

# Intel Xeon Pre-Silicon Validation

## *Introduction & Challenges*

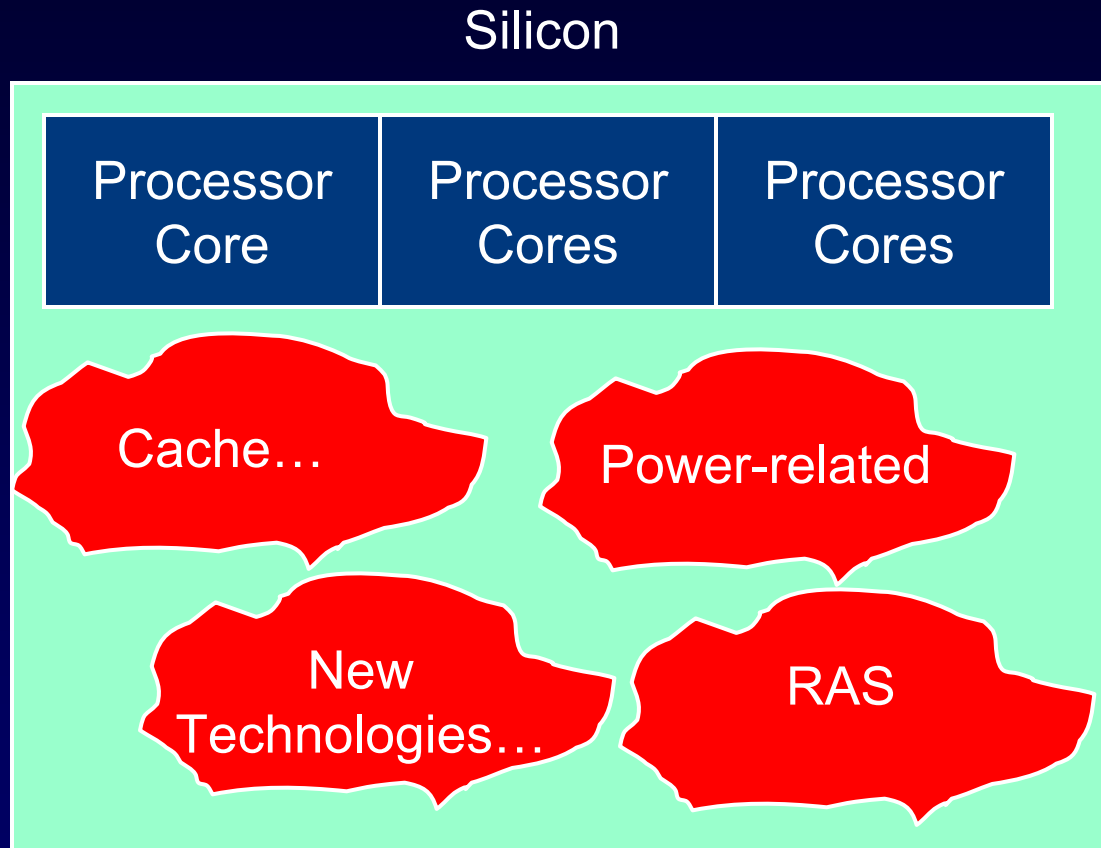
Paul Zehr

December 5, 2006

# Agenda

- This talk is intended to be an intro to Xeon pre-silicon validation work & challenges:
  - Please stop me & ask questions ←
- Introduction
  - What is a “typical” Xeon processor
- Challenges
  - Re-Use
  - Some managerial issues
  - The usual...

# What is a Xeon Processor?



Can be anything from a simple cache size increase to full blown from-scratch development of features

# Parameters Driving Xeon Validation

- TTM is the key driving factor
  - Quick ramp time → new features need comprehending quickly
- Relatively “small” teams
- Validation “dominates” wrt team size
  - Bonus: high demand for talented DV engineers
- Focus is on what’s new & changed
  - Still need to validate full chip
  - Beware: minor design changes can have broad impact
- Heavy re-use of other design environments
  - Combining environments always sounds easier at proj. conception...
- Tend to be cross-site efforts
- Trend: More & more from-scratch design being added

# Challenges: Re-Use

- First, a few examples:
  - Merced's bus cluster
  - Whitefield
- Expectation is that gluing various pieces of design together is fast & easy
  - State of the “parent” design can be an issue...
- Varying design languages / environments
- A lot of code implemented for “single use”
  - Poorly written code from parent projects
  - Never intended to be proliferated in the first place
- Engineers need to understand original intent
- Re-use of unfinished code

# Challenges: Managerial

- Influence of validation on project design during concept phase
  - Stop feature creep at the source...
  - “simple” arch deltas can cause inordinate validation
- Cross-site teams
- Documentation & training from parent project(s)
- Retention
  - Need *experienced* teams to meet goals, but,
  - Also need *development* challenges to retain...

