VESPA: Variability Emulation for System-on-Chip Performance Analysis

Vivek J. Kozhikkottu, Rangharajan Venkatesan, Anand Raghunathan
School of ECE
Purdue University
{vkozhikk,rvenkate,raghunathan}@purdue.edu

Sujit Dey
Department of ECE
UC San Diego
dey@ece.ucsd.edu

***Abstract*—We address the problem of analyzing the performance of System-on-chip (SoC) architectures in the presence of variations. Existing techniques such as gate-level statistical timing analysis compute the distributions of clock frequencies of SoC components. However, we demonstrate that translating component-level characteristics into a system-level performance distribution is a complex and challenging problem due to the inter-dependencies between components' execution, indirect effects of shared resources, and interactions between multiple system-level "execution paths". We argue that accurate variation-aware system-level performance analysis requires repeated system execution, which is prohibitively slow when based on simulation. Emulation is a widely-used approach to drastically speedup system-level simulation, but it has not been hitherto applied to variation analysis. We describe a framework - *V*ariability *E*mulation for *S*oC *P*erformance *A*nalysis (VESPA) - that adapts and applies emulation to the problem of variation aware SoC performance analysis. The proposed framework consists of three phases: component variability characterization, variation-aware emulation setup, and Monte-carlo driven emulation. We demonstrate the utility of the proposed framework by applying it to design variation-aware architectures for two example SoCs - an 802.11 MAC processor and an MPEG encoder. Our results suggest that variability emulation has great potential to enable variation-aware design and exploration at the system level.**

I. Introduction

Variations in the characteristics of transistors and interconnect are widely regarded as one of the major obstacles to continued scaling of integrated circuits (ICs) [1], [2]. Variation-aware design – the process of understanding the impact of variations on ICs and designing systems that are resilient to them – has therefore emerged as one of the major active research areas in circuits, architecture, and design automation.

Since variations are inherently a bottom-up phenomenon, most efforts on addressing them have understandably focused on the later stages of the design cycle. These include mask-level techniques such as Resolution Enhancement Technologies (RET) and Optical Proximity Correction (OPC), circuit-level techniques such as variation-aware transistor sizing, variation-aware placement and routing, and statistical timing analysis and synthesis at the logic level. Unfortunately, inevitable increases in variations make it difficult to fully contain their effect at the later stages of the design cycle. Recognizing this, there have been several recent efforts on variation-aware design at the architectural and system levels [3]-[17]. These efforts have shown great potential for effectively addressing variations with substantially lower design overhead and effort. However, we believe that adoption of variation-aware design techniques at the system level will require the development of concomitant variation-aware analysis tools. A commonly used paradigm for variation-aware analysis at any level of abstraction is to repeatedly perform simulation in a Monte-Carlo loop, while varying the performance/power characteristics of circuit components in each iteration. While this is the most accurate and general approach, the need to run simulation in an iterative loop drastically reduces efficiency and scalability, limiting the possibilities for design space exploration.

This material is based upon work supported in part by the National Science Foundation under Grant No. 0916117.

978-3-9810801-7-9/DATE11/(c)2011 EDAA

A. Paper Overview and Contributions

In this work, we address the problem of variation-aware system-level performance analysis, and specifically target the challenge of improving efficiency and scalability, while maintaining the generality and accuracy of the iterative simulation paradigm. Emulation is a widely-used approach to drastically speedup system-level simulation, but it has not been hitherto applied to variation-aware performance analysis. We propose an emulation-based framework — **V**ariability **E**mulation for **S**oC **P**erformance **A**nalysis (VESPA) — for analyzing the impact of variations on performance at the system level. The significant contributions of our work are as follows:

- We study the challenges involved in system-level performance analysis under variations. We demonstrate that translating component-level characteristics into a system-level performance distribution is a complex and challenging problem due to the inter-dependencies between components' execution, indirect effects of shared resources, and interactions between multiple system-level "execution paths". We therefore argue that accurate variation-aware system-level performance analysis requires repeated system execution, which is prohibitively slow when based on simulation.
- We describe the VESPA framework that applies emulation to the problem of variation-aware SoC performance analysis. A key attribute of the proposed framework is that mechanisms for adaptation of component frequencies and the control loop for iterative (Monte-Carlo) analysis are embedded within the emulation platform, eliminating the need for re-synthesis of the design or FPGA re-configuration within the iterative loop. The inherent speed

of emulation is therefore fully exploited for variation-aware performance analysis.

