Successful Application of System Level Emulation

For Server CPU Validation

George Powley
Intel Corporation

September 11, 2014
SNUG Boston

Agenda

Server Design Challenges

System Level Emulation Overview

System Level Emulation Challenges

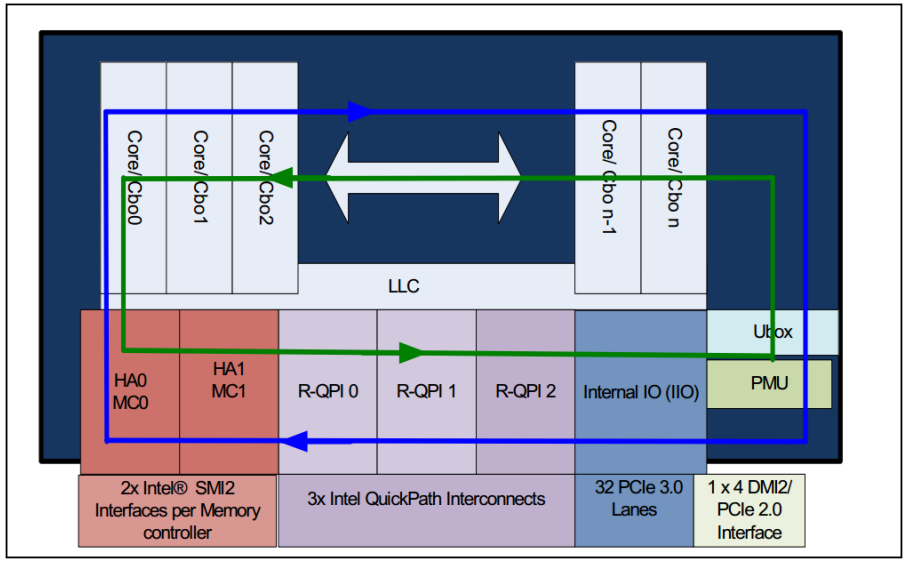
Delivering a System Level Emulation Model

Emulation Usage Models and Debug Methodology

Conclusions

Server Design Challenges

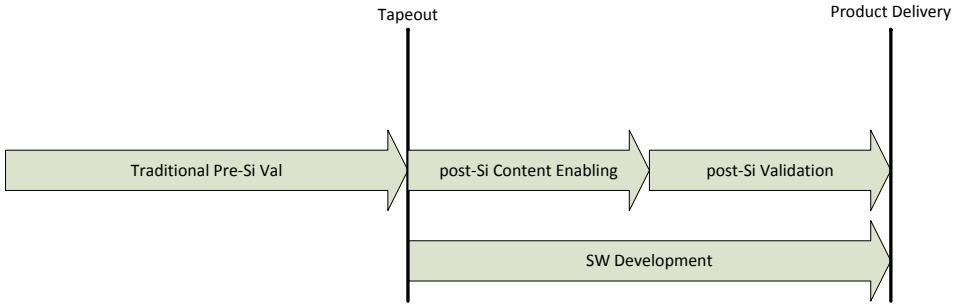
Complex Design & Slow Simulation



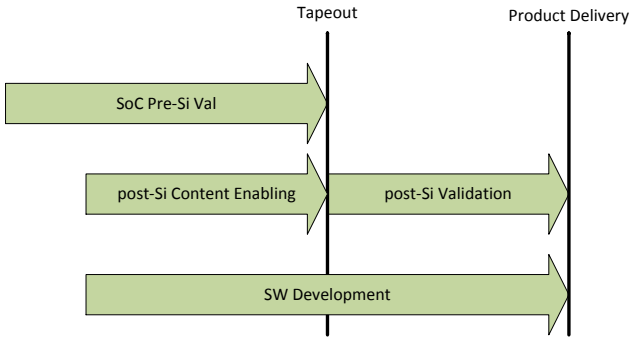
- Multiple Intel Xeon cores (up to 15)
- High-performance, server specific uncore
- High-speed interfaces (QPI and PCIe 3.0)
- Complex Power Management features
- Validating long flows and cross products would take *weeks* of RTL simulation time – *PER TEST!*

