Throughput Analysis and Voltage-Frequency Island Partitioning for Streaming Applications under Process Variation

Davit Mirzoyan[1], Sander Stuijk[2], Benny Akesson[2], Kees Goossens[2]
[1]Delft University of Technology, [2]Eindhoven University of Technology
D.Mirzoyan@tudelft.nl, S.Stuijk@tue.nl, K.B.Akesson@tue.nl, K.G.W.Goossens@tue.nl

***Abstract*—Variability in the manufacturing process results in variation in the maximum supported frequency of individual cores in a Multi-Processor System-on-Chip (MPSoC). This variation needs to be considered when performing statistical timing analysis in the system-level design. As our first contribution, we present a framework to estimate the probability distribution of application throughput (e.g. frames per second in video decoding) in a system with Voltage-Frequency Island (VFI) partitions in the presence of process variation. The novelty of the framework lies in the computation of the probability distribution of throughput, based on a user-specified set of clock-frequency levels per VFI domain considering *both within-die and die-to-die variations* of cores. As a second contribution, we provide a methodology to perform variation-aware partitioning of the cores of an MPSoC into VFIs for maximized *timing yield* (percentage of chips that satisfy a given throughput requirement). On a case study, we demonstrate how our methodology can be used by system designers for two purposes: 1) to make trade-offs between the number of VFI partitions (design cost) and timing yield; 2) to estimate the impact of reducing circuit design margins on the number of good dies on a wafer. We illustrate that the proposed variation-aware partitioning provides up to 18% improvements in the timing yield compared to a deterministic partitioning.**

I. Introduction

Aggressive technology scaling has enabled the integration of multiple processors and hardware accelerators on a single silicon die, known as a Multi-Processor System-on-Chip (MPSoC). Present-day MPSoCs are implemented by means of the Globally Asynchronous, Locally Synchronous (GALS) design style, which was introduced to alleviate the bottleneck of global clock distribution. The GALS architecture is composed of synchronous blocks, communicating with each other on an asynchronous basis. The concept of Voltage-Frequency Islands (VFI), within the GALS design paradigm, enables scaling the frequency (voltage) of each individual hardware component (clusters of components) in a chip to reduce power.

On the other hand, scaling technology into minimum feature size nodes has made it practically impossible to precisely control the manufacturing process. This leads to variability in key design parameters, such as device channel length and threshold voltage, and interconnect width. Parameter variability, in turn, affects the performance characteristics of cores in a Multi-Processor System-on-Chip (MPSoC) [2]. Variability of up to 20% in the longest path delay of a processor at 32 nm technology is demonstrated in [16].

Conventionally, circuits are implemented with conservative design margins or guard-bands, often referred to as worst-case design, to guarantee the target frequency of each core in the manufactured chips. From an application's perspective, the cores have deterministic frequencies, leading to a conventional application mapping, such that a certain timing requirement (e.g. throughput or latency) imposed on the system is satisfied. It is shown that guard-band reduction provides the benefits of reduced circuit area (i.e. a higher number of gross dies on a wafer), and lower dynamic and leakage power [10] [12]. With reduced guard-bands, the target frequency of cores is not guaranteed any more, and the probability distribution of frequencies needs to be considered when analyzing the system timing. Therefore, models and a methodology is required by system designers for evaluating the statistical timing of a system. Ultimately, system designers need to be able to estimate the impact of guard-band reduction on the number of good dies (dies that satisfy the timing requirement of a system) on a wafer.

This work has two main contributions. As our first contribution, we present a framework to estimate the probability distribution of application throughput (e.g. frames per second in video decoding) in a system with VFI partitions in the presence of process variation. The novelty of the framework lies in the computation of the probability distribution of throughput, based on a user-specified set of clock-frequency levels per VFI domain considering *both within-die and die-to-die variations*. The specified clock-frequency levels are (to be) provided by Clock Generation Units (CGU) to VFI domains in hardware. We use Synchronous Data-Flow (SDF) to model a system consisting of a real-time streaming application mapped to an MPSoC. SDF graphs are well-suited for modeling and analysis of streaming applications. We allow resource sharing, and assume Periodic Static-Order Scheduling (PSOS) among tasks of an application allocated to the same core. As a second contribution, we provide a methodology to perform variation-aware partitioning of the cores of an MPSoC into VFIs for maximized *timing yield*. The timing yield is a system-level metric showing the percentage of chips that satisfy a given throughput requirement imposed on the system. To the best of our knowledge, this is the first work to address the problem of VFI partitioning for improved timing yield in the presence of process-driven variations. On a case study, we demonstrate how our methodology can be used by system designers for two purposes: 1) to make trade-offs between the number of VFI partitions (design cost) and timing yield; 2) to estimate the impact of reducing circuit design margins on the number of good dies (dies that satisfy the throughput requirement of a system) on a wafer. We illustrate that the proposed variation-aware partitioning provides up to 18% improvements in the timing yield compared to a deterministic partitioning.

The rest of the paper is organized as follows: Section II presents the related work; system modeling is introduced in Section III; in Section IV, we demonstrate how the probability distribution of throughput is computed; Section V presents the VFI partitioning algorithm; Section VI experimentally evaluates our methods; and Section VII concludes the work.

II. Related work

The related work can be divided into two categories: VFI partitioning; and system-level throughput analysis, considering the impact of process variation. With regard to VFI partitioning, the works in [11] and [9] address the problem of partitioning a tile-based Network-on-Chip (NoC) architecture into VFIs for minimized energy consumption, subject to performance constraints. The authors in [18] solve a similar problem, and propose a methodology for a run-time energy management through voltage (frequency) scaling, given workload and technology-related variations. None of these works consider process variation in the partitioning process. Majzoub *et al.* include process, voltage and temperature variations in the VFI partitioning process to minimize energy consumption [13]. They estimate expected voltage and temperature variations, and assume a given core-frequency map across a chip due to within-die process variation. In contrast, we consider both within-die and die-to-die variations, such that different chips have different core-frequency maps with associated probabilities; this is how the variation in reality behaves, as demonstrated by measurements in [19]. Moreover, the authors in [13] perform VFI partitioning to minimize energy consumption, while we maximize timing yield in a probabilistic setting. We are not aware of any other work addressing the problem of VFI partitioning for improved timing yield, considering process-driven variations.

With regard to variation-aware system-level throughput analysis, Marculescu *et al.* analyze the probability distribution of *latency* of systems with multiple VFIs, considering within-die variation [14]. Their approach is only applicable to systems specified as *acyclic* task graphs. Acyclic task graphs are not able to capture the iterative and overlapping execution of many real-life streaming applications (e.g. our modem benchmark application presented in Section A), which are primarily constrained by a throughput requirement. We allow arbitrary task graphs that may include *cyclic* data dependencies. We model a system by means of an SDF graph, which is well-suited for modeling and analysis of real-time streaming applications with *throughput* requirements. A methodology to perform system-level throughput analysis of multiple VFI designs, considering process variation, is presented by Garg *et al.* [7]. However, they account for only within-die variation, while we consider both within-die and die-to-die variations. Their approach is based on Homogeneous SDF (HSDF) graphs, which is a special case of an SDF graph, where all token rates associated with edges are equal to 1. We use an SDF graph for system modeling. A SDF graph provides much more compact application models, which is why many real-time streaming applications are modeled in an SDF formulation. To be able to use the approach in [7] for an application specified as an SDF graph, a conversion from the SDF graph to an equivalent HSDF graph is required [22]. This can lead to an exponential increase in the graph size (in terms of the number of actors and edges), as compared to the original SDF graph. For example, the SDF graph of our MP3 Playback benchmark application consists of only four actors, while the number of actors in an equivalent HSDF graph after conversion becomes 10601. Performing throughput analysis on a large graph results in prohibitively high computation times, making the approach in [7] unsuitable for many applications. Additionally, the work in [7] assumes a one-on-one mapping of tasks to processing elements, while we allow resource sharing and assume Periodic Static-Order Scheduling (PSOS) among tasks of an application allocated to the same core. Mirzoyan *et al.* [17] solve an application to MPSoC mapping problem for improved timing yield, considering the impact of process variation. We assume a specified mapping, and analyze the throughput of a system with VFI partitions. As in our work, they also use an SDF graph for system modeling. They introduce a set of *operating points* (possible frequency values) to model the impact of process variation. In contrast, we explicitly model within-die and die-to-die variations with Probability Density Functions (PDF), and compute the probability distribution of application throughput based on a user-specified set of clock-frequency levels.