- We apply the proposed framework to two example SoCs - an 802.11 MAC processor and an MPEG encoder, and use it to explore variation-aware architectures based on multiple frequency islands. For these SoCs, VESPA is one to two orders of magnitude faster than HW/SW co-simulation, and four orders of magnitude faster than register-transfer level (RTL) simulation. Our results also demonstrate the utility of the proposed framework in driving variation-aware design at the system level.

The rest of the paper is organized as follows. Section II summarizes prior work on variation-aware design at the system level. Section III describes the challenges involved in variation-aware system-level performance analysis. Section IV describes the proposed VESPA framework. Section V presents the application of the framework to the example SoCs, evaluates its benefits, and shows how the proposed framework can be applied to drive variation-aware architectural exploration.

II. Related Work

A large body of work over the last decade has analyzed and addressed the impact of variations on integrated circuits at various levels of abstraction. We describe some representative efforts that focus on the earlier stages of the design cycle. Variation-tolerant design techniques for SoC components, including processors, memories, and on-chip buses, have been proposed [3], [4], [5], [6], [7], [8]. Techniques for optimizing system-level power management policies under variations were presented in [9]. A variation-aware task scheduling and allocation technique for multi-processor SoCs was presented in [10]. Modern SoCs are frequently divided into islands that operate at different clock frequencies, supply voltages, or body bias voltages [11], [12]. Several research efforts have exploited the multi-island design style to mitigate the effects of variations [13], [14]. The use of software adaptation to deal with hardware variations was proposed in [15].

A limited body of work has explored incorporating the impact of variations during system-level performance and power analysis. An analytical framework for performance analysis of multi-island systems under variations was presented in [16]. Techniques to perform variation-aware power analysis at the system level were presented in [17].

Our work addresses the problem of variation-aware performance analysis, for which the only known prior work takes an analytical approach [16]. Analytical techniques have proven their utility in limited contexts in system-level design (such as worst-case timing analysis), and are much faster than simulation when they are applicable. However, the most widespread approaches to system-level performance estimation in practice (regardless of variations) are based on simulation. This is because simulation-based approaches are very general (no limiting assumptions made regarding the system), offer the flexibility to tradeoff accuracy for efficiency (through the level of detail at which the model is specified), and are the most accurate in accounting for complex effects such as shared system resources, contention, data-dependent execution times, timing-dependent changes in execution paths, *etc.* To the best of our knowledge, ours is the first proposal to apply emulation to the problem of variation-aware performance analysis. The key benefit of our approach is that we significantly improve the speed of variation-aware performance analysis, without compromising either generality or accuracy.

III. Complexity of System-Level Variation Analysis

In this section we use an example SoC to illustrate the challenges involved in analyzing the impact of process variations on system level performance. The SoC that we consider implements the 802.11b MAC protocol (shown in Fig. 1).

