System-Level Process Variation Driven Throughput Analysis for Single and Multiple Voltage-Frequency Island Designs*

Siddharth Garg
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
sgarg1@andrew.cmu.edu

Diana Marculescu
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
dianam@ece.cmu.edu

Abstract

*Manufacturing process variations are the primary cause of timing yield loss in aggressively scaled technologies. In this paper, we analyze the impact of process variations on the throughput (rate) characteristics of embedded systems comprised of multiple voltage-frequency islands (VFIs) represented as component graphs. We provide an efficient, yet accurate method to compute the throughput of an application in a probabilistic scenario and show that systems implemented with multiple VFIs are more likely to meet throughput constraints than their fully synchronous counterparts. The proposed framework allows designers to investigate the impact of architectural decisions such as the granularity of VFI partitioning on their designs, while determining the likelihood of a system meeting specified throughput constraints. An implementation of the proposed framework is accurate within **1.2%** of Monte Carlo simulation while yielding speedups ranging from **78X-260X**, for a set of synthetic benchmarks. Results on a real benchmark (MPEG-2 encoder) show that a nine clock domain implementation gives **100%** yield for a throughput constraint for which a fully synchronous design only yields **25%**. For the same throughput constraint, a three clock domain architecture yields **78%**.*

1. Introduction

Manufacturing process variations have emerged as a major design concern for aggressively scaled technologies. These variations manifest themselves across a single die (WID or within-die) or across several dies (D2D or die-to-die). Furthermore, the source, and therefore the statistical nature, of these variations can be random or systematic, static or dynamic [4].

There is a significant body of work that analyzes the impact of process variations on static timing analysis [1, 3] at the circuit level. Such techniques are extremely useful in helping designers predict timing yield and in optimizing their designs at the circuit/gate level. There is, however, very little work that makes variability models available to system and microarchitecture level designers. In [11], the authors take a first step in that direction and introduce a framework that uses variability information from low level circuit analysis to build variability models for system level performance parameters, such as end-to-end system latency. Using this framework, they show that systems comprised of multiple voltage-frequency islands (VFI) are more likely to meet a specified *latency constraint* than their fully synchronous, single clock, single voltage (SSV) counterparts.

*This research was supported in part by Semiconductor Research Corporation contract no. 2005-HJ-1314.

VFI systems tend to outperform fully synchronous designs because they provide more flexibility in dealing with WID process variability. While the clock frequency of a fully synchronous design is limited by the slowest critical path on the entire chip, the clock frequency of each clock domain in a multiple VFI system is only limited by the slowest path in that particular domain.VFI systems are thereby able to isolate the impact of critical paths that have been negatively affected by process variations to the frequency domains in which these speed constraining paths lie.

In this work, we complement the analysis framework proposed in [11] and outline an algorithm to determine the *throughput* distribution of a multiple VFI system, given frequency distributions for each clock domain. Combined with results from [11], our proposed technique would allow designers to specify throughput *and* latency constraints for their designs and determine the percentage of manufactured chips that will meet these constraints.

2. Related Work

While statistical timing analysis has become a hot area of research over the past few years, the problem has, with the exception of [11], been addressed only at the gate/circuit level. Techniques have been proposed to deal with correlated gate delay distributions [3] and to provide stochastic bounds on worst-case circuit delay [4]. [11] provides stochastic bounds on the end-to-end latency of a directed acyclic task graph implemented using multiple VFIs. Our work extends the scope of [11] by analyzing graphs that have cyclic paths and consequently determining the throughput distribution of the design.

In [9], the authors 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. They propose a software based self-test scheme to run each PU at its optimal clock frequency. [5] proposes 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 maximum clock speed. Both works focus on implementation issues, while our focus is on *evaluating* the benefits of such implementations. We note that while [5] uses a fully synchronous design, the technique described could as easily be used in a multiple VFI system to adapt the clock speed of each frequency island.