# Challenges: The Usual

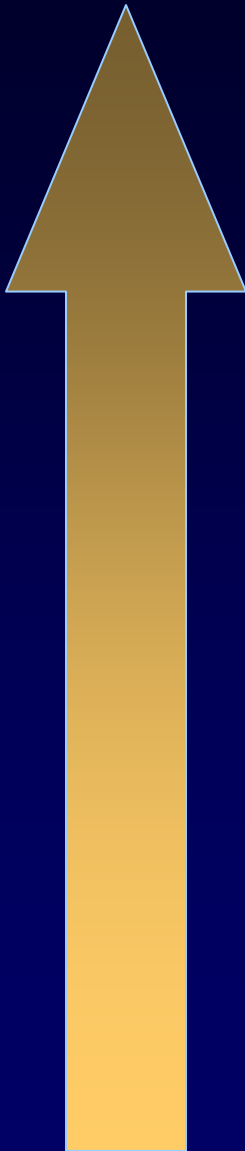
- Model Size & Speed
  - Emulation & Formal help, but...
  - Speed exacerbated by ever-growing number of clock domains
  - Ever-increasing sets of tools doesn't help either
- Emulation:
  - Long initial RTL model build time (months).
  - Less granular checking
  - Debug requires extra steps
  - Not a real effective way to get out early bugs.
- Formal for specific areas, but isn't mature
  - Meaning: We still simulate everything

# Challenges (cont.)

- Stimulus: We tend to be very repetitive:
  - “Start-Up” code is almost identical for 1000’s of tests
  - RIT example: S/W semaphores (or barriers) & interrupt handlers
    - *Many RIT generators code “macros” – same code each time*
    - *Some people have gotten creative here, but needs more*
  - Challenge is to minimize duplicate code – especially in the large bug-hunting runs.
- Checking: Escapes rarely due to lack of checking
  - Checker overhead still an issue
    - *Code ‘em into RTL – helps emulation as well*
    - *Could aid in post-Si debug if done properly*
- Coverage: Again, adds overhead to the model
  - A lot of philosophical differences here – internally & externally...
  - Need better solutions with less overhead – See checking.
- Debug: Vendors need to understand varying needs
  - Large, monolithic all-in-one solutions don’t suit all

# A bit more on stimulus & coverage

~100 %  
Covered



Low %  
Covered

	Pro	Con
<b>Formal Verification</b>	<ul style="list-style-type: none"><li>• 100% coverage</li><li>• Proves absence of bugs</li></ul>	<ul style="list-style-type: none"><li>• Requires special skills</li><li>• Constrained by complexity</li></ul>
<b>Directed Random Tests</b>	<ul style="list-style-type: none"><li>• Targets areas most likely to be of concern</li><li>• Greatly reduces cycle requirements</li><li>• Develops strong uArch knowledge</li></ul>	<ul style="list-style-type: none"><li>• Requires strong uArch knowledge</li></ul>
<b>Generic Random Tests</b>	<ul style="list-style-type: none"><li>• After generator created, easy to write</li><li>• Requires little uArch knowledge</li><li>• Can create things no one would ever think of</li></ul>	<ul style="list-style-type: none"><li>• Requires almost <math>\infty</math> cycles / time</li><li>• Difficult / impossible to avoid broken features</li></ul>
<b>Directed Tests</b>	<ul style="list-style-type: none"><li>• Easy to write</li><li>• Easy to understand</li><li>• Easy to reuse</li></ul>	<ul style="list-style-type: none"><li>• Requires almost <math>\infty</math> number of tests</li><li>• Difficult to hit uArch conditions (esp OOO, SMT machines)</li></ul>

# Challenges (cont.)

- Cross over between pre & post Si
  - Portability of tests between post & pre silicon
    - *Implications on environment*
  - Circuits/phys. design & implementation need functional tests to find timing & noise
    - *Intent verification requires ability to run on pre-si models*