Shrinking Time-to-Market

Traditional Design Cycle



Shift Left Design Cycle



- Market pressures require a “shift left” approach to delivering products

System Level Emulation Overview

ZeBu Emulation Platform

Comprehensive SoC verification system

Synopsys Users Group

Automated Software Flow

RTL Synthesis	Memory Compiler	Transactor Compiler
Instrumentation	Auto-Partition	Place and Route

Libraries

- Transactors
- Memories
- DesignWare

Interfaces

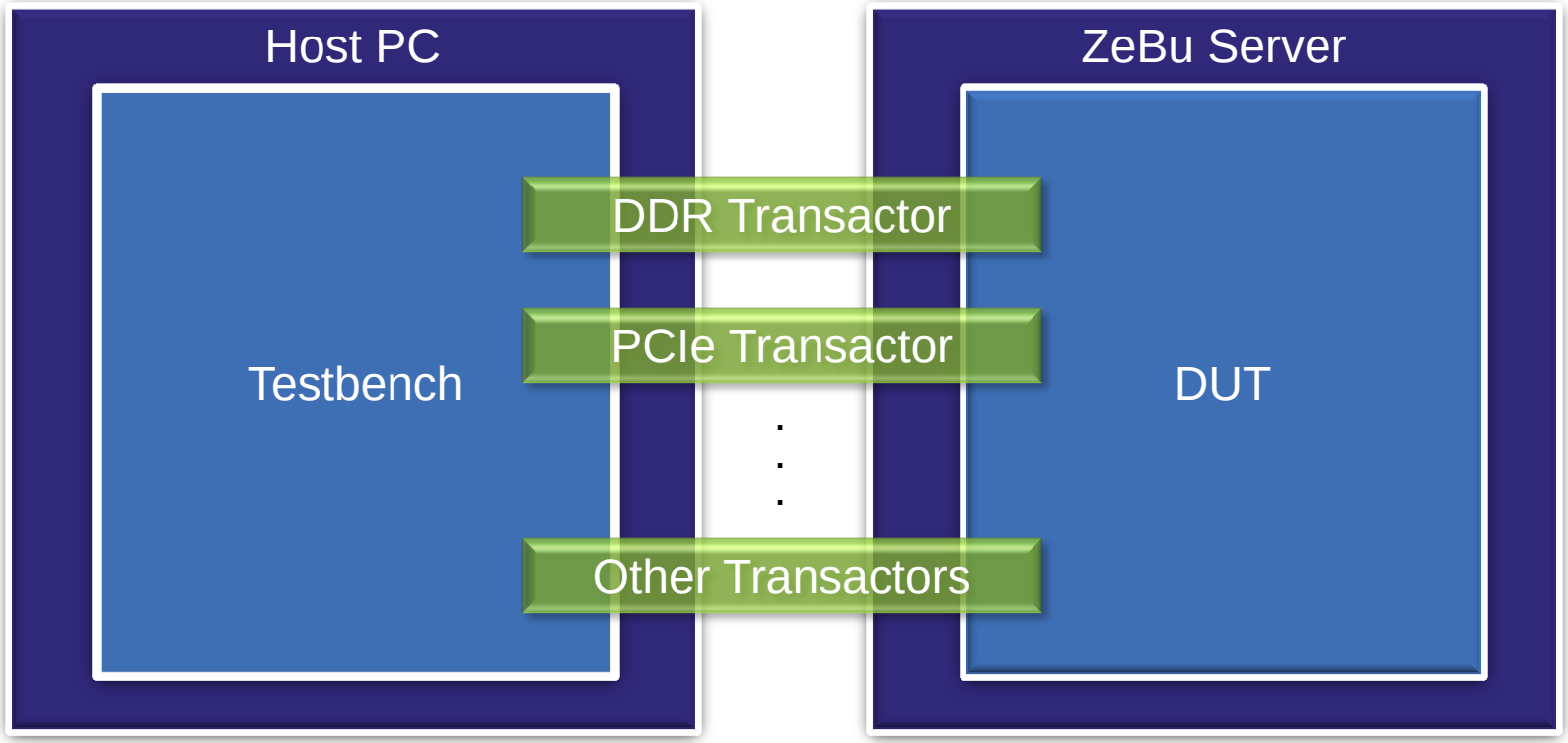
- In-Circuit I/O
- Virtual Adapters
- Co-Simulation Channel