978-3-9810801-2-4/DATE07 © 2007 EDAA

3. Paper Contributions

This work makes the following contributions:

- We consider the case of applications specified as *cyclic task (or component) graphs* implemented on multiple VFIs and derive distributions for the best case throughput (alternatively rate) 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.

Before proceeding further, we will now discuss the assumption we make about the hardware implementation of single and multiple VFI systems and introduce the mathematical notation that will be used in the rest of the paper.

4. Preliminaries and Assumptions

We consider the case of systems comprised of a number of synchronous cores, IPs or processing elements (PE). Henceforth, we will refer to all three generically as PEs. We now consider two cases: (1) A fully synchronous (or an SSV) system that has a single global clock that drives all the PEs. Communication between PEs is assumed to be point-to-point and synchronous. (2) A multiple VFI system in which each voltage-frequency island is controlled by an independent clock source. Each clock domain (or VFI) can have multiple PEs within it, all running locally synchronous to each other. PEs in different clock domains communicate via mixed-clock token ring FIFOs modified to support voltage level conversion if required [6].

For both VFI and SSV systems, we assume that the implementation supports fine grained frequency scaling, and there exists either hardware or software based support, as discussed in [9] or [5] 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 is clocked globally with a single clock and represents an SSV system, while the one on the right has an two independent clocks that operate two separate VFIs. As in Figure 1, each clock domain can contain more than one PE.

5. Theoretical Formulation

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

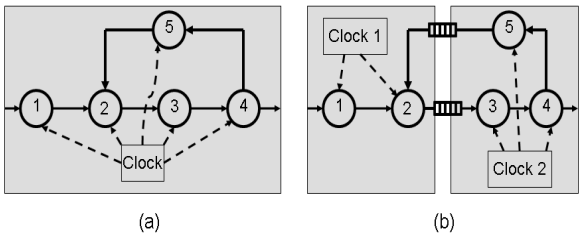

Figure 1. Fully Synchronous and multiple VFI architecture (a) An SSV system with a single global clock source (b) A multiple VFI system with two VFIs and more than one PE in each VFI.

- 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 an IP implemented as a simple linear pipeline, for example, the number of cycles will be equal to the number of stages in that pipeline. We assume, as in [11] that the communication latency is lumped into the number of execution cycles, and that the architecture is partitioned to minimize inter-domain communication. Note that, in general, the number of execution cycles for a PE can vary dynamically depending on the workload. In this work, we are not interested in modeling workload or application driven variability and therefore we restrict C_i to be a fixed number.

- Each PE is characterized in terms of the probability density function (*pdf*) of its cycle time T_i, where the cycle time 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 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. While, our approach is general and can handle both discrete and continuous distributions, we note that continuous distributions serve well to answer *what-if* kind of questions that frequently arise in system level design. 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 $F_{T_i}(t) = \int_{-\infty}^{t} f_{T_i}(\tau)\, d\tau$.

- 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 PEs. 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 j^{th} 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 $L_i = C_i.T_j^{VFI}$. 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.

Having introduced the mathematical notations and assumptions, we will now outline our algorithm to compute the throughput of the component graph efficiently.

5.1. Throughput Analysis for VFI Systems

The throughput (or rate) of a component graph is restricted by the presence of cycles in the graph [7]. 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 [7]. 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 the to the latency (as defined in the previous Section) of the source node of that edge. Specifically

$$w(u, v) = L_u, \forall (u, v) \in E$$

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) [7]. 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:

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

where $|C|$ represents the number of edges in cycle C. In [7], the authors use Karp's algorithm [10] to compute the MCM. According to Karp's algorithm, the MCM ($\Delta^* = \frac{1}{\lambda^*}$) is given as:

$$\Delta^* = \frac{1}{\lambda^*} = \max_{v \in V} \min_{0 \le k \le n-1} \frac{D_v^n - D_v^k}{n - k} \quad (1)$$

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

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

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 $1 \le k \le n$). 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 (1). 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 use a recently proposed SSTA technique that models correlations by representing all intermediate random variables as linear (or quadratic) functions of the input random variables, and uses a moment matching based propagation scheme [3]. 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:

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

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

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

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 $1 \le k \le n$ and for all $v \in V$ as:

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

where the coefficients $a_{v,i}^k \in \Re$ for $0 \le i \le n$. 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 $1 \le i \le n$. We can now solve the recurrence relationship to get

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

but we also know that:

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

We now need to determine the coefficients $a_{v,i}^k$ for $1 \le i \le n$. Without any loss in generality, consider a generic *max* function of that takes as input two variables A and B that are linear combinations of the random variables L_i, $1 \le i \le n$:

$$D = max(A, B) = max(\alpha_0 + \sum_{1 \le i \le n} \alpha_i T_i^{'}, \beta_0 + \sum_{1 \le i \le n} \beta_i T_i^{'})$$

We want to write:

$$D = \gamma_0 + \sum_{1 \le i \le n} \gamma_i T_i^{'}$$

This can be accomplished by noting that:

$$E(T_i^{'} D) = \gamma_i = E(T_i^{'} max(A, B)), \forall (1 \le i \le n) \quad (2)$$

and

$$E(D) = \gamma_0 = E(max(A, B)) \quad (3)$$

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 in [3] and can be evaluated numerically. Having computed the coefficients for each D_v^k for $1 \le k \le n$ and $v \in V$, we can now rewrite equation (1) as:

$$\frac{1}{\lambda^*} = \max_{v \in V} \min_{0 \le k \le n-1} \frac{a_{v,0}^n - a_{v,0}^k + \sum_{1 \le i \le n} (a_{v,i}^n - a_{v,i}^k) T_i^{'}}{n - k} \quad (4)$$

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 $(0 \le i \le n)$, that allow us to write:

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

We can now write the *cdf* of random variable of Δ^* as:

$$F_{\Delta^*}(\tau) = F_{T_1^{'}}(\frac{\tau - \delta_0}{\delta_1}) * F_{T_2^{'}}(\frac{\tau - \delta_0}{\delta_2}) \ldots F_{T_n^{'}}(\frac{\tau - \delta_0}{\delta_n}) \quad (5)$$

Therefore the *cdf* for the throughput of G, represented as $F_{\lambda^*}(\lambda)$, is given by:

$$1 - F_{\lambda^*}(\lambda) = F_{T_1^{'}}(\frac{1 - \delta_0 \lambda}{\delta_1 \lambda}) * F_{T_2^{'}}(\frac{1 - \delta_0 \lambda}{\delta_2 \lambda}) * \ldots F_{T_n^{'}}(\frac{1 - \delta_0 \lambda}{\delta_n \lambda}) \quad (6)$$

In equations (5) and (6), $*$ represents the convolution operation. 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 i^{th} 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:

$$1 - F_Z(z) = (1 - F_X(z))(1 - F_Y(z))$$

We can therefore write:

$$1 - F_{\lambda^*}(\lambda) = (1 - F_{\lambda_1^*}(\lambda))(1 - F_{\lambda_2^*}(\lambda)) \ldots (1 - F_{\lambda_m^*}(\lambda)) \quad (7)$$

Equations (2),(3), (5),(6) and (7) can be computed efficiently using the techniques outlined in [1]. 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 $O(|V||E|)$ [7], 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 $p \le n$).

Algorithm 1 Statistical MCM

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

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

$$T_G = \max_{1 \le i \le n} T_i \quad (8)$$

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

$$F_{T_G}(t) = F_{T_1}(t).F_{T_2}(t) \ldots F_{T_n}(t) \quad (9)$$

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:

$$\lambda^* = Q(C_1T_1, C_2T_2 \ldots C_nT_n)$$

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

$$\begin{aligned} \lambda^*_{SSV} &= Q(C_1.T_G, C_2.T_G \ldots C_n.T_G) \\ &= T_G.Q(C_1, C_2, \ldots, C_n) \end{aligned}$$

Since the cycle counts are single values, and we have already computed the *cdf* (and therefore *pdf*) of T_G in equation (9), we just need a single run of the classic Karp's MCM algorithm over input values that are fixed numbers.

