Accurate and Inexpensive Performance Monitoring for Variability-Aware Systems

Liangzhen Lai

Dept. of Electrical Engineering
University of California Los Angeles
Los Angeles, CA 90095
e-mail: liangzhen@ucla.edu

Puneet Gupta

Dept. of Electrical Engineering
University of California Los Angeles
Los Angeles, CA 90095
e-mail: puneet@ee.ucla.edu

***Abstract*—Designing reliable integrated systems has become a major challenge with shrinking geometries, increasing fault rates and devices which age substantially in their usage life. The proposed research is motivated by the observation that many of the in-field failures are delay failures and several variability signatures are also delay-related. The origins of temporal delay fluctuations include manufacturing variability, voltage/temperature changes, negative or positive bias temperature instability-related Vth degradation, etc. Since the actual delay changes depend on process variations as well as workload, on-chip monitoring may be the best way of predicting them. There is a need to monitor circuit performance during manufacturing as well as at runtime to predict achievable performance and warn against impending failures. Adaptive mechanisms in hardware and/or software can optimize the trade-off between errors, energy and performance based on the feedback from runtime circuit performance monitors.**

This paper presents approaches for automated synthesis of design-dependent performance monitors. These monitors can be used to predict impending delay failures relatively inexpensively. For low-overhead monitoring, we propose multiple design-dependent ring oscillators (*DDROs*) as smart canary structures which can reliably predict achievable chip frequency but with margins for local variations. Early silicon results indicate that *DDROs* can reduce delay monitoring error by 35% compared to conventional ring oscillators. To further improve the prediction (albeit at a higher overhead), we propose in-situ slack monitors (*SlackProbe*) which can match local variations as well at overheads much smaller than monitoring all sequential elements. *SlackProbe* reduces the number of monitors required by over 15X with 5% additional delay margin in several commercial processor benchmarks. Finally, we show an example of software testbed that demonstrates a variability-aware system that utilizes the hardware monitors and operates with both hardware and software adaptation.

I. Introduction

With CMOS technology scaling, hardware variability continues to increase due to increasing amounts of manufacturing variability, ambient fluctuation, and circuit wear-out (e.g., NBTI, HCI etc.) degradation. These increased variations have led to increased margining in designing reliable integrated systems. If these variations can be captured and exposed to the software/system level during runtime, corresponding hardware and software adaptation can opportunistically reduce or even eliminate the design margin [1].

There are different approaches to capture hardware variability at runtime. On-line self-test and diagnostics allow a system to test itself concurrently during normal operation [2]–[4]. Software-based inference methods [5]–[7] use software-implemented test operations to capture variation and detect errors. Other than testing approaches, hardware monitors can also be used to capture variations. In this work, we focus on designing and utilizing hardware monitors to capture circuit performance variations. The proposed research is motivated by the observation that many of the in-field failures are delay failures and several variability signatures are also delay-related.

There are mainly two classes of monitors that aim at monitoring circuit performance (i.e., measuring circuit critical path delay). They are replica monitors and in situ monitors.

Replica monitors, also known as canary circuits, [8]–[11] are stand-alone circuits which are designed to mimic the timing behavior of the original circuits. By measuring the replica circuit delay, one can estimate the delay of the original circuits. Typically, the replicas are simple circuits, e.g., simple paths or ring oscillators. Therefore, replicas are usually non-intrusive and with low overhead, but hard to cover the heterogeneity within the original circuits. Furthermore, replica monitors may fail to capture the variation that are local to real circuits such as random manufacturing variations and circuit aging.

In situ monitors measure the delay directly from the circuit paths [12]–[14].They can accurately capture the real path delay, but with significant overhead, especially when a large number of registers are timing critical.

In this work, we first introduce two circuit performance methodologies. Design-dependent ring oscillator (*DDRO*) [15] designs and leverages *multiple* replicas to accurately monitor circuit performance. *SlackProbe* [16] inserts in situ monitors at both path intermediate nets and path endpoints, which can greatly reduce the overhead of in situ monitoring. Then we present our implementation of an end-to-end variability-aware system [17] which utilizes the hardware monitors and operates with both hardware and software adaptation.

The rest of the paper is organized as follows: Section II presents *DDRO* replica monitoring methodology. Section III presents *SlackProbe* in situ monitoring methodology. The variability-aware system implementation and demonstration is described in Section IV, and Section V concludes the paper.

II. Design-Dependent Ring Oscillator (*DDRO*)

A. Motivation and Overview

In order to accurately estimate the circuit delay, replica monitors should be designed to follow the timing behavior of original circuits under variations. We know that the circuit's performance is determined by delay of the slowest path, i.e., critical path. Therefore, the replica should be designed to follow the timing behavior of the critical path.