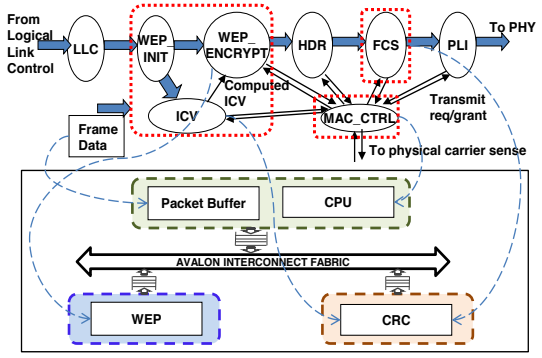


Fig. 1: SoC implementing 802.11b MAC protocol

The system takes incoming packets from an on-chip RAM (Packet Buffer), performs a Cyclic Redundancy Check (CRC) on it, the packet data is then encrypted using the Wired Equivalent Privacy (WEP) encryption scheme, another CRC computation is performed on this encrypted packet data and the packet is stored back to the Packet Buffer. We implemented both WEP and CRC components as hardware accelerators. The components are connected to each other using the Avalon Interconnect Fabric [18]. Since we are interested in designing variation-tolerant SoCs, we utilize the multi-island design style for the SoC [12] [1].

We first find the delay (clock frequency) distribution of the components of this SoC, by synthesizing each component and using commercial gate-level statistical timing analysis tools. Fig. 2 shows that there is a large diversity in both the mean and the standard-deviation of the delays across components. The delay distribution at the component level is mostly a function of logic depth and the number of critical paths [3]. When the component has a large number

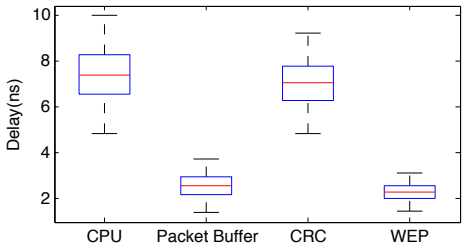


Fig. 2: Component-wise delay distribution for the 802.11 MAC SoC

[1]For SoCs that contain a single frequency island, we note that the performance analysis problem degenerates to simply determining the frequency distribution, which can be performed using conventional gate-level SSTA tools without any system-level analysis. However, such a design would incur significant performance degradation or yield loss due to variations.

of critical paths, calculating the statistical maximum of individual path distributions tends to increase the mean and decrease the standard deviation of the component clock distribution. Similarly higher logic depth contributes to decreasing the σ/μ ratio of the clock period distribution [3]. In general different components in an SoC tend to have different circuit level characteristics, we might find a large diversity in their delay or frequency profiles under variations. Given the delay distribution of each component, we evaluate the sensitivity of the system's performance to variations using the VESPA framework. We measure the system performance of the MAC system under varying frequencies of two of its components (WEP and CRC). Fig. 3 shows that, in most of the frequency space, system performance is more sensitive to variations in the WEP frequency as it is the dominant component in the system's critical path. However, there exists a region in this space (highlighted by a rectangle in Fig. 3) where the system is more sensitive to the CRC component than the WEP. In this region the CRC component starts to dominate the system critical path.

To further analyze the sensitivity of system performance to delay variations of components, we consider two different instances of the MAC system with nominal frequencies corresponding to Points A and B in Fig. 3 and measure the minimum throughput that would guarantee 95% yield with varying σ/μ of component frequencies. Fig. 4 shows that a system designed around Point A as the nominal case is most sensitive to variations in the WEP block's frequency whereas the system designed with Point B as the nominal case is most sensitive to the CRC block. In general different components have different sensitivities and even these sensitivities are a function of the frequencies of operation of the other components of the system.

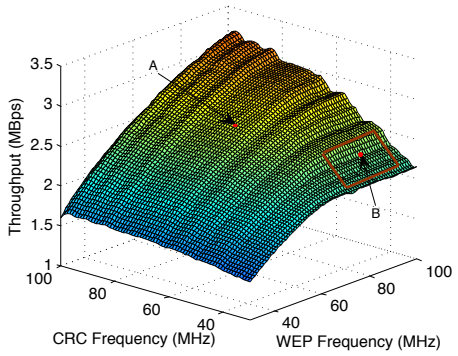
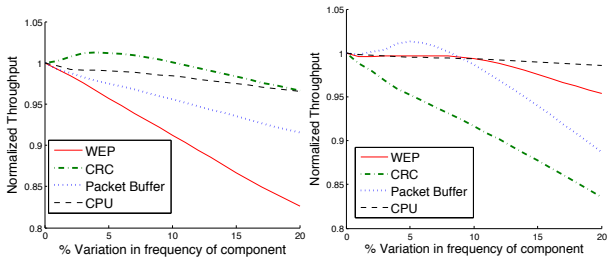


