Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors

Qiang Wu   Philo Juang   Margaret Martonosi   Douglas W. Clark
Dept. of Computer Science, Dept. of Electrical Engineering
Princeton University, Princeton, NJ 08544
{jqwu, pjuang, mrm, doug} @princeton.edu

ABSTRACT

Multiple Clock Domain (MCD) processors are a promising future alternative to today's fully synchronous designs. Dynamic Voltage and Frequency Scaling (DVFS) in an MCD processor has the extra flexibility to adjust the voltage and frequency in each domain independently.

Most existing DVFS approaches are profile-based offline schemes which are mainly suitable for applications whose execution characteristics are constrained and repeatable. While some work has been published about online DVFS schemes, the prior approaches are typically heuristic-based. In this paper, we present an effective online DVFS scheme for an MCD processor which takes a formal analytic approach, is driven by dynamic workloads, and is suitable for all applications.

In our approach, we model an MCD processor as a queue-domain network and the online DVFS as a feedback control problem with issue queue occupancies as feedback signals. A dynamic stochastic queuing model is first proposed and linearized through an accurate linearization technique. A controller is then designed and verified by stability analysis. Finally we evaluate our DVFS scheme through a cycle-accurate simulation with a broad set of applications selected from MediaBench and SPEC2000 benchmark suites. Compared to the best-known prior approach, which is heuristic-based, the proposed online DVFS scheme is substantially more effective due to its automatic regulation ability. For example, we have achieved a 2-3 fold increase in efficiency in terms of energy-delay product improvement. In addition, our control theoretic technique is more resilient, requires less tuning effort, and has better scalability as compared to prior online DVFS schemes.

We believe that the techniques and methodology described in this paper can be generalized for energy control in processors other than MCD, such as tiled stream processors.

Categories and Subject Descriptors: C.1.3 [Computer System Organization]: Processor Architectures

General Terms: Design

Keywords: Dynamic Voltage/Frequency