Stochastic Processors (or processors that do not always compute correctly by design)

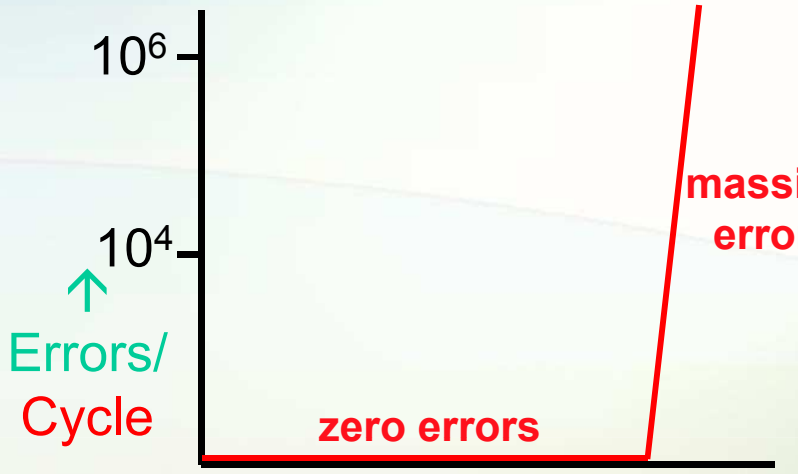
Rakesh Kumar

Department of Electrical and Computer Engineering

University of Illinois, Urbana-Champaign

Insisting on Correctness Always is Expensive

- Traditional CMOS-based computing engines take too much power because they are designed to always compute correctly
  - E.g., Guard-banding, redundancy, etc. increase power significantly
  - Insistence on correctness creates designs that fail catastrophically (below a certain critical voltage, for example)
    - Severely limits opportunities to reduce processor power. Eg., voltage can't be reduced below critical voltage



Increase Clock Frequency →
Or
Decrease Supply Voltage
Or
Increase Aging Degradation
Or
Increase Process Variations

(Critical Operating Point Hypothesis).
(courtesy Janak Patel, Illinois)

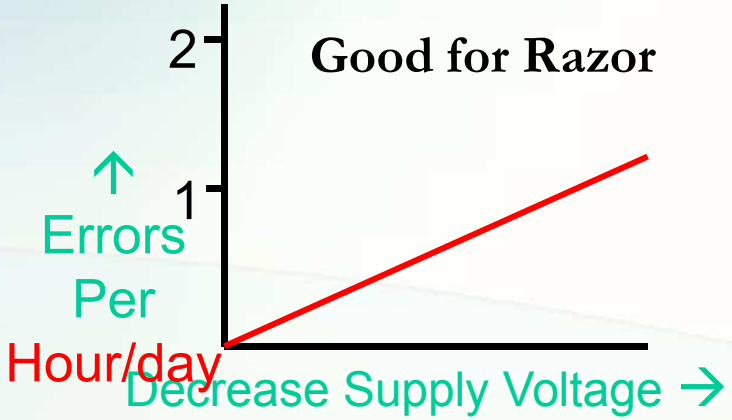
Insisting on Correctness Always is Expensive

- Cost of "always correct" computation even higher for nanoscale and post-CMOS technologies
  - Substrates exhibit high levels of parameter variations and other non-idealities
  - Cost of redundancy or guardbanding potentially enormous

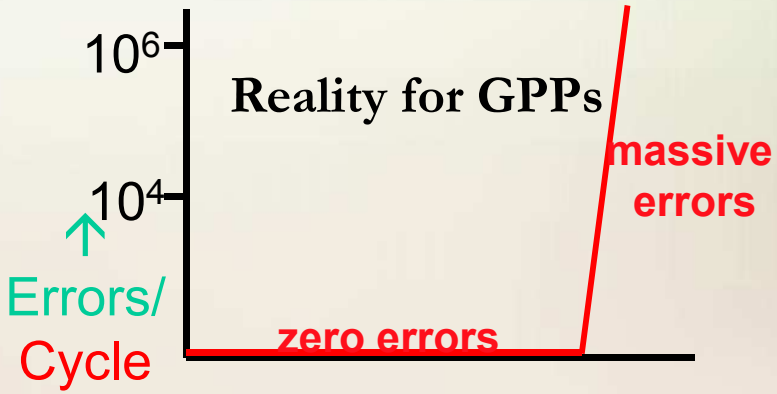
- **Hypothesis:** Extremely low power designs possible that do not always compute correctly, but still produce acceptable results due to the nature and number of errors.
  - We call such architectures stochastic processors.