Fig. 3: System performance *vs.* component frequency



(a) Point A (b) Point B

Fig. 4: System sensitivity to variations at different nominal frequencies

The sensitivity of system performance to component variations is a very complex problem due to several factors.

- Component interdependence: System sensitivity to a component is determined by the percentage of time a component spends in the system critical path, which depends on a component's interaction (synchronization and communication) with other SoC components.
- Shared resources: Resources such as a system bus that are shared by all components result in contentions that lead to variable latencies. These contention profiles and thus latencies change in the presence of component variations.
- Multiple paths of execution: In systems that have multiple paths of execution, variations can cause a given path to speed up or slow down and as a result it may no longer even be a part of the system's critical path, causing discontinuities in the system's sensitivity to a component.

Any accurate and general system-level performance analysis technique must consider these intricacies. One approach is Monte-Carlo simulation of the system for sufficiently large number of samples, so as to have a high level of confidence in the performance distribution. Using sampling theory [19], we estimate that the MAC system discussed above requires 900 samples for a confidence level of 99% on the estimated mean value with margin of error less than 1%. Performing such a large number of system level simulations would require a prohibitively large run-time and is not practical.

Emulation is a well known technique for improving the runtime of system level simulation. It provides orders of magnitude improvement in performance over system simulation at various levels of abstraction. However, using standard emulation techniques for variation aware analysis poses a number of challenges. Variation aware analysis requires emulating the system at multiple sample frequencies. This in turn would require synthesis of the system and downloading the bit-stream for every sample repeatedly. The overhead involved would defeat the benefits obtained through emulation. In contrast, the VESPA framework performs these operations exactly once. This is accomplished by down-scaling the frequency distribution of each component to the operating frequency range of the FPGA, using reconfigurable PLLs whose frequency can be changed on the fly and a software control routine to vary the operating frequencies of the component and perform repeated emulation of the application program at different sample frequencies. This process is discussed in detail in the next section.

IV. VESPA Framework

In this section, we present an overview of our proposed variation aware emulation framework. The framework takes as its input, the SoC architecture, the application software, RTL component models, variation-enhanced cell libraries and outputs the system performance distribution. The flow as shown in Fig. 5 can be divided in to three distinct phases. The Component Variability Characterization phase generates the delay distribution for each component of the SoC. The Variation-aware Emulation Setup phase consists of adding various hardware components and software routines essential for