Due to variations, there may be a number of potential critical paths and they may behave differently under variations, which makes a single replica monitor inadequate. The motivation of *DDRO* is shown in Fig. 1, in which each dot represents the delta delay of one critical path under variations of PMOS threshold voltage $Vthp$ (y-axis) and NMOS threshold voltage $Vthn$ (x-axis). The critical path delay sensitivities form natural

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

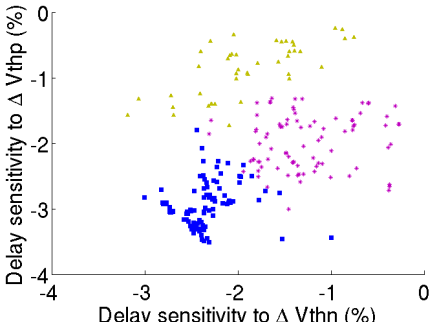

Fig. 1. Every dot represents delay sensitivities of a critical path in a design. In this example, we cluster the paths into 3 different clusters indicated by different colors. The results are based on SPICE simulation using commercial 45nm process technology. Critical paths are extracted from AES [18].

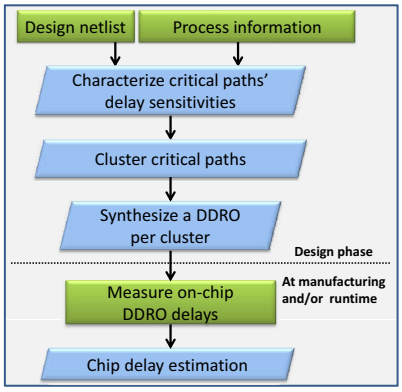

Fig. 2. Overview of *DDRO* design methodology.

clusters, which implies that *multiple* replica monitors can be designed to cover these clusters.

An overview of *DDRO* monitoring strategy is shown in Fig. 2. First, we extract critical paths of a design and characterize their delay sensitivities to variation sources. Second, we cluster the critical paths based on the sensitivities, and synthesize *DDROs* to match delay sensitivity of the clusters. By matching *DDRO* and cluster delay sensitivities, we ensure that the synthesized *DDROs* have good correlation with the critical paths. Since we use only standard cells (gates) to synthesize the *DDROs*, the design and placement of *DDROs* can be easily integrated with conventional implementation flows. Based on *DDRO* frequencies, we can estimate chip delay during manufacturing or runtime.

B. Path Sensitivity Extraction and Clustering

Delay sensitivity of path i (V_i^{path}) can be obtained using finite differences, i.e., taking the $\Delta delay$ by perturbating each variation source by 1σ. After characterizing all path delay sensitivities, we can cluster the critical paths. The objective function of the clustering is defined as

$$\text{minimize} \sum_{i=1}^{N} (w_i \times |\mathbf{V}_i^{path} - \mathbf{V}_k^{ro}|) \quad (1)$$

where the summation is taken over paths i in cluster k, and w_i is the probability of a critical path delay exceeding the clock period of the design. The weight factor w_i is added so that we can impose a higher penalty for having mismatched delay sensitivities on a path with higher probability to fail (less timing slack). Detailed derivation of w_i can be found in [15].

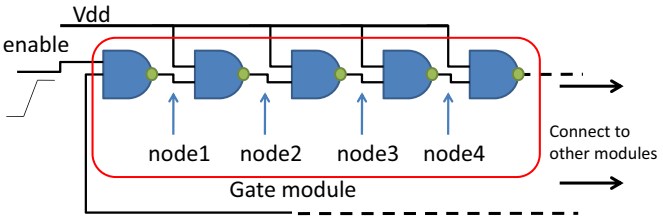

Fig. 3. Illustration of a gate module in a *DDRO*.

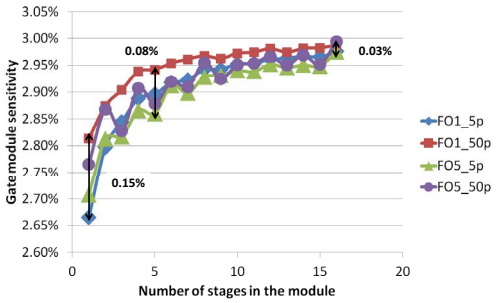

Fig. 4. Simulation results show that the sensitivities under different input slews {5ps, 50ps} and output loads {FO1, FO5} combinations converge as the number of stages in a gate module increases.

C. DDRO Synthesis

1) Gate Module: For the ease of synthesizing *DDRO*, we use gate module as basic building block. A gate module is constructed as several identical gates connected in series as illustrated in Fig. 3.

This repeated pattern can help decouple the load and slew interaction between gates. Simulation results in Fig. 4 show that the sensitivity difference due to different input slew and output load is reduced from 0.15% to 0.03%, as the number of stages in a gate module increases from 1 to 15. In this work, we use 5-stage gate modules as a trade-off between stability of sensitivity and total area of a gate module. For a gate with multiple input pins, gate delays through different input pins will have different delay sensitivities. Thus, each gate module type is defined with respect to a specific input pin. Extra input pins of a multi-input gate are assigned to high or low to make a gate module inverting or buffering (see Fig. 3).

Since the interconnect also affects path delay sensitivity, we use different wirelength in building our gate modules. Gate modules with different wirelength are considered as different instance types even if they have the same gate type.