Waveform	Transactor	Memory
Save and Restore	Verdi^3	Auto-Trace to Disk
Coverage	Power Analysis	Assertion

Comprehensive Debug

Server Emulation Overview

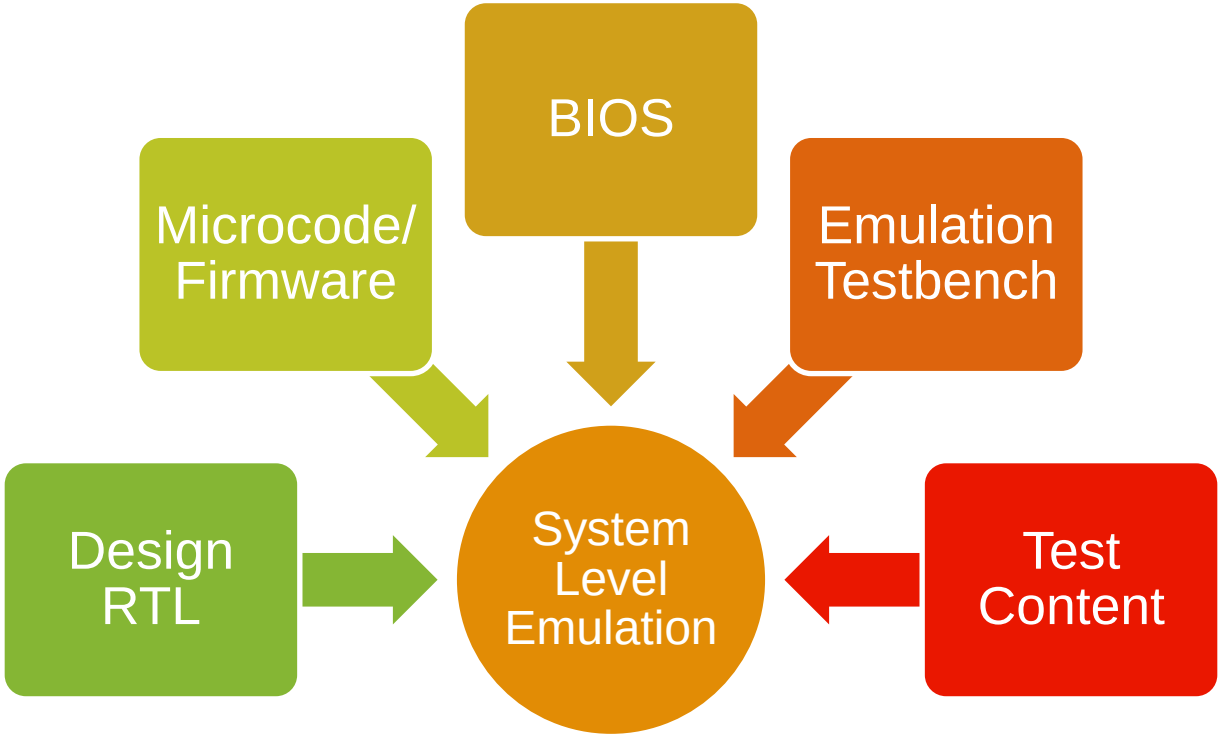
- High-speed transaction based model
- Custom C/C++ testbench



