The Variability Expeditions:
Variability-Aware Software for Efficient Computing With Nanoscale Devices.

Rajesh K. Gupta

Nikil Dutt, UCI
Punit Gupta, UCLA
Mani Srivastava, UCLA
Lucas Wanner, UCLA
Steve Swanson, UCSD

Lara Dolecek, UCLA
Subhashish Mitra, Stanford
YY Zhou, UCSD
Tajana Rosing, UCSD
Alex Nicolau, UCI
Ranjit Jhala, UCSD
Sorin Lerner, UCSD
Rakesh Kumar, UIUC
Dennis Sylvester, UMich
To a software designer, all chips look alike

To a hardware engineer, a chip is delivered as per contract in a data-sheet.

5.5 Current Consumption

All of the below current consumption data is lab data measured on a single device using an evaluation board. Table II shows the typical current consumption in low-power modes at various $f_{CPU}$ frequencies. Current measurements are taken after executing a STOP instruction.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Voltage (V)</th>
<th>$f_{CPU}$ (MHz)</th>
<th>40 MHz</th>
<th>56 MHz</th>
<th>64 MHz</th>
<th>72 MHz</th>
<th>83.33 MHz</th>
<th>83.33 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop Mode 1</td>
<td>3.3</td>
<td></td>
<td>1.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 2</td>
<td>2.5</td>
<td></td>
<td>15.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 3</td>
<td>1.8</td>
<td></td>
<td>0.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 4</td>
<td>2.5</td>
<td></td>
<td>15.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 5</td>
<td>2.5</td>
<td></td>
<td>0.92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 6</td>
<td>2.5</td>
<td></td>
<td>15.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 7</td>
<td>2.5</td>
<td></td>
<td>1.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 8</td>
<td>2.5</td>
<td></td>
<td>1.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 9</td>
<td>2.5</td>
<td></td>
<td>15.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 10</td>
<td>2.5</td>
<td></td>
<td>15.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 11</td>
<td>2.5</td>
<td></td>
<td>1.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Mode 12</td>
<td>2.5</td>
<td></td>
<td>0.92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.8.1 SDR SDRAM AC Timing Characteristics

The following timing numbers indicate when data will be latched or driven onto the external bus, relative to the memory bus clock, when operating in SDR mode on write cycles and relative to SD_DQS on read cycles. The SDRAM controller is a DDR controller with an SDR mode. Because it is designed to support DDR, a DQS pulse must remain supplied to the device for each data burst of an SDR read. The COMFIRE processor accomplishes this by asserting a signal called SD_SDRAM_DQS during read cycles. Take care during board design to adhere to the following guidelines and specs with regard to the SD_SDRAM_DQS signal and its usage.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Characteristic</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Clock Period ($t_{CC}$)</td>
<td>12</td>
<td>16.67</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>Pulse Width High ($t_{OH}$)</td>
<td>0.45</td>
<td>0.55</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>Pulse Width Low ($t_{OL}$)</td>
<td>0.45</td>
<td>0.55</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>Address, SD_CLK, SD_RAS, SD_WE, SD_BA, SD_CKE - Output Valid ($t_{OH}$)</td>
<td>—</td>
<td>0.5 × SD_CLK + 0.5 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>Address, SD_CLK, SD_RAS, SD_WE, SD_BA, SD_CKE (1.0) - Output Hold ($t_{OH}$)</td>
<td>2.6</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>SD_SDR_DQS Output Valid ($t_{OH}$)</td>
<td>—</td>
<td>Self timed</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td>SD_DQS(3) Input relative to SD_CLK (100%)</td>
<td>0.25 × SD_CLK + 0.25 ns</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S8</td>
<td>SD_DQS(3) Read relative to SD_CLK (100%)</td>
<td>0.25 × SD_CLK + 0.25 ns</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S9</td>
<td>Data (B1:0) Input relative to SD_CLK (100%)</td>
<td>0.25 × SD_CLK + 0.25 ns</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S10</td>
<td>Data (B1:0) Input relative to SD_CLK (100%)</td>
<td>0.25 × SD_CLK + 0.25 ns</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>S11</td>
<td>Data (B1:0) Input relative to SD_CLK (100%)</td>
<td>0.25 × SD_CLK + 0.25 ns</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
From Chiseled Objects to Molecular Assemblies

The paradigm simulation yesterday
At 32nm (physical gate length) MOSFET in production today
At 9nm MOSFET in production by 2023

Temperature Clock
actual circuit delay guardband
Aging

V_{CC} Droop
Across-wafer Frequency

Nominal scaling
Oversized scaling

Performance
Technology Generation

Clock

20-May-13
Abbas Rahimi/ UC San Diego
What if?