2) ILP-based Synthesis: Given a delay sensitivity target ($\mathbf{V}^{ro}$) obtained from path clustering, we want to choose the number of each gate module in a *DDRO*, so that the delay sensitivities of the *DDRO* match the targeted delay sensitivities. Because of its unique structure, gate module delay variation is less sensitive to input slew and output load. Since each gate module type is instantiated a discrete number of times, we can formulate *DDRO* synthesis as an integer linear programming (ILP) problem, where the integer variable is the number of certain gate modules in the synthesized *DDRO*. Appropriate constraints are applied to limit the maximum RO length. Detailed description of gate module and ILP formulation can be found in [15].

D. Delay Estimation

1) Path-based Estimation: As shown in Fig. 2, at manufacturing and/or runtime, *DDRO* delay is measured and used

to estimate chip delay.

Each critical path delay can be estimated by *DDROs*. Given M *DDROs*, we can represent delay sensitivity of each path ($\mathbf{V}_i^{path}$) as a linear combination of *DDRO* delay sensitivity($\mathbf{V}_k^{ro}$ $(k = 1, ..., M)$) as in Equation (2).

$$\mathbf{V}_i^{path} = \sum_{k=1}^{M} b_{ik} \cdot \mathbf{V}_k^{ro} + \mathbf{V}_i^{res\_path} \tag{2}$$

where b_{ik} is a $[1 \times M]$ matrix containing constant coefficients and $\mathbf{V}_i^{res\_path}$ is a $[1 \times Q]$ matrix that represents the decomposition residue.

Using b_{ik}, we can estimate the path delay by:

$$d_i^{path} = d_i^{nom\_path}(1 + \sum_{k=1}^{M} \overbrace{(b_{ik}\mathbf{V}_k^{ro} \cdot \mathbf{g})}^{\text{measurable}} + \underbrace{u_i}_{\text{uncertainty}}) \tag{3}$$

$$\text{where} \quad u_i = l_i^{path} + \mathbf{V}_i^{res\_path} \cdot \mathbf{g}$$

where $\mathbf{g}$ is global variation vector, l_i^{path} is the local variation of the path. Equation (3) shows that d_i^{path} consists of a measurable term and an uncertainty term. While the value of the *measurable* term can be determined from the delays of *DDROs*, the value of the *uncertainty* term cannot be measured directly.

2) Cluster-based Estimation: The path-based estimation method requires one estimation for each critical path, which can cause significant computation and storage overhead. To reduce the overhead, we propose to utilize the clustering nature of critical paths and to have one estimation for each path cluster. We calculate the maximum delay of paths in each cluster using the method in [19], assuming that the means of path delays correspond to their nominal values. The outcome of this step gives us the expected maximum delay. But more importantly, it also extracts the sensitivity of the maximum delay to variation sources ($\mathbf{V}^{max}$). Similar to the path-based approach, we treat the cluster as a pseudo path with delay sensitivity $\mathbf{V}^{max}$. Simulation results show that the estimation error of this approximation approach compared to the reference method is very small.

E. Simulation and Silicon Result Highlights

To validate *DDRO* performance monitoring methodology, we synthesized, placed and routed processor benchmark circuits using a commercial 45nm SOI technology. Some simulation results of *DDRO* on Cortex-M0 [20] are highlighted in Fig. 5. For global variation only case, using more *DDROs* can dramatically decrease mean overestimation (i.e., required delay margin). But this benefit becomes less in presence of local variation. The results show that the estimation error of cluster-based approximation, compared to the path-based approach, is very small.

A testchip was taped out with *DDRO*-based performance monitoring using a 45nm IBM SOI technology with dual-V_{th} (RVT and HVT) libraries. The testchip has an ARM Cortex-M3 microprocessor [21] with five *DDROs*. For comparison, we also implemented several inverter-based ROs in the testchip. The silicon measurement results are shown in Fig. 7. The measurement results show that by using five *DDROs*, we can reduce the mean delay estimation error by 35% (from 2.3% to 1.5%) compared to generic inverter-based ROs.

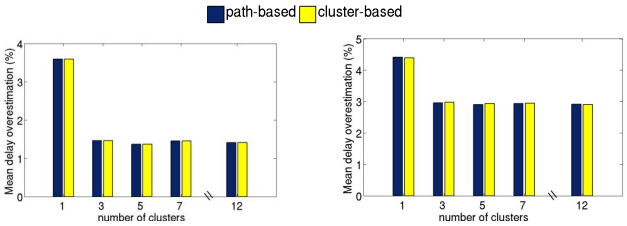

Fig. 5. Simulation results of *DDRO* on Cortex-M0 with global variation only (left) and both global and local variations (right).

