An Accelerated System Level CPU Verification through Simulation-Emulation Co-Existence

Ruchi Misra, Samridh Deva, P Sai Krishna, Alok Kumar, Garima Srivastava
YoungSik Kim, Seonil Brian Choi
Outline

• Need for adoption of Emulation in early Design Verification
• Pre-Silicon Verification flow
• Emulator Compilation flow
• Methodology
• Case Studies
• Enhancements done
• Results
• Conclusion
• Future Scope and Care Abouts
• Acknowledgement
Need for adoption of Emulation in early Design Verification

- Very long simulation time
- Creating corner case scenarios
- Coverage Closure
- Security and Functional Safety
- Low Power Verification
- Huge Design Complexity
- Large T-A-T for iterations

Introduction

- Shrinking turn around times
- Need for faster simulations and quicker results
- Delay in time to market
Motivation

- Extremely fast, full System-on-Chip testing
- Emulation enables longer test cases to be run in less time
- Accelerated Verification flow
- Quicker debug iterations
Pre-Silicon Verification Flow

- Design Specs and Requirements
- Verification
- Test Cases and TLM
- DUT

Simulation and Emulation (Iterative Process)

Calculate the Coverage
Emulation Compilation Flow

- Front end compilation
- System level back compilation
- Core level back-end compilation
- Place and Route

Design Files

Emulation Compiler
- Front End
- Back End

Bitstream

Compilation Parameters

Runtime Environment
Motivation for Power Aware Verification at SoC

• Power-aware Verification of SoCs has been around for over a decade ever since IEEE released the UPF standard.

• Today, we not only have multi-core SoC designs but also multi-cluster SoCs with dual architecture processors implemented on the same die.

• Needless to say, this increase in complexity due to SoC architecture evolution has seeped into the requirements of power-aware verification and is fast becoming the long-pole of the verification lifecycle.

• We divided the power-aware verification requirements into a number of test categories and successfully accelerated the power-aware verification lifecycle with the use of emulators by incorporating UPF design libraries and power domain control into the design images used for emulation-based verification.

• This has resulted in up to 40x savings in time and resources.
Basic Methodology

Preparing the Test-Plan

Setting the Test Priority

Compile and Run

Select Desired Scenario and generate .vrom file

Determine which blocks are required and which are faked. Choose Base or Full Image accordingly

Base Image

Full Image

Script to generate a binary file using design data

Separate the Binary files with desired headers and sizes

Prepare a .tcl file with all the information and paths

Apply for License and run
Case-Study 1: Complex Coherency Scenario

SoC teams are connecting multiple clusters together

Processor vendor may not provide coherent connection mechanism for multiple clusters

System level Cache coherency is fundamentally hard to verify!

Stressed scenarios mixing these:

- Page Table Properties
- Memory Ordering
- Power Scaling
- Cache-line Evictions
Case-Study 2: Dynamic Voltage and Frequency Scaling

Ways to reduce power consumption of chips on the fly

Different Design Blocks at SoC work on different frequency

Verification team has to traverse through all the frequencies for testing

Testing/Scaling of frequencies for memory interface, cores, coherent interconnects etc take huge simulation time!
Case-Study 3: Low Power Scenarios

✓ Various low power scenarios at Emulation level
✓ Core/Cluster Clock Gating schemes, Memory Retention schemes as well as complete power down and sleep scenarios for various blocks of SoC.
✓ Figure shows different processor states possible and the waveform shows a typical Power Down scenario for Cores and Cluster which can be verified using emulator in an efficient manner.
✓ These low power scenarios are also run in Power-Aware setup with UPF incorporated.
Methodology : Low Power Set-up

- Both UPF files and RTL files are fed to the EDA tool which understands the power intent of the design through UPF files and functional intent from design files.
- After the process of analysis, elaboration and splitting, and then synthesis, EDIF files are produced which is a vendor-neutral file format used to store netlists.
- The power-aware emulation model compilation flow has:
  - Similar UPF constructs are supported between simulation and emulation environments.
  - Debug continuity across PA-Simulation and PA-Emulation.
Compiling a PA Emulation Design Target

- Makefile changes and identifying compilation options that need to change for running low-power scenarios.
- While compiling, we need to fix any errors that may come due to any power related ports for any IP instances.
- We also need to ensure the availability of complete list of UPF libraries to be included.
- Next step is to use the available utility to convert present libraries to Emulation compatible libraries.
- After that we can run lib to DB conversion script and convert libraries to the format needed.
- After all this, once we fix all the UPF compilation issues related to version or any specific tool option, we can run our basic single/multi core power down scenarios to verify that the intent of the test has been achieved.
This methodology ensures that all synthesizable design components are re-used between PA Simulation and PA Emulation.

This means now we can use both platforms to complement each other in the Validation lifecycle with a high degree of confidence.

We have efficiently offloaded long running power aware scenarios to the PA Emulation environment to bring down the testing time to a fraction of what it was in the PA Simulation environment.
Some Common Issues

✓ **Clash of Rules and Design Attributes amongst various IPs:**

Enabling/disabling of power well biases varies from one IP to another and sometimes clashes with the power bias enabled/disabled rule at the SoC level. To resolve this, we had set enable_bias to false for all IPs and the SoC top.

