Title

RedCooper: Hardware Sensor Enabled Variability Software Testbed for Lifetime Energy Constrained Application

Authors

Agarwal, Yuvraj
Bishop, Alex
Chan, Tuck-Boon
et al.

Publication Date

2014-06-02

RedCooper: Hardware Sensor Enabled Variability Software Testbed for Lifetime Energy Constrained Application

Yuvraj Agarwal[†], Alex Bishop[†], Tuck-Boon Chan[†], Matt Fotjik[‡], Puneet Gupta[*], Andrew B. Kahng[†], Liangzhen Lai[*], Paul Martin[*], Mani Srivastava[*], Dennis Sylvester[‡], Lucas Wanner[*] and Bing Zhang[*]
UCLA[*], UCSD[†], University of Michigan[‡]
E-mails: liangzhen@ucla.edu

***Abstract*—Conventional hardware uses overdesigned margins to guardband against variability, which incurs significant amounts of power and performance overhead. If the variations can be captured and exposed to the higher levels (e.g., system/software levels), the margin can be reduced or even eliminated through opportunistic hardware/software adaptation. In this paper, we present our end-to-end implementation of an software testbed with built-in hardware sensors and adaptive software. The measurement results show that using our novel performance sensor, *Design-Dependent Ring-Oscillator* (DDRO), can reduce the mean delay estimation errors by up to 35% (from 2.5% to 1.5%) compared to using generic inverter-based ring-oscillator. By utilizing the sensing infrastructure on our *RedCooper* testbed, a demonstration shows that the hardware and software adaptation can achieve up to 2.7X total active time increase for lifetime energy constrained, as compared to sensorless system.**

I. Introduction

As semiconductor manufacturing process advances and feature size shrinks, hardware sees an increasing amount of variability. Performance variation and power variation are the two major manifestations of hardware variability. Conventional approaches use overdesign and guardbands for the variations, which leaves increasing amounts of power and performance potential untapped.

If the variations can be captured and exposed to the higher levels of stacks (e.g., system and software levels), the margin can be reduced or even eliminated through opportunistic hardware/software adaptation [1]. For example, a processor typically has some margin on its operating frequency to guarantee correct functionality across the worst-case process, voltage and temperature (PVT) variations. The margin can be reduced if we can estimate the processor's performance through on-chip performance monitors (e.g., ring-oscillators). When the performance monitor reports an estimation better than worst-case, which is likely, we can operate the processor at a higher frequency. Alternatively, we can utilize the "better than worst-case" scenario by lowering the supply voltage for power reduction.

In addition to hardware adaptations enabled by performance sensors, software adaptation can also help eliminate the overdesigned guardband. For example, [2] proposes software duty-cycling for lifetime energy constrained embedded sensing applications. By exposing system's current power consumption information to the software, the application can dynamically adjust its duty cycle rates (i.e., ratio of chips active time to its lifetime) so that overall *quality of service* (QoS), as determined by system active time, can be optimized within pre-specified energy budget and lifetime requirements.

There are three major issues in the implementation of such an adaptive system: 1).How to design efficient and accurate sensors to capture the hardware variability signature? 2). How to expose the hardware sensors to the system and software? 3). How can the system and software adapt to the sensed hardware variability?

In this work, we present our end-to-end implementation of such systems which addresses the above implementation issues by:

- We implement a testchip of *Design-Dependent Ring-Oscillator* (DDRO), a systematic way of designing and leveraging *multiple* replica monitors. The silicon measurement results show that using multiple DDROs can reduce the mean delay estimation errors by up to 35% (from 2.5% to 1.5%) compared to using generic inverter-based ring-oscillator.
- We implement a testbed *RedCooper* based on our testchip, which offers the sensing infrastructure, and more importantly, capability of exposing the sensor readings to the software.
- We demonstrate an adaptive duty-cycling application based on a embedded operating system running on *RedCooper* testbed. Our demonstration shows that the hardware and software adaptation can achieve up to 2.7X total active time increase for lifetime energy constrained, as compared to sensorless system.