5.3. SSV vs. VFI

Though it seems intuitive to assume that multiple VFIs will always perform better than SSV systems under variability, we would like to prove formally that this is indeed the case.

Lemma: The probability that a multiple VFI system meets a given throughput constraint, λ_c, is always greater than or equal to the probability that its SSV counterpart will meet the same constraint.

Proof: Assume there that exists a probability space Ω, from which samples of the random vector of cycle times $T = (T_1, T_2 \ldots T_n)$ are drawn. Now we define

$$Q_{SSV}(T) = \lambda^*_{SSV} = Q(C_1T_G, C_2T_G \ldots C_nT_G)$$

and correspondingly

$$Q_{VFI}(T) = \lambda^*_{VFI} = Q(C_1T_1, C_2T_2 \ldots C_nT_n)$$

Since the $Q(.)$ function is a monotonically decreasing function of its inputs [7], and from equation (8) we know that $T_G \geq T_i$, ($\forall i$ s.t. $1 \leq i \leq n$), we can say that:

$$Q_{SSV}(T) \leq Q_{VFI}(T), \forall T \in \Omega \tag{10}$$

Now consider the probability of an SSV system meeting the throughput constraint λ_c,

$$Pr(Q_{SSV}(T) \geq \lambda_c) = Pr(\Omega_{SSV})$$

and the corresponding probability for VFI systems

$$Pr(Q_{VFI}(T) \geq \lambda_c) = Pr(\Omega_{VFI})$$

where Ω_{SSV} and Ω_{VFI} are the regions of Ω where the corresponding throughputs of SSV and VFI systems respectively are larger than λ_c. From equation (10), we know that

$$\Omega_{SSV} \subset \Omega_{VFI}$$

and therefore, $Pr(\Omega_{VFI}) \geq Pr(\Omega_{SSV})$, or equivalently $Pr(\lambda^*_{VFI} \geq \lambda_c) \geq Pr(\lambda^*_{SSV} \geq \lambda_c)$. In other words, a VFI system is more likely to meet a given throughput constraint than an SSV system. We note that we have not made any assumptions about the nature (discrete or continuous) or statistics of the cycle time distributions. The proof is therefore applicable for any arbitrary distribution of cycle times. $\diamond$

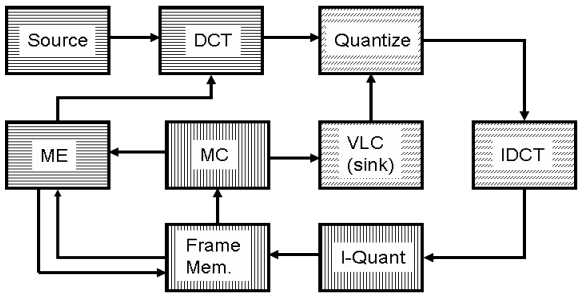

Figure 2. The MPEG-2 Encoder Benchmark. For MCV-3, nodes with similar background patterns are clustered into VFIs.

Benchmark	*n*	*p*	Error(μ)	Error(Y.P.)	Speed-Up
synth-1	15	5	1.21%	2.1%	260X
synth-2	30	10	0.87%	2.74%	147X
synth-3	45	15	1.8%	0.65%	97X
synth-4	60	20	0.92%	3.08%	78X

Table 1. Results for synthetic benchmarks. All comparisons are with respect to Monte Carlo simulations

6. Results

We implemented the proposed algorithm for determining the throughput distribution of single and multiple voltage-frequency island systems in C. 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 [12]. We therefore validate the accuracy and efficiency of our proposed techniques on a set of *synthetic benchmarks* that are generated using the algorithm presented in [12]. 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 [3, 1, 2].

