High Performance and SoC Timing verification challenges

Tau 2015

KS Ramesh
Andalib Khan
Fritz Rothacker

Outline

- Processor design landscape
- Background
- Wish list
- Current status and issues
- Call for help

Processor Design Landscape

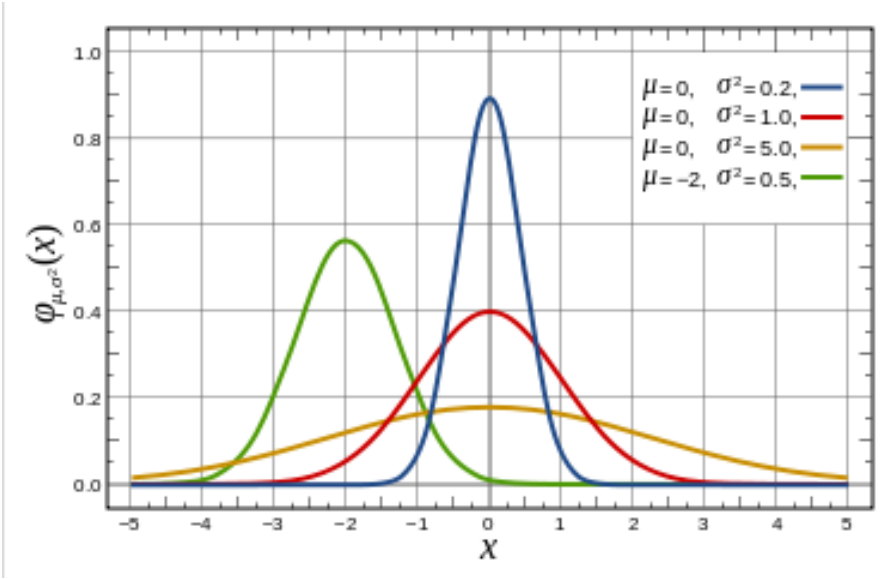
- **Scope**
  - Client: 100+$, Tick-Tock, Early process/product co-optimization/bring-up, highest Performance, adaptive methods
  - SoC: <<50$, process node specific, typically mature process, performance becoming bigger factor …
  - Wireless SoC: 10$, Fast TTM, Mature Process, IP re-use
  - PHYs? like DDR USB, …
- **Floorplans and Layout**
  - Client: Custom pin placement: Budgeting, Divide and Conquer, channels, repeaters, and more. Big die
  - SoC: abutted pins, channels, Bottom-up, Chip-level balancing. Sizes growing
- **Optimization**
  - Pttt, multiple voltage level, Proprietary Transistor-level timing tool, tax function, relative placement, latch/phase-based …
    Post silicon optimization/adaptation, binning, …
  - Pttt, multiple voltage level, ICC/PT
  - MCMM design convergence/verification