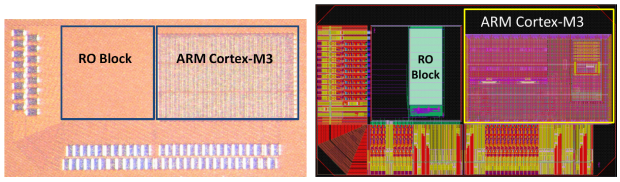

Fig. 6. Testchip die photo and layout illustration

III. *SlackProbe*: An Efficient and Flexible In Situ Monitoring Methodology

With *DDRO* methodology, replica monitors can predict global variation near perfectly. But the accuracy is still limited and constrained by local variation. In situ monitors can capture the actual path delay, including local variation, but usually incur significant overhead. We observe that most of existing work focuses on monitoring destination registers. In this section, we introduce a novel and flexible in situ monitoring methodology, *SlackProbe*, which inserts monitors at both path endpoints and path intermediate nets.

A. Monitor Working Principle and Overview

The monitor working principle is shown through an example in Fig. 8. If a monitor is inserted at an intermediate node *A*, a "probe", which consists of delay matching gates and a transition detector, is connected to *A* through a minimum size inverter. Signal transitions at node *A* are transferred through the delay chain to the transition detector and compared with incoming clock edge. If the transition is close to its required arrival time (RAT), i.e., within the margin window as in Fig. 8, a corresponding signal transition will arrive at node *E* after the clock edge. This triggers the transition detector and flags a signal indicating an impending delay failure.

The monitor inserted at node *A* is capable to monitor the delay of all critical paths passing through *A*. As shown in Fig. 8, in stead of monitoring all four destination registers,

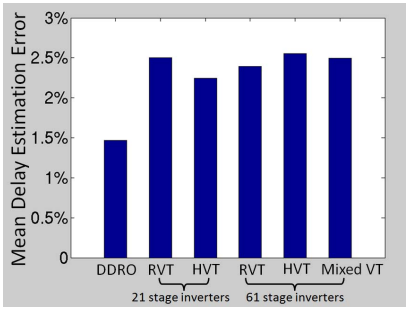

Fig. 7. Mean delay estimation error obtained from *DDROs* and inverter-based ROs. Estimation errors are calculated by taking the absolute difference between normalized estimation and normalized chip delay. RVT and HVT are the two Vt options for the inverters.

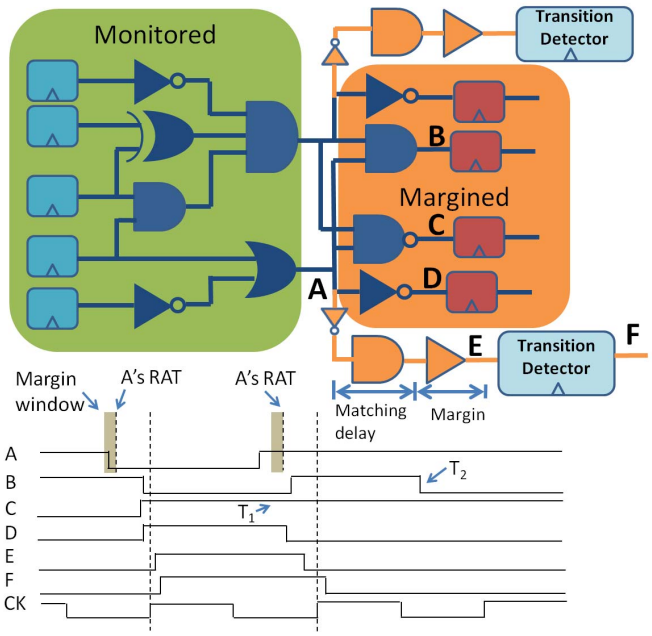

Fig. 8. *SlackProbe* working principle. As shown in the timing diagram, compared to inserting monitors at destination registers, the monitor inserted at A can monitor the path delay even when the transition does not propagate to the destination register (i.e., T_1 at C). But the monitor inserted at node A cannot capture transitions that do not pass through A (i.e., T_2 at B).

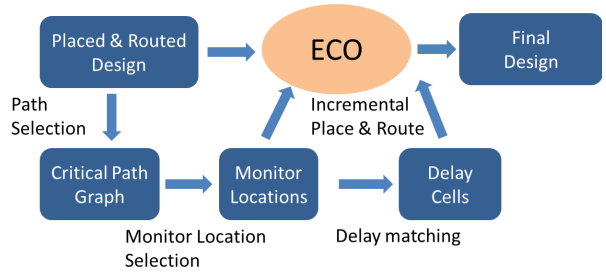

Fig. 9. Monitor insertion flow

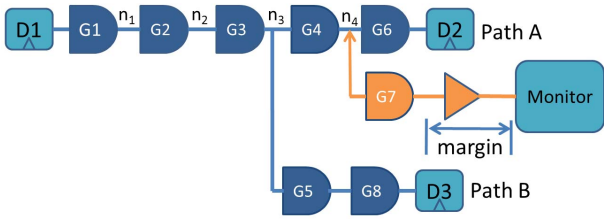

Fig. 10. Example of path-monitor pairs

SlackProbe can use only two monitors while achieving the same path coverage.