- Application
- Hardware Abstraction Layer (HAL)
- Operating System

Time or part

underdesigned hardware
Builds upon a 50-year rich research in fault tolerance.
UNO Computing Machines Seek Opportunities based on Sensing Results

- Do Nothing
- Change Hardware Operating Point
- Change Program Parameters
- Change Runtime Parameters
- Change Algorithms

Metadata Mechanisms: Reflection, Introspection

Models
- Sensors
Building Machines that leverage move from Crash & Recover to Sense & Adapt

Machines that consist of parts with variations in performance, power and reliability

Machines that incorporate sensing circuits

Machines w/ interfaces to change ongoing computation & structures

New machine models: QOS or Relaxed Reliability parts
Example: Procedure Hopping in Clustered CPU, Each core with its voltage domain

- Statically characterize procedure for PLV
- A core increases voltage if monitored delay is high
- A procedure hops from one core to another if its voltage variation is high
- Less 1% cycle overhead in EEMBC.

\[ V_{\text{DD}} = 0.81\text{V} \]

\[ V_{\text{DD}} = 0.99\text{V} \]

\[ \text{VA-V}_{\text{DD}}\text{-Hopping} = (0.81\text{V}, 0.99\text{V}) \]
• The code is easily accessible via the shared-L1 I$/.
• The data and parameters are passed through the shared stack in TCDM.
• A procedure hopping information table (PHIT) keeps the status for a migrated procedure.
ViPZonE: Exploiting Memory Power Variability

- App developers can optimize dynamic allocations for reduced power
- Linux + Glibc implementation
Example: UnO Stack for Duty-cycled Sensors

Many Sensors: \( P_{\text{sleep}}, P_{\text{active}}, \text{Memory Speed}, \text{Temp}, \text{Battery}, \ldots \)

Module SenseAndForward {
  provides energylevel LowFid<1>;
  provides energylevel MidFid<2>;
  provides energylevel HiFid<3>;
  ( On_event Timer
    call SensorRead();
    On_event LowFid
    call Timer(2500);
    On_event MidFid
    call Timer(2000);
    On_event HiFid
    call Timer(1650);)
}

Module SenseAndForward {
  provides energylevel LowFid<1>;
  provides energylevel MidFid<2>;
  provides energylevel HiFid<3>;
  ( On_event Timer
    call SensorRead();
    On_event MonitorTimer
    call SysinfoRead(&sysinfo);
    If Error > Delta
    call Time(DownSample);)
}

Module SenseAndForward {
  provides energylevel LowFid<1>;
  provides energylevel MidFid<2>;
  provides energylevel HiFid<3>;
  ( On_event SysinfoChanged
    call SysinfoRead;
    If Error > Delta
    call Timer(DownSample);)
}
RESEARCH AND ITS ORGANIZATION

GRAND CHALLENGE, QUESTIONS AND RESEARCH PROGRESS
Expedition Grand Challenge & Questions

“Can microelectronic variability be controlled and utilized in building better computer systems?”

Three Goals:

a. Address fundamental technical challenges (understand the problem)

b. Create experimental systems (proof of concept prototypes)

c. Educational and broader impact opportunities to make an impact (ensure training for future talent).

I.D. Overview of Expedition’s Plan

Our Expedition plan has three goals: (a) to address the fundamental technical challenges in the realization of the UnO computing machines; (b) to create experimental systems at different scales to evaluate the idea in real-life application contexts; and, (c) to leverage the educational and other broader impact opportunities offered by such a rethinking of traditional computing machines.