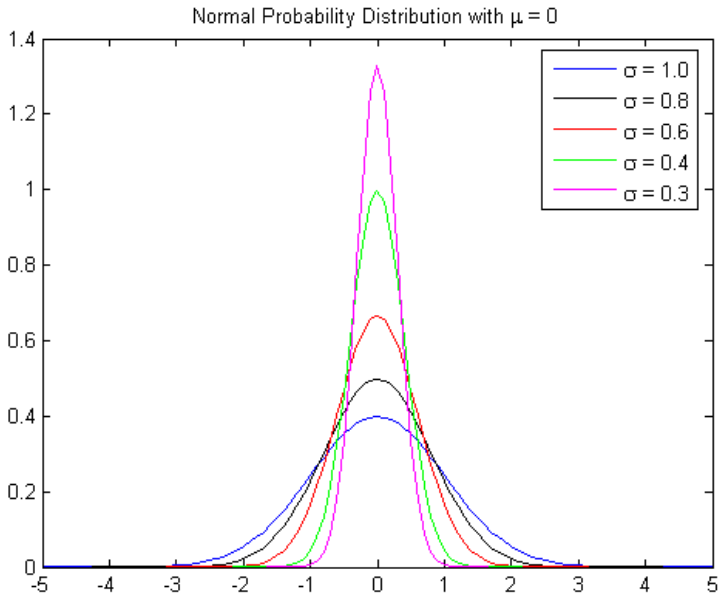
Traditional Client Processor design

- **Capitalizes on advanced process technology to the highest level**
- **Deep pipeline with a lot of transparent latches**
  - every ‘ps’ timing matters; accuracy in timing signoff tools very important
  - inaccurate transparency handling ; accumulate a lot of pessimism
  - In-house tools developed and optimized over multiple process nodes
- **Process-Design Co-optimization**
  - Design has to make a lot of assumptions about process technology, which can change significantly by product launch time
  - Requires timing analysis tools to add guard-bands for unknown
  - Typically design re-verified for process changes more than once in lifetime
  - Multiple “steppings” for multiple product targets

Very tight loop between process, design, test, and HVM

Design vs Process targets





Start of life
Middle of life
End of life
Different centering for same design

Different versions of stable process
mature and tight process
Mature but wider control

Custom to full automation design

- **CPU projects have**
  - more regular, synchronous clocking,
  - shallow clock tree to get performance,
  - Fewer cross domain clocking topologies
  - pretty similar clock frequency designs across part & family
  - Custom manual handling of design/analysis
    - Full custom designed global clock network
    - Hand routed FC and critical nets
    - Controlled placement
    - Pitch matched block pin alignment ; not limited to boundaries
    - Stricter max transition limits
  - incrementally tuned for performance and power with post-silicon feedback

- **SoC design has widely different frequencies, architecture, topologies, asynchronous logic, and clock structures.**
  - short lived; with full re-synthesis in each product/stepping.
  - Cannot afford custom design tweaks
  - Clock design and analysis is a different ball game!

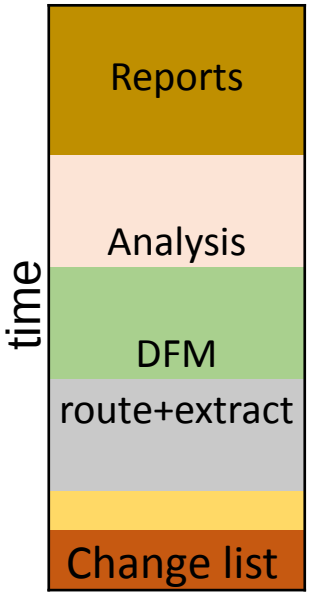
Transitioning to industry timing tools

- **Intel has long history with its timing tools and post-silicon correlations**
  - Many of these learnings are proprietary and not easily portable
  - changing verification tools drastically can lead to costly silicon escapes.
- **Intel has developed a lot of silicon learning based quality checks**
  - design methods are all embedded in various tools.
  - It is challenging to implement these using existing hooks provided by industry tools.
- **CPU STA tools are designed for hierarchal signoff with relatively smaller blocks.**
  - SoC designs generally involve much bigger design blocks
  - Meeting runtime and capacity for flat full-chip signoff requires industry tools that CPU tools cannot support.

What would we all want ?

- **Design Creation (MCMM)**
  - Optimize design for [Max, Min] constraints across corners
- **Scalable architecture**
  - Partition-Full Chip : can we get 24 hr loop time?
  - Synth -> Fill on partition (>1M instances) : 1-2 weeks currently
    - Can this be brought down to <48 hrs
  - Multi_threading support?
- **Standard API for custom design flows**
  - DFM verification
  - Variation
- **Scalable system for Spec/Verf convergence across hierarchies**
- **Support for multiple X'tor types & LV flows**
  - This is needed for leading process nodes
- **Clock design:**
  - #clocks, Hybrid, CTMesh/CTS
  - Ability to chose implementation style

Where do we stand?