III. Formalization

This section introduces the formal models that the proposed framework relies on. We first define a hardware multi-processor platform and present the modeling of within-die and die-to-die variations in hardware resources. Later, an SDF model of a real-time throughput-constrained application, named an *unbound graph*, is introduced. This model is unaware of the binding of application actors to hardware resources, and is hence decoupled from hardware variation. Finally, for a specific binding and static-order schedule of actors on shared resources, we define an SDF model of a system named a *parametric bound graph*. This model is used to derive the probability distribution of system-level throughput. The presented techniques are general and apply to any system that implements the models in this section. Examples of such systems are CoMPSoC [8] and CA-MPSoC [21].

Although, the presented models are inspired by the ones in [17], our modeling differs in the following ways. They model variation as a set of operating points (frequency values), but we explicitly model die-to-die and within-die variations by means of Probability Density Functions (PDF) of the maximum supported frequency of a resource (this is what is known from a statistical characterization). Our models allow users to specify any set of clock-frequency levels in the frequency range covering both die-to-die and within-die variations (these are the clock-frequency levels to be implemented in hardware and to be provided to each island). This set of clock-frequency levels is used to construct the probability distribution of throughput, as described in Section IV.

A. Hardware Platform

We refer to a hardware multi-processor platform as a set R of resources connected to each other by an interconnection network. Each resource is a generic processing element, such as a processor, DSP or a hardware accelerator. We assume a Globally Asynchronous, Locally Synchronous (GALS) architecture, where the resources are partitioned into VFIs. Communication between islands is done by means of mixed-clock First-In-First-Out (FIFO) buffers. A Clock Generation Unit (CGU), which provides a set of clock-frequency levels, is dedicated to each island. Depending on the desired clock-frequency levels, the complexity of a CGU can vary. Starting from a simple hardware clock divider, a CGU can consist of a programmable-length ring oscillator controlled by an on/off-chip voltage regulator, for high resolution clock-frequency generation. For simplicity, we assume zero-latency data communication between resources over the network. However, non-zero-latency communication can be dealt with by modeling the delay for sending data over the network in the application SDF graph, as shown in [24].

Manufacturing process variation can be classified into die-to-die and within-die variations. Die-to-die variation, also referred to as global variation, acts globally on the entire chip die, affecting parameters of all devices (i.e. transistors) on the die identically. Global variation is seen between dies within a wafer and across wafers (due to wafer-to-wafer variation); therefore, overall global variation presumes multiple wafers. In contrast, within-die variation, also known as local variation, affects parameters of devices on the same die differently. It can be classified into systematic and random components. Systematic local variation exhibits spatial correlation, such that nearby devices possess similar parameter values due to high correlation, which dies out quickly as a function of distance [6]. While the parameter correlation between adjacent devices is high, the correlation between larger adjacent logic blocks on a die, such as cores, is typically much lower. Furthermore, random local variation, which is purely random from device to device, further reduces this spatial correlation. For simplicity, we assume zero correlation between maximum supported frequencies of resources (cores) in a chip. To further support our assumption, Pang *et al.* [19] show that at 45 nm technology, systematic local variation is insignificant, as random local variation has increased, resulting in insignificant spatial correlation. The models given below can be extended, such that it is possible to specify correlation between maximum supported frequencies of resources. This extension will not affect the rest of the models and the methodology presented in this work. The treatment of correlation is left as future work.

To reflect the impact of global variation, we describe the maximum supported frequency of each resource $r \in R$ in a chip by a random variable f_g^r distributed normally with f_n^r mean and σ_g^r standard deviation (i.e. $f_g^r = N(f_n^r, (\sigma_g^r)^2)$), where f_n^r is the nominal maximum supported frequency of the resource. Global variation affects the maximum supported frequency of all resources on a chip die identically. This results in equally faster or equally slower resources on each manufactured die. Therefore, we can say that the correlation between $(f_g^{r_i}, f_g^{r_j})$ for any $r_i, r_j \in R$ is equal to 1. Additionally, the standard deviation to mean ratio (σ_g^r / f_n^r) is the same for all resources.

For a given global frequency value $f_g^r = f_0$ of a resource, we introduce a normally distributed random variable $f_l^r = N(f_0 - \delta^r, (\sigma_l^r)^2)$ to model the local variation with respect to f_0. Here, σ_l^r is the standard deviation and δ^r models the reduction in mean frequency of the resource due to multiple critical paths [2]. As we assume no spatial correlation between the maximum supported frequencies of resources, the covariance between $(f_l^{r_i}, f_l^{r_j})$ for any $r_i, r_j \in R$ is equal to 0. Figure 1 illustrates an example PDF $\phi(f_g^r)$ of f_g^r for a resource with $f_n^r = 500$ MHz, where $\sigma_g^r = 25$ MHz. The same figure shows PDFs $\phi(f_l^r)$ of f_l^r with respect to $f_g^r = 475$, 500 and 525 MHz, where $\delta^r = 25$ MHz and $\sigma_l^r = 16$ MHz; these numbers are representative for current technology nodes [19].

To describe the maximum supported frequency of a resource by a single distribution, global and local distributions are combined by convolution, as given by Equation (1).

$$\phi(f^r) = \phi(f_g^r) * \phi(\Delta f_l^r) \tag{1}$$

In Equation (1), $\Delta f_l^r = N(-\delta^r, (\sigma_l^r)^2)$ is the deviation from the global frequency value $f_g^r = f_0$, and is obtained by shifting the distribution $\phi(f_l^r)$ by f_0. It is known that the convolution of two normal distributions is also a normal distribution with added means and variances. Therefore, the maximum supported frequency of a resource due to both global and local variations is described by a normally distributed random variable given by Equation (2). The combined distribution for the example described in Figure 1 is shown in the figure.

$$f^r = N(f_n^r - \delta^r, \left(\sigma_g^r\right)^2 + (\sigma_l^r)^2) = N(\mu^r, (\sigma^r)^2) \tag{2}$$

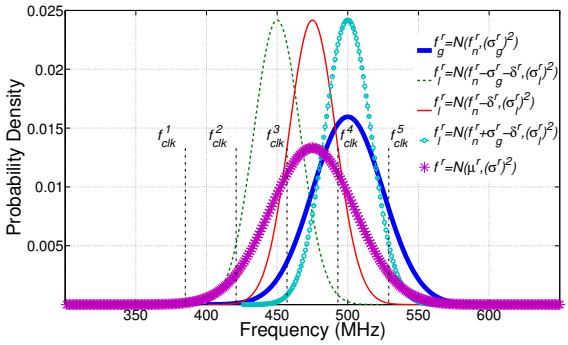


Fig. 1. f_g^r PDF (due to global variation) for a resource with $f_n^r = 500$ MHz, $\sigma_g^r = 25$ MHz; f_l^r PDFs (due to local variation) with respect to $f_g^r = 475$, 500 and 525 MHz, $\delta^r = 25$ MHz, $\sigma_l^r = 16$ MHz; combined PDF of f^r

As explained, global and local variations in frequency are modeled by means of normal distributions. However, arbitrary distributions can be dealt with. In the case of arbitrary distributions, the combined distribution is not given by Equation (2), but needs to be derived by performing convolution of distributions (Equation (1)).