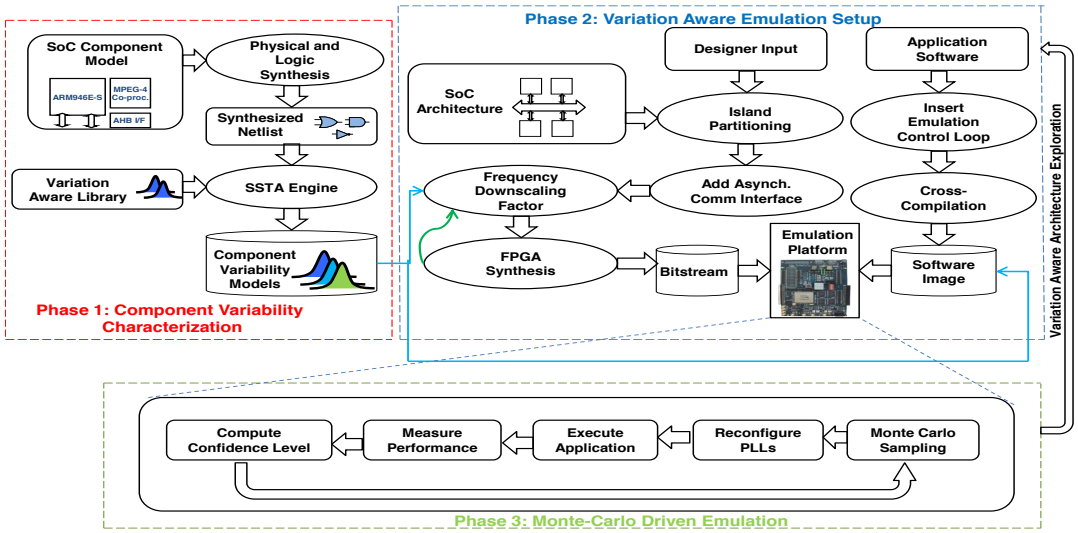


Fig. 5: VESPA Framework

performing variation-aware performance analysis. The Monte-carlo Driven Emulation phase describes the various steps the runtime control software executes so as to obtain the desired performance distribution.

A. Component Variability Characterization

In this phase, we synthesize different components of the system from their RTL models using commercial logic synthesis tools. A gate-level variation-aware technology library is fed to a statistical static timing analysis (SSTA) engine along with the synthesized component netlist to compute the frequency distribution for different components of the SoC, taking into account the structural correlation that may exist between gates and paths within each component. At the end of this phase, we obtain the component level delay distributions for each component in the SoC.

B. Variation-aware Emulation Setup

In this phase, we perform the design-time operations required for setting up our emulation framework. Based on designer input, the SoC architecture is first partitioned into multiple frequency islands. We then insert FIFOs and other asynchronous interface logic so that the different frequency islands can communicate efficiently with each other. In order to perform variation-aware analysis efficiently, we need to change the frequency of each island without performing repeated synthesis. This is accomplished by generating reconfigurable PLLs whose frequencies can be controlled at runtime by programming them from software executing on a microprocessor. The frequency distribution obtained from the first phase needs to be down-scaled by an appropriate factor to ensure correct functioning in the emulation environment. Each component is synthesized individually such that correct operation is guaranteed for frequencies ranging from $\mu-3\sigma$ to $\mu+3\sigma$ and its corresponding down-scaling factor is obtained. The down-scaling factor of the entire system is then computed by finding the maximum of each component's down-scaling factor. We then take the given SoC application program and encapsulate it within a software control routine that performs the operations required to measure system performance.

C. Monte-carlo Driven Emulation

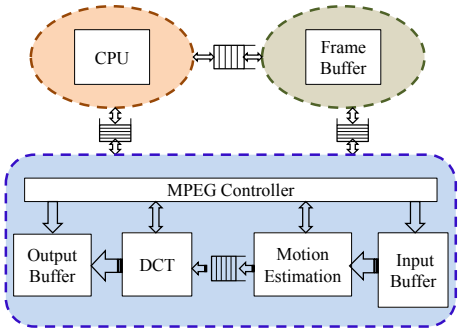
In the Monte-carlo Driven Emulation phase, the software control routine first samples the component distributions to obtain their operating frequencies. It then reconfigures each island's PLL to its sampled frequency point. The given SoC application is then executed and its performance is measured with the help of hardware counters. This process is repeated for a fixed number samples, or until a a given level of confidence is achieved in the generated distribution. Once the software control loop terminates, the desired performance distribution is obtained.

V. Experimental Results

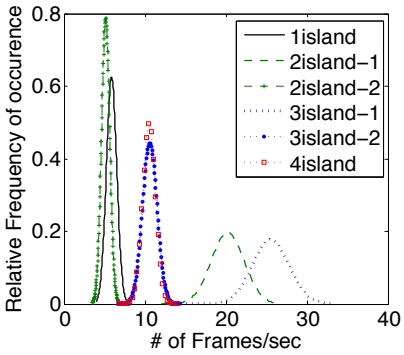
In this section, we first describe our experimental set up and show that considerable speed up can be achieved using the VESPA framework compared to existing techniques. We then present two case studies to illustrate the utility of our framework for variation-aware architecture exploration.