Comparing against Better-than-Worst-Case Designs

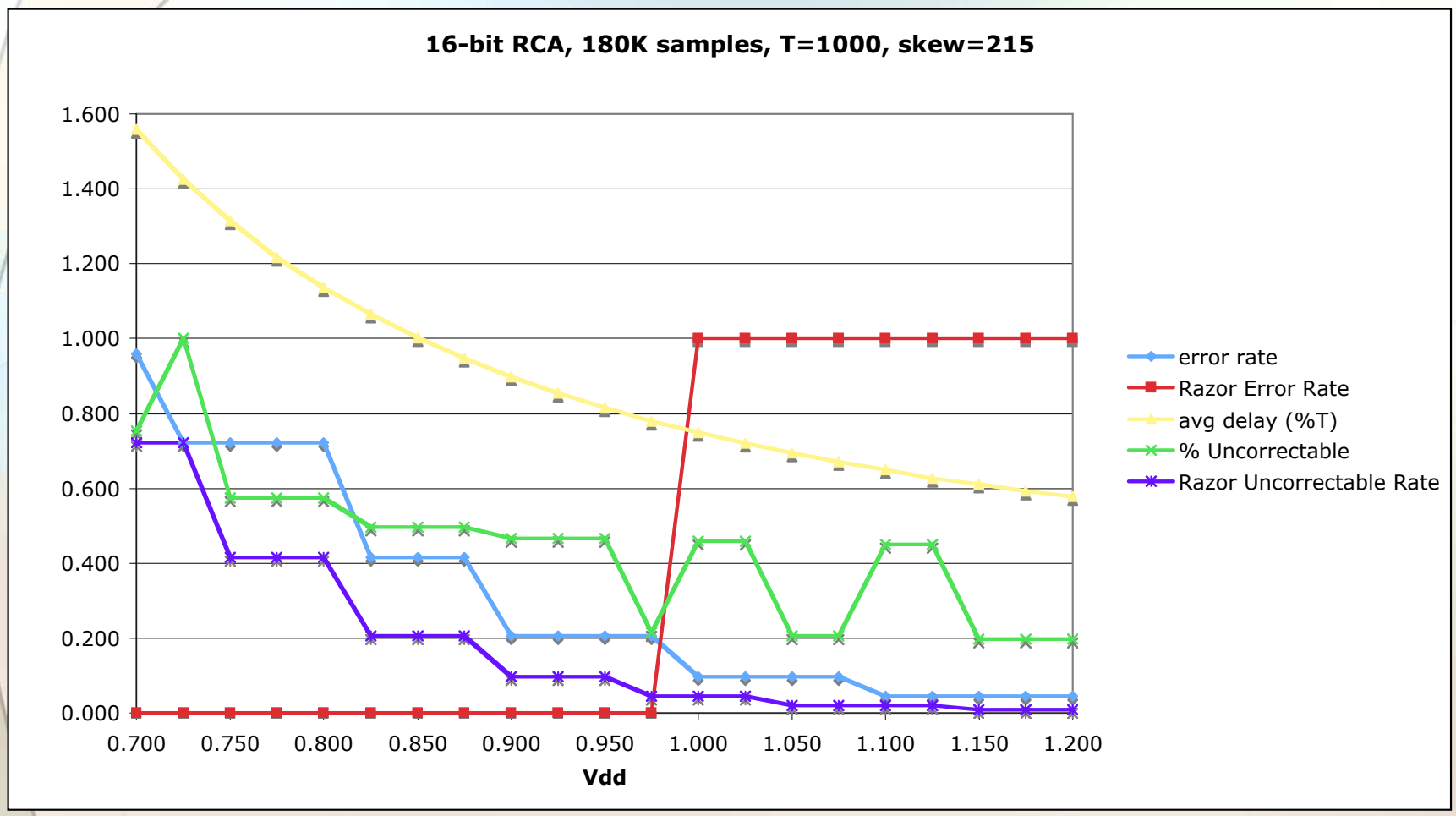
- Better than worst-case Designs (e.g., Razor) allow occasional errors to save power.
  - Allow aggressive voltage scaling, for example