B. *Unbound Graph*

We model a real-time streaming application by means of an SDF graph, which provides a good compromise between expressiveness, modeling ease, analysis potential and implementation efficiency. With an SDF model, an application is captured by a directed graph, where the nodes, called actors, represent computation; actors communicate with each other by sending streams of data elements, called tokens, over their edges. Figure 2 illustrates an example SDF model of an H.263 encoder application. It consists of five actors connected to each other by seven dependency edges; edges d_3, d_6 and d_7 contain initial tokens, illustrated by black dots in the figure. The execution of an actor is called a *firing*. When an actor fires it removes a number of tokens from all its input ports and at the end of the firing (after its execution), it produces a number of tokens on each output port. These numbers of tokens are called rates, and are shown near the channel ends in Figure 2.

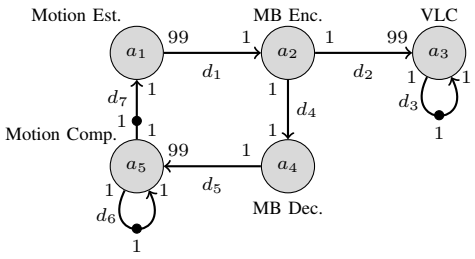


Fig. 2. Example SDF model of H.263 encoder

Formally, we denote the set of all actors as A, where each actor requires a number of clock cycles to finish its execution (Definition 1). For real-time applications, execution cycles can be derived using worst-case execution-time estimation tools, as explained in [27]. Note that the number of clock cycles required for an actor's execution can be different for each resource if the platform is heterogeneous.

Definition 1: (Execution cycles) The number of cycles required to execute an actor $a \in A$ on a resource $r \in R$ it can be bound to is given by $ec(a, r) : A \times R \rightarrow \mathbb{N}$.

A SDF model that is unaware of the binding of actors to resources is given in Definition 2. Each actor in the graph is characterized by a number of clock cycles of a resource, for all resources to which it can be bound.

Definition 2: (Unbound graph) An unbound graph *gu* is a 3-tuple $\langle A, D, ec(a, r)\rangle$ consisting of a set A of actors, a set $D = A^2 \times \mathbb{N}^3$ of dependency edges with rates and initial tokens and the function $ec(a, r)$ that describes each actor $a \in A$ with execution cycles on a number of resources in the set R.

C. Parametric Bound Graph

We assume that each actor can be bound to a number of resources from the set R. Therefore, there are multiple bindings of a set A of actors to a set R of resources. A given binding of actors to resources is captured in a *binding vector*, denoted as b, and is an N-dimensional vector for N actors (Definition 3). Each element in b specifies the resource each actor is bound to. For example, let us assume that the unbound graph shown in Figure 2 is mapped to a multi-processor platform comprising two resources; a_2, a_3 are bound to r_1 and a_1, a_4, a_5 are bound to r_2. This is given in a binding vector $(a_1, a_2, a_3, a_4, a_5) \rightarrow (r_2, r_1, r_1, r_2, r_2)$.

Definition 3: (Binding vector) A binding vector of a set A of actors specifies the resource $r \in R$ every actor $a \in A$ is bound to, and is given by $b(a) : A \rightarrow R$.

For a given binding vector b, the execution cycles $ec(a, r)$ of each actor $a \in A$ is known from Definition 1. The execution time $et(a, r)$ in seconds is obtained by the division of $ec(a, r)$ by the frequency f^r of the resource the actor is bound to (Definition 4). As f^r is a normally distributed random variable, $et(a, r)$ can have a range of values depending on f^r.

Definition 4: (Execution time) The execution time in seconds of an actor $a \in A$ on a resource $r \in R$ that has a frequency f^r is given by $et(a, r) : A \times R \rightarrow \mathbb{R}$, and is defined as $et(a, r) = ec(a, r)/f^r$

Several actors, as specified by a binding vector, can be bound to the same resource; therefore, the actors have to be scheduled on the shared resource. This is accomplished by a Periodic Static-Order Schedule (PSOS), which specifies the sequence of actor firings repeated indefinitely (Definition 5). For the example binding vector $(a_1, a_2, a_3, a_4, a_5) \rightarrow (r_2, r_1, r_1, r_2, r_2)$, presented for the graph shown in Figure 2, the PSOSs for the two resources can be $s(r_1) = ((a_2)^{99}a_3)^*$ and $s(r_2) = (a_1(a_4)^{99}a_5)^*$; where $(a_i)^j$ specifies that a_i fires j times, and * indicates that the schedule is repeated indefinitely.

Definition 5: (Periodic static-order schedule) A periodic static-order schedule of the actors bound to a resource $r \in R$ is given by $s(r) : R \rightarrow S$, where S is the set of all schedules.

For a binding vector b and a periodic static-order schedule $s(r)$, given for each resource $r \in R$, an SDF model, called parametric bound graph, is defined (Definition 6). In a parametric bound graph, the execution time of each actor $a \in A$, bound to a resource $r \in R$, is a function of the resource frequency f^r, and is given by $et(a, r)$ (Definition 4).

Definition 6: (Parametric bound graph) A parametric bound graph *gb* is a 4-tuple $\langle gu, b, et(a, r), s(r)\rangle$ consisting of an unbound graph *gu*, a binding vector b of actors A to resources R, the function $et(a, r)$ that describes the execution time in seconds of each actor as a function of f^r and the function $s(r)$ that determines the periodic static-order schedule for each resource $r \in R$.

An approach is presented in [3] to model given PSOSs in an SDF graph by adding additional actors with zero execution time and edges to enforce the actor firings according to the schedule. We use this approach in our work to model the schedules $s(r)$ in the parametric bound graph *gb*.

A technique is introduced by Damavandpeyma *et al.* [4] to perform throughput analysis of an SDF graph, where the actor execution times are specified as a function of parameters. The outcome is a set of parametric expressions describing the throughput for specified parameter intervals. They use the theory of Max-Plus algebra to perform fast throughput computations of an SDF graph, without converting it to an equivalent HSDF graph that can be much larger in size. We use their approach to perform throughput analysis of a parametric bound graph, where the execution time of actors is a function of frequency f^r. We specify a parameter interval of $[\mu - 3\sigma,\ \mu + 3\sigma]$ for f^r; the interval covers 99.7% of samples in a normal distribution. The result is a set E of parametric throughput expressions, where each expression $e \in E$ describes the throughput of a cycle in the graph. Equation (3) shows the representation of an expression, where c^r are constants. An expression $(c^{r_i}/f^{r_i} + c^{r_j}/f^{r_j})^{-1}$ shows that there is a cycle in the graph mapped across resources r_i and r_j $(i \neq j)$.

$$e = \frac{1}{\sum_{r \in R} c^r/f^r} \tag{3}$$

IV. Throughput distribution

This section details how the probability distribution of the throughput of a parametric bound graph is constructed. From an implementation perspective, each CGU, associated with a VFI, provides a set of discrete clock-frequency levels. We chose the clock-frequency levels from the combined distribution $\phi(f^r)$ (Equation (1)), so that frequency levels are available in the overall resource-frequency range that covers both global and local variations. Each resource is operated at one of the clock-frequency levels depending on its maximum supported frequency. In this work, we select the clock-frequency levels equidistantly in the range $[\mu - 3\sigma,\ \mu + 3\sigma]$ (Definition 7), for the reason that equidistant clock-frequency levels are easier to implement in hardware (e.g. hardware clock-dividers, programmable ring oscillators) than non-equidistant clock frequencies. However, in principle the user can specify any frequency levels. Figure 1 illustrates how five equidistant clock-frequency levels are obtained from the combined distribution.