The rest of the paper is organized as follows: Section II describes our implementation of DDRO testchip and present the silicon measurement results. Section III describes the variability-aware software duty-cycling methodology. Section IV explains our board implementation of the *RedCooper* testbed. Section V presents our software implementation and the demonstration of application running on the testbed with both hardware and software adaptations. We conclude the paper in Section VI.

II. DDRO Testchip Implementation and Results

There are in general two classes of performance monitors: in situ monitors and replica monitors. In situ monitors directly probe/measure the delay of the actual circuit paths. Thus, implementation of in situ monitors requires changes in the original design. Conversely, replica monitors are stand-alone circuits which are designed to mimic the delay behavior of the original circuit under variations. They are in general non-intrusive, but with limited accuracy. [3]

To accurately estimate the circuit delay, replica monitors should be designed with respect to the timing behavior of the actual circuits. We know that the circuit's delay is determined by delay of the slowest path, i.e., critical path. Due to variations and different path structures, there are a number of potential critical paths and they might behave differently under variations. The motivation of DDRO is shown in Fig. 1, in which each dot represents the delta delay of one critical path under variations of supply voltage (y-axis) and temperature (x-axis). Since the path delay sensitivities form natural clusters, multiple DDROs can be designed to match these clusters correspondingly. A final delay estimation is made by leveraging all DDROs' results. Detailed design and estimation methods are described in [3]. A flow chart illustration of DDRO design methodology is shown in Fig. 2.

To validate the DDRO-based performance monitoring, we implement a testchip using a 45nm IBM SOI technology with dual-V_{th} libraries. The testchip has an ARM Cortex-M3 microprocessor with DDROs. Five DDROs are synthesized using the method described in [3].

To control DDRO oscillation, a NAND (or AND) gate is added in each RO as shown in the schematic in Fig. 4. An on-chip digital counter is used to obtain the RO frequencies. For comparison, we also implemented inverter-based ROs. The testchip layout and die photo are shown in Fig. 3.

The measurement of the processor F_{max} is done through running a test program. Since ROs are of different length and nominal delay, we normalized them across all 15 chips, e.g., if RO1 delay on chip A is 10% slower than mean delay of RO1 across all chips, it will estimate the processor delay to be 10% slower than mean delay of processors across all chips. The testchip measurement results are shown in Fig. 5. The silicon measurement results show that using multiple DDROs can reduce the mean delay estimation errors by up to 35% (from 2.5% to 1.5%) compared to using generic inverter-based ROs.

III. Variability-Aware Software Duty-Cycling

For battery-powered embedded sensing system, the total lifetime energy is usually constrained. In order to meet the lifetime requirements, one particularly common power management techniques is duty cycling, where the system is at default in a sleep state and woken up periodically to attend to pending tasks and events. A system with higher duty cycle may, for example, sample sensors for longer intervals or at higher rates, increasing data quality. A typical application-level goal is to maximize quality of data through higher duty cycles,

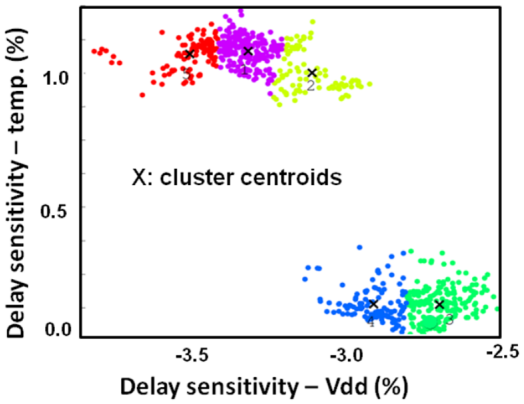

Fig. 1. DDRO Motivation: each dot represents delta delay of a critical path under variations. The critical path delay sensitivities form natural clusters. The paths are extracted from Cortex-M3 microprocessor. Delay values are simulated using SPICE.