- Benefits limited by the existing design of the processors
  - Most power benefits in the range where there are no errors
    - Very small voltage range where Razor is useful in face of errors
  - Even if processor were designed to degrade gracefully, can't do much scaling beyond critical voltage/frequency



16-bit Ripple Carry Adder



Razor only works in size T window

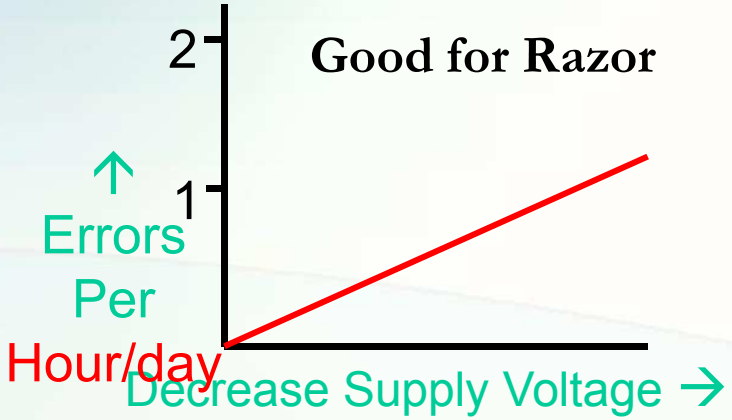
100% False Razor-induced errors when min < skew

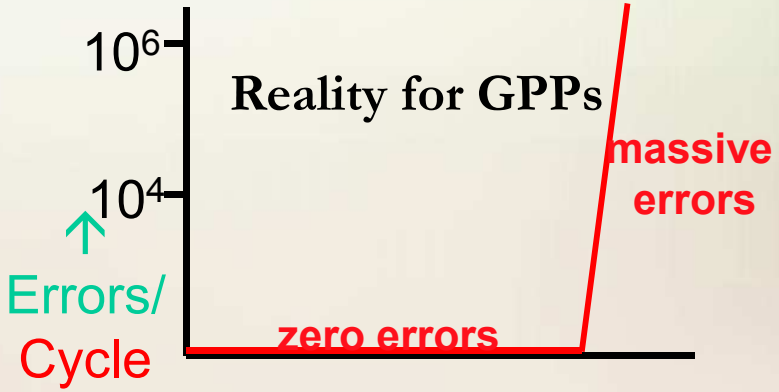
Must turn off Razor and accept some level of error

Many uncorrectable errors when T+skew not large enough

Comparing against Better-than-Worst-Case Designs

- Better than worst-case Designs (e.g., Razor) allow occasional errors to save power.
  - Allow aggressive voltage scaling, for example
- Benefits limited by the existing design of the processors
  - Most power benefits in the range where there are no errors
    - Very small voltage range where Razor is useful in face of errors
  - Even if processor were designed to degrade gracefully, can't do much scaling beyond critical voltage/frequency
- Still do not allow errors to be exposed to the system/application
  - So not really *allowing* errors