Definition 7: (Clock-frequency levels) A set of n equidistant clock-frequency levels available to a resource $r \in R$ is given by $c(r, n) : R \times \mathbb{N} \rightarrow \mathcal{P}(\mathbb{R}^+)$, and is defined as

$$c(r, n) = \{(\mu^r - 3\sigma^r) + (k-1) \cdot (6\sigma^r/n) |\ k = 1, 2, .., n\} \tag{4}$$

Given that each resource can be operated at any clock-frequency level in the set $c(r,n)$, for a set R of resources in a chip, there are multiple possible combinations of clock-frequency levels. An instance of clock-frequency levels for the overall number of resources in a chip is captured in a *chip-frequency vector*, denoted as *fc*, and is an M-dimensional vector for M resources (Definition 8). Each element in *fc* represents a clock-frequency level $f_{clk} \in c(r,n)$ for a corresponding resource $r \in R$. The set of all possible chip-frequency vectors is obtained by the Cartesian product of individual sets $c(r,n)$ (Definition 9).

Definition 8: (Chip-frequency vector) A chip-frequency vector for a set R of resources specifies a frequency $f_{clk} \in c(r,n)$ for every resource $r \in R$, and is given by $fc(r) : R \rightarrow \mathbb{R}^+$.

Definition 9: (All chip-frequency vectors) The set of all possible chip-frequency vectors for a set R of resources is given by

$$FC = \prod_{r \in R} c(r,n) \tag{5}$$

Each chip-frequency vector $fc \in FC$ is associated with a probability, which is the probability that resources in a chip are clocked at the particular clock-frequency levels specified by *fc*. By computing the probability and the resulting throughput of a parametric bound graph for each $fc \in FC$, the probability distribution of system-level throughput is constructed. From probability theory, it is known that the joint probability of independent events equals the product of their individual probabilities. Frequencies of resources in a chip described by random variables f^r are not independent due to the correlated global variation in the resources. In contrast, resource frequencies described by random variables f_l^r are independent, as we assume no spatial correlation due to local variation. For this reason, the joint probability of a chip-frequency vector *fc* is represented as a sum of components. Each component is the *joint local probability*, which is the probability that the resources are clocked at the particular clock-frequency levels based on local distributions with respect to a global frequency value.

Depending on the maximum supported frequency, each resource is operated at the highest possible clock-frequency level. Let us consider the local frequency distribution shown by the dashed line in Figure 1; it is with respect to a global frequency value $f_g^r = 475$ MHz. For any actual f_l^r in the range $[f_{clk}^2,\ f_{clk}^3)$, the resource is operated at f_{clk}^2. The probability of the resource being operated at f_{clk}^5 equals zero. Therefore, the probability that a resource is operated at a clock-frequency level f_{clk}^i in a local distribution is given by the probability of f_l^r being in the interval $[f_{clk}^i,\ f_{clk}^{i+1})$ if $i \neq n$ or $[f_{clk}^i,\ \infty)$ otherwise (Definition 10).

Definition 10: The probability of a resource $r \in R$ being operated at f_{clk}^i in a local distribution with respect to a global frequency value $f_g^r = f_0$ is given by $pf(r, f_0, f_{clk}^i) : R \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$, and is defined as

$$\begin{aligned} pf(r, f_0, f_{clk}^i) &= \int_I \phi(f_l^r = N(f_0, (\sigma_l^r)^2))\, \mathrm{d}f_l^r \\ \text{where } I &= \begin{cases} [f_{clk}^i,\ f_{clk}^{i+1}) & \text{if } i \neq n \\ [f_{clk}^i,\ \infty) & \text{otherwise} \end{cases} \end{aligned} \tag{6}$$

Based on local distributions with respect to a global frequency value (a frequency value for each resource), the joint probability of a chip-frequency vector *fc* is computed by the product of individual probabilities $pf(r, f_0, fc(r))$ and the probability of the global frequency value (Definition 11).

Definition 11: (Local probability of *fc*) The probability of a chip-frequency vector $fc \in FC$ with respect to a global frequency value f_0 is given by $p(fc, f_0) : FC \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$, and is defined as

$$p(fc, f_0) = \phi(f_0) \cdot \prod_{r \in R} pf(r, f_0, fc(r)) \tag{7}$$

The overall probability of a chip-frequency vector *fc* is obtained by adding the joint local probabilities for all possible global frequency values (Definition 12).

Definition 12: (Probability of *fc*) The probability of a chip-frequency vector $fc \in FC$ is given by $pc(fc) : FC \rightarrow \mathbb{R}^+$, and is defined as

$$pc(fc) = \sum_{\forall f_0 \mid \phi(f_g^r = f_0) \neq 0} p(fc, f_0) \tag{8}$$

The throughput of a parametric bound graph for a chip-frequency vector *fc* is given by the minimum of all throughput expressions (Definition 13). When multiple resources are placed in a single voltage-frequency domain, the frequency of the island (resources) is decided by the minimum frequency of all resources in the island. Therefore, to compute the throughput of a parametric bound graph for a given partitioning of resources into VFIs, each chip-frequency vector $fc \in FC$ is modified according to the partitioning.

Definition 13: (Throughput of *bg* at *fc*) The throughput of a bound graph at a chip-frequency vector $fc \in FC$ is given by $t(fc) : FC \rightarrow \mathbb{R}^+$, and is defined as

$$t(fc) = \min_{e \in E}(e) \tag{9}$$

By computing the probability of each chip-frequency vector $fc \in FC$ (Definition 12) and the resulting throughput of a parametric bound graph (Definition 13), the probability distribution of system-level throughput is constructed.

V. VFI PARTITIONING ALGORITHM

In this section, we present a methodology to partition a hardware platform comprising multiple resources into VFIs. Each island is associated with a CGU that provides a set of clock-frequency levels. By reducing the number of VFIs, we effectively reduce the number of CGUs and FIFO buffers, resulting in a lower cost (i.e. area and power) design. The partitioning is performed for a given parametric bound graph, and thus for a given mapping and scheduling of an application to platform resources. When minimizing the number of VFIs, the objective is to have maximized timing yield. The timing yield for a given throughput requirement can be evaluated by constructing the probability distribution of system-level throughput, as detailed in Section IV.

We define a metric called *resource criticality* that guides the process of partitioning resources into VFIs. It quantifies the sensitivity of the throughput of a parametric bound graph to the frequency of a resource. Figure 3 illustrates the normalized throughput range for twelve parametric expressions describing

the throughput of an application mapped to a platform comprising eight resources. Each expression describes the throughput of a cycle in the graph. Each number in the figure represents a resource index; multiple numbers in each bar show that the cycles in the graph are mapped across different resources; the same number in multiple bars indicates that the resource is present in multiple cycles. As cycles 10 and 12 have the lowest normalized throughput, they have higher probability of limiting the throughput of the bound application, which is decided by the minimum throughput of all cycles (Definition 13). Resources r_1 and r_7 (indices 1 and 7 in the figure) have high weights on cycles 12 and 10, respectively. This is shown by the length of corresponding rectangles in a bar. Therefore, the throughput is more sensitive to the frequency changes in these resources, as captured by the resource-criticality metric. It is defined as the worst-case reduction in the throughput of a parametric bound graph as a result of reducing the frequency of the resource from $(\mu^r + 3\sigma^r)$ to $(\mu^r - 3\sigma^r)$ (for the rest of the resources $f^r = (\mu^r + 3\sigma^r)$) (Definition 14).