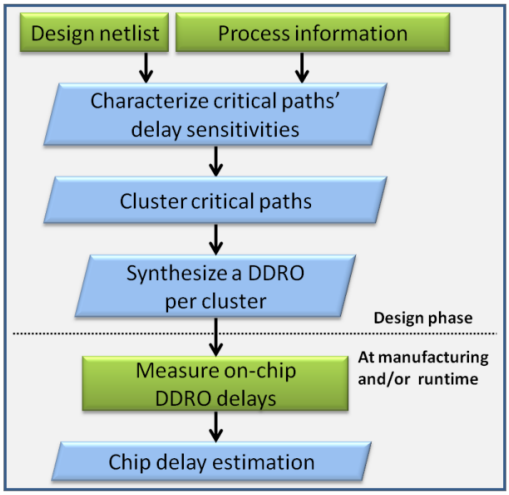

Fig. 2. Overview of DDRO design methodology.

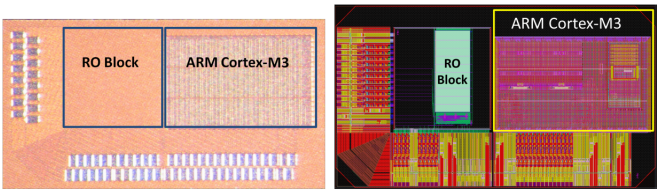

Fig. 3. DDRO Testchip die photo (left) and layout illustration (right).

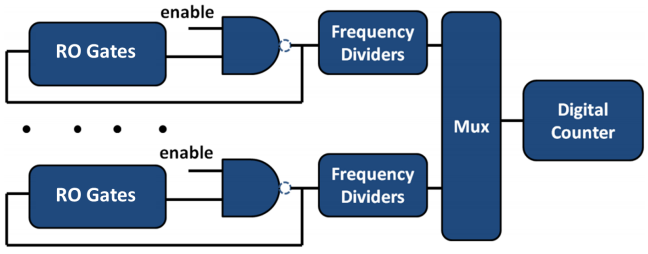

Fig. 4. RO schematics for the RO block. Each RO is connected to a NAND/AND gate followed by a 12-stage frequency divider. A digital counter is used to measure the RO frequency.

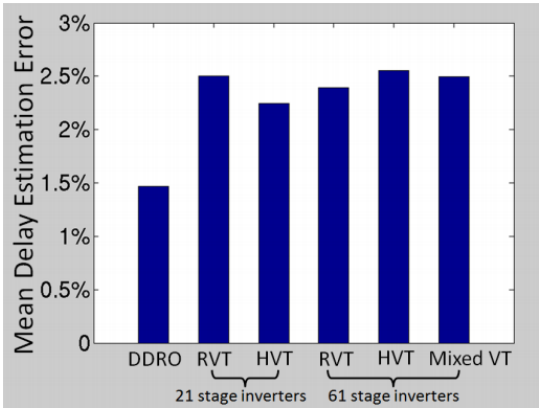

Fig. 5. DDRO Results: RVT stands for RO using regular V_{th} inverters and HVT stands for RO using high V_{th} inverters. Mixed VT stands for RO using both type of inverters.

while meeting a lifetime goal. Conventional approach determines duty cycles by either worst-case power specifications or datasheet power values, which may be heavily guardbanded. If the system's power consumption can be measured and exposed to the software, the application may be able to adapt its duty cycle rate and increase its QoS opportunistically according to the hardware power consumption status.

For example, for a fixed lifetime energy budget E and specified targeting lifetime constraints L, the software duty cycle rate DC can be calculated through the following equation:

$$\begin{aligned} P_A \cdot DC + P_S \cdot (1 - DC) &= \frac{E}{L} \\ DC &= \frac{E - L \cdot P_S}{L \cdot (P_A - P_S)} \end{aligned} \tag{1}$$

where P_A and P_S are the active and sleep power consumption respectively. By determining P_A and P_S on a per-instance basis, the duty cycle may be tailored to maximize active time for each individual sensor under a given deployment scenario (temperature profile, lifetime requirement, battery capacity).

There are several different ways that such an opportunistic stack may be organized as shown in Fig. 6. The scenarios differ in how the sense-and-adapt functionality is split between applications and the operating system. Scenario 1 relies on the application polling the hardware for its current variability signature. In the second scenario, the application handles variability events generated by the operating system. In the last scenario, handling of variability is largely offloaded to the operating system.