In pursuit of these goals, our objectives include addressing the following interlinked questions:

1. What are most effective ways to detect variability?
   - Sensors embedded in the circuit and software instrumentation, which poses the challenge of minimizing area, time, and energy costs.

2. What are software-visible manifestations?
   - The trade-off between quality and overhead of information exchanged from hardware to software (termed “hardware signatures”).

3. What are software mechanisms to exploit variability?
   - Explicitly provide alternative algorithms optimized for different hardware presentations but which share as much code as possible to improve code density, debuggability, etc. Alternatively, compilers may automatically generate different code configurations, perhaps even dynamically at run time without algorithm intervention. In either case, some level of run-time assist from the OS will be needed.

4. How can designers and tools leverage adaptation?
   - About the application behavior (such as the quality metrics and the reaction to variable performance and error rate) to be passed down to the design flow, as well as effective design automation algorithms for incorporating this information as soft constraints during synthesis, placement, routing etc. This operation may need to be done at run-time in the case of hardware platforms that expose circuit-level “knobs” such as sleep modes, voltage scaling, and frequency scaling, or are implemented on in-field reconfigurable devices, e.g., soft processor cores on FPGAs.

5. How do we verify and test hw-sw interfaces?
   - One might allow under-verification of hardware by ensuring the correctness of the overall behavior of an opportunistic application and its associated software stack rather than that of the hardware alone.
Research Organization

• Four thrust areas
  1. Measurement and Modeling
  2. Design Tools and Testing Methodologies
  3. Microarchitecture and Compilers
  4. Runtime Support

• Two Cross-cutting thrusts
  5. Applications and Testbeds
  6. Outreach and Education

Thrusts span teams across universities, usually in pairs.
Thrusts traverse institutions on testbed vehicles seeding various projects

**Group A: Signature Detection and Generation**
- Characterizing variability in power consumption for modern computing platforms, and implications
- Runtime support and software adaptation for variable hardware
- Probabilistic analysis of faulty hardware
- Understanding and exploiting variability in flash memory devices
- FPGA-based variability simulator

**Group B: Variability Mitigation Measures**
- Mitigating variability in solid-state storage devices
- Hardware solutions to better understand and exploit variability
- VarEmu emulation-based testbed for variability-aware software
- Variability-aware opportunistic system software stack
- Application robustification for stochastic processors

**Group C: Opportunistic Software and Abstractions**
- Effective error resilience
- Negative bias temperature instability and electromigration
- Memory-variability aware runtime systems
- Design-dependent ring oscillator and software testbed
- Executing programs under relaxed semantics
Two years of building an Expedition

• Kickoff, review, tape-outs and builds-ins
  – 82 peer-reviewed publications, 21% collaborative
  – 54 events/releases on variability.org/news
  – 64 presentations on variability.org/presentations

• A collaborative community
  – 15 faculty, 25 GSRs, 1 postdoc, 10+ UG, 300 K-8-12
Timeline in Progress

- **Oct**
  - Y1 Review
  - IMEC/ESWeek
  - ATS
  - Research Review (UCSD)
  - Industry Advisory (Stanford)
  - Y2 Review
  - Complete Eval Boards w/ S-ARM
  - 28nm Test Chips
  - CUDA Simulator
  - Samsung (Tapeout Measurements)
  - ARM, TSMC (Benchmarking)
  - Samsung (Tapeout Measurements)
  - ARM, TSMC (Benchmarking)

- **Nov**
  - IMEC/ESWeek
  - ATS
  - Research Review (UCSD)
  - Industry Advisory (Stanford)
  - Y2 Review
  - Complete Eval Boards w/ S-ARM
  - 28nm Test Chips
  - CUDA Simulator
  - Samsung (Tapeout Measurements)
  - ARM, TSMC (Benchmarking)

- **Dec**
  - IMEC/ESWeek
  - ATS
  - Research Review (UCSD)
  - Industry Advisory (Stanford)
  - Y2 Review
  - Complete Eval Boards w/ S-ARM
  - 28nm Test Chips
  - CUDA Simulator
  - Samsung (Tapeout Measurements)
  - ARM, TSMC (Benchmarking)
• Year 1 was mostly focused on **characterization of variability** (IC designer centric)
  – What is the extent of variation and can it be sensed? Can it be used in the HW/SW stack?
• Year 2 focused on proof-of-concept **methods to use variability information** (Programmer centric)
  – From observation to systematic control.
  – Can we construct **useful signatures** that can enable systematic observability (and controllability) of variation?
• Year 3 sees the two streams coming together: expanding collaborations across teams, emerging testbeds & tools.
Important Takeaways

To ensure effective use by software, we need accurate characterization (of performance, power).