Definition 14: (Resource criticality) The criticality of a resource $r \in R$ in a set of expressions $e \in E$ is given by $cr(r) : R \rightarrow \mathbb{R}^+$, and is defined as

$$\begin{aligned} cr(r_i) &= (t(fc') - t(fc))/t(fc') \\ \text{where } fc'(r) &= (\mu^r + 3\sigma^r)\ \forall r \in R \\ fc(r) &= \begin{cases} (\mu^r + 3\sigma^r) & \text{if } r \neq r_i \\ (\mu^r - 3\sigma^r) & \text{otherwise} \end{cases} \end{aligned} \tag{10}$$

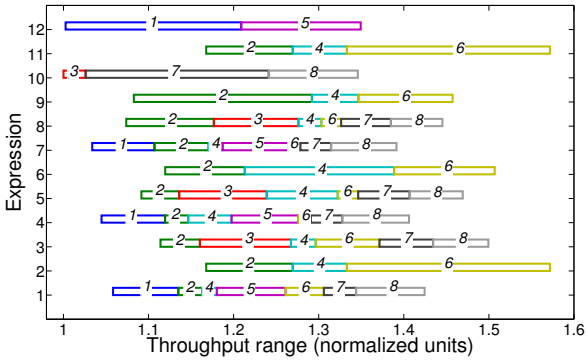


Fig. 3. Throughput range of parametric expressions; each bar corresponds to an expression; each number represents a resource index; the different lengths of rectangles in a bar represent resource weights

Figure 4 demonstrates our proposed heuristic algorithm for partitioning the resources into VFIs. Initially, all resources are placed in separate VFIs. The islands are sorted based on their criticality. The criticality of an island is given by the resource criticality that is the highest among all resources comprising the island. The algorithm consists of iterations. In each iteration, two islands are merged together, forming a single island. This process continues until all resources are placed in a single VFI. As such, the algorithm evaluates all possible granularities of partitions. Intuitively, islands having the lowest criticality values, as given by the order of islands based on criticality, are merged in each iteration. This allows resources with higher criticality values, which are more likely to limit the throughput of a parametric bound graph, to remain in separate VFIs, resulting in maximized timing yield. However, merging two adjacent islands (adjacent in the ordered list of islands based on criticality) with higher criticality values may result in lower throughput reductions (e.g. when these islands have equal or close criticality values). For this reason, the timing yield as a result of merging each pair of adjacent islands is evaluated in each iteration. The grouping providing the highest timing yield is chosen. This results in an overall number of $(M/2 \cdot (M-1) - 1)$ evaluations of the timing yield, M being the number of resources. Note that an exhaustive evaluation of all possible groupings of resources into islands requires $(2M)!/((M+1)! \cdot n!)$ (given by the Catalan number) evaluations, resulting in prohibitively long computation times for a large number of resources.

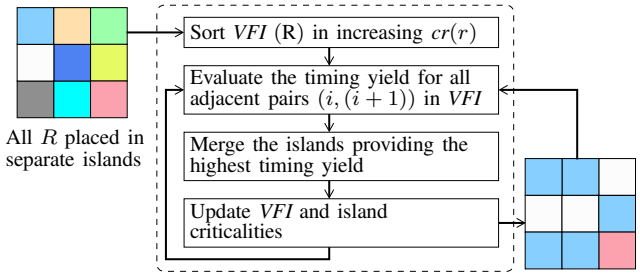


Fig. 4. VFI partitioning algorithm

VI. Experimental Results

On a case study, this section demonstrates how our framework can be used by system designers for two purposes: to make trade-offs between the number of VFI partitions (design cost) and timing yield; to assess the impact of guard-band reduction on the number of good dies. We additionally illustrate that the proposed VFI partitioning algorithm effectively groups resources into islands for maximized timing yield.

A. Experimental setup

We consider a platform comprised of eight homogeneous resources. Each resource has a nominal maximum supported frequency of $f_n^r = 500$ MHz. As reported in [5], at high computing frequencies (in the order of GHz), the variation is large. In our work, we target embedded systems running streaming applications, where the typical frequencies are in the order of hundreds of MHz. Measurements at 45 nm technology in the frequency range of interest are provided by Pang *et al.* [19]. Based on this data, we assume standard deviation to mean ratios $(3\sigma_l^r/f_n^r = 10\%)$ and $(3\sigma_g^r/f_n^r = 12\%)$ for within-die and die-to-die variations, respectively. We assume no reduction in mean frequency due to multiple critical paths (i.e. $\delta^r = 0$); this assumption does not affect the results and observations presented in this section. We select a set of eight equidistant clock-frequency levels for each island (resource) as described in Section IV.

To have fair trade-offs between the granularity of partitions and timing yield, the application mapped to the platform is required to have enough parallelism to be exploited by all eight resources. We used the approach in [23] to generate a cyclic synthetic SDF graph consisting of seventeen actors, which has the required parallelism. To determine the mapping of the application to the platform (i.e. binding vector b), we used the variation-aware mapping algorithm given by Mirzoyan *et al.* in [17]. The periodic static-order schedules $s(r)$ for actors on shared resources are determined by the common list scheduler [15]. To construct the parametric bound graph and perform parametric throughput analysis, the SDF3 tool set is used [23]. The models and the algorithms presented in this work

are implemented in Matlab. In Appendix A, we additionally present case studies based on real applications modeled as SDF graphs. The reason a synthetic application is chosen over a real application in this section is that it has a level of parallelism that allows to demonstrate the important concepts.

B. VFI granularity and yield trade-offs

The parametric expressions describing the system-level throughput for this use-case are graphically shown in Figure 3. The order of resources, based on decreasing criticality $cr(r)$, is $(r_7, r_1, r_5, r_8, r_2, r_4, r_6, r_3)$ with respective criticality values $(0.18, 0.17, 0.12, 0.095, 0.09, 0.032, 0.03, 0.025)$. As expected, r_7 and r_1 have higher criticality values than the rest of the resources. This shows that the throughput of the parametric bound graph is more sensitive to the frequency changes in these resources. Figure 5 illustrates the Cumulative Distribution Function (CDF) of the inverse of the system-level throughput for four different system implementations. Namely, VFI-8, VFI-5, VFI-2 and FS. VFI-8 is an eight voltage-frequency domain architecture, where each resource is in a separate island. VFI-5 and VFI-2 are architectures, where the resources are partitioned into five and two voltage-frequency domains, respectively; the partitioning is decided by the proposed heuristic VFI partitioning algorithm presented in Section V. For the given throughput requirement (denoted as t_{req}), the grouping of the resources into islands for VFI-5 and VFI-2 is shown in Figure 5. Finally, FS is a fully synchronous design, where all resources are in a single VFI. It can be seen that the throughput distribution for VFI-5 closely tracks the one for VFI-8. This is because the resources r_2, r_4, r_6, r_3 have low criticality values and grouping them in a single VFI results in negligible throughput reductions. In contrast, the VFI-2 architecture results in noticeable reductions in the timing yield. Note that resources $r7$, $r1$ with high criticality values are placed in a single island. As shown above, these resources have close criticality values (compared to the criticality values of other resources). As suggested by the partitioning algorithm, placing the resources in a single island provides the lowest degradations in the timing yield. For the given throughput requirement (indicated by the vertical line in Figure 5), VFI-8 provides a 71% timing yield, while VFI-2 achieves a 64% yield. Note that for a different throughput requirement, the reduction in the yield can be higher. For example, for a requirement resulting in a 66% yield with VFI-8, the reduction with VFI-2 is 16% (the partitioning algorithm provides the same resource grouping for this requirement). The FS architecture results in a sub-sampled throughput distribution, and can lead to considerable reductions in the timing yield. For the given throughput requirement, FS gives only a 44% yield compared to the 71% with VFI-8. This information can be used by system designers to make informed trade-offs between the granularity of VFIs, i.e. design cost, and timing yield.