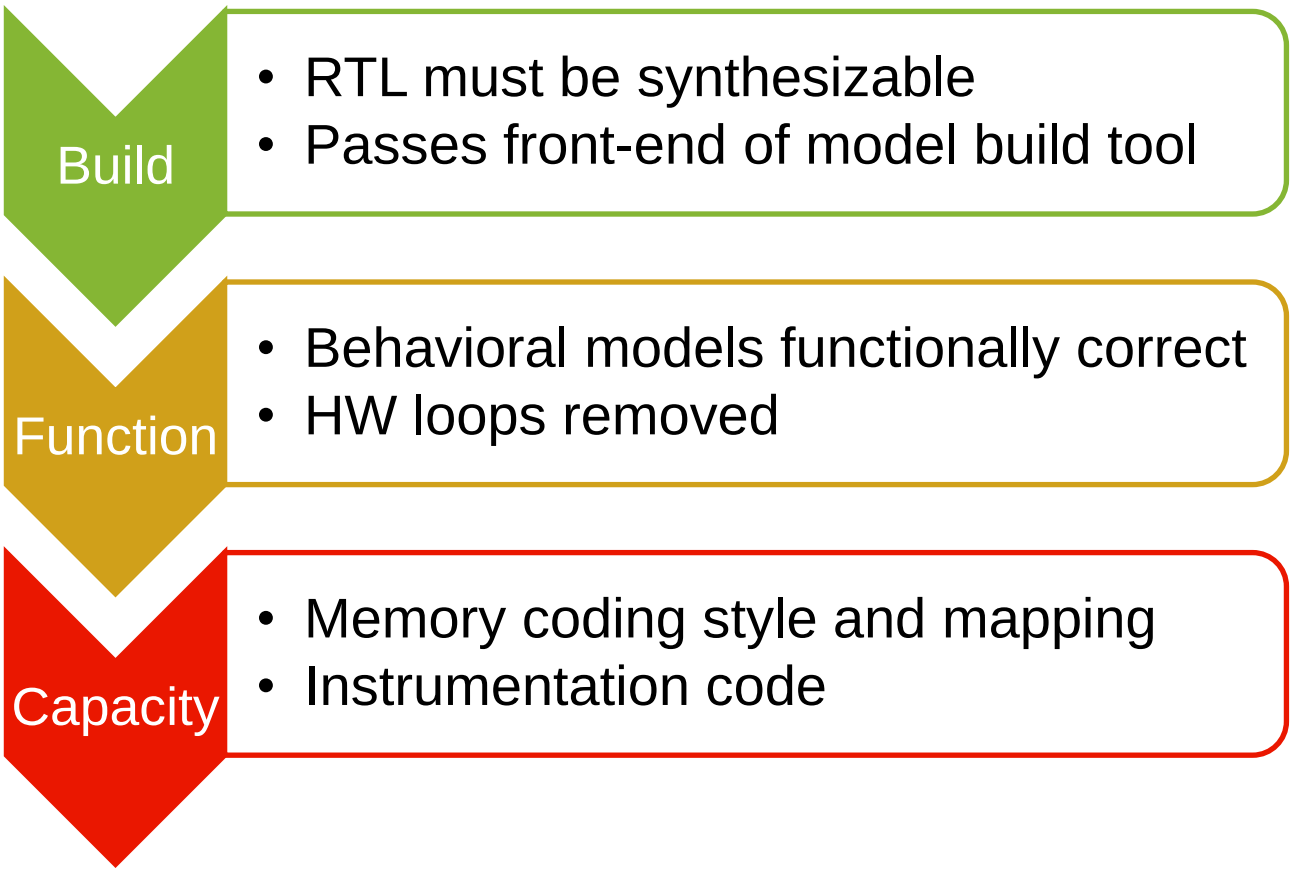
System Level Emulation Challenges

Emulation Ingredient Availability

- System level emulation requires a number of ingredients from different teams
- Quality of ingredients varies throughout the design cycle



Emulation Friendly RTL



Large Model Issues

- Team productivity and emulation utilization issues are magnified by large models