1. Variability imposes a limit on how accurate the models can get to
   - Mean error ~20% + 12% due to variability for 34% overall error in Nehalem 45nm CPUs
   - 15-20% variation across 22 DIMMs
   - 20-24% read, 40-67% write variation in Flash
   - Rooted in inherent non-observability of power states.
Important Takeaways (continued)

2. Instrumentation and sensing is necessary to ensure ‘high-level’ observability of variation
   – “High enough for semantic value.” Averages may not be sufficient.

3. Sensing for delay, power, aging and degradation is feasible and indeed necessary
   – Important difference between failure prediction and error detection. Notion of static & dynamic variability management.

4. Variability can be leveraged in software
   – media applications, duty cycle, security sensitive applications. Notion of ‘tunable error’ and its observability criteria.
2. Instrumentation and sensing is necessary to ensure 'high-level' observability of variation – "High enough for semantic value." Averages may not be sufficient.

3. Sensing for delay, power, aging and degradation is feasible and indeed necessary – Important difference between failure prediction and error detection. Notion of static & dynamic variability management.

4. Variability can be leveraged in software – media applications, duty cycle, security sensitive applications. Notion of 'tunable error' and its observability criteria.

At the end of two years, we have a complete end-to-end initial realization of an embedded system platform with sensing chip, board-level feedback, OS supporting duty-cycled tasks driven by variability, and API for such machines.
Expedition Experimental Platforms & Artifacts

• Interesting and unique challenges in building research testbeds that drive our explorations
  – Mocks up don’t go far since variability is at the heart of microelectronic scaling. Need platforms that capture *scaling* and *integration* aspects.

• Testbeds to observe (Molecule, GreenLight, Ming), control (Oven, ERSA)
Red Cooper Testbed: *in-situ* visibility

- Customized chip with processor + speed/leakage sensors available since April 2011
- Testbed board to finish the sensor feedback loop on board
Ferrari Chip: Closing Loop On-Chip

- **On-Chip Sensors**
  - Memory mapped i/o and control
  - Leakage sensors, DDROs, temperature sensors, reliability sensors
- **Better support for OS and software.**

Available April 2013
From Control to Software Abstractions

Going forward

• Leon3 (Sparc) sensorized chip tapeout

• Software abstractions: PL and Runtime
  – A formal/consistent way of exposing hardware signatures
  – A full Linux software stack working

• Verification methods
  – Performance & power invariants at RT-level in the presence of variability (with TI) using probabilistic model checking
    • Similar to property checking against Monte Carlo simulations
  – Automatic generation of invariants and assertion synthesis.
Reaching out and building a community

Building our teams across 6 six sites
Building our mentors and champions
Creating early adopters
Inspiring talent
## Emerging Synergies

<table>
<thead>
<tr>
<th></th>
<th>UCSD</th>
<th>UCLA</th>
<th>UCI</th>
<th>UIUC</th>
<th>UM</th>
<th>Stanford</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Cooper</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Molecule</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIPZONE</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VarEMU</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Ferrari</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ERSA/LLVM</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>Systems</th>
<th>LL Code</th>
<th>LL Code</th>
<th>Chips</th>
<th>Sensors</th>
</tr>
</thead>
</table>

- **Examples of collaborative discovery**
  - Lara Dolecek working with Steve Swanson & Mitra
  - Dennis Sylvester at the center of chip/platform characterization
  - Nik Dutt, Alex Nicolau and Rakesh Kumar on code scheduling
  - Rakesh Kumar, Sorin Lerner, Ranjit Jhala on code analysis and programming language support for variability.
Thank You!