IV. *RedCooper* Testbed

In this section, we describe our implementation of our testbed that uses DDRO testchip and on-board senors to enable the demonstration of variability-aware hardware and software adaptation. The overall block diagram of the testbed is shown in Fig. 8.

On our testchip, there are on-chip performance sensors (DDRO) and leakage sensors. For the purpose of demonstrating software duty-cycling, we also implement on-board precision resistors to measure the current drawn by the processor and the (on-chip) SRAM memory separately.

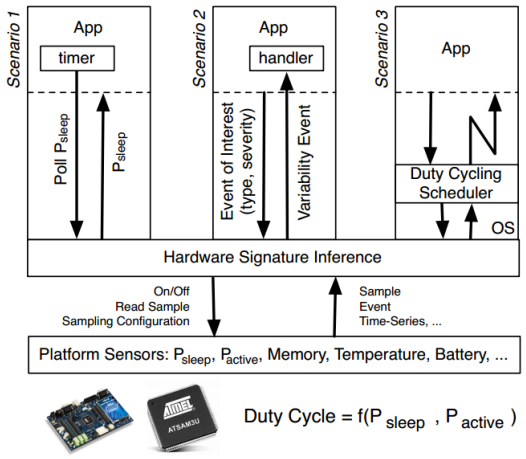

Fig. 6. Designing a software stack for variability-aware duty cycling

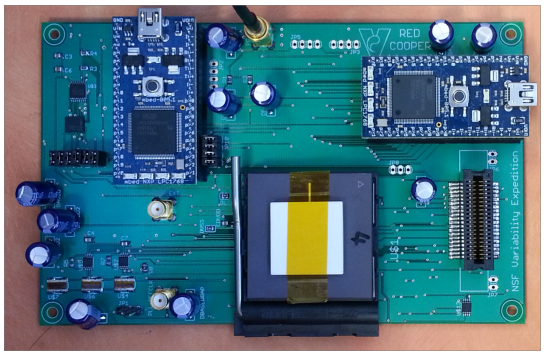

Fig. 7. *RedCooper* Testbed.

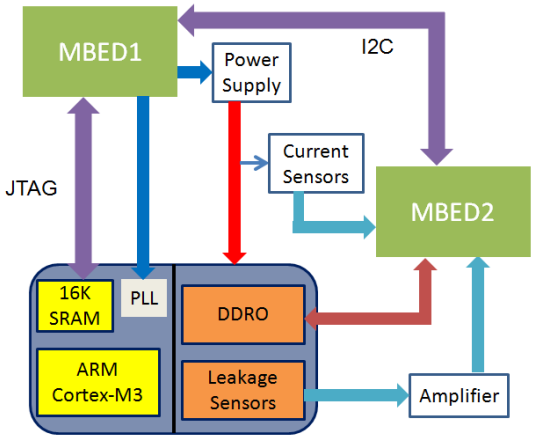

Fig. 8. Block Diagram of *RedCooper* Testbed.