Different transition detector designs as in [14], [22]. can be applied here. *SlackProbe* also allows monitors to be inserted at path endpoints where monitors as in [23]–[26] can be used as well. Since the additional margin makes the monitor detect an impending timing failure rather than an actual one, there is no datapath metastability issue as raised and discussed in [14]. The metastability issue of the monitor signal either results in a more pessimistic detection or is guardbanded by the monitor delay margin.

With the proposed monitoring strategy, the problem now becomes *when*, *where* and *how* to insert these monitors. In this work, we propose the monitor insertion flow as in Fig. 9. Different alternatives will also be discussed and compared against conventional approaches.

Monitor insertion starts with a placed and routed design, as the timing information is more accurate at this stage. Since we only care about timing-critical paths, a path selection process is applied to extract timing-critical paths and to construct the critical path graph. Then, monitor locations are picked from the graph using our proposed method. For each of the monitor locations, a delay cell path is synthesized. The final insertion flow is similar to Engineering Change Order (ECO) where the monitors are incrementally placed and routed. ECO metrics like those in [27] are applied when picking monitor locations to minimize the interference to the original design.

B. Monitor Delay Margin

If the monitor is placed at a path intermediate net, the path delay before the monitor can be captured. But some extra delay margin will be required for the remaining part of the path. As shown in Fig. 10, there are three types of relations between monitor and a path:

1) The path passes through the net, for example path A in Fig. 10. Since the delay up to G4 can be captured by the monitor, the delay path should account for the delay uncertainty of G6.
2) The path branches out at some net before the monitor, for example path B in Fig. 10. Depending on the application and gate type of G4, the monitor may be treated as being inserted between G3 and G5 with G4 as part of the delay matching. If the application is speed sensing, i.e., monitoring slow delay changes, path B can be considered as being monitored with delay uncertainty of G4, G5 and G8. If the application is event detection, only G4 with gate types that are transparent to signal transitions (e.g., inverter, buffer) are allowed.
3) None of the path instances fall in the fan-in cone of the monitor net. In this case, we consider that the path is not monitored.

Though different paths may require different delay margin, each monitor will have only one margin matching chain. The monitor margin should account for worst delay uncertainty after monitor insertion point and guarantee that the delay chain is always slower than margined part of monitored circuit paths.

In the example in Fig. 10, the best case delay of the delay chain (i.e., n_4 to the monitor) should match the worst case delay of the original path (i.e., $G6$). But this may be too pessimistic since the delay is likely to be correlated. Similar to on-chip variation modeling, in this work the delay chain is designed so that its delay at typical process corner matches the worst case delay of the original path. The equivalent delay margin in this case equals the delay of $G6$ at slow process corner (i.e., delay of the delay chain at typical process corner) minus the delay of $G6$ at typical process corner. This margin is considered as the delay uncertainty of $G6$.

The final delay margin for the entire circuit will be dominated by the monitor with the largest margin. Therefore, in this work, we define the delay margin cost as the maximum monitor delay margin constraint ϵ on each monitor. For a given ϵ, we can identify the feasible monitor candidate locations. Larger ϵ will give more flexibility in choosing monitor insertion location, which can potentially reduce the number of monitors required. But it will also reduce the monitoring benefit. The trade-off of delay margin ϵ will be discussed together with critical path selection in Section III-C.

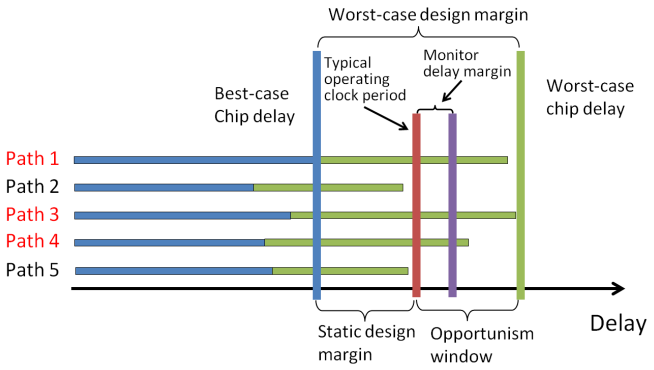

Fig. 11. Opportunism window is the margin saving comparing to worst-case design. Both monitoring benefit and overhead is affected by opportunism window size and monitor delay margin size.

C. Opportunism Window

As shown in Fig. 9, given a placed and routed design, the first step is to identify the part of the design that may be timing critical. Depending on the application, different criticality criteria may be applied for the selection process.

In this work, we propose a flexible path selection method by introducing user-defined *opportunism window*. As illustrated in Fig. 11, *opportunism window* and monitor margin will dictate the potential monitoring benefits. Typical worst-case design margins for the worst-case scenario, i.e., all chips will by default run at the worst-case chip delay regardless of what their actual delays are. In the presence of monitors, we may reduce the design margin and decrease the default operating clock period. The paths whose worst-case delay exceeds the default operating clock period should be selected and monitored. The amount of design margin reduction is called *opportunism window*, within which the circuit will operate opportunistically at its best effort.

This path selection method does not require any knowledge of correlation in the variations between the delay of different paths. Therefore, it can be used to select paths for applications like aging sensors, where exact delay degradations are context dependent with little pre-assumed correlation.