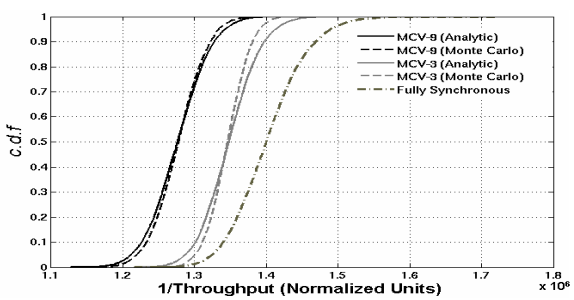

Figure 3. *cdfs* of throughput obtained for the MPEG-2 encoder benchmark for three different architectures

6.1. Synthetic Benchmarks

In [12], the authors outline an algorithm to generate cyclic task (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 the number of PEs (n) in the graphs from 15 to 60 and the number of clock domains (p) from 4 to 20. The number of execution cycles for each PE is chosen randomly from a uniform distribution between 50 and 100. Finally, we assume that the cycle times of the PEs are normally distributed with a 3σ of 20% of the mean. Table 1 shows the error between the mean and the 99% yield points of the throughput distributions for each of the benchmarks. We note that the average error in the mean of the throughput distribution, compared to the Monte Carlo results, is **1.2%** (maximum **1.81%**) and the average error in the 99% yield point is **2.14%** (maximum **3.08%**). This comes at a speed-up ranging from **78X** to **260X** (average **145X**). We note that the speed-up is greater for systems with fewer clock domains, as predicted by the time complexity analysis.

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 run time. Using an example of an MPEG-2 encoder from [8], we now demonstrate how such a framework can be used by system level designers to evaluate multiple VFI systems. Figure 2 shows the component graph of the MPEG-2 encoder. To determine the execution cycles for each of the components, we simulated a software version of the MPEG-2 encoder on an ARM7TDMI core and obtained cycle counts for each module. We note 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. 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.

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. MCV-3 has three clock domains, with three PEs in each clock domain, as represented by the shaded regions in Figure 2. Finally SSV is a fully synchronous design with a single clock domain. Figure 3 show 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 **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 textbf99.8%) 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.

7. Conclusions and Future Work

We provide an efficient and accurate algorithm to compute the system throughput of an embedded application, implemented as a VFI design, under manufacturing process variations. The proposed framework allows system level designers to make variability aware architectural decisions for their VFI designs. Results on synthetic benchmarks demonstrate the accuracy of the proposed technique (on average **1.2%** error in mean and **2.14%** error in the 99% yield point) for a speed up ranging from **78X** to **260X**. Furthermore, using an MPEG-2 benchmark application, we show that multiple VFI island designs are more likely to meet throughput constraints than their fully synchronous counterparts and that the yield advantage diminishes gradually as the number of clock domains are reduced (PEs are clustered together). Our future research directions involve modeling spatial and systematic sources of WID variability.

References

[1] A.Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *Proceedings of ICCAD*, 2003.

[2] H. Chang and S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *Proceedings of ICCAD*, 2003.

[3] Y. Zhang, A.J. Strojwas, X. Li and L.T. Pileggi. Correlation-aware statistical timing analysis with non-gaussian delay distributions. In *Proceedings of DAC* ,2005.

[4] M. Orshansky and K.Kuetzer. A general probabilistic framework for worst case timing analysis. In *Proceedings of DAC*, 2002.

[5] D. Ernst, N.S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner and T. Mudge. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[6] T. Chelcea and S. Nowick. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In *Proceedings of DAC*, 2001.

[7] A. Mathur, A. Dasdan and R.K. Gupta. Rate analysis for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 3(3), 1998.

[8] L.P. Carloni and A.S. Vincentelli. Performance analysis and optimization of latency insensitive systems. In *Proceedings of DAC*, 2000.

[9] A. Dutta, N. Bhunia, A. Banerjee and K. Roy. A power-aware GALS architecture for real-time algorithm-specific tasks. In *Proceedings of ISQED*, 2005.

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

[11] D. Marculescu and S. Garg. System level process-driven variability analysis for single and multiple voltage-frequency islands. In *Proceedings of ICCAD*, 2006.

[12] S. Stuijik, M. Gielen and T. Basten. SDF3: SDF for free. In *Proceedings of ACSD'06*, 2006.