System-Level Throughput Analysis for Process Variation Aware Multiple Voltage-Frequency Island Designs

SIDDHARTH GARG and DIANA MARCULESCU
Carnegie Mellon University, Pittsburgh

The increasing variability in manufacturing process parameters is expected to lead to significant performance degradation in deep submicron technologies. Multiple Voltage-Frequency Island (VFI) design styles with fine-grained, process-variation aware clocking have recently been shown to possess increased immunity to manufacturing process variations. In this article, we propose a theoretical framework that allows designers to quantify the performance improvement that is to be expected if they were to migrate from a fully synchronous design to the proposed multiple VFI design style. Specifically, we provide techniques to efficiently and accurately estimate the probability distribution of the execution rate (or throughput) of both single and multiple VFI systems under the influence of manufacturing process variations. Finally, using an MPEG-2 encoder benchmark, we demonstrate how the proposed analysis framework can be used by designers to make architectural decisions such as the granularity of VFI domain partitioning based on the throughput constraints their systems are required to satisfy.

Categories and Subject Descriptors: B.8.2 [**Hardware**]: Performance and Reliability—*Performance analysis and design aids*

General Terms: Performance, Design, Theory

Additional Key Words and Phrases: Globally asynchronous locally synchronous, maximum cycle mean, performance analysis, manufacturing process variations, system-level design, voltage-frequency islands

ACM Reference Format:

Garg, S. and Marculescu, D. 2008. System-level throughput analysis for process variation aware multiple voltage-frequency island designs. ACM Trans. Des. Automat. Elect. Syst. 13, 4, Article 59 (September 2008), 25 pages, DOI = 10.1145/1391962.1391967 http://doi.acm.org/10.1145/1391962.1391967

This research was supported in part by Semiconductor Research Corporation contract number 2005-HU-1314.
Author's address: S. Garg, email: sgarg1@andrew.cmu.edu.

1. INTRODUCTION

Progressive scaling of device dimensions has made it increasingly harder to strictly control manufacturing process parameters such as gate length, gate width and channel doping concentration. Variations in these process parameters translate to variations in the performance characteristics of the manufactured dies. The problem is further exacerbated by the fact that these variations manifest themselves as either lot-to-lot (L2L), wafer-to-wafer (W2W), die-to-die (D2D), or within-die (WID) variations. While L2L, W2W, and D2D variations impact each device on a die in exactly the same fashion, WID variations can impact each device on the same die differently. Furthermore, WID device variations can be entirely random in nature or spatially correlated. Traditionally, the impact of WID variations on the system performance has been neglected, but as the minimum device dimensions become smaller than the wavelength of light used in the lithographic process, they can no longer be ignored. Bowman et al. [2002] demonstrate that in scaled technologies, WID variations can indeed have a significant impact on the operating frequency (and thereby performance) of a design. They predict that unless a solution is found to the WID variability problem, as much as 39% reduction in die frequency may be observed at the 32-nm technology node.

A number of solutions have been proposed to mitigate the impact of process variation on system performance. They are divided broadly into two categories—pre-silicon and post-silicon techniques [Kulkarni et al. 2006]. Pre-silicon techniques adjust design time parameters such as device widths, threshold voltages and layout styles to make the circuit less susceptible to process variation. On the other hand, post-silicon techniques adjust the characteristics of each individual die, after it has been manufactured, to compensate for the impact of process variations on that specific die. For example, Adaptive Body Biasing (ABB) [Tschanz et al. 2002] is a popularly used post-silicon technique that sets the body bias of each die based on its leakage power dissipation. Dies that exhibit greater (lesser) than the expected leakage power dissipation are set at a reverse (forward) body bias, thereby increasing (decreasing) the threshold voltage of all devices on the die, leading to a reduction (increase) in the overall leakage power dissipation. Another technique, called Speed Binning [Belete et al. 2002], allows each die to operate at its maximum possible clock frequency dictated by the extent to which process variations have sped-up or slowed-down critical paths on that die.

In this article, we consider the case of embedded systems implemented as multiple voltage and frequency island (VFI) designs with post-silicon tuning capabilities to cope with WID process variations. Specifically, we assume that each VFI can operate at a frequency that is limited only by the slowest critical path within that particular island—that is, we allow for independent speed-binning of each VFI in the design instead of the traditional, die-level speed-binning that is performed for fully-synchronous systems. Therefore, in the proposed architecture, even if some parts of the die contain frequency-limiting critical paths due to WID process variations, only those parts would be required to run at lower frequencies while the others could continue to operate at their optimum clock speeds. This is in contrast to a fully synchronous design in which the operating frequency is restricted by the slowest critical path on the entire die, thereby forcing faster parts of the die to run at speeds limited by the slowest part.

While it seems intuitively obvious that multiple VFI designs would be less susceptible to WID process variations, it is essential to provide tools that allow designers to quantify the performance benefits of moving to such a design style. Furthermore, the performance benefits must be reported using a metric that system-level designers can use in a practical manner; for example, the frame encoding rate in *frames/second* would be a useful metric for an MPEG encoding embedded system. To this end, the performance metric that we explore in this paper is the execution rate or throughput of an embedded system whose execution characteristics have been specified using a *component graph*. The proposed framework allows designers to *accurately and efficiently determine the percentage of dies that will meet a specified throughput constraint*, given the system component graph, its partitioning into VFIs and the impact of process variability on the clock speed of each VFI.

2. RELATED WORK

Multiple voltage island designs were originally proposed as a solution to the increasing power dissipation concerns for digital ICs [Lackey et al. 2002]. The authors argue that each functional block in a complex SOC need only be run at the minimum voltage (and frequency) that is required to meet the specified performance constraints, thereby leading to a decrease in chip power dissipation. On a related note, the Globally Asynchronous Locally Synchronous (GALS) design style, introduced by Chapiro [1984], allows clock-domains, or locally synchronous blocks (LSB), to be clocked independently and asynchronously with respect to each other, while data transfer between the LSBs is orchestrated using asynchronous communication interfaces. GALS systems can significantly reduce the power dissipated in the clock distribution network [Hemani et al. 1999], and, at the same time allow for independent dynamic voltage and frequency scaling (DVFS) of each locally synchronous block for increased power savings. This idea has been exploited previously to reduce the dynamic power consumption of an out-of-order superscalar GALS processor [Semeraro et al. 2002; Marculescu and Iyer 2002], and of VFI-based embedded systems [Niyogi and Marculescu 2005a]. While both these techniques use fine-grained voltage and frequency adaptation to deal with workload driven variability, the idea of using fine-grained frequency scaling to combat the impact of WID process variations was introduced later in the context of a GALS processor [Marculescu and Talpes 2005]. The authors note that since each clock domain of the GALS processor has fewer critical paths than the processor as a whole, WID process variations would have a greater impact on a fully synchronous processor, which would therefore run slower, on average, than each clock domain of the GALS version. Based on this observation, the authors show that, a GALS processor demonstrates superior performance, measured using a metric that combines the impact of instructions-per-cycle (IPC), energy, and die area than its fully synchronous counterpart. However, since the IPC of a processor cannot be measured analytically with sufficient accuracy, the authors use exhaustive simulation to evaluate the impact of moving to a GALS design style.

A significant body of research has emerged over the past few years that analyzes the impact of process variation on system performance characteristics such as circuit timing and leakage power dissipation at the transistor/gate level. Researchers have only recently started looking at process variation at the microarchitecture or system level. Marculescu and Garg [2006] analyze the effect of WID process variations on the execution *latency* of embedded systems implemented as both fully synchronous and multiple VFI designs. The proposed techniques, however, are only applicable to embedded systems specified as acyclic component graphs. This work introduces techniques to analyze the impact of variability on the execution *rate* of throughput-constrained systems with *cyclic* component graphs. Wang et al. [2007] solve the variation aware task mapping and scheduling problem for heterogeneous multiprocessor platforms, in which each PE can have differing variability characteristics. Again, while this work uses an underlying system performance model that takes into account the impact of process variations, the model is restricted to acyclic graphs only. Hung et al. [2006] and Wang et al. [2008] both consider the variability-aware module selection problem in high-level synthesis for fully synchronous systems. Recent work on variability-aware design for general-purpose high performance microprocessor systems has focused on the use of pipeline time borrowing [Tiwari et al. 2007], fine-grained body-biasing [Teodorescu et al. 2007] and novel cache architectures [Liang et al. 2007] to mitigate the impact of process variations.