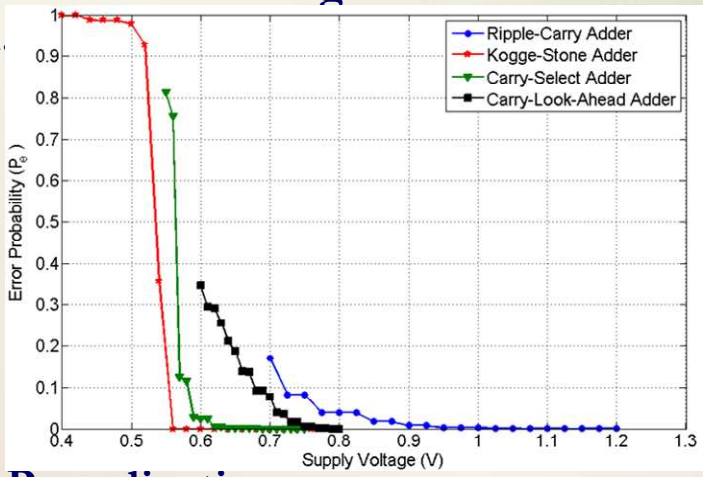


Need something better than better-than-worst-case designs

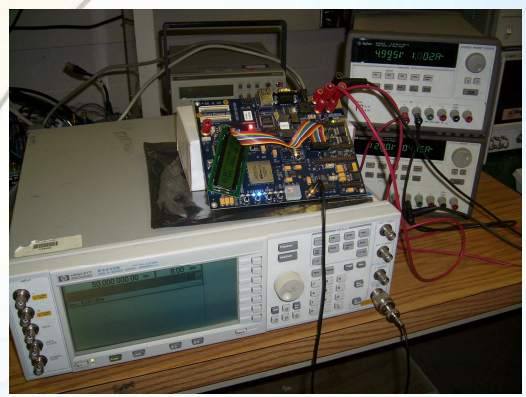
Stochastic Processors: Insights and Research Plan

- Insight#1:
  - A large class of emerging client-side (*in field*) applications have inherent algorithmic/cognitive noise tolerance.
  - So, processors can be optimized for very low-power instead of always preserving correctness.
    - Errors tolerated by the applications instead of spending power in detecting/correcting errors at the circuit/architecture level.
- Insight#2:
  - If processor designed to make errors gradually instead of catastrophically, significant power savings possible
    - E.g., when input voltage is decreased below critical voltage (*voltage overscaling*). for power reduction.
- Research Plan
  - Develop stochastic architectures that produce graceful degradation in terms of errors
  - Define the CAD flow for implementation stochastic processor architectures
  - Develop a library of error-tolerant kernels

that implement (Mobile Augmented Reality) MAR applications.