A. Experimental Setup

In the component variability characterization phase, Synopsys Design Compiler was used to synthesize various SoC components using the IBM 45nm technology library. Variations were modeled in accordance with [20]. The SSTA analysis was performed using Synopsys Primetime-VX [21] to obtain the frequency distributions of different components. Access time of the memory components were estimated using CACTI5.3 [22] and variations were modeled in accordance with [23]. We used an Altera DE3 board with a Stratix III EPS3SL150 FPGA and the Quartus 10.0 FPGA design kit. The Nios2 IDE was then used to create the software framework and control the runtime emulation flow. The speed up of the proposed framework was compared with models of the systems simulated using the hardware/software co-simulation tool Gezel [24] and RTL simulation using Modelsim.



Architecture	Component Mapping			
1 island	CPU, Frame Buffer, MPEG Controller, ME, DCT			
2 island-1	CPU	Frame Buffer, MPEG Controller, ME, DCT		
2 island-2	CPU, Frame Buffer	MPEG Controller, ME, DCT		
3 island-1	CPU	Frame Buffer, MPEG Controller	ME, DCT	
3 island -2	CPU	Frame Buffer	MPEG Controller, ME, DCT	
4 island	CPU	Frame Buffer	MPEG Controller, ME	DCT



(a) Block Diagram (b) Different Island Partitions (c) Performance Distribution

Fig. 6: Exploring multi-island architectures for the MPEG encoder

TABLE I: Runtime Comparison

System	Cycles executed per sec.(x10^6)			Speed up	
	RTL	Gezel	VESPA	VESPA/RTL	VESPA/ISS
MPEG	0.0015	2.07	133	88666	64
MAC	0.0017	2.16	133	78235	61

We consider two example SoCs - an 802.11b MAC processor and an MPEG Encoder. The MAC system was described in detail in Section III. Fig. 6a shows the block diagram of the MPEG encoder system. The input frames are stored in the Frame Buffer. The MPEG controller coordinates the transfer of blocks from the Frame buffer to the Input Buffer and motion estimation is performed by comparing the current and previous frames. The DCT block computes the Discrete Cosine Transform and stores the compressed data in the Output Buffer. The performance of the system is primarily determined by the time required to read packets from the Frame Buffer and the time taken to perform motion estimation. The CPU and DCT spend a relatively small amount of time in the system critical path.

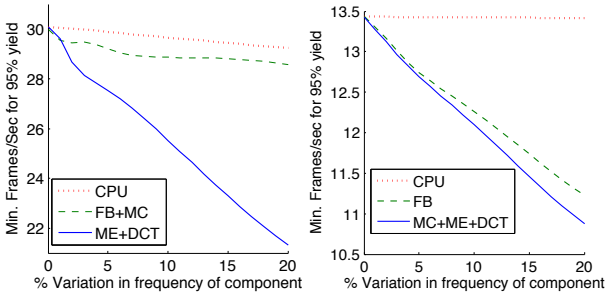
B. Speed Up

Table I compares the number of cycles executed per second using RTL simulation, HW/SW co-simulation using Gezel and VESPA. On average, our proposed frame work was found to have a speed up of 80000x compared to RTL simulation and 60x with respect to HW/SW co-simulation using Gezel. These results clearly demonstrate that the proposed framework can be used to efficiently model the effect of process variations on system performance.

C. Case Study 1: Island Partitioning and Component Mapping

In this case study, we use VESPA to study island partitioning for the MPEG system. We demonstrate the tradeoff between the adaptivity offered by larger numbers of islands and the overhead introduced by asynchronous communication interfaces. We also show that, for a given number of islands, the system performance greatly depends on the mapping of the components to islands.