✓ **Corrupt cell libraries:**

Cell libraries for various IPs have vastly different release schedules and hence during integration, sometimes we can end up with stale and corrupt lib paths which need to be updated manually.
Some Enhancements Done

1. **Single Step Run of the Scenario**
   - Automate the process of launching the run
   - Waveform dump using the script
   - Binary file split and use for full image runs

2. **Changing signal value during run time**
   - Deposit
   - Inject
   - Force

3. **Enable Checkers/Monitors**
   - Some SV-C handshakes should be enabled in Emulator run
   - Specific Tracer/Monitor enablement
### Results

#### CADENCE PLATFORM

<table>
<thead>
<tr>
<th>Test Category</th>
<th>No. of clock cycles</th>
<th>RTL Simulation Time (sec)</th>
<th>Emulation Run Time (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category 1</td>
<td>8,641</td>
<td>20529</td>
<td>1020</td>
<td>95.014</td>
</tr>
<tr>
<td>Category 2</td>
<td>5,182</td>
<td>10715</td>
<td>645</td>
<td>93.9804</td>
</tr>
<tr>
<td>Category 3</td>
<td>6,526</td>
<td>18654</td>
<td>840</td>
<td>95.4969</td>
</tr>
<tr>
<td>Category 4</td>
<td>7,951</td>
<td>23279</td>
<td>960</td>
<td>95.8761</td>
</tr>
<tr>
<td>Category 5</td>
<td>7,878</td>
<td>21782</td>
<td>971</td>
<td>95.5422</td>
</tr>
<tr>
<td>Category 6</td>
<td>7,250</td>
<td>17458</td>
<td>911</td>
<td>94.7818</td>
</tr>
<tr>
<td>Category 7</td>
<td>6,404</td>
<td>18880</td>
<td>765</td>
<td>95.8827</td>
</tr>
<tr>
<td>Category 8</td>
<td>8,530</td>
<td>16381</td>
<td>785</td>
<td>95.2976</td>
</tr>
</tbody>
</table>

#### SYNONYS PLATFORM

<table>
<thead>
<tr>
<th>Test Category</th>
<th>No. of clock cycles</th>
<th>RTL Simulation Time (sec)</th>
<th>Emulation Run Time (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category 1</td>
<td>9,199</td>
<td>20492</td>
<td>307</td>
<td>98.5019</td>
</tr>
<tr>
<td>Category 2</td>
<td>9,056</td>
<td>23335</td>
<td>288</td>
<td>97.7605</td>
</tr>
<tr>
<td>Category 3</td>
<td>7,374</td>
<td>18065</td>
<td>322</td>
<td>98.2176</td>
</tr>
<tr>
<td>Category 4</td>
<td>8,641</td>
<td>20529</td>
<td>331</td>
<td>98.3876</td>
</tr>
<tr>
<td>Category 5</td>
<td>6,526</td>
<td>18654</td>
<td>268</td>
<td>95.5633</td>
</tr>
<tr>
<td>Category 6</td>
<td>7,951</td>
<td>23279</td>
<td>275</td>
<td>98.8187</td>
</tr>
<tr>
<td>Category 7</td>
<td>7,375</td>
<td>21782</td>
<td>273</td>
<td>98.7467</td>
</tr>
<tr>
<td>Category 8</td>
<td>7,374</td>
<td>18432</td>
<td>275</td>
<td>98.5080</td>
</tr>
</tbody>
</table>

#### Additional Data

<table>
<thead>
<tr>
<th>Test Category</th>
<th>RTL Simulation Time (sec)</th>
<th>Emulation Run Time (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category 1</td>
<td>116877</td>
<td>2417</td>
<td>97.9820</td>
</tr>
<tr>
<td>Category 2</td>
<td>119090</td>
<td>2733</td>
<td>97.7051</td>
</tr>
<tr>
<td>Category 3</td>
<td>79703</td>
<td>4027</td>
<td>94.3043</td>
</tr>
<tr>
<td>Category 4</td>
<td>267216</td>
<td>7782</td>
<td>97.0877</td>
</tr>
</tbody>
</table>
Conclusion

• Multiple long-running scenarios can be smoothly ported onto an emulation platform and are much easier and faster than simulation.

• The experiments were carried out for several functional and low power scenarios along with simulation for the same scenario on same design drop for comparison.

• The run times were recorded and analyzed. The corresponding plots are shown in Figure 16 as an indicator of the results.

• By using this method, we significantly reduced the number of iterations needed, debug time, and the verification schedule. Current results are indicative and based on two emulator platforms from vendors like Cadence and Synopsys.

• We also narrowed down the areas of critical bugs since we specifically aim to find relevant bugs early through this environment and not the complete coverage.

• Further enhancing the process by adding automation, verification engineer can maneuver between the two environments very easily as per the requirement.

• The results achieved in some of the Exynos Mobile SoCs and Automotive SoCs were demonstrative of the fact that we saved at least 50X time in closure of certain critical long pole features.
**BENEFITS**

- Huge saving in SOC simulation run time
- Scalable to various platforms across different designs
- Major use case in low power functionalities
- Has the potential to be applied to IO Coherent scenarios

**LIMITATIONS**

- Licenses are limited for parallel runs
- Debug prints, logs and checkers are difficult to integrate and use
- Randomization of signals and memories is difficult
- Complete waveform dump with all hierarchies and full time duration limits the performance
Future Scope

- 4 State runs on Emulator
- Unified compile flow
- Hybrid Emulation
- Enabling SV assertions in ZeBu

Care abouts

- Usage of software code
- Choosing the scenarios wisely
- ZTDB/FSDB dump duration
- Additional task of model enablement
The authors would like to thank Samsung Semiconductors India Research for enabling the work mentioned in this paper. We would also like to thank DVCon Europe team for giving us the opportunity to participate in the conference and present our work.