As illustrated in Fig. 11, there is a natural trade-off between monitoring benefits and monitoring overhead when deciding the *opportunism window* and monitor delay margin (ϵ). Larger opportunism window size and smaller monitor delay (ϵ) will increase the monitoring benefit, but also will increase the monitoring overhead.

After picking the critical paths, monitor location selection can be formulated as a Linear Programming (LP) problem, which resembles a max-flow problem. Detailed problem formulation and solution can be found in [16].

D. Experimental Result Highlights

To evaluate the effectiveness of our monitoring methodology, we apply *SlackProbe* on commercial processor benchmarks using a commercial sub-32nm process technology and libraries. For comparison purpose, three different monitoring methods are implemented:

- Baseline: A monitor is inserted at every critical path endpoint
- *SlackProbe* Event Detection: Monitors are inserted at both path intermediate nets and path endpoints. Monitors are inserted to capture every timing-critical signal switching events.

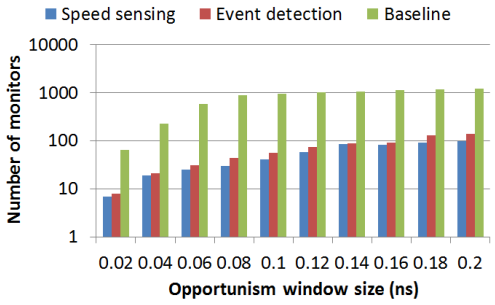

Fig. 12. Monitor count comparison between *SlackProbe* and baseline. The y-axis is plotted in log scale.

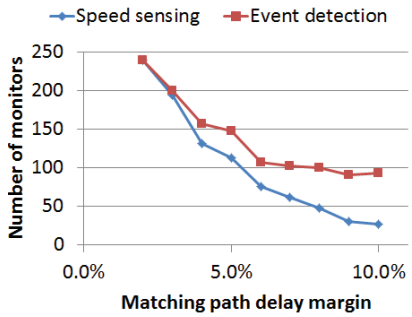

Fig. 13. Monitor count and cost vs. delay margin for processor A

- *SlackProbe* Speed Sensing: Monitors are inserted at both path intermediate nets and path endpoints. Monitor are inserted to capture the delay changes of all critical paths, given sufficient time of operations.

To evaluate our methodology and show the opportunism window trade-off, we implement all three methods on a processor benchmark. *SlackProbe* monitors are inserted with monitor delay margin ϵ as 5% of the clock period. The results are shown in Fig. 12. Compared to the baseline endpoint monitoring, with the 5% additional delay margin, *SlackProbe* can achieve up to 16X reduction in total number of monitors.

To show the trade-off between delay margin and monitor count, we also sweep the delay margin ϵ with opportunism window size of 0.2ns, i.e., equivalently 20% of the clock period. By allowing more delay margin and flexibility in selecting monitor candidate location, the number of monitor reduces for both *SlackProbe* methods (see Fig. 13).

IV. *RedCooper*: A Testbed for Variability-Aware System

In this section, we will present our implementation of a complete end-to-end system [17] that demonstrates the use of hardware monitors with both hardware and software adaptation. We first describe the software adaptation concept of variability-aware software duty-cycling. Then we present our testbed implementation and variability-aware system demonstration.

A. 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

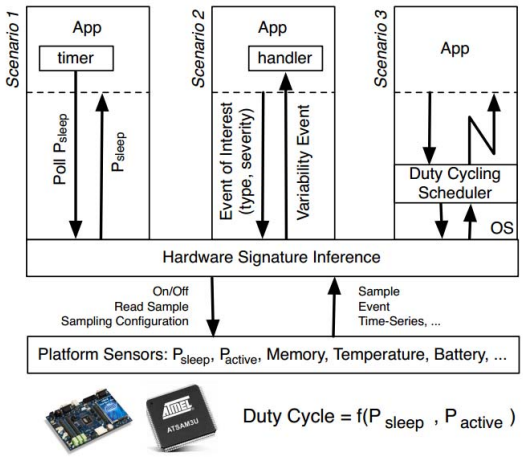

Fig. 14. Designing a software stack for variability-aware duty cycling [28]

higher rates, increasing data quality. A typical application-level goal is to maximize quality of data through higher duty cycles, 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 [28].

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 [28]:

$$\begin{aligned} P_A \cdot DC + P_S \cdot (1 - DC) &= \frac{E}{L} \\ DC &= \frac{E - L \cdot P_S}{L \cdot (P_A - P_S)} \end{aligned} \tag{4}$$

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. 14. The scenarios differ in how the sense-and-adapt functionality is split between applications and the operating system. In this work, we use the last scenario, where variability is largely offloaded to the operating system.