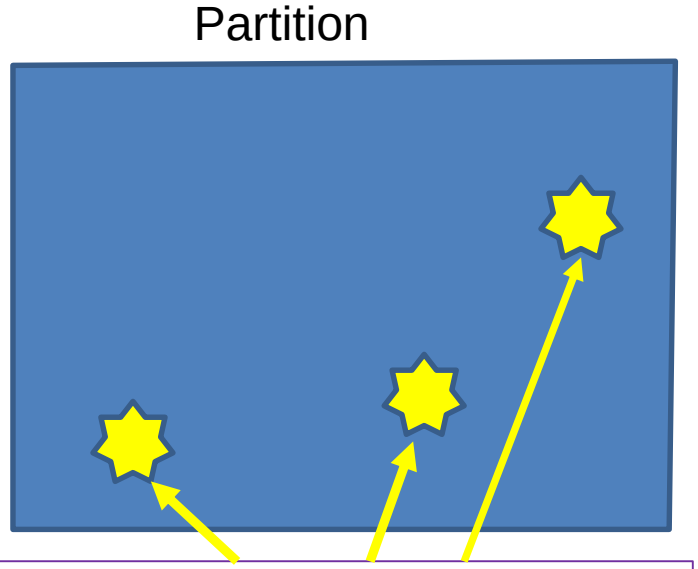


Huge runtime costs
Report writing long
DFM takes lot of time
Everything done on full DB

ECOs

Timing differences between ICC and PT post detailed route
Setup fixes
Corner fixes

Min + quality fixing loops
Block + FC reconciliation loops



Localized ecos
Tend to aggregate ecos to save on runtime penalty

Memory footprint of big partitions: > 70 G on a 1.5M instance block
Compute resources needed : server sessions = #corners analyzed

Hierarchical design challenges:

flat full chip level timing and quality models mandatory

- Flat fullchip timing shows very different timing
  - AOCVM mismatch
  - Cross-coupling mismatch if CRPR (point of divergence) outside the block
  - Pulse-width evaporation not modeled at the block level run
- Full-chip/block level loops are long and hurts iterations
- Difficult to do staggered block level execution.
  - All blocks need to be in a good shape for interface convergence
  - paths with external CRPR
- Stitching global clock network is difficult.

Need a good timing budgeting method

- Accurately allocate budgets for lower level blocks
- Can push down exceptions
- Comprehend clock source latencies
- Comprehend multiple clocks on the same clock pin

Global clock design:

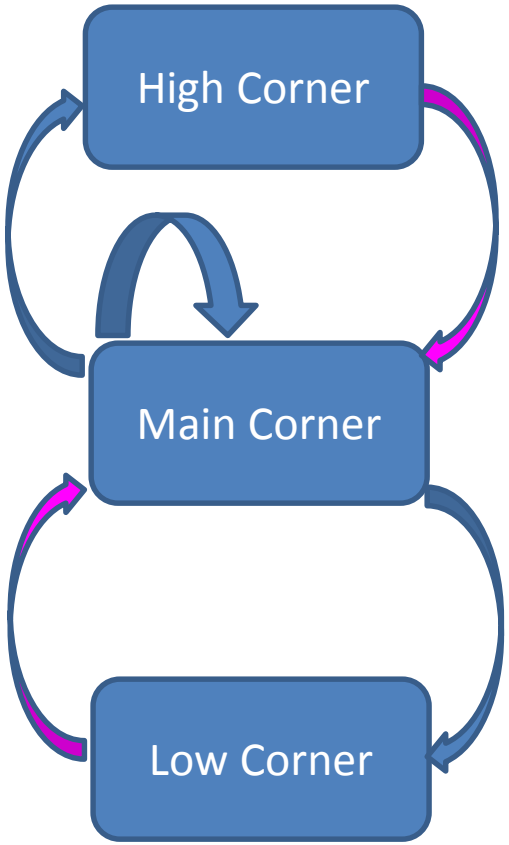
This is one of the biggest challenges in SoC design

- Design tools (CTS) cannot hit network latency values
  - Clk latencies of >500ps
  - Clk skew very high
  - becomes a clock design and timing nightmare
- especially for clocks routed as feed-throughs in other blocks
- Requires all blocks stable to model clock timing accurately
- Need ability to superimpose global clock timing while the design is in progress
- Need ability to mix ‘propagated vs ideal clocking’ block by block
  - pre and post cts/propagated clocks
  - supports staggered design execution