Hu et al. [2001] and de Veciana and Jacome [1998] study the impact of variability in the *application characteristics* of the tasks running on an embedded system on the final performance of the system. On the other hand, we assume deterministic application characteristics, but are interested in variations that arise from the variability in manufacturing process parameters. There are several key differences between the work by Hu et al. [2001] and de Veciana and Jacome [1998] and our work—first, both these prior works concentrated on execution latency as a performance metric while we care about the application throughput, which requires the computation of the maximum cycle mean in a probabilistic setting. To the best of our knowledge, this has not been attempted before, and we do not see any direct way the techniques from Hu et al. [2001] and de Veciana and Jacome [1998] can be adapted to our problem. Second, one of the primary goals of this work is to study the impact of the granularity of VFI partitioning on the performance of variability aware multiple VFI designs, to aid system-level designers in making informed trade-offs between design complexity and performance. This scenario does not arise during performance characterization under application variability (since it is purely a physical design issue), and has not been dealt with before.

Finally, from an implementation perspective, Datta et al. [2005] propose a globally asynchronous locally synchronous (GALS) architecture consisting of a number of processing units (PU), each implemented as a separate VFI. The authors note that intra-die process variations can cause the maximum clock frequency of each PU to shift by different margins and outline a software-based self-test scheme to run each PU at its optimal clock frequency. Ernst et al. [2003] propose a hardware based technique that uses shadow flip-flops to detect timing violations, thereby allowing the synchronous logic to operate close to or at its minimum possible supply voltage. The same technique can as well be used to run each VFI in a multiple VFI system at its maximum clock frequency. Both works focus on implementation issues, while our focus is on *evaluating* the benefits of such implementations.

3. ARTICLE CONTRIBUTIONS

This work makes the following contributions:

—We consider the case of *cyclic component graphs* implemented using a multiple VFI design style and derive distributions for the execution rate of these systems under manufacturing process variations. Our technique offers significant speed-up over Monte Carlo based simulation at the expense of marginal loss in accuracy.
—In the process of determining system throughput, we describe, to the best of our knowledge, the *first algorithm* that solves the Maximum Cycle Mean (MCM) problem in a *probabilistic setting*.
—Using a case study, we demonstrate how our framework can be used to evaluate the trade-off between performance and clock domain granularity, and compare the performance of a multiple VFI design versus that of a fully synchronous design.

The rest of the article is organized as follows: in Section 4 we outline the assumptions we make about the underlying multiple VFI implementation under consideration and introduce the mathematical notation we use in the rest of the paper, in Section 5 we outline the algorithm we use to study the impact of WID variability on the throughput of single and multiple VFI systems, in Section 6 we provide some experimental results on synthetic and a real embedded benchmark, and finally, we conclude in Section 7 with some comments and possible future work.

4. PRELIMINARIES AND ASSUMPTIONS

We consider the case of embedded systems implemented using multiple processing elements (PE), with each PE executing one of more of the tasks into which the application has been partitioned. Based on the required performance, each PE could be an embedded processor, a custom digital implementation or an imported Intellectual Property (IP) component. We now examine two implementation alternatives for such systems—a fully synchronous design (henceforth referred to as an SSV design in keeping with the terminology introduced by Marculescu and Garg [2006]) with a single global clock that drives all the PEs, or a multiple VFI design in which each voltage-frequency island is controlled by an independent clock source. We note that for SSV designs, the PEs

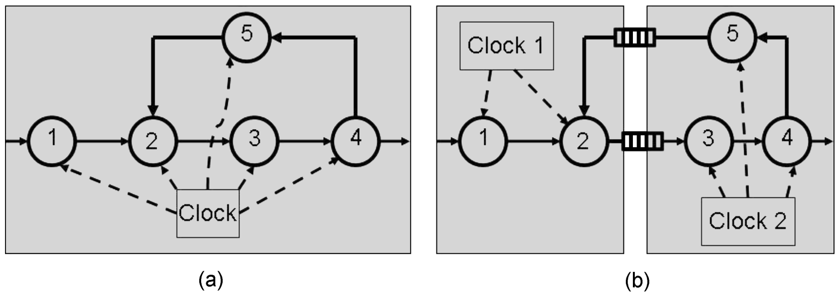


Fig. 1. Component graph representation of an embedded system with five PEs implemented as (a) an SSV design with a single global clock source; or (b) a multiple VFI design with two VFIs. Mixed-clock FIFOs are used to communicate across clock domains.

communicate using point-to-point synchronous links, while in the multiple VFI design case, links that cross clock domains communicate via mixed-clock FIFO interfaces [Chelcea and Nowick 2001] modified to support voltage level conversion. A similar network of point-to-point FIFOs was proposed by Carloni and Sangiovanni-Vincentelli [2000] as part of the Latency Insensitive Design paradigm, and is particularly well-suited for multiple VFI architectures. Finally, for both VFI and SSV systems, we assume that the implementation supports fine grained frequency control, and there exists either hardware or software based support [Datta et al. 2005; Ernst et al. 2003] to allow each clock domain to run at or near its optimal clock frequency under the impact of process variations.

We model a system comprising of a number of communicating PEs using a *component graph*, represented as a directed graph $G(V, E)$. Vertices in a component graph represent PEs and edges represent control or data dependencies between vertices. Figure 1 shows an example of a component graph with five PEs. The graph on the left represents an SSV design that is clocked globally with a single clock source, while the one on the right represents a multiple VFI design with two voltage-frequency islands, each with its own local clock source. Finally, for notational simplicity, we will assume throughout this article that each PE has only one task mapped to it, thereby making the concept of a "PE" inter-changeable with that of a "task." In general, however, there may be multiple tasks mapped to each PE; we will show in Section 5 how this case can be handled by our framework.

5. THEORETICAL FORMULATION

Without any loss of generality, for a given component graph $G(V, E)$, we make the following assumptions:

—The component graph $G(V, E)$ of a system with n PEs comprises the set of nodes $V = 1, 2, \ldots, n$ and edges $E = (i, j) : i \rightarrow j, i, j \in V$.

—Each node i, $(1 \leq i \leq n)$, is characterized by the number of cycles C_i it takes to produce an output data token after all its input data dependencies are satisfied. For a PE implemented as an embedded processor, for example, the number of cycles can be estimated by profiling the code for the task the PE is required to perform on a cycle accurate simulator of the embedded processor. Note that, in general, the number of execution cycles for a PE can vary dynamically depending on the workload. However, in this work, we do not model workload or application driven variability and therefore restrict C_i to be a fixed number. From a practical perspective, C_i could represent the average-case or the worst-case number of cycles, depending on whether the metric of interest is average or worst-case execution rate.

—The number of cycles, C_i ($1 \leq i \leq n$), can be written as the sum of the number of cycles the PE spends performing computation, that is, *computation cycles* (C_i^{comp}), and the number of cycles it needs to communicate the output data to its consumer nodes, that is, *communication cycles* (C_i^{comm}). Therefore, $C_i = C_i^{comp} + C_i^{comm}$. Furthermore, the communication-to-computation ratio at node i, represented by α_i, can be written as

$$\alpha_i = \frac{C_i^{comm}}{C_i^{comp}}. \tag{1}$$

—Each PE is characterized in terms of the probability density function (*pdf*) of its cycle time T_i, where the cycle time, T_i, is defined as the inverse of the clock frequency for that PE. If the PE is an external IP, the *pdf* of cycle time could be provided by the IP vendor or it could be obtained using detailed circuit level Statistical Static Timing Analysis (SSTA) assuming probability distributions for the underlying process parameters. We note that from an implementation perspective, the clock frequency of a PE is likely to be controlled in discrete steps. If that is the case, the *pdf* of T_i will actually be a discrete distribution, obtained by quantizing the continuous cycle time distribution into discrete time steps based on the available frequency values. As we will show later, the proposed analysis techniques are valid for both discrete and continuous cycle time distributions. However, we note that continuous cycle time distributions provide the asymptotic limit of performance improvements that can be gained by moving to multiple VFI designs, and can therefore be an extremely useful design tool. The *pdf* of T_i is represented as $f_{T_i}(t)$ and its corresponding cumulative density function (*cdf*) as $F_{T_i}(t)$, where $F_{T_i}(t) = \int_{-\infty}^{t} f_{T_i}(\tau)\,d\tau$.

—The cycle time random variables for each PE are assumed to vary independently of each other. While spatial correlations between process parameters may lead to some degree of correlation between the PE cycle time random variables, the magnitude of correlations seen between PE cycle times is typically much smaller than between individual gates in the circuit during traditional statistical timing analysis, since spatial correlations tend to die out quickly as a function of distance (for example, Friedberg et al. [2006] showed that spatial correlations go to zero at a distance of half the die length). Related work at gate-level [Chang and Sapatnekar 2003] that has considered spatial correlations also determined that only adjacent gates are highly correlated—this translates to negligible correlations between large PEs, like the ones considered in our work. Furthermore, while gate length has been shown to posses spatial correlation characteristics, an important source of threshold voltage variation is random dopant fluctuation (RDF) that is purely random from transistor to transistor—this random component tends to further decrease the correlation between PE cycle times. We demarcate the precise treatment of correlations between cycle time random variables as future work.

