events
  - Also records the duration

#### Modeling power requires
- Addition of handful of API calls inside existing model
- High-level power characterization of the blocks being modeled

<table>
<thead>
<tr>
<th>Mode</th>
<th>Power (uW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE mode</td>
<td>20</td>
</tr>
<tr>
<td>Processing Mode</td>
<td>500</td>
</tr>
<tr>
<td>Data Transmit Mode</td>
<td>300</td>
</tr>
</tbody>
</table>

**Example Calculation**

- **I/O interaction**
  - 64b data transfer
  - Write energy per bit: 0.5nJ
  - Total energy: 32nJ

\[
64b \times (0.5nJ/b \text{ write energy}) = 32nJ
\]

- **16 word burst timing**
  - Initial 5 clocks: 100ns
  - 15 more clocks: 300ns
  - Total time: 400ns

\[
16 \text{ word burst} = 5\text{clks(init)} + 15\times1\text{clks(beat)} = 20\text{ clks * 20ns period} = 400\text{ns}
\]
System Power Allocation

- Power consumption elements
  - CPU
    - Based on clock frequency
  - Memory accesses
    - Read / write set by range
  - Memory controller
  - DMA engine
  - Other processing units (accelerators)
  - I/O elements
    - Including bus bridges
  - Advanced interconnects

Form-based (or custom power models)

Power enabled TLM models
Seamless: Verify Hardware *and* Software at the Same Time with the Same Data

- Drops in to existing RTL environment
  - Customer’s Simulator
  - Seamless Bus Model + Memories
  - Bus Monitors

- Embedded software provides hardware testbench

- Common environment eliminates ambiguity

- Customer’s Embedded Code
  - Code profiling

- RTL description replaces the need for stub code
Profiler Views

Software Profile
Software Gantt
Bus Delay
Bus Load
Power
Memory Heat Map
Progressive Refinement to RTL

- Assemble Transaction Level Models
- Create new elements
- Simulate to validate functionality, performance & power
- Substitute IP models
- Synthesize algorithms
- Add specific RTL blocks
- Floorplan, Power Islands & Implement
System-Level Software - BridgePoint
Executable, Translatable UML (xtUML)

- Application Models in xtUML:
  - Platform-independent
  - Executable
  - Capture subject-matter expertise
  - Enable large-scale reuse
  - Integrate with legacy code

- Early verification before:
  - HW/SW partitioning
  - Processor selection
  - Target language selection
  - RTOS selection
  - Target model compiler is available

- Model Compilers for xtUML:
  - Application-independent
  - Provide complete code generation
  - Generate optimized code
  - Delivered in source for customization
  - Verify independent of application models

Model Construction

Model Compilation

Model Execution and Verification

Model Debugging

Cause and Effect:

my Code
switch (m_state)
{
  case RES_WAIT:
    if (rsp_fifo._get(rsp))
    {
      send_resp(rsp);
    }
    break;

BridgePoint

SLE Division, A view of system level design, September 2005
BridgePoint Flow for System Specification

- Use functional requirements specification as input
- Build platform-independent executable models of:
  - Application
  - Test Suite
- Verify application behavior on host:
  - Application models
  - C Algorithms
  - Legacy SW, commercial middleware, RTOS
- Buy or build model compiler for target
- Mark models as hardware and software
With a unique breadth of expertise in the key domains that make up System Level Design and ESL today, Mentor Graphics is well positioned to lead in its evolution.

A strong partnership between design automation solutions, system IP suppliers and design teams defines the direction of next generation systems.