B. Testbed Implementation and System Demonstration

The testbed is built upon *DDRO* testchip (Fig. 6). On the testchip, there are on-chip performance monitors (*DDRO*) and leakage sensors. For the purpose of demonstrating software duty-cycling, we also implement on-board power sensors to measure the power consumption of the testchip. the power of the Cortex-M3 processor and the (on-chip) SRAM memory are measured separately. A picture of the testbed board is shown in Fig. 17. Because *DDRO* and the processor are implemented as separate blocks when taping-out the testchip, we use on-board MCUs [29] to sample the monitor readings and feed into the on-chip SRAM.

The software running on the M3 core is shown in Fig. 16. The operating system (OS) is based on CoOS [30]. Three tasks are implemented within the OS:

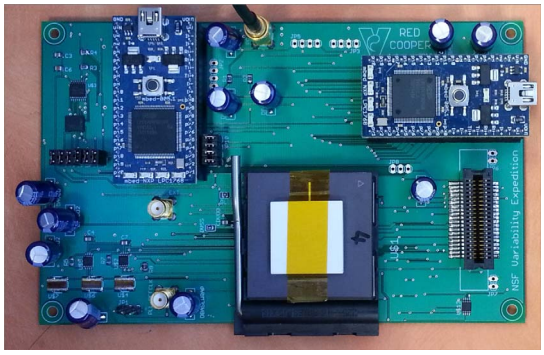

Fig. 15. *RedCooper* Testbed.

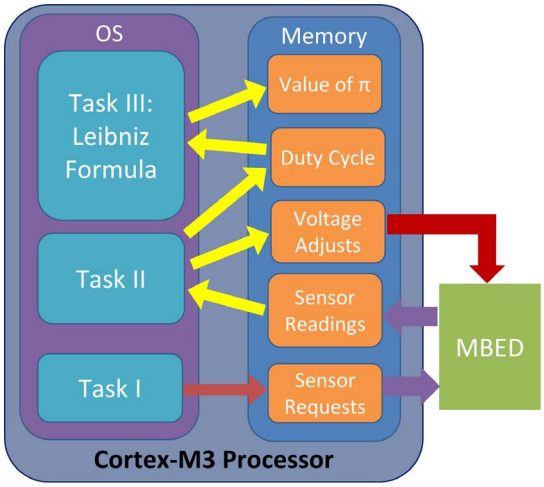

Fig. 16. Software Adaptation Illustration.

- 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 (4). 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 π with its best effort under the constrained duty cycle.

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. 17. We have two copies of *RedCooper* running side-by-side. Each testbed is equipped with an LCD reporting the system status. Some results are highlighted in Table I.[1] At room temperature, Instance B system has smaller sleep power than

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

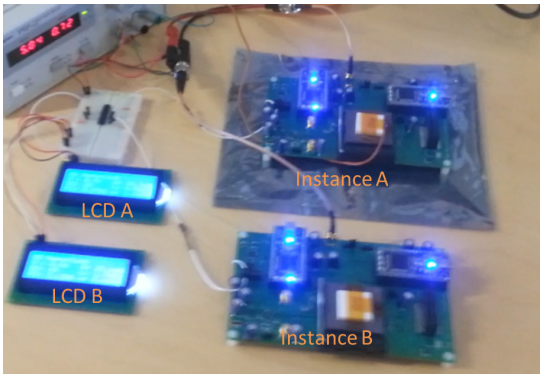

Fig. 17. A snapshot of the demo hardware.

TABLE I
SYSTEM DEMONSTRATION RESULT HIGHLIGHTS

	Instance A (room temperature)	Instance B (room temperature)	Instance A (after heat-up)
Supply Voltage	0.93 V	0.96 V	0.96 V
Active power	23.81 mW	24.29 mW	26.50 mW
Sleep power	4.74 mW	1.63 mW	7.58 mW
DC	27	49	18
Calculated π	3.1028	3.1206	3.2002
Calculation error	1.235%	0.668%	1.865%

Instance A, which implies the potential of achieving high duty cycle rate. Therefore, the adaptive software set the number of iterations (DC) at 49 for Instance B and the calculation error is smaller. After heating up Instance A from room temperature to about $45°C$, the system shows its capability for both runtime software and hardware adaptation (see last column in Table I). The voltage is boosted to compensate the performance loss. Software duty-cycle is reduced to compensate the increased power consumption. If the system is without hardware sensors and designed for the worst-case scenario (including process variations and temperature fluctuations), the number of iterations (DC), as in this demo, will be at most 18 (the case for Instance A after heat-up) 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%.

V. CONCLUSION

In this paper, we first present *DDRO*, an accurate replica performance monitoring methodology. Then we present *SlackProbe*, an efficient and flexible in situ performance monitoring methodology. Last, we demonstrate an end-to-end variability-aware system that utilizes hardware monitors for both hardware and software adaptation.

ACKNOWLEDGMENTS

The work in Section II was jointly done with Tuck-Boon Chan and Andrew B. Kahng. The author would like to thank Vikas Chandra and Robert Aitken for collaboration for work presented in Section III. Finally, *RedCooper* testbed was jointly developed by the authors and Yuvraj Agarwal, Alex Bishop, Matt Fotjik, Paul Martin, Mani Srivastava, Dennis Sylvester, Lucas Wanner, and Bing Zhang.