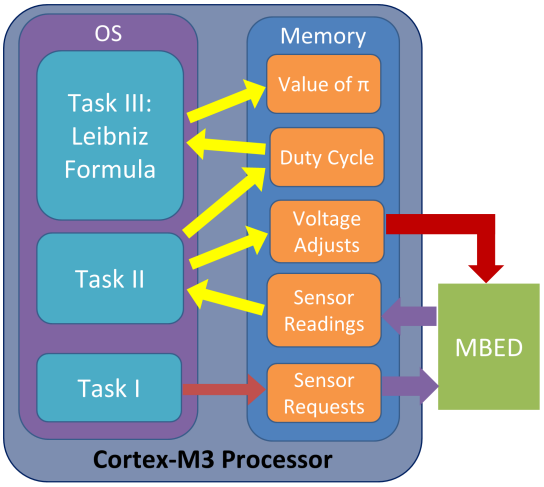

Fig. 9. Software Adaptation Illustration.

```
OS Task 1 {
  if (sensor_request>1) {
    sensor_request = 0
  }
}
OS Task 2 {
  if (sensor_request==1) {
    update(&duty_cycle,
           voltage_adjust,
           sensor_readings)
    sensor_request = 2
  }
}
OS Task 3 {
  acc = 0
  for i=1:duty_cycle {
    acc += (2*(i%2)-1) * 1/(2*i-1)
  }
}
```

```
MBED1 main{
  req = jtag_read(sensor_request)
  vol = jtag_read(voltage_adjust)
  if (req==0) {
    voltage_adapt(vol)
    reqest_i2c();
    reading = read_i2c();
    jtag_write(sensor_readings,
               reading)
    jtag_write(sensor_request,1)
  }
}
MBED2 main{
  while(1) {
    if ( check_request() ) {
      reading = pull_sensor()
      write_i2c(reading)
    }
  }
}
```

Fig. 10. Pseudo code showing programs running on the processor and mbeds.

Due to testchip implementation issues, there are two major challenges in the implementation of the testbed:

1) They only way of off-chip communication is through the JTAG interface (i.e., memory read/write)
2) the on-chip sensors (including DDRO and leakage sensors) are implemented as a self-contained block, therefore cannot be directed exposed to the processor.

To address these issues, we put two mbed MCUs [4] on the board. One MCU (MBED2) is used to control both on-chip and off-chip sensors and transfer the sensor readings upon sensing request. The other MCU (MBED1) is used to accept the actions requested by the processor, including adjusting operating frequency/voltage, requesting for sensor readings, writing sensor readings to certain memory locations, and acknowledging sensor reading updates. A photo of the testbed board is shown in Fig. 7.

V. Adaptation Demonstration

In this section, we will describe our implementation of the software running on *RedCooper* testbed that demonstrate the use of hardware sensors for hardware and software adaptation.

The software running on the M3 core is shown in Fig. 9. The operating system (OS) is based on CoOS [5]. Three tasks

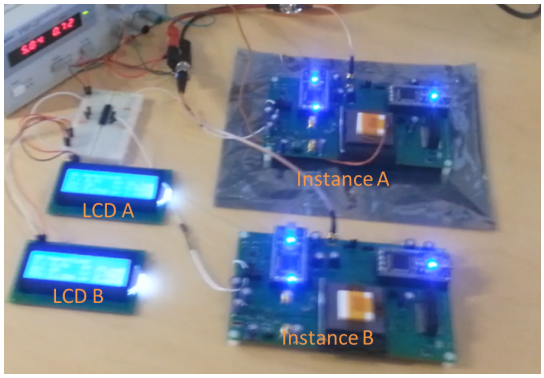

Fig. 11. A snapshot of the demo hardware.

are implemented within the OS:

- Task I sends the sensing request by writing to a pre-specified memory location. Upon seeing the request, the on-board MCUs will start reading the sensor values and write the sensor readings directly to certain memory locations.
- Task II acts as the central adaptation center, which reads the sensor readings, including DDRO frequencies, current drawn by the M3 core, current drawn by the on-chip SRAM, and the on-chip leakage sensor. DDRO frequencies are used to calculate the performance slack and determine the adjusted voltage. The current and leakage sensor values are used to calculate the feasible duty cycle using Equation (1). The duty cycle rate is further translate to the number of iterations for Task III.
- Task III is the main application running in the OS, which calculates the value of pi with its best effort under the constrained duty cycle.

The pseudo code of the OS tasks and MBEDs are shown in Fig. 10. In this demo, all three OS tasks are fired every 10 seconds. We set the processor to run at a fixed 600 MHz clock frequency. The active power includes both the core and SRAM power consumption. The sleep power includes the SRAM power and the projected leakage power of the core.

A snapshot of the entire hardware is shown in Fig. 11. We have two copies of *RedCooper* (i.e., Instance A and Instance B) running side-by-side. Each testbed is equipped with an LCD reporting the system status. The demo results is highlighted with the snapshot of the LCD as in Fig. 12 and 13. [1] The LCD reports current system status, including supply voltage (V), current drawn by processor core (A), current drawn by SRAM plus the projected leakage current from on-chip leakage sensor (L), current number of iterations for Task III (DC), calculated value of pi (PI), and error of calculation in percentage (e).