To show the benefits of variation-aware VFI partitioning, we compare the proposed heuristic VFI partitioning algorithm to a deterministic partitioning approach. From the perspective of clock-tree routing, it is desirable to have a low number of resources in each island, which can reduce the complexity of local clock wiring in the partitions. Therefore, for a specified number of voltage-frequency domains, the deterministic partitioning tries to equally distribute the resources into islands, thus reducing the maximum number of resources in a single partition. Figure 6 shows the CDF of the inverse of the system-level throughput for four and two domain architectures,

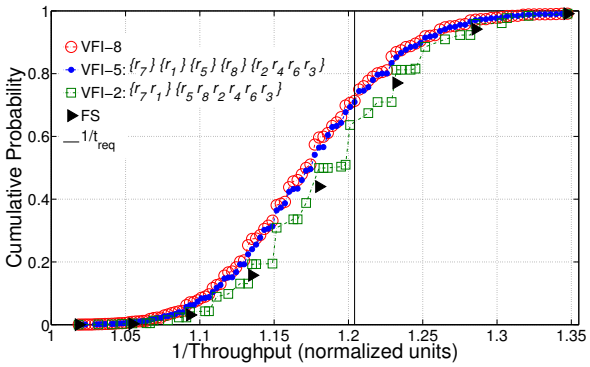


Fig. 5. Throughput CDF for VFI-8, VFI-5, VFI-2 and FS architectures; the grouping of resources into islands for VFI-5 and VFI-2 is decided by the proposed variation-aware VFI partitioning algorithm

based on both variation-aware and deterministic VFI partitions. As shown, with the deterministic partitioning (DVFI-4 and DVFI-2), the resources are equally distributed in the islands. The choice of resources in the islands is based on the adjacent resource index values. For the given throughput requirement, the deterministic four-domain partitioning (DVFI-4) results in an 11% lower yield than the one provided by the proposed variation-aware algorithm (VFI-4). Similarly, an 18% lower yield is achieved by DVFI-2, as compared to VFI-2. This shows the importance of variation-aware voltage-frequency partitioning for improved timing yield.

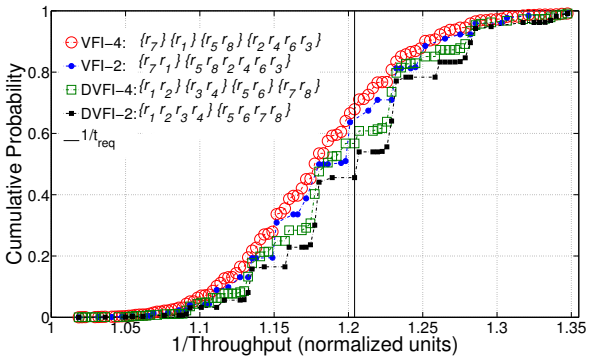


Fig. 6. Throughput CDF for four and two domain architectures, based on both variation-aware (VFI-4 and VFI-2) and deterministic partitions (DVFI-4 and DVFI-2)

C. Guard-band reduction impact on good dies

Reducing the design margins or guard-bands when implementing a circuit provides the benefit of decreased circuit area, resulting in a higher number of gross dies on a wafer. This in turn may provide a higher number of good dies, which satisfy the throughput requirement imposed on the system. In this section, we demonstrate on the presented case study, how our framework is used to estimate the impact of guard-band reduction on the number of good dies on a wafer. The number of gross dies on a wafer is given by Equation (11), where l is the radius of the wafer and V is the die area [10]; the second term in the equation accounts for wasted area around the edges of the circular wafer.

$$N_{\text{gross}} = \pi \cdot \left(\frac{l^2}{V} - \frac{2l}{\sqrt{2V}} \right) \tag{11}$$

The number of good dies on a wafer (i.e. dies (chips) that satisfy the system throughput requirement) is then given by the product of timing yield (Y_t) and the number of gross dies (Equation 12). Note that random defect yield can also be taken into account in the equation, as shown in [10]. However, due to low error density values and a relatively small die size, as assumed in our work, the random defect yield is high and has a negligible impact on the number of good dies [10]. Furthermore, guard-band reduction has almost no impact on random defect yield [10]. For these reasons, we do not take random defect yield into consideration.

$$N_{\text{good}} = Y_t \cdot N_{\text{gross}} \tag{12}$$

Circuit guard-banding is typically done by using corner-files during the design and verification stages; these files describe the worst-case (WC) and best-case (BC) delay values of standard-cells, corresponding to slow and fast process corners, respectively. The change in circuit area when reducing these guard-bands (i.e. implementing the circuit with reduced WC and increased BC delay values) is assessed by Jeong *et al.* in [10]. They use open-source cores and an industrial embedded processor core with target clock frequencies ranging from 300 to 600 MHz; the cores are synthesized using 90, 65 and 45 nm technology model libraries. Based on measured data, the authors provide a linear regression model for circuit-area reduction versus guard-band reduction. Equation (13) shows the model, where v is the area reduction factor and u is the guard-band reduction in percent. We use this model to compute the number of gross dies on a wafer due to reduced guard-bands, and thus circuit area.

$$v = 1 - 0.0033 \cdot u \tag{13}$$

As mentioned, each guard-band value results in a particular circuit implementation with a certain area, after the design and verification stages. By performing statistical characterization (Monte Carlo simulations) on each circuit implementation, the PDFs of the maximum supported frequency of the resources is obtained. A statistical characterization flow is proposed by Miranda *et al.* in [16]. As no data on the probability distribution of the frequency of a core for different guard-band values is given by Jeong *et al.* in [10], we make the following intuitive assumptions. With the original guard-band (i.e. 0% guard-band reduction), we assume that $\approx 99.7\%$ of manufactured resource instances satisfy the target frequency, which is assumed to be 500 MHz in this analysis. This corresponds to a combined normal distribution of the maximum supported frequency, where $(\mu_0^r - 3\sigma_0^r) = 500$ MHz, as depicted in Figure 7. The standard deviation to mean ratio for the combined distribution is $3\sigma_0^r/\mu_0^r \approx 15.6\%$, as $3\sigma_l^r/f_n^r = 10\%$ and $3\sigma_g^r/f_n^r = 12\%$ (based on the available data at 45nm technology [19]). Therefore, $\mu_0^r \approx 592$ MHz. With no guard-band (i.e. 100% guard-band reduction), the combined distribution has a mean $\mu_1^r = (500 - \delta^r) = 500$ MHz (δ^r is assumed to be zero), as shown in Figure 7. A $u\%$ guard-band reduction results in a new combined normal distribution with a decreased mean frequency of $\mu^r = \mu_0^r - u \cdot (\mu_0^r - \mu_1^r)/100$. Figure 7 shows the combined distribution for $u = 40\%$. We assume that the standard deviation to mean ratio of the combined normal distribution for any $u\%$ guard-band reduction is constant (i.e. $3\sigma^r/\mu^r \approx 15.6\%$). In reality, this ratio may change due to local variation that depends on the design, which is different for each $u\%$ guard-band reduction (global variation does not depend on the design). However, only small differences in a close nominal-frequency range are expected (e.g. the difference between 500 MHz and 592 MHz nominal-frequency designs is much less than between 500 MHz and 2 GHz).