Fig. 6b shows six different partitions of the MPEG system and the corresponding system performance distributions are shown in Fig. 6c. Initially, there is an improvement in the system performance as the number of islands increases. However, when the system is partitioned beyond a certain threshold, the overhead of communication interfaces begins to dominate, causing a reduction in system performance. The 1island, 2island-1, 3island-1 and 4island architectures demonstrate this.



(a) 3 island-1 (b) 3 island-2

Fig. 7: Impact of component mapping on sensitivity

For a given number of islands, both the system performance and its sensitivity to components depend on the component-to-island mapping. For example, let us compare the 3island-1 and 3island-2 architectures in Fig. 6c and Fig. 7. In the 3island-1 configuration, Frame Buffer and MPEG controller operate at the same frequency and there is no need for additional asynchronous communication interfaces, which decreases the read latency for macroblocks. System performance is determined mainly by the ME block which makes it more sensitive to variations in ME+DCT than FB+MC. On the other hand, in the 3island-2 configuration, the Frame Buffer and MPEG controller are in different islands and memory read latency becomes a substantial part of the system critical path. This makes system performance sensitive to variations in frequency of the Frame Buffer and ME block.

D. Case Study 2: Design Space Exploration

In this case study we demonstrate how VESPA can be used to efficiently perform exhaustive design space exploration and also show the need for performing variation-aware system design using the MAC system described in Section III.

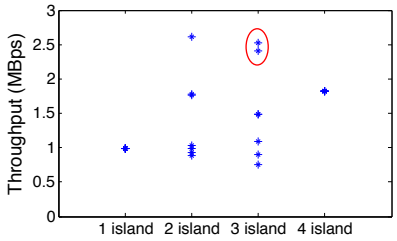


Fig. 8: Multi-island design space for the 802.11 system

Fig. 8 shows the throughput that can guarantee 95% yield for all possible island partitionings and component-to-island mappings. System performance again increases initially with increasing number of islands (until 3 islands). However, due to high overheads involved in inter-island communication, further increasing the number of islands deteriorates system performance. For a given fixed number of islands, the component-to-island mapping also has a huge impact on system performance, as illustrated by the vertical spread in Fig. 8.

Consider the two points encircled in Fig. 8, which correspond to two three-island configurations, one in which the Packet Buffer and the CRC component are mapped to same island (CRC+PB) and another in which the Packet Buffer is grouped along with the WEP component (WEP+PB). Fig. 9 shows the system performance distribution profile for these two configurations. When variations are ignored, both configurations would yield nearly identical performance. In both configurations the individual component's throughput is limited by memory access times. In the WEP+PB configuration, the CRC component has to communicate across an island partition, leading to higher memory access times. As a result system performance becomes highly sensitive to CRC variations and similarly in the CRC+PB configuration system performance is more sensitive to the WEP component. From SSTA analysis, it turns out the WEP component has much larger σ in the clock period than the CRC block. Thus, as shown in Fig. 9, the CRC+PB configuration is more severely impacted by variations than the WEP+PB configuration. As a result, for the same yield requirement the WEP+PB mapping is clearly better than CRC+PB mapping. In general, architectures in which high variability components are not a large part of the system critical path, tend to be more tolerant to variations.

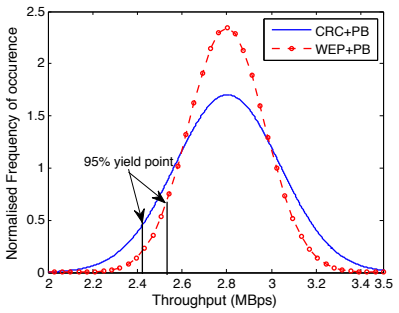


Fig. 9: Performance distributions for two 3-island designs

In summary, our experiments clearly establish the value of the proposed system-level variation analysis framework in driving variation-tolerant design.