—Given C_i and T_i for a PE, its execution latency $L_i = C_i.T_i$ will also be a random variable. Since we have assumed C_i to be a fixed number, the *pdf* for L_i can be computed directly from the *pdf* of the cycle time. We will refer to the *pdf* of L_i as $f_{L_i}(t)$ and its *cdf* as $F_{L_i}(t)$.

—We point out that the notation and assumptions described above hold if each PE lies in a separate VFI. In general, assume that there are p VFIs, where $p \leq n$. If $p < n$, there will be at least one domain with more than one PE. The cycle time of the VFI j is given as T_j^{VFI}, where $1 \leq j \leq p$. Without loss of generality, let the nodes $(1, 2, \ldots, r)$ belong to the jth VFI. Since the cycle time of a VFI can be no smaller than the largest cycle time of its constituent PEs, we can write T_j VFI $= \max(T_1, T_2 \ldots, T_r)$. Furthermore, we need to make the following changes in the definition of the latency L_i of a PE: if PE i lies in VFI j, its latency $L_i = C_i.T_j$. Though our proposed algorithm is presented for the case in which each PE lies in a separate VFI (to avoid notational complexity), these modifications make it equally valid for the case when there is more than one PE in a clock domain.

—If the number of VFIs is less than the number of PEs in the system, we assume that the mapping of PEs to VFIs is predetermined. However, the analysis techniques presented in this article could be used in an optimization framework to determine the best possible (from a variability perspective) PE to VFI mapping for a fixed number of VFIs, though we leave this as future work.

—As mentioned in Section 4, we assume, for notational simplicity, that each PE has only one task mapped to it. For the general case, in which multiple tasks are mapped to each PE, we can define a mapping function, M, that maps tasks to PEs—that is, $M(i)$ is the PE on which task i is mapped. Therefore, the latency of task i can be written as $L_i = C_{M(i)}T_{M(i)}$, where $C_{M(i)}$ is the number of clock cycles that task i takes to execute on PE $M(i)$, and $T_{M(i)}$ is the cycle time of PE $M(i)$. It is clear that for the general case, the latency distributions of tasks mapped to the same PE will be correlated since they share the same cycle time random variables (in fact, they will be *perfectly correlated*). However, the proposed algorithm only requires that the latency of each task be specified as a linear function of a set of *independent* random variables. This condition is clearly valid for the case in which each PE has only one task mapped to it; more importantly, it is *also* valid for the general case described above, since the latency of each task is still a linear function of the cycle time random variables. The proposed algorithm can, therefore, be easily extended to handle the general case as well.

5.1 Throughput Analysis for VFI Systems

In this section, we describe in detail the algorithm we propose to compute the execution rate of an embedded system implemented using multiple VFI design style under the impact of manufacturing process variations. For clarity, the following discussion assumes that data communication over the asynchronous interfaces between the clock-domains of a VFI design does not incur any performance overhead. However, in Section 5.3, we remove this assumption by appropriately modifying the proposed algorithm to account for the communication overheads in a multiple VFI design.

The execution rate (or throughput) of a component graph is restricted by the presence of cycles in the graph [Mathur et al. 1998]. Cycles in component graphs can *only* be found within *strongly connected components* (SCC). A SCC is a set of nodes in which it is possible to traverse from every node to every other node. While a graph can have more than one SCC, no two distinct SCCs can have a node in common. Furthermore, all SCCs in a graph can be found in linear time with respect to the number of nodes in the graph [Mathur et al. 1998]. It is, therefore, sufficient to individually compute the throughput constraining cycles of each SCC in a component graph to determine the system throughput, which will just be the minimum throughput across all SCCs. In the following discussion, therefore, we only discuss throughput analysis on a component graph that is strongly connected, and later show how graphs with multiple SCCs can be analyzed.

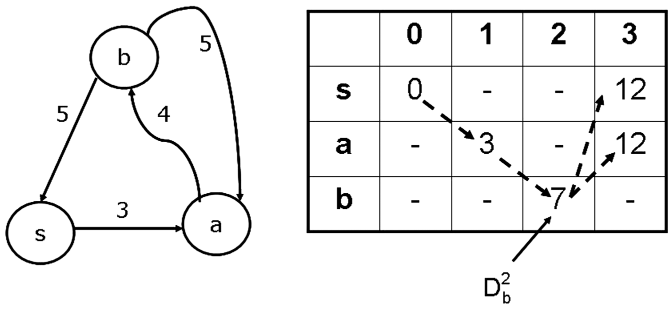
We start with a component graph $G(V, E)$ with n nodes, as described in the previous section. We make an additional assumption that $G(V, E)$ is strongly connected. We note that if the graph is not strongly connected, we can run the proposed algorithm on each of its SCCs individually and take the statistical minimum of the resulting distributions from each SCC, as we will demonstrate in the final step of the proposed algorithm. Finally we associate weights $w(u, v)$ to every edge $e \in E$ that connects nodes u and v. The weight assigned to edge e is equal to the latency (as defined in the previous Section) of the source node of that edge. Specifically:

$$w(u, v) = L_u, \forall (u, v) \in E. \tag{2}$$

Since the latencies are random variables, the edge weights are random variables also. We can now compute the throughput for the graph by computing its *maximum cycle mean* (MCM) [Mathur et al. 1998]. The cycle mean (CM) of a cycle C in $G(V, E)$ is defined as the sum of the weights of the edges in the cycle divided by the number of edges in the cycle. The MCM can then be computed by determining the maximum value of the cycle mean over all cycles in the graph. The throughput for the graph is then inversely proportional to the MCM. Formally, if λ^* is the throughput for $G(V, E)$, then:

$$\lambda^* = \max_{C \in G} \frac{|C|}{\sum\limits_{(u,v) \in C} w(u, v)}, \tag{3}$$

where $|C|$ represents the number of edges in cycle C. Mathur et al. [1998], use Karp's algorithm [Karp 1978] to compute the MCM. According to Karp's



	0	1	2	3
s	0	-	-	12
a	-	3	-	12
b	-	-	7	-

Fig. 2. An example graph with four vertices and annotated edge weights. The table shows how the distance variables in Karp's algorithm are computed.

algorithm, the MCM ($\Delta^* = \frac{1}{\lambda^*}$) is given as:

$$\Delta^* = \frac{1}{\lambda^*} = \max_{v \in V} \min_{0 \le k \le n-1} \frac{D_v^n - D_v^k}{n-k}, \tag{4}$$

where $D_v^k (0 \le k \le N)$ is defined as the maximum k step distance between node v and an arbitrarily picked node $s \in V$ in the graph. This can be computed by enumerating all paths between s and v that contain exactly k edges and picking the path that has the maximum sum of edge weights. The algorithm begins by $D_s^0 = 0$ (since the node s can reach itself in zero steps) and $D_i^0 = -\infty$ for every other node $i \in V$. Now D_v^k can be computed for $1 \le k \le n$ and all $v \in V$ using the following recurrence relation

$$D_v^k = \max_{u \in V, (u,v) \in E} \left(D_u^{k-1} + w(u,v) \right). \tag{5}$$

This completes the description of Karp's MCM algorithm for *fixed* edge weights. Before proceeding further, we will now demonstrate how Karp's algorithm works using a simple example. Consider the graph shown in Figure 2 with nodes a, b, and s. As mentioned earlier, each edge is assigned a weight equal to the latency of the source node of that edge; in our example these weights have been randomly picked. Looking at the graph, it is clear that it contains only two cycles, $C_1 = (s \to a \to b \to s)$ and $C_2 = (a \to b \to a)$. The cycle means of C_1 and C_2 can therefore be calculated as $\frac{5+4+3}{3} = 4$ and $\frac{5+4}{2} = 4.5$ respectively. Finally the maximum cycle mean is given by $max(4, 4.5) = 4.5$.

To see how the same result can be obtained from Karp's algorithm, in Figure 2, we first compute and tabulate the values of the distance variable, D_v^k, for $k \in [0, 3]$ and $v \in \{s, a, b\}$ using Equation (5). To further illustrate how the distance variables are computed, the table in Figure 2 has the value of distance variable D_b^2 marked specifically ($D_b^2 = 7$), along with the path that is followed from source node s to node b in the component graph. It can be verified that this is indeed the longest path, and in this example the only path, from node s to node b that traverses exactly two edges. We can now use the computed values of the distance variables in Equation (4) to calculate the maximum cycle mean as $max(\frac{12-3}{2}, \frac{12-0}{3}) = 4.5$. Note that not only does Karp's algorithm provide the correct result, but it also computes the final *max* over exactly the same values as those determined via visual inspection.