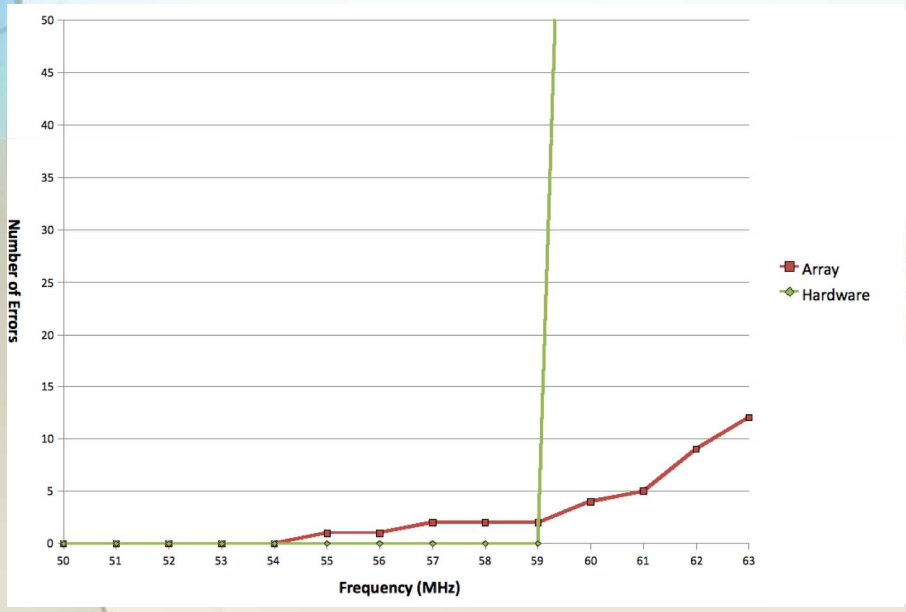


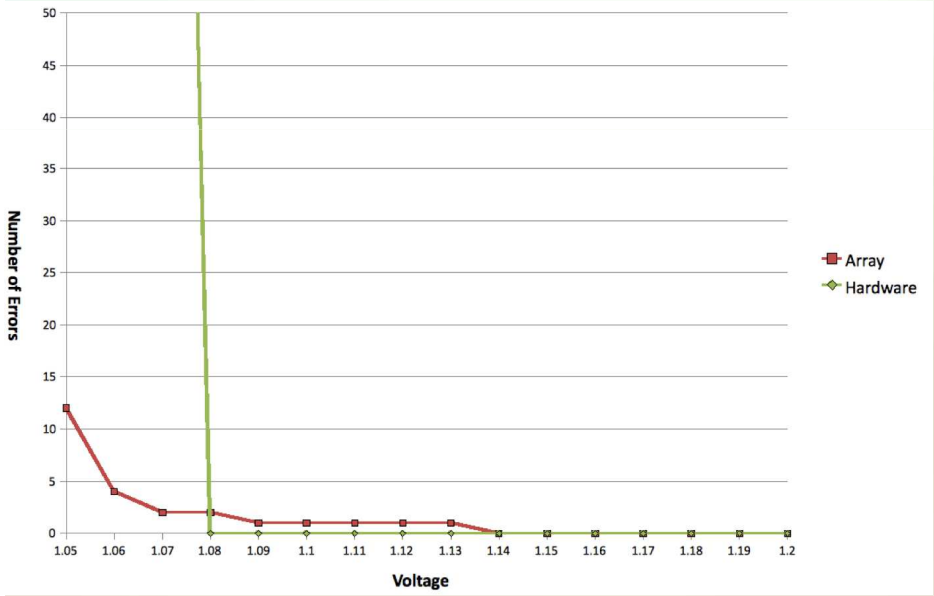
Stochastic Processors: An example microarchitectural solution



FPGA testbed consisting of soft-core processor

- **Modified to allow frequency/voltage overscaling**
- **Utilizes Leon3 processor**
  - **7-stage in-order pipeline, SPARC 8 ISA**
- **Stratix II FPGA**
  - **Baseline uses On-chip signed 18x18 hardware multiplier**
  - **Stochastic version uses an soft array-based multiplier with staggered delay characteristics**