This work is supported in part by NSF Variability Expedition grant CCF-1029030.

REFERENCES

[1] P. Gupta *et al.*, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2012, Keynote Paper.
[2] Y. Li *et al.*, "Overcoming early-life failure and aging for robust systems," *IEEE Design and Test of Computers*, vol. 26, no. 6, pp. 28–39, 2009.
[3] H. Inoue *et al.*, "Vast: Virtualization-assisted concurrent autonomous self-test," in *Proc. IEEE International Test Conference*. IEEE, 2008, pp. 1–10.
[4] Y. Li *et al.*, "Casp: concurrent autonomous chip self-test using stored test patterns," in *IEEE/ACM Design, Automation and Test in Europe*. ACM, 2008, pp. 885–890.
[5] D. Lin *et al.*, "Quick detection of difficult bugs for effective post-silicon validation," in *Proc. ACM/IEEE Design Automation Conference*. IEEE, 2012, pp. 561–566.
[6] S. K. Sahoo *et al.*, "Using likely program invariants to detect hardware errors," in *Dependable Systems and Networks, IEEE International Conference on*. IEEE, 2008, pp. 70–79.
[7] M.-L. Li *et al.*, "Understanding the propagation of hard errors to software and implications for resilient system design," *ACM Sigplan Notices*, vol. 43, no. 3, pp. 265–276, 2008.
[8] A. Drake *et al.*, "A distributed critical-path timing monitor for a 65nm high-performance microprocessor," in *Proc. IEEE International Solid State Circuits Conference*, feb. 2007.
[9] M. Bhushan *et al.*, "Ring oscillators for cmos process tuning and variability control," *IEEE Transactions on Semiconductor Manufacturing*, vol. 19, no. 1, pp. 10–18, 2006.
[10] Q. Liu *et al.*, "Capturing post-silicon variations using a representative critical path," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 211–222, 2010.
[11] T.-B. Chan *et al.*, "Tunable sensors for process-aware voltage scaling," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*. ACM, 2012, pp. 7–14.
[12] D. Fick *et al.*, "In situ delay-slack monitor for high-performance processors using an all-digital self-calibrating 5ps resolution time-to-digital converter," in *Proc. IEEE International Solid State Circuits Conference*, feb. 2010.
[13] S. Kim *et al.*, "Razor-lite: A side-channel error-detection register for timing-margin recovery in 45nm soi cmos," in *Proc. IEEE International Solid State Circuits Conference*. IEEE, 2013, pp. 264–265.
[14] K. Bowman *et al.*, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE Journal of Solid State Circuits*, jan. 2009.
[15] T.-B. Chan *et al.*, "Synthesis and analysis of design-dependent ring oscillator (ddro) performance monitors," *IEEE Transactions on Very Large Scale Integration Systems*, 2013, accpted for publication.
[16] L. Lai *et al.*, "Slackprobe: A low overhead in situ on-line timing slack monitoring methodology," in *IEEE/ACM Design, Automation and Test in Europe*, 2013, pp. 282–287.
[17] Y. Agarwal *et al.*, "Redcooper: Hardware sensor enabled variability software testbed for lifetime energy constrained application," Tech. Rep., http://nanocad.ee.ucla.edu/pub/Main/Codesign/red_cooper.pdf.
[18] [Online]. Available: http://opencores.org
[19] C. Visweswariah *et al.*, "First-order incremental block-based statistical timing analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2170–2180, 2006.
[20] [Online]. Available: http://www.arm.com/products/processors/cortex-m/cortex-m0.php
[21] [Online]. Available: http://www.arm.com/products/processors/cortex-m/cortex-m3.php
[22] B. Rebaud *et al.*, "Digital timing slack monitors and their specific insertion flow for adaptive compensation of variabilities," in *International Conference on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation*, ser. PATMOS'09, 2010.
[23] D. Ernst *et al.*, "Razor: a low-power pipeline based on circuit-level timing speculation," in *IEEE/ACM International Symposium on Microarchitecture*, dec. 2003.
[24] S. Das *et al.*, "RazorII: In situ error detection and correction for pvt and ser tolerance," *IEEE Journal of Solid State Circuits*, jan. 2009.
[25] H. Fuketa *et al.*, "Adaptive performance compensation with in-situ timing error prediction for subthreshold circuits," in *IEEE Custom Integrated Circuits Conference*, sept. 2009.
[26] M. Eireiner *et al.*, "In-situ delay characterization and local supply voltage adjustment for compensation of local parametric variations," *IEEE Journal of Solid State Circuits*, vol. 42, no. 7, pp. 1583–1592, 2007.
[27] J. Lee *et al.*, "Eco cost measurement and incremental gate sizing for late process changes," *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 1, p. 16, 2012.
[28] L. Wanner *et al.*, "Hardware variability-aware duty cycling for embedded sensors," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 21, no. 6, pp. 1000–1012, 2013.
[29] [Online]. Available: http://mbed.org/
[30] [Online]. Available: http://www.coocox.org/CoOS.htm