The discussion so far has demonstrated how the MCM of a component graph can be computed for fixed values of edge weights in an SCC. We now shift our focus to the primary contribution of this work—computing the MCM for graphs where the edge weights (derived from the node latencies of the component graph) are random variables. Before proceeding, we note that it is critically important for a statistical version of Karp's MCM algorithm to keep track of *correlations* between the distance variables (D_v^k for all $v \in V$ and $1 \leq k \leq n$). Unlike SSTA, that needs to operate only on independent variables *if* there are no structural or spatial correlations between critical paths, statistical MCM *always* needs to account for correlations. This is due to the existence of the term $(D_v^n - D_v^k)$ in Equation (4). Specifically, the variables D_v^n and D_v^k will *always* be correlated for throughput constraining cycles in the graph, since the edges represented in D_v^k will be a subset of those in D_v^n. To ensure that we keep track of correlations at all stages in the algorithm, we represent all intermediate random variables in the algorithm as linear functions of the input random variables, and use a moment matching based scheme [Zhan et al. 2005] to propagate the random variables across operations.

For each random variable T_i, we introduce a new random variable $T_i^{'}$ that is a normalized version of T_i. If μ_{T_i} is the mean of T_i and σ_{T_i} is its standard deviation, we can write $T_i^{'}$ as:

$$T_i^{'} = \frac{T_i - \mu_{T_i}}{\sigma_{T_i}}. \tag{6}$$

The *cdf* of $T_i^{'}$ can now be written in terms of $F_{T_i}(t)$ as:

$$F_{T_i^{'}}(t) = F_{T_i}(\sigma_{T_i} t + \mu_{T_i}). \tag{7}$$

Since the only random variables we take as input to our algorithm are the cycle times T_i, or equivalently $T_i^{'}$, we would like to express the intermediate variables D_v^k for $1 \leq k \leq n$ and for all $v \in V$ as:

$$D_v^k = a_{v,0}^k + \sum_{1 \leq i \leq n} a_{v,i}^k T_i^{'}, \tag{8}$$

where the coefficients $a_{v,i}^k \in \Re$ for $0 \leq i \leq n$. The goal is to determine these coefficients for D_v^k. We start by assigning $D_s^0 = 0$ as in Karp's MCM algorithm by setting $a_{s,i}^0 = 0$ for $0 \leq i \leq n$. We can now solve the recurrence relationship to get

$$D_v^k = \max_{u \in V, (u,v) \in E} \left(a_{u,0}^{k-1} + \sum_{1 \leq i \leq n} a_{u,i}^{k-1} T_i^{'} + C_u(\sigma_{T_u} T_u^{'} + \mu_{T_u}) \right), \tag{9}$$

but we also know that:

$$D_v^k = a_{v,0}^k + \sum_{1 \leq i \leq n} a_{v,i}^k T_i^{'}. \tag{10}$$

We now need to determine the coefficients $a_{v,i}^k$ for $0 \leq i \leq n$. Without any loss in generality, consider a generic *max* function that takes as input two variables

A and B that are linear combinations of the random variables $T_i^{'}$, $1 \leq i \leq n$:

$$D = max(A, B) = max\left(\alpha_0 + \sum_{1\leq i\leq n} \alpha_i T_i^{'}, \beta_0 + \sum_{1\leq i\leq n} \beta_i T_i^{'}\right). \tag{11}$$

We want to write:

$$D = \gamma_0 + \sum_{1\leq i\leq n} \gamma_i T_i^{'}. \tag{12}$$

This can be accomplished by noting that:

$$E(T_i^{'}D) = \gamma_i = E(T_i^{'}max(A, B)), \forall(1 \leq i \leq n) \tag{13}$$

and

$$E(D) = \gamma_0 = E(max(A, B)), \tag{14}$$

where $E(X)$ represents the expectation of random variable X. Exact algebraic expressions for the terms $E(max(A, B))$ and $E(T_i^{'}max(A, B))$ are provided by Zhan et al. [2005] and can be evaluated numerically. Having computed the coefficients for each D_v^k for $1 \leq k \leq n$ and $v \in V$, we can now rewrite Equation (4) as:

$$\frac{1}{\lambda^*} = \max_{v\in V} \min_{0\leq k\leq n-1} \frac{a_{v,0}^n - a_{v,0}^k + \sum_{1\leq i\leq n} (a_{v,i}^n - a_{v,i}^k) T_i^{'}}{n-k}. \tag{15}$$

This equation needs, again, a series of *max* and *min* operations over inputs that are linear combinations of random variables, where, at each stage we express the output as another linear combination over the same random variables. Even though we have only described in detail how this can be done using moment matching for the *max* operation, the expressions for the *min* operation can be derived in exactly the same fashion as for the *max* operation.

Algorithm 1 is the formal description of our proposed technique and yields the desired coefficients δ_i, where $(0 \leq i \leq n)$, that allow us to write:

$$\Delta^* = \frac{1}{\lambda^*} = \delta_0 + \sum_{i=1}^{n} \delta_i T_i^{'}. \tag{16}$$

We can now write the *cdf* of the random variable Δ^*, as[1]:

$$F_{\Delta^*}(\tau) = F_{T_1^{'}}\left(\frac{\tau - \delta_0}{\delta_1}\right) * F_{T_2^{'}}\left(\frac{\tau - \delta_0}{\delta_2}\right) * \ldots * F_{T_n^{'}}\left(\frac{\tau - \delta_0}{\delta_n}\right). \tag{17}$$

Before proceeding further, we now demonstrate using a simple example how the *cdf* of the maximum cycle mean of a given component graph is derived using

[1]Throughout this paper, we use $*$ operator to denote the convolution operation.

Algorithm 1. Statistical MCM

```
  *Inputs:* Number of cycles for each PE ($C_i$), *cdfs* of cycle time random variables
($T_i, T_i^{'}$).
  *Outputs:* $\delta_i; \forall i : 0 \leq i \leq n$
  **for** $k = 0$ to $n$ **do**
    **for** each node $v \in V$ **do**
      $a_{v,i}^{k} = -\infty; \forall i : 0 \leq i \leq n$
    **end for**
  **end for**
  $a_{s,i}^{0} = 0, \forall i : 0 \leq i \leq n$
  **for** $k = 1$ to $n$ **do**
    **for** each node $v \in V$ **do**
      $t_i = 0; \forall i : 0 \leq i \leq n$
      **for** each node $u$ s.t. $(u, v) \in E$ **do**
        $A = t_0 + \sum_{i=1}^{n} t_i T_i^{'}$
        $B = a_{u,0}^{k-1} + \sum_{i=1}^{n} a_{u,i}^{k-1} T_i^{'} + \sigma_{T_u} C_u T_u^{'} + \mu_{T_u} C_u$
        $t_i = E(T_i^{'} \max(A, B)); \forall i : 1 \leq i \leq n$
        $t_0 = E(\max(A, B))$
      **end for**
      $a_{v,i}^{k} = t_i; \forall i : 0 \leq i \leq n$
    **end for**
  **end for**
  $\delta_i = 0; \forall i : 0 \leq i \leq n$
  **for** each node $v \in V$ **do**
    $t_i = \infty; \forall i : 0 \leq i \leq n$
    **for** $k = 0$ to $n - 1$ **do**
      $A = t_0 + \sum_{i=1}^{n} t_i T_i^{'}$
      $B = \frac{a_{v,0}^{n} - a_{v,0}^{k}}{n-k} + \sum_{i=1}^{n} \left( \frac{a_{v,i}^{n} - a_{v,i}^{k}}{n-k} \right) T_i^{'}$
      $t_i = E(T_i^{'} \min(A, B)); \forall i : 1 \leq i \leq n$
      $t_0 = E(\min(A, B))$
    **end for**
    $C = \delta_0 + \sum_{i=1}^{n} \delta_i T_i^{'}$
    $D = t_0 + \sum_{i=1}^{n} t_i T_i^{'}$
    $\delta_i = E(T_i^{'} \max(C, D)); \forall i : 1 \leq i \leq n$
    $\delta_0 = E(\max(C, D))$
  **end for**
```

the statistical version of Karp's MCM Algorithm. Figure 3(a) shows an example component graph with three PEs. Each PE is annotated with its cycle count and the mean and variance of its cycle time distribution (which is assumed to be a normal distribution). Furthermore, it is assumed that each PE maps to a different VFI, that is, the system has three VFIs. In the first step, we normalize the cycle time random variables (T_i) to zero-mean unit-variance random variables