VI. Conclusion

We presented a framework for system-level performance analysis under variations, which leverages emulation to significantly speedup analysis without sacrificing generality and accuracy of simulation. Key to preserving the inherent efficiency of emulation is the ability to perform repeated system executions without re-configuring the emulation platform or re-synthesizing the design. We applied the proposed framework to two example SoCs, and demonstrated its utility in exploring variation-aware multi-island architectures for both systems.

References

[1] S. Borkar et. al. Parameter variations and impact on circuits and microarchitecture. In *Proc. DAC*, pages 338–342, 2003.
[2] International Technology Roadmap for Semiconductors. http://public.itrs.net/reports.html.
[3] K. A. Bowman, S. G. Duvall, and J. D. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE JSSC.*, 37(2):183 –190, Feb. 2002.
[4] N. S. Kim et. al. Total power-optimal pipelining and parallel processing under process variations in nanometer technology. In *Proc. ICCAD*, pages 535–540, 2005.
[5] A. Agarwal et. al. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE TVLSI*, 13(1):27 – 38, 2005.
[6] D. Ernst et. al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proc. MICRO*, pages 7 – 18, Dec. 2003.
[7] P. Ndai et. al. Trifecta: A nonspeculative scheme to exploit common, data-dependent subcritical paths. *IEEE Trans. VLSI*, 18(1):53 –65, Jan. 2010.
[8] S. Pandey, R. Drechsler, T. Murgan, and M. Glesner. Process variations aware robust on-chip bus architecture synthesis for MPSoCs. In *Proc. ISCAS*, pages 2989 –2992, May 2008.
[9] S. Chandra, K. Lahiri, A. Raghunathan, and S. Dey. System-on-chip power management considering leakage power variations. In *Proc. DAC*, pages 877–882, June 2007.
[10] F. Wang and Y. Xie. Embedded multi-processor system-on-chip (MP-SoC) design considering process variations. In *Proc. IPDPS*, pages 1–5, 2008.
[11] D. E. Lackey et. al. Managing power and performance for system-on-chip designs using voltage islands. In *Proc. ICCAD*, pages 195–202, 2002.
[12] K. Niyogi and D. Marculescu. Speed and voltage selection for gals systems based on voltage/frequency islands. In *Proc. ASP-DAC*, pages 292–297, 2005.
[13] B. Stefano et. al. Process variation tolerant pipeline design through a placement-aware multiple voltage island design style. In *Proc. DATE*, pages 967–972, Mar. 2008.
[14] U. Y. Ogras, R. Marculescu, and D. Marculescu. Variation-adaptive feedback control for networks-on-chip with multiple clock domains. In *Proc. DAC*, pages 614–619, June 2008.
[15] A. Pant, P. Gupta, and M. van der Schaar. Software adaptation in quality sensitive applications to deal with hardware variability. In *Proc. Great Lakes Symp. on VLSI*, pages 85–90, 2010.
[16] S. Garg and D. Marculescu. System-level throughput analysis for process variation aware multiple voltage-frequency island designs. *ACM TODAES.*, 13(4):1–25, 2008.
[17] S. Chandra, K. Lahiri, A. Raghunathan, and S. Dey. Considering process variations during system-level power analysis. In *Proc. ISLPED*, pages 342–345, 2006.
[18] ALTERA. http://www.altera.com/.
[19] R. A. Fisher. Statistical methods and scientific inference. *Oliver and Boyd*, 1956.
[20] Y. Cao and L. T. Clark. Mapping statistical process variations toward circuit performance variability: an analytical modeling approach. In *Proc. DAC*, pages 658–663, June 2005.
[21] Primetime-VX. Synopsys inc.
[22] CACTI-5.3. http://quid.hpl.hp.com:9081/cacti/detailed.y.
[23] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaledCMOS. *IEEE Trans. CAD*, 24(12):1859 – 1880, Dec. 2005.
[24] GEZEL Hardware/Software Codesign Environment. http://rijndael.ece.vt.edu/gezel2/.