Multi-corner design and verification

Challenges: runtime impact, iteration loops, disruptive eco's

- Do design convergence in a single corner
- Optimizing in the others
  - Only be left with outlier violations
  - best productivity and fewer iterations
- Current overhead is quite high
  - to always run all corners during design and verification; especially when the design is not clean.
- Final signoff will happen in native corners



Setup and DRC ECO tools

- Better ECO capability to do more comprehensive timing ECOs
- Need to fix setup paths not just by sizing, but removing buffers, adjusting clocks, improving routing etc.
- Of course all this needs to be physically aware,
  - a feature that exists in many tools today. Ex; PT 2014
  - Intel is still in the evaluation phase on this

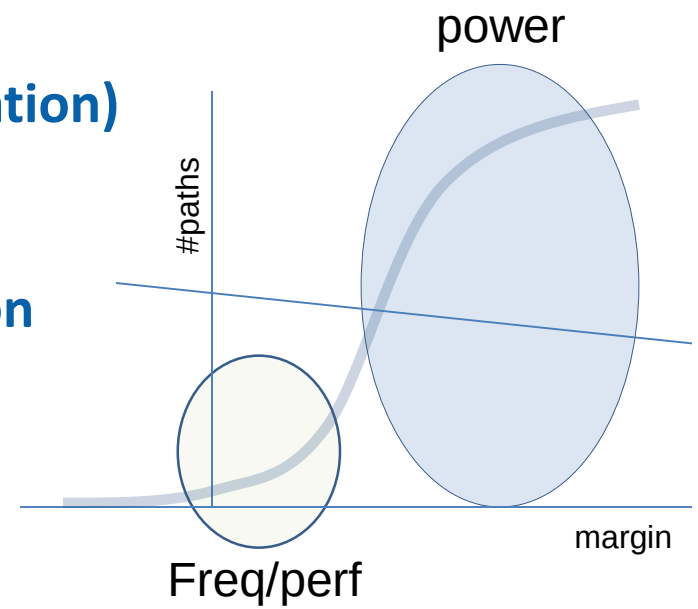
- A hook/callback function support to apply path based “tax” is missing in current PV tools
  - The function needs to access attributes like depth, distance, clock uncertainty, power domain, RC etc.
  - ability to tag a path and apply sticky guard-band/tax
    - present throughout the PT session
    - Currently only delay derates are available

Hold time fixing:

- **Hold fixing becomes a huge bottleneck for design closure**
  - Need hold fixing across multiple PVT,
    - physically aware,
    - power domain aware,
    - handle big design like flat full-chip
  - Runtime is a key issue here
    - algorithms work well at the block level, but runtime/crash is a big issue when running at FC
    - Looping over corners; prohibitive times
  - Smarter algorithms like
    - fixing hold using clock tuning,
    - creating area by getting rid of un-needed buffers,
    - side-load fixing etc.
  - Some vendor tools already have this !

Power

- **Power Info:**
  - Large Runtime
  - Large #tests ----- design DB (backend) ---- RTL (front end)
  - Datasize
- **Need AWI (automated window Identification)**
  - ICC/PT Run power reducing algorithms
- **RTL model based Estimation/Optimization**
- **Power reduction in the finfet era**
  - Different challenges



Challenges where we need help

- **Incremental extraction timing model with new SPEF/netlist**
- **Fill insertion and extraction runtime**
- **Complex Multi-via insertion to reduce resistance**
- **Routing optimization using metals with huge differences in RC**
- **Better support of latch based design and transparency modeling**
- **MCO design constraint verification**
- **Complexity must be abstracted/managed**
- **Power estimation and reduction – both BE and RTL**
- **Multi-mode support: further improvements**
  - Can overwhelm by throwing hardware resources
    - Not just costly from hardware,
    - big manual overhead to monitor runs(crashes)
    - But need smarter approach

Thank You!

Opens