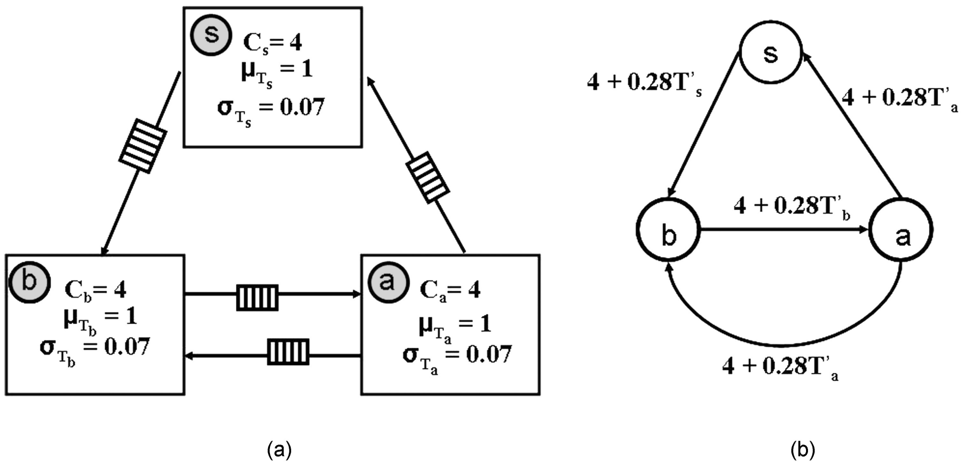


Fig. 3. (a) A three-PE component graph. Each PE is annotated with its cycle count and the mean and variance of its cycle time distribution. (b) Equivalent directed cyclic graph representing the component graph in (a). Each edge is annotated with an edge weight which is equal to the latency of the PE from which the edge originates.

as shown in Equation (6) to obtain the normalized variables T_i'. The latency of each PE is then computed in terms of these normalized random variables to obtain the edge weights of all the outgoing edges of each PE. This yields a directed cyclic graph, shown in Figure 3(b) with annotated edge weights that can now be used as an input to the statistical MCM algorithm.

The first step of the statistical MCM algorithm is to compute the distance variables, D_v^k, using the recurrence relationship in Equation (5). It is important to note that unlike the previous example in which the table of distance variables consisted of fixed numbers (because the inputs to the algorithm were fixed numbers) each entry of the distance variable table will be a linear function of the normalized cycle time random variables T_i'. Figure 4(a) shows the distance variable table for this example. Note that the distance variable table implicitly contains the values of $a_{v,i}^k$ coefficients, which are simply the coefficients of the normalized cycle time random variables of the appropriate distance variable. For example, since $D_s^3 = 12 + 0.28T_s' + 0.28T_a' + 0.28T_b'$, the corresponding coefficients take the values $a_{s,0}^3 = 12$, $a_{s,1}^3 = 0.28$, $a_{s,2}^3 = 0.28$ and $a_{s,3}^3 = 0.28$. Now, using Equation (4), we can write the MCM of the given component graph as:

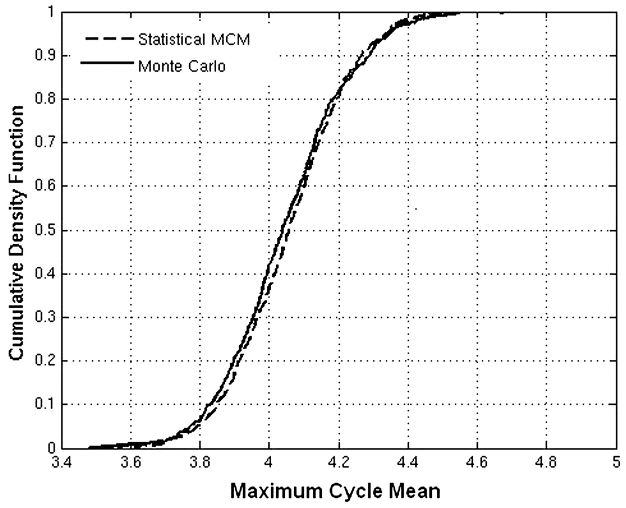
$$\Delta^* = \max\left(\frac{D_s^3 - D_s^0}{3-0}, \frac{D_b^3 - D_b^1}{3-1}\right). \tag{18}$$

Furthermore, by substituting the expressions corresponding to the distance variables in Equation (18) from the distance variable table in Figure 4, we get:

$$\Delta^* = \max(4 + 0.093T_s' + 0.093T_a' + 0.093T_b', 4 + 0.14T_a' + 0.14T_b'). \tag{19}$$

	0	1	2	3
s	0	--	--	$12 + 0.28T'_s + 0.28T'_a + 0.28T'_b$
a	--	--	$8 + 0.28T'_s + 0.28T'_b$	--
b	--	$4 + 0.28T'_s$	--	$12 + 0.28T'_s + 0.28T'_a + 0.28T'_b$

(a)



(b)

Fig. 4. (a) The distance variable table for the component graph in Figure 3. (b) The computed *cdf* of the maximum cycle mean of the component graph in Figure 3 versus the *cdf* obtained for the same system using Monte-Carlo simulations.

Finally, using the proposed moment matching approach, Δ^* can further be approximated as:

$$\Delta^* = 4.052 + 0.065T'_s + 0.107T'_a + 0.107T'_b. \tag{20}$$

Since Δ^* has been approximated as a weighted sum of Gaussian random variables, it is also a Gaussian random variable whose mean and variance are easily computed. In Figure 4(b), we plot the *cdf* of Δ^* as computed in Equation (20) (Statistical MCM) versus the *cdf* of maximum cycle mean obtained from 10,000 runs of Monte-Carlo simulations for the component graph. We observe that the two *cdfs* are almost indistinguishable. In addition, note that besides approximating the *cdf* of the maximum cycle mean of the given component graph, Equation (20) also yields the sensitivity of the maximum cycle mean distribution to the cycle times of each of the PEs. Specifically, note that in Equation (20), the random variable representing the cycle time of PE b, that is, T'_b, has the minimum weight. This is to be expected since PE b only contributes to the cycle mean of one of the two cycles in the component graph, while the other two PEs impact both cycles.

The final step of the statistical MCM algorithm is to compute the *cdf* for the throughput of G, represented as $F_{\lambda^*}(\lambda)$, using the following expression:

$$1 - F_{\lambda^*}(\lambda) = F_{T_1'}\left(\frac{1-\delta_0\lambda}{\delta_1\lambda}\right) * F_{T_2'}\left(\frac{1-\delta_0\lambda}{\delta_2\lambda}\right) * \ldots F_{T_n'}\left(\frac{1-\delta_0\lambda}{\delta_n\lambda}\right). \tag{21}$$

Finally, the description above applies to a component graph that is an SCC. If there are more than one such SCCs in the graph, we run the steps described above on each SCC individually and obtain the *cdfs* of the throughput for each SCC. These *cdfs* can then be combined using a simple statistical *min* operation to yield the final result. If $F_{\lambda_i^*}(\lambda)$ represents the *cdf* for the ith SCC in the graph, we can write the *cdf* of throughput for the entire graph by taking the statistical minimum across the throughput distributions from each SCC. If X, Y, and Z are some arbitrary random variables, and $Z = \min(X, Y)$, the *cdf* of Z can be written in terms of the *cdf* of X and Y as:

$$1 - F_Z(z) = (1 - F_X(z))(1 - F_Y(z)). \tag{22}$$

We can therefore write:

$$1 - F_{\lambda^*}(\lambda) = \left(1 - F_{\lambda_1^*}(\lambda)\right)\left(1 - F_{\lambda_2^*}(\lambda)\right)\ldots\left(1 - F_{\lambda_m^*}(\lambda)\right). \tag{23}$$

Equations (13), (14), (17), (21), and (23) can be computed efficiently using the techniques outlined by Devgan and Kashyap [2003]. This completes the description of the proposed algorithm. We note that the time complexity of the algorithm is $O(p|V||E|)$, since Karp's MCM is itself $O(|V||E|)$ [Mathur et al. 1998], and we replace the *max* function in Karp's MCM with the computation of p coefficients (the description in this section assumes $p = n$, but in the general case $p \leq n$).