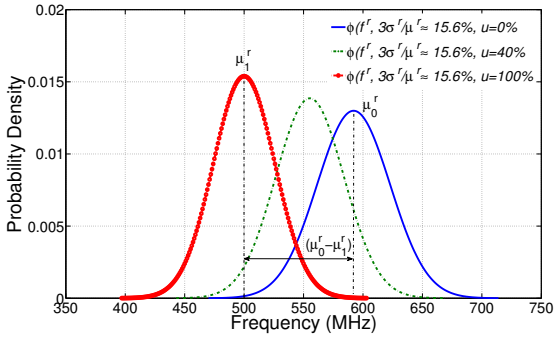


Fig. 7. Combined f^r PDF of a resource due to a 0%, 40%, 100% guard-band reduction; target frequency is 500 MHz; $3\sigma^r/\mu^r \approx 15.6\%$

We assume that the eight-resource chip has a die area of 10 mm^2, out of which 7 mm^2 consists of standard logic cells and 3 mm^2 comprises of embedded SRAM and IO cells. This data is assumed based on an eight-core ARM Cortex-A5 processor at 45 nm technology. Note that, with original guard-bands ($u = 0\%$), all resources can be clocked at 500 MHz. This requires a simple CGU providing a fixed clock frequency for all eight resources. In contrast, a higher cost (i.e. area) CGU per island is necessary with reduced guard-bands, as described in this work. We account for this additional area due to CGUs, and assume that a typical fine-grained CGU has a die area of 0.03 mm^2 [20]. For each guard-band reduction, the corresponding die area and the number of gross dies on a wafer are computed using Equations (13) and (11), respectively. The timing yield is evaluated by means of the proposed framework, using the PDFs of resource-frequency corresponding to the guard-band reduction. The throughput requirement of the cyclic synthetic application is set based on the target clock frequency of the resources, being 500 MHz for all eight resources. With the specified target frequencies, we assume that the application just satisfies its throughput requirement. This assumption enables us to have fair results when estimating the impact of guard-band reduction on timing yield (a relaxed throughput requirement creates a large slack in performance). We use the approach in [17] and perform a mapping of the synthetic cyclic application to the platform for the specified 500 MHz frequencies, such that the throughput is maximized. The result is a throughput value, which is taken as the requirement. The timing yield for different $u\%$ guard-band reduction, based on eight (VFI-8) and five (VFI-5) domain architectures, is shown in Table I.

TABLE I. TIMING YIELD FOR U% REDUCED GUARD-BANDS

u%	0	10	20	30	40	50	60	70	80	90	100
Y_t% (VFI-8)	99	99	98	97	94	88	79	69	51	38	27
Y_t% (VFI-5)	99	99	98	97	94	88	78	67	49	36	24

Figure 8 illustrates the change in the number of good dies per wafer against guard-band reduction for VFI-8 and VFI-5 architectures and for two different assumptions: 1) a design with fixed blocks, where the area of embedded SRAM

and IO cells does not change with guard-band reduction; and 2) a design without fixed blocks (i.e. hard macros are newly designed corresponding to the guard-band reduction). Figure 8 shows that the number of good dies is maximized at a 30% guard-band reduction for the VFI-5 architecture and fixed blocks. This results in a 3.7%[1] more dies that satisfy the throughput requirement imposed on the system. When there are no fixed blocks, a 40% reduction in the guard-band leads to a 7.7% increase in the number of good dies. Beyond 30%, 40% guard-band reduction, the number of good dies gradually decreases; this is because the reduction in the timing yield becomes considerable (see Table I). As Table I illustrates, the VFI-5 architecture does not reduce the timing yield of VFI-8 at 30% and 40% guard-band reductions. As VFI-5 requires a lower number of CGUs (a smaller die size), a 1% higher number of good dies is provided by VFI-5 at 30% or 40% guard-band reductions, as compared to VFI-8. Note that, guard-band reduction also benefits in reduced dynamic and leakage power. These results show that a higher number of good dies with reduced guard-bands is obtained, increasing profit.

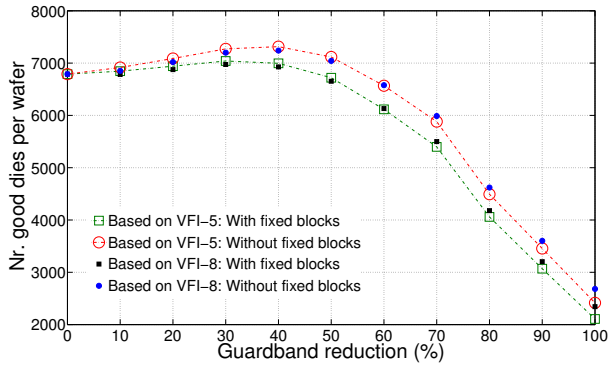


Fig. 8. Number of good dies per wafer against reduced guard-band for VFI-8 and VFI-5 architectures; designs with and without fixed blocks are considered

VII. Conclusions

This work presents a framework to assess the timing yield of a system with VFI partitions under the influence of within-die and die-to-die process-driven variations. We model a system consisting of a real-time streaming application mapped to variation-affected resources in an MPSoC by means of an SDF formulation. As a second contribution, we provide a methodology to partition the resources in an MPSoC into VFIs for maximized timing yield. On a case study, we demonstrate how our methodology can be used by system designers to make trade-offs between the number of VFI partitions (design cost) and timing yield. To this end, we show that not all reductions in the granularity of partitions reduce the timing yield. Specifically, the five domain architecture, where our partitioning algorithm groups resources with low criticality values into a single island, results in only a negligible reduction in the timing yield, as compared to the eight domain implementation. In contrast, the yield is lowered by 7% and 27% when moving to two clock-domain and fully synchronous architectures, respectively. Additionally, we illustrate how the framework can be used to estimate the impact of reducing circuit design margins on the number of good dies on a wafer. We show that 30% guard-band reduction results in a 3.7% increase in the number of good dies per wafer. Finally, we show that the proposed variation-aware partitioning provides up to 18% improvements in the timing yield compared to a deterministic partitioning.

[1] A 3.7% increase in the number of good dies is significant. For example, if 4 K wafers are required to produce 30 M good dies, a 3.7% higher number of good dies per wafer translates into 143 less wafers for the same 30 M good dies. For a wafer cost of 3000 $, the cost saving is 429000 $.

References

[1] S. S. Bhattacharyya *et al.* Synthesis of embedded software from synchronous dataflow specifications. *IJVSPA*, 21, 1999.

[2] K. Bowman *et al.* Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *JSSC*, 37(2), 2002.

[3] M. Damavandpeyma *et al.* Modeling static-order schedules in synchronous dataflow graphs. In *Proc. DATE*, 2012.

[4] M. Damavandpeyma *et al.* Parametric throughput analysis of scenario-aware dataflow graphs. In *Proc. ICCD*, 2012.

[5] S. Dighe *et al.* Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core TeraFLOPS processor. *JSSC*, 46(1), 2011.

[6] P. Friedberg *et al.* Modeling within-die spatial correlation effects for process-design co-optimization. In *Proc. ISQED*, 2005.

[7] S. Garg *et al.* System-level throughput analysis for process variation aware multiple voltage-frequency island designs. *TODAES*, 13(4), 2008.

[8] A. Hansson *et al.* Compsoc: A template for composable and predictable multi-processor system on chips. *TODAES*, 14, 2009.

[9] W. Jang *et al.* A voltage-frequency island aware energy optimization framework for networks-on-chip. *JETCAS*, 1(3), 2011.