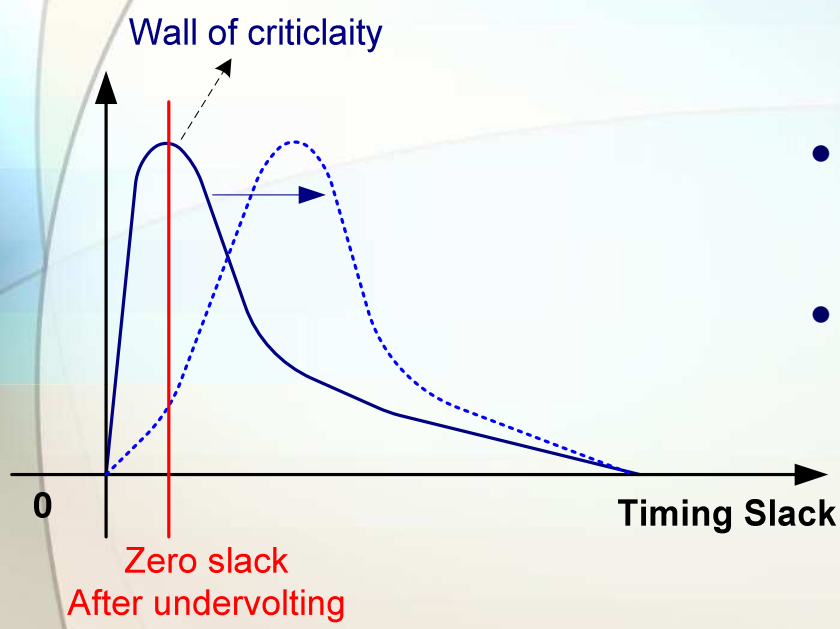




Significant throughput/power benefits of a stochastic processor design

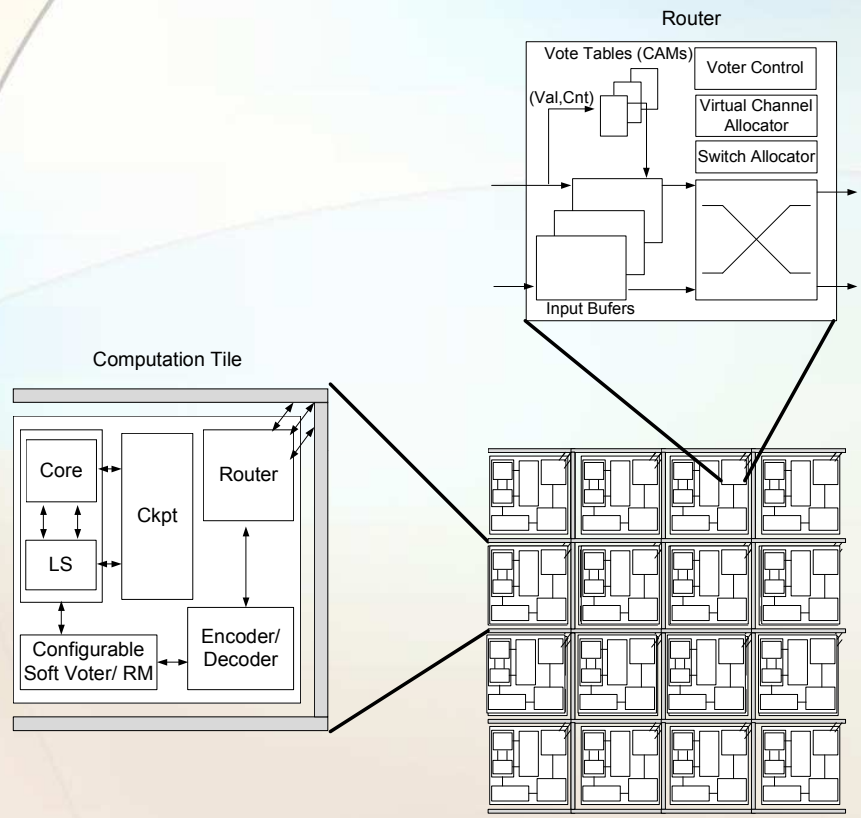
(More details in our SLESE 2009 paper)

Stochastic Processors: An example CAD-level solution



- For a slow rising slack, we have to move the slack of some paths to the right (positive) position by applying a tighter constraint.
- There are two methods on this; path based and cell based.
- In the path based method, we can use a set_max_delay from to constraints on some selected paths in SP&R.
  - Using this tighter constraints on some paths, the shape of slack distribution could be changed.
- In the cell based method, we can multiply a derating factor to the delay of cells on the target paths. This method will be easier to implement than the path based method.

Stochastic Processors: An example architecture-level solution



Microarchitecture allows maximal separation of datapath and control

- E.g., GALS

A shared-nothing/message passing architecture with configurable routers and voting logic

- Allows fault containment and tolerance to timing errors due to asynchrony

Dynamic NMR allows adaptation to different reliability targets

In-network voting reduces the overhead of voting

Related Work

- Probabilistic System-on-chip Architectures
  - Partition applications into probabilistic and deterministic components
    - Run probabilistic components on a PCMOS co-processor (powered near sub-threshold voltage)
- Stochastic Processors vs PSOC
  - Our approaches target power reduction in general purpose processors
  - PSOC designs are hand-partitioned and application specific
  - Applications
    - Stochastic processors useful for a large class of applications with no explicit probabilistic components
    - PSOC requires strict partitioning between probabilistic and deterministic components
  - Error Characteristics
    - PSOC requires controlled randomness/errors
    - We focus more on efficient techniques to eliminate or deal with errors rather than controlling their characteristics
  - Accelerator / coprocessor design of PSOC incurs communication cost
    - This can become an issue when probabilistic step is critical to the application

Summary and conclusions

- Significant power/throughput benefits may be possible if correctness in all situations is not a requirement
- Cannot simply use the better than worst-case designs
- Stochastic processors allow aggressive undervolting/overscaling even beyond critical voltage/frequency
  - May also expose errors to system software/applications
  - May be the only way to do design for nanoscale/post-CMOS technologies
- Preliminary results are promising
- Open up an entire area of research that is interesting and with potentially very high payoffs