5.2 Throughput Analysis for SSV Systems

An SSV system has only one global clock frequency. Let the cycle time of the global clock be T_G. Since T_G is constrained by the cycle times of each of the individual PEs, we can write

$$T_G = \max_{1\leq i\leq n} T_i. \tag{24}$$

If we assume that the individual cycle times vary independently due to random variations we can write the *cdf* of T_G as

$$F_{T_G}(t) = F_{T_1}(t).F_{T_2}(t)\ldots F_{T_n}(t). \tag{25}$$

In the previous section, we outlined an algorithm to compute the distribution of λ^* given the input latencies $L_i = C_i T_i$ for all nodes in V. Formally:

$$\lambda^* = Q(C_1T_1, C_2T_2 \ldots C_nT_n), \tag{26}$$

where the function $Q(.)$ represents the proposed probabilistic version of Karp's MCM algorithm. We note that $Q(ax, ay) = aQ(x, y)$ since scaling the latency

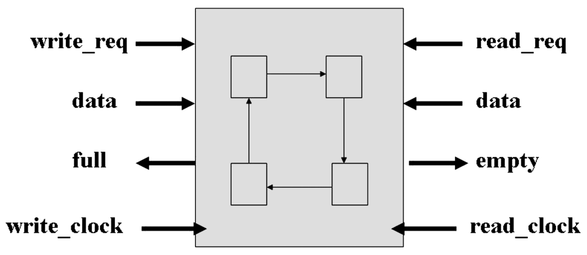


Fig. 5. Black box model of the mixed-clock FIFO.

of each node by a fixed amount can only scale the output by the same amount. Now, for an SSV system

$$\lambda^*_{SSV} = Q(C_1.T_G, C_2.T_G \ldots C_n.T_G) \tag{27}$$

$$= T_G.Q(C_1, C_2, \ldots, C_n). \tag{28}$$

Since the cycle counts are single values, and we have already computed the *cdf* (and therefore *pdf*) of T_G in Equation (25), we just need a single run of the classic Karp's MCM algorithm over input values that are fixed numbers.

5.3 Communication Overhead

We now discuss how we model the communication overhead of crossing clock domain boundaries in the case of multiple VFI designs. As mentioned before, we assume that in the case of a fully synchronous design, PE i $(1 \leq i \leq n)$ uses C_i^{comm} cycles out of its total number of execution cycles C_i to write data to the input buffers of its consumer nodes. The amount of data transferred from the producer to the consumer in a single cycle is referred to as a *token* of data. Now, if the producer (node i) and consumer (node j) lie in different clock-domains, instead of writing directly (and synchronously) to the input buffer of the consumer node, the producer node will have to communicate C_i^{comm} data tokens to the consumer node j via a mixed-clock FIFO. Figure 5 shows a black-box model of the mixed-clock FIFO that we use in the multiple VFI design. For a more detailed description of the design, we refer the reader to Chelcea and Nowick [2001]; however, for the purpose of this discussion, it is important to point out that the FIFO *write_clock* signal is driven by the producer clock-domain while the FIFO *read_clock* signal is driven by the consumer clock. Furthermore, in the steady state, the producer can enqueue a data item in every producer clock cycle and the consumer can dequeue a data item in every consumer clock cycle. Consider a case when the producer node i is running slower than the consumer node j, that is, $T_i > T_j$, and the producer tries to send C_i^{comm} tokens of data to the consumer. In this case, even though the consumer may stall occasionally due to an *empty* signal from the FIFO, the producer will always be able to insert a data token into the FIFO in every producer clock cycle. A similar argument can be made if the consumer runs slower than the producer node, in which case the consumer will always be able to withdraw a data token from the FIFO in every clock cycle. Therefore, the mixed-clock FIFO is able to offer a bandwidth dictated by the *maximum* of the cycle times of the producer and consumer clocks.

Table I. Results for Synthetic Benchmarks

Benchmark	n	p	$\alpha = 0.1$		$\alpha = 0.3$		$\alpha = 0.5$		Speed-up
			Error(μ)	Error(Y.P.)	Error(μ)	Error(Y.P.)	Error(μ)	Error(Y.P.)	
synth-15	15	5	0.38%	0.59%	0.23%	0.87%	0.36%	0.09%	260X
synth-30	30	10	0.44%	1.88%	0.55%	2.1%	0.29%	1.27%	147X
synth-45	45	15	1.18%	0.41%	1.36%	0.24%	1.31%	0.68%	97X
synth-60	60	20	1.22%	0.06%	1.05%	0.33%	1.30%	0.55%	78X

As a result, the latency incurred in sending C_i^{comm} tokens of data from the producer to the consumer can now be written as $C_i^{comm} * \max(T_i, T_j)$. This has been experimentally verified using cycle-accurate simulations of a mixed-clock FIFO by Niyogi and Marculescu [2005b].

To account for this additional communication latency, we need to modify the weights of the edges in the component graph. Recall that in Section 5.1, we assign the weight to the edge between nodes i and j, $w(i, j) = L_i = C_i T_i$, for all $(i, j) \in E$. This is now modified as follows:

$$w(i, j) = C_i^{comp} T_i + C_i^{comm} \max(T_i, T_j), \forall (i, j) \in E. \tag{29}$$

Note that Equation (29) can be directly incorporated into the proposed Statistical MCM framework by using the described moment matching approach to represent $w(i, j)$ $(\forall (i, j) \in E)$ as a weighted linear combination of the T_i and T_j (equivalently $T_i^{'}$ and $T_j^{'}$) random variables.

6. EXPERIMENTAL RESULTS

We implemented the proposed algorithm for determining the throughput distribution of single and multiple voltage-frequency island systems in C, and ran experiments on a workstation equipped with a 2.4 GHz Intel Pentium Processor running a Linux OS. The implementation takes as input the component graph for the given application, the number of execution cycles and the cycle time distribution for each PE in the graph, the number of clock domains and the allocation of PEs to clock domains and provides the *cdf* of the throughput for the application. There is, unfortunately, an acknowledged lack of embedded system benchmarks that have cyclic component graphs [Stuijk et al. 2006]. We therefore validate the accuracy and efficiency of our proposed techniques on a set of *synthetic benchmarks* that are generated using the algorithm presented by Stuijk et al. [2006]. We then demonstrate the impact of our proposed analysis framework on the design of multiple VFI systems with a case study on a real embedded benchmark (MPEG-2 encoder). All results are compared to an exhaustive simulation that consists of 10,000 runs of Monte Carlo simulation [Zhan et al. 2005; Devgan and Kashyap 2003; Chang and Sapatnekar 2003].

6.1 Synthetic Benchmarks

Stuijk et al. [2006] outline an algorithm to generate cyclic component graphs with specified properties. Using this approach, we generate a set of four synthetic benchmarks that we label *synth-1,synth-2,synth-3, and synth-4*. We vary

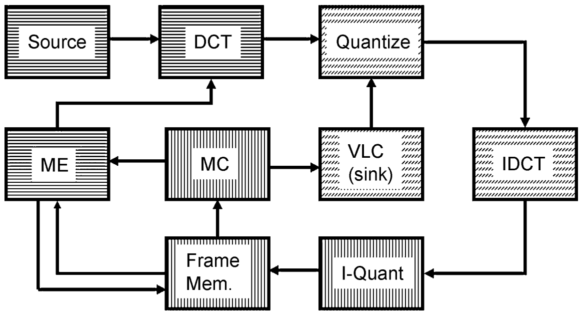


Fig. 6. The MPEG-2 Encoder Benchmark. For MCV-3, nodes with similar background patterns are clustered into VFIs.

the number of PEs (n) in the graphs from 15 to 60 and the number of clock domains (p) from 5 to 20, since these numbers are, in our opinion, representative of the increasingly complex SOC designs that will emerge in future scaled technologies. The number of execution cycles for each PE is chosen randomly from a uniform distribution between 50 and 100. Furthermore, for each benchmark, we vary the communication to computation ratio of the PEs, as defined in Equation (1), by setting $\alpha_i = (0.1, 0.3, 0.5)$ for all $i \in (1, n)$. Finally, we assume that the cycle times of the PEs are normally distributed with a 3σ of 20% of the mean [Marculescu and Garg 2006]. Table I shows the error between the mean and the 99% yield points of the throughput distributions for each of the benchmarks for different values of the computation to communication ratio. We note that the average error in the mean of the throughput distribution, compared to the Monte Carlo results, is **0.80%** (maximum **1.36%**) and the average error in the 99% yield point is **0.71%** (maximum **2.1%**). This comes at a speed-up ranging from **78X** to **260X** (average **145X**).