Fig. 12 highlights some results at room temperature. Instance B has smaller sleep power than Instance A, which implies the potential of achieving high duty cycle rate. Therefore the number of iterations is set at 49 and the calculation error is smaller than that of the Instance A.

[1]The complete demo video can be found at http://nanocad.ee.ucla.edu/Main/Codesign

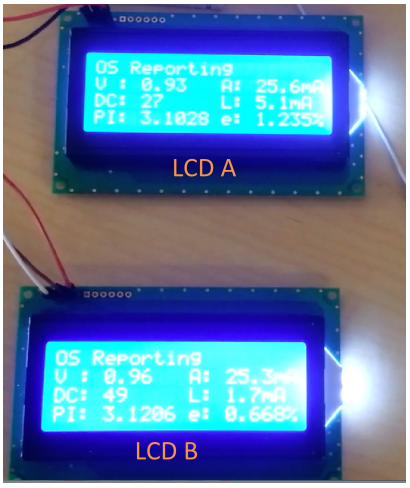

Fig. 12. A snapshot of the demo results under room temperature.

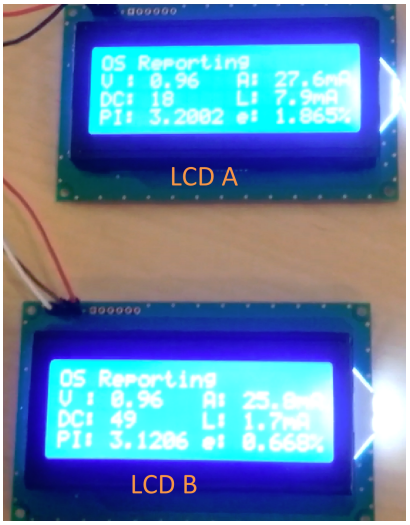

Fig. 13. A snapshot of the demo results after heating up Instance A (reported by LCD A). If the system is "designed for worst case", it has to margin for scenario of Instance A and set $DC = 18$. While our system shows its capability of reducing the margin and run at $DC = 49$, as the scenario of Instance B.

Fig. 13 highlights some results after heating up the Instance A. At higher temperature, Instance A shows its capability to adapt its supply voltage based on the performance sensors. Higher temperature and supply voltage increases Instance A's power consumption significantly, especially the sleep power. Instance A is able to adapt to this change dynamically and adjust its software duty cycle to meet the lifetime constraints. If the system is without hardware sensors and designed for the worst-case scenario (including process variations and temperature fluctuations), the number of iterations, as in this demo, will be at most 18 (the case for Instance A in Fig. 13) with 1.865% calculation error. With the hardware sensors and adaptations, we are able to achieve 49 iterations and calculation error as small as 0.668% (the case for Instance B).

VI. Conclusion

In this paper, we first described out testchip implementation of DDRO and present the testchip measurement results. The silicon measurement results show that using multiple DDROs can reduce the mean delay estimation errors by up to 35% (from 2.5% to 1.5%) compared to using generic inverter-based ring-oscillator. Then we described our implementation of the *RedCooper* testbed using DDRO testchip. We demonstrated a variability-aware system, which utilized hardware performance and power sensors, exposes them to the software level, and reduce the design margin through hardware and software adaptations. By utilizing the sensing infrastructure, our demonstration shows that the hardware and software adaptation can achieve up to 2.7X total active time increase for lifetime energy constrained, as compared to sensorless system.

References

[1] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava *et al.*, "Underdesigned and opportunistic computing in presence of hardware variability," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 1, pp. 8–23, 2013.

[2] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava, "Hardware variability-aware duty cycling for embedded sensors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 1000–1012, 2013.

[3] T.-B. Chan, P. Gupta, A. B. Kahng, and L. Lai, "Ddro: A novel performance monitoring methodology based on design-dependent ring oscillators," in *Quality Electronic Design (ISQED), 2012 13th International Symposium on*. IEEE, 2012, pp. 633–640.

[4] [Online]. Available: http://mbed.org/

[5] [Online]. Available: http://www.coocox.org/CoOS.htm