[10] K. Jeong *et al.* Impact of guardband reduction on design outcomes: A quantitative approach. *SM*, 22(4), 2009.

[11] L.-F. Leung *et al.* Energy-aware synthesis of networks-on-chip implemented with voltage islands. In *Proc. DAC*, 2007.

[12] S. Londono *et al.* A better-than-worst-case circuit design methodology using timing-error speculation and frequency adaptation. In *Proc. SOCC*, 2012.

[13] S. Majzoub *et al.* PVT variation impact on voltage island formation in mpsoc design. In *Proc. ISQED*, 2009.

[14] D. Marculescu *et al.* Process-driven variability analysis of single and multiple voltage frequency island latency-constrained systems. *TCAD*, 27(5), 2008.

[15] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. 1st edition, 1994.

[16] M. Miranda *et al.* Variability aware modeling of SoCs: From device variations to manufactured system yield. In *Proc. ISQED*, 2009.

[17] D. Mirzoyan *et al.* Process-Variation Aware Mapping of Real-Time Streaming Applications to MPSoCs for Improved Yield. In *Proc. ISQED*, 2012.

[18] U. Ogras *et al.* Design and management of voltage-frequency island partitioned networks-on-chip. *VLSI*, 17(3), 2009.

[19] L. T. Pang *et al.* Measurement and analysis of variability in 45nm strained-Si CMOS technology. In *Proc. CICC*, 2008.

[20] A. Rylyakov *et al.* A wide tuning range (1 ghz-to-15 ghz) fractional-n all-digital pll in 45nm soi. In *Proc. CICC*, 2008.

[21] A. Shabbir *et al.* CA-MPSoC: An automated design flow for predictable multi-processor architectures for multiple applications. *JCA*, 56, 2010.

[22] S. Sriram *et al. Embedded Multiprocessors: Scheduling and Synchronization*. 2000.

[23] S. Stuijk *et al.* SDF3: SDF for free. In *Proc. ACSD*, 2006.

[24] S. Stuijk *et al.* Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Proc. DAC*, 2007.

[25] S. Stuijk *et al.* Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *TC*, 57(10), 2008.

[26] M. Wiggers *et al.* Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proc. DAC*, 2007.

[27] R. Wilhelm *et al.* The worst-case execution-time problem overview of methods and survey of tools. *TECS*, 7(3), 2008.

APPENDIX

A. EXTENDED RESULTS

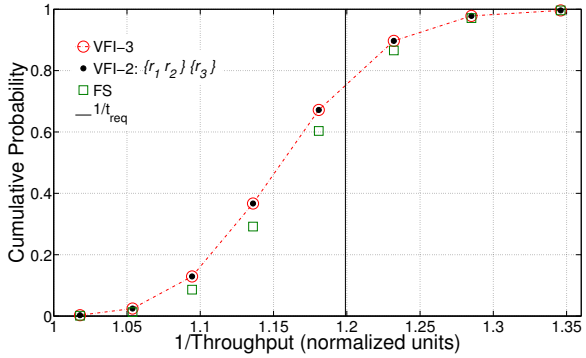
In this supplementary section, we present additional case studies based on a set of four real applications modeled as SDF graphs. The set contains an MP3 decoder [25], modem [1], H.263 decoder [25] and MP3 playback [26]. An overview to the application SDF graphs, showing the number of actors and cycles, is shown in Table II. All four applications have parallelism that can be exploited by three resources. Therefore, they are mapped to a multi-processor platform consisting of three resources. The assumptions on resource frequencies and on the variation in them made in Section VI-A also hold in this section. A set of eight equidistant clock-frequency levels is selected for each island (resource). The mapping and the scheduling of actors on shared resources is determined as described in Section VI-A. The same section also details how the proposed models are implemented.

TABLE II. APPLICATION OVERVIEW

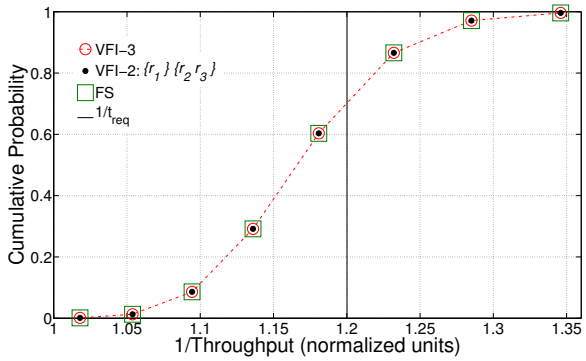
SDF graph	# actors	# cycles
MP3 decoder	14	0
Modem	16	5
H.263 decoder	4	0
MP3 playback	4	1

Figure 9 illustrates the CDFs of the system-level throughput for the MP3 decoder, modem, H.263 decoder and MP3 playback applications. For all applications, architectures with three (VFI-3) and two (VFI-2) voltage-frequency domains, as well as a fully synchronous (FS) system are considered. For the MP3 decoder, VFI-3 and VFI-2 result in identical throughput distributions. This is due to two resources having equal criticality values. Grouping them in a single island does not reduce the throughput of the parametric bound graph. In contrast, FS can cause up to an 8% reduction in the timing yield. For the modem application (Figure 9b), all three implementations result in an identical distribution. This shows that all three resources in the platform have equal criticality values. Therefore, only a single CGU unit is sufficient for achieving the maximum yield for this application; this results in reduced design costs. For the H.263 decoder (Figure 9c), the throughput distribution with VFI-2 closely follows the one with VFI-3. In contrast, FS can lead to a considerable decrease in the timing yield of up to 22%. For the MP3 playback application (Figure 9d), the throughput distributions with VFI-3 and VFI-2 are identical, again showing that two resources are equally critical. The FS architecture results in a sub-sampled throughput distribution over the one with VFI-3, and can lead to an up to 6% reduction in the timing yield. However, based on the throughput requirement, FS can also provide a timing yield equal to the one for VFI-3.

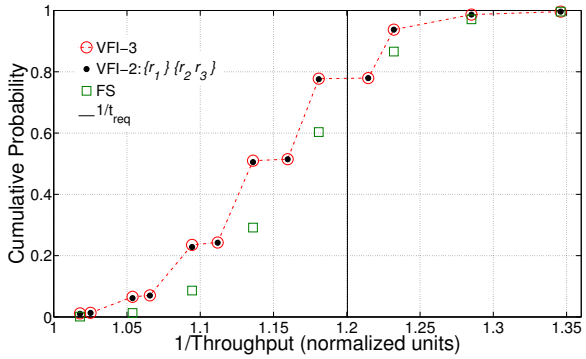
The discussion presented in this section demonstrates on realistic applications how the proposed framework can be used by system designers to make trade-offs between the number of VFI partitions, i.e. design cost, and timing yield.



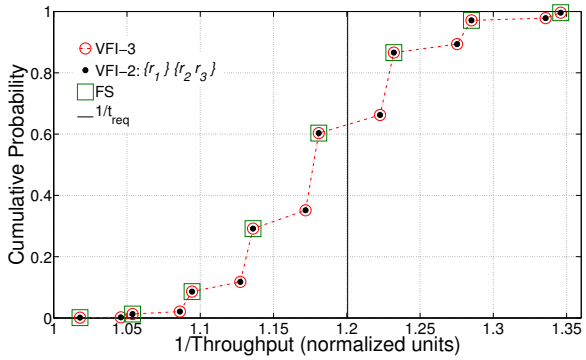
(a) MP3 decoder



(b) Modem



(c) H.263 decoder



(d) MP3 playback

Fig. 9. Throughput CDF for VFI-3, VFI-2 and FS architectures; the grouping of resources into islands for VFI-2 is decided by the proposed variation-aware VFI partitioning algorithm