6.2 Case Study: MPEG-2 Encoder

The results from the previous section demonstrate that the proposed technique is able to accurately estimate the throughput distribution of multiple VFI systems with an appreciable speed-up in runtime. Using an example of an MPEG-2 encoder [Carloni and Sangiovanni-Vincentelli 2000], we now demonstrate how such a framework can be used by system level designers to evaluate multiple VFI systems. Figure 6 shows the component graph of the MPEG-2 encoder. To determine the number of execution cycles for each of the components, we simulated a software version of the MPEG-2 encoder on an ARM7TDMI core using the publicly available *sunflower* embedded system simulation tool [Stanley-Marbell and Marculescu 2007] (the tool performs cycle-accurate simulation of embedded systems consisting of multiple ARM7TDMI or Hitachi SuperH cores) to obtain cycle counts for each module. We note that for the software implementation we used, the DCT and Quantizer modules were implemented together, as were the IDCT and IQ modules. Instead of rewriting the software to separate the two modules, we divided the cycle counts equally between the modules that were implemented together. The cycle counts for each of the MPEG-2 encoder

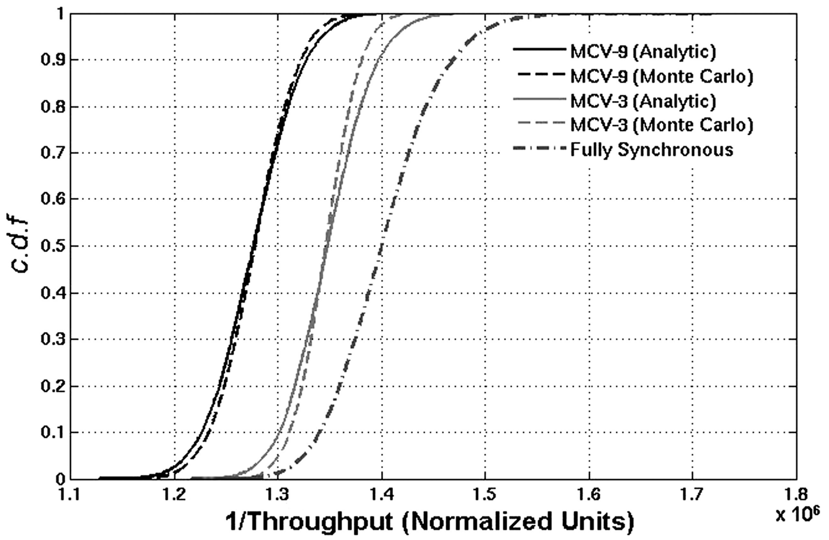


Fig. 7. *cdfs* of throughput obtained for the MPEG-2 encoder benchmark for three different architectures.

modules are shown in Table II. As in the previous section, we assumed the cycle time of each PE to be normally distributed with a 3σ of 20% of the mean (by 2008 and beyond, gate length variation, and thus delay variation, will reach and possibly exceed this limit [Nassif 2000]).

Using these simulation parameters, we considered three possible implementations of the MPEG-2 encoder: MCV-9, MCV-3, and SSV. MCV-9 is a nine clock domain architecture in which each PE lies in its own VFI and has a mean frequency of 133 MHz (the nominal frequency of the ARM7TDMI core simulated). MCV-3 has three VFIs, with three PEs in each VFI, as represented by the shaded regions in Figure 6. The mean frequency of each VFI for the MCV-3 architecture is 129.3 MHz, which is, as expected, lower than the MCV-9 architecture since the frequency of each VFI in MCV-3 is limited by the *slowest* of the three PEs in that VFI. Finally, SSV is a fully synchronous design with a single clock domain that runs at a mean frequency of 121 MHz.

Figure 7 shows the *cdf* of throughput obtained (normalized to the nominal cycle time of a PE) using our approach and using Monte Carlo simulations for each of the three architectures (for SSV the proposed method always yields exact results and therefore we only show the Monte Carlo curve for SSV). The results allow us to quantify the yield of any of the three designs for a given throughput constraint. We can see that for a throughput constraint that gives 50% yield for a fully synchronous system, a nine clock domain architecture (MCV-9) achieves

Table II. Cycle Counts for Each Component in the MPEG-2 Encoder Example

MPEG-2 Encoder	Src	DCT+Q	ME+MC	VLC	IDCT+IQ	F.M.
Cycles/Macro-block	3188	370060	101282	43222	351259	16722

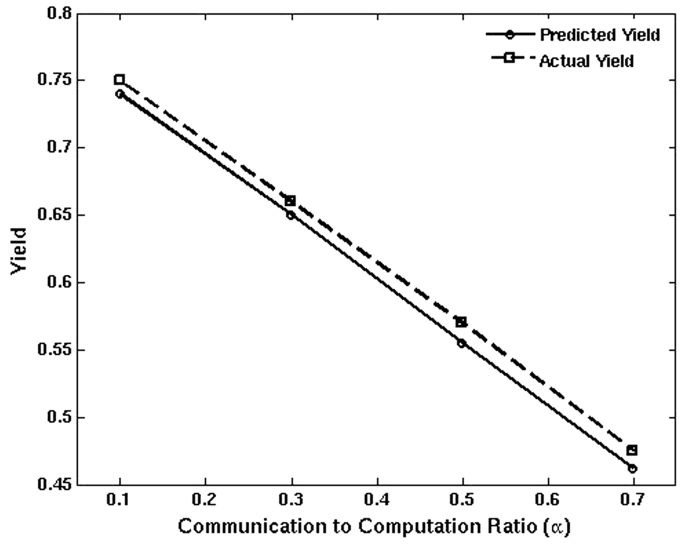


Fig. 8. Predicted and actual yield of the MCV-9 architecture as a function of the communication to computation ratio α for each PE.

100% yield (proposed scheme also predicts **100%**) while a three clock domain architecture (MCV-3) achieves **98%** yield (proposed scheme predicts **92%**). The improvements are more dramatic when we consider the 25% yield point of the SSV architecture- MCV-9 again achieves **100%** yield (predicted **99.8%**) while MCV-3 is able to achieve **77%** yield (predicted **71%**). Such information could be used by designers, in conjunction with the throughput constraints that the design is expected to meet, to decide on the number of clock domains for their design or even choose between a fully synchronous and a multiple VFI design style.

We point out that it is not *always* the case that multiple VFI designs are better than their fully synchronous counterparts—for example, if the throughput of the component graph is determined only by the cycle time of the slowest PE in the system, both the MCV and SSV design styles will have exactly the same performance distribution in the absence of any overheads due to crossing clock domains, and MCV designs may actually perform *worse* than SSV designs if these are accounted for. To demonstrate the impact of the overhead of crossing clock domain boundaries on the performance of VFI systems, we performed additional experiments on the MPEG-2 encoder benchmark in which we increased the communication to computation ratio, α, of each PE from 0.1 to 0.7 in steps of 0.2 for the nine clock-domain architecture (MCV-9). In Figure 8, we plot the actual (obtained from Monte-Carlo simulations) and predicted (from the proposed analytical techniques) yield for the MCV-9 architecture as a function of the communication to computation ratio for a throughput constraint that provides 75% yield for $\alpha = 0.1$. As expected, the yield decreases with increasing communication to computation ratio, falling to about 46% for $\alpha = 0.7$. We note that since the nine clock-domain architecture has the finest granularity of VFI partitioning, it is the most susceptible to performance loss due to the overhead

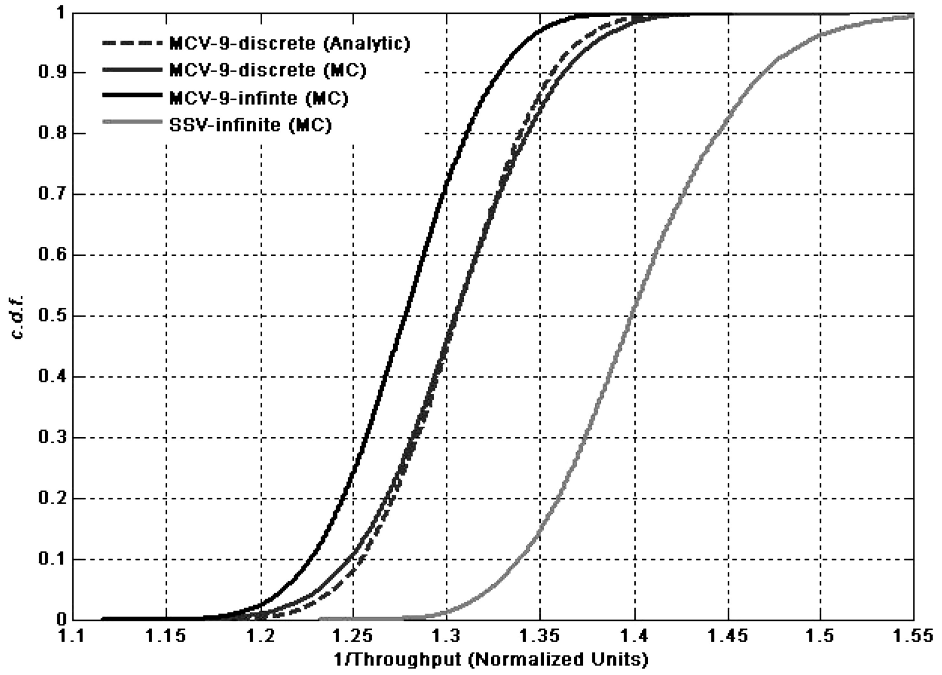


Fig. 9. *cdfs* of throughput obtained for the MPEG-2 encoder benchmark for MCV-9 with a continuous range of frequency values (MCV-9-infinite), MCV-9 with a discrete set of frequency values (MCV-9-discrete) and a fully-synchronous design with a continuous range of frequency values (SSV-infinite). For MCV-9-discrete, we plot *cdfs* obtained from Monte-Carlo simulations as well as from the proposed analytical framework.

of crossing clock domains; in fact, if the volume of communication across clock boundaries is high enough, it is possible that the performance gained by moving to a fine-grain VFI implementation may be entirely eclipsed by the loss due to communication over mixed-clock interfaces.