Long Compile Times

HW Capacity vs. FPGA Convergence

Model Speed/Performance

Delivering a System Level Emulation Model

Planning for Emulation

- Emulation Customers develop test plans
  - Specify model topologies, configurations, transactors, test content
- Emulation Team distills requests into a manageable set
- Staging plan for emulation model delivery
  - HW/FW availability for initial model and new features
  - BIOS availability for model configuration
- Execution planning
  - Model build, bring-up, and release
  - Plan for a lot of debug, need debug support from designers
  - Emulation customer support
  - Expect new capability requests

Emulation Model Build

- Static RTL checks gate check-in
  - Emulation related lint rules
  - Emulation model build front-end analysis and elaboration
- Build flow automation
  - Leverage RTL build flow for code generation
  - Parameterize model configuration (# sockets, # cores, SKU, etc.)
  - Parameterize emulation testbench and instrumentation
  - Provide emulation model synthesis options (# boards, fill rate, etc.)
- Monitor emulation capacity and performance

Ideally, provide a push-button model build flow

Emulation Model Bring-Up

- Bring the model out of reset
  - Debug why the model fails to get out of reset
  - Develop workarounds for issues and continue to make progress
- Issues found may require a model rebuild
  - Very common during early model bring-up
  - Impact of long emulation build times felt here
- Once model is out of reset, prepare for release

Emulation Model Release

- Model Regression
  - Run suite of tests covering different use models before release
  - Improve regression suite as model matures
- Model Refresh
  - Release a new model periodically (i.e. every 1-2 weeks)
  - Pull in new features
  - Avoid debugging failures caused by known bugs
- “Nightly” Model Regression
  - Fully automated build and test
  - Needs more manual intervention early in the design flow
  - Avoid surprises when moving to a new model
  - Prime the model refresh pipeline

Emulation Usage Models and Debug Methodology

Emulation Usage Model

- Emulation customers develop content and run tests
  - Pre-Si validation of HW/FW/SW (Reset, Power Management, etc.)
  - Early development and testing of post-Si debug tools
  - Pre-Si validation of manufacturing test vectors
  - Post-Si bug reproduction
- Debug becomes a team effort

Emulation Debug

- Debug of emulation test failures can be challenging
  - Millions of cycles, numerous sources of potential errors
  - Design HW/FW, Model Configuration, Synthesizable Behavioral Code, HW Loops, Test Content, Emulation Testbench, Emulation SW, Emulation HW
- Emulation debug toolkit
  - Synthesizable event monitors generate log messages
  - High-speed, reduced visibility signal dump over long time windows
  - Low-speed, full visibility signal dump over short time windows
  - Post-processing checkers
  - More capabilities available

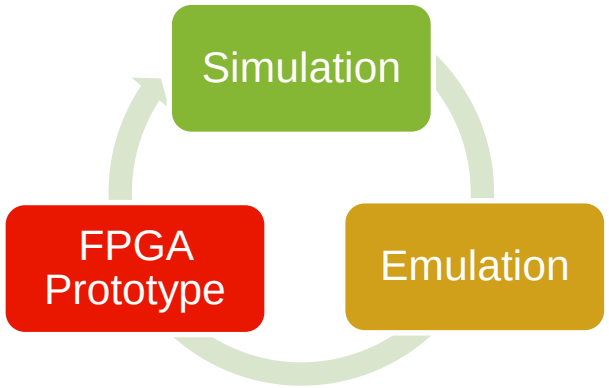
Conclusions

Result = Success!

- Found multiple complex, critical HW bugs that would have escaped to silicon → Saved silicon steppings
- Found numerous FW and SW bugs before A0 tape-in → Saved time in post-Si
- Shift left of post-Si tool development and manufacturing test vector checkout → Saved time in post-Si
- Reproduced post-Si bugs → Accelerated post-Si debug
- Tapeout gate → Improved quality of tapeout

Future Improvements and Trends

- Emulation Methodology
  - More sharing between simulation, emulation, and post-Si
  - Robust synthesizable behavioral models
- Emulation SW
  - Reduced compile time
  - Automated model speed improvement
- Adopt more emulation features
  - UPF based Power Aware Emulation
  - System Verilog Assertions
- Validation Continuum




Thank You