Finally, to determine the performance impact of controlling the VFI clocks in discrete steps, we performed experiments for the MCV-9 architecture in which each clock domain was only allowed to select from a discrete set of ten equally spaced frequency values. In Figure 9, we plot the *cdf* of throughput obtained for the MCV-9 architecture with a continuous range of frequency values for each VFI (this *cdf* was also plotted in Figure 7 and is replicated here for comparison purposes) along with the *cdfs* obtained from Monte Carlo simulations and the proposed analytical techniques for the case in which only discrete frequency values are allowed. Finally, we also plot the *cdf* of throughput for the fully synchronous design (SSV) with a continuous range of frequency levels. From the figure, we can see that the *cdfs* for the discrete case obtained from Monte Carlo simulations and from the proposed framework are almost indistinguishable. Furthermore, even though the restriction of choosing from a discrete range of frequency values leads to some performance degradation with respect to the case in which we allow a continuous range of frequency values, the performance of the MCV-9 design with discrete frequency steps is still significantly superior to the SSV design with *infinite* frequency levels.

7. CONCLUSIONS AND FUTURE WORK

In this article, we propose a framework that allows embedded-system designers to analyze the performance impact of WID variability on their designs at the *system level*, with a specific focus on variability adaptive multiple VFI designs. Using the proposed framework, designers can choose between either a fully synchronous design or an equivalent multiple voltage-frequency island implementation based on the extent of improvement in execution rate or throughput yield that migrating to a multiple VFI design provides them. Furthermore, as shown in the experimental results section, the proposed techniques can also be used to examine the trade-off between performance yield and granularity of VFI island partitioning. Our results indicate that multiple VFI implementations can offer a significant performance improvement over fully-synchronous designs in the face of WID manufacturing process variations. Specifically, we show that for a throughput constraint for which a fully synchronous implementation of an MPEG-2 encoder benchmark yields only 50%, a nine clock domain architecture (MCV-9) can yield 100% while a three clock domain design (MCV-3) would yield 99.8%. While these results assume that the each VFI can precisely tune its frequency to any value within a given range, we also show that moving to a more practical implementation in which the frequency can only be set in discrete steps is still significantly superior to a fully synchronous implementation with precise frequency control.

As future work, we intend to explore more sophisticated models of WID variability that include systematic and correlated sources of variations on process parameters. We would also like to examine the impact a VFI design style could have on reducing variability in circuit leakage power dissipation.

REFERENCES

Belete, D., Razdan, A., Schwarz, W., Raina, R., Hawkins, C., and Morehead, J. 2002. Use of DFT techniques in speed grading a 1 GHz+ microprocessor. In *Proceedings of the International Test Conference*. 1111–1119.

Bowman, K., Duvall, S., and Meindl, J. 2002. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE J. Solid-State Circ. 37,* 2.

Carloni, L. and Sangiovanni-Vincentelli, A. 2000. Performance analysis and optimization of latency insensitive systems. In *Proceedings of the 37th Conference on Design Automation*. 361–367.

Chang, H. and Sapatnekar, S. 2003. Statistical timing analysis considering spatial correlations using a single pert-Like traversal. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 621–625.

Chapiro, D. 1984. Globally-asynchronous locally-synchronous systems. Ph.D. thesis, Stanford University.

Chelcea, T. and Nowick, S. 2001. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In *Proceedings of the 38th Conference on Design Automation*. 21–26.

Datta, A., Bhunia, S., Banerjee, N., and Roy, K. 2005. A power-aware GALS architecture for real-time algorithm-specific tasks. In *Proceedings of the 6th International Symposium on Quality of Electronic Design*. 358–363.

de Veciana, G. and Jacome, M. 1998. Hierarchical algorithms for assessing probabilistic constraints on system performance. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*.

Devgan, A. and Kashyap, C. 2003. Block-based static timing analysis with uncertainty. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 607–614.

Ernst, D., Flautner, K., Mudge, T., Kim, N., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., et al. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 7–18.

Friedberg, P., Cheung, W., and Spanos, C. 2006. Spatial modeling of micron-scale gate length variation. In *Data Analysis and Modeling for Patterning Control III*. SPIE, vol. 6155.

Hemani, A., Meincke, T., Kumar, S., Postula, A., Olsson, T., Nilsson, P., Oberg, J., Ellervee, P., and Lundqvist, D. 1999. Lowering power consumption in clock by using globally asynchronous locally synchronous design style. In *Proceedings of the 36th ACM/IEEE Conference on Design Automation Conference*, 873–878.

Hu, X. S., Zhou, T., and Sha, E. H.-M. 2001. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Trans. TVLSI*.

Hung, W.-L., Wu, X., and Xie, Y. 2006. Guaranteeing performance yield in high-level synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 303–309.

Karp, R. 1978. A characterization of the minimum cycle mean in a digraph. *Discrete Math 23*.

Kulkarni, S., Sylvester, D., and Blaauw, D. 2006. A statistical framework for post-silicon tuning through body bias clustering. *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 39–46.

Lackey, D., Zuchowski, P., Bednar, T., Stout, D., Gould, S., and Cohn, J. 2002. Managing power and performance for system-on-chip designs using voltage islands. *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, 195–202.

Liang, X., Canal, R., Wei, G.-Y., and Brooks, D. 2007. Process variation tolerant 3t1d-based cache architectures. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*. 15–26.

Marculescu, D. and Garg, S. 2006. System level process-driven variability analysis for single and multiple voltage-frequency islands. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 541–546.

Marculescu, D. and Iyer, A. 2002. Power and performance evaluation of globally asynchronous locally synchronous processors. *Proceedings of the 29th International Symposium on Computer Architecture (ISCA-02)*. *30*, 158–168.

Marculescu, D. and Talpes, E. 2005. Variability and energy awareness: a microarchitecture-level perspective. In *Proceedings of the 42nd Annual Conference on Design Automation*. 11–16.

Mathur, A., Dasdan, A., and Gupta, R. 1998. Rate analysis for embedded systems. *ACM Trans. Des. Automat. Electron. Syst. 3,* 3, 408–436.

Nassif, S. 2000. Design for manufacturability in DSM technologies. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*.

Niyogi, K. and Marculescu, D. 2005a. Speed and voltage selection for GALS systems based on voltage/frequency islands. In *Proceedings of the Conference on Asia South Pacific Design Automation*. 292–297.

Niyogi, K. and Marculescu, D. 2005b. System level power and performance modeling of GALS point-to-point communication interfaces. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 381–386.

Semeraro, G., Albonesi, D. H., Dropsho, S. G., Magklis, G., Dwarkadas, S., and Scott, M. L. 2002. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'35)*.

Stanley-Marbell, P. and Marculescu, D. 2007. Sunflower: Full-system, embedded microarchitecture evaluation. In *Proceedings of the International Conference on High Performance Embedded Architectures and Compilers (HiPEAC)*.

Stuijk, S., Geilen, M., and Basten, T. 2006. SDF 3: SDF for free. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design*. 276–278.

Teodorescu, R., Nakano, J., Tiwari, A., and Torrellas, J. 2007. Mitigating parameter variation with dynamic fine-grain body biasing. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*. 27–42.

Tiwari, A., Sarangi, S. R., and Torrellas, J. 2007. Recycle:: pipeline adaptation to tolerate process variation. In *Proceedings of the 34th annual international symposium on Computer architecture (ISCA'07)*. 323–334.

Tschanz, J., Kao, J., Narendra, S., Nair, R., Antoniadis, D., Chandrakasan, A., and De, V. 2002. Adaptive Body Bias for Reducing Impacts of Die-to-Die. *IEEE J. Solid-State Circ. 37,* 11.

Wang, F., Nicopoulos, C., Wu, X., Xie, Y., and Vijaykrishnan, N. 2007. Variation-aware task allocation and scheduling for mpsoc. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 598–603.

Wang, F., Wu, X., and Xie, Y. 2008. Variability-driven module selection with joint design time optimization and post-silicon tuning. In *Proceedings of the Conference on Asia and South Pacific Design Automation*. 2–9.

Zhan, Y., Strojwas, A., Li, X., Pileggi, L., Newmark, D., and Sharma, M. 2005. Correlation-aware statistical timing analysis with non-gaussian delay distributions. In *Proceedings of the 42nd Annual Conference on Design Automation*, 77–82.

Received April 2007; revised January 2008, April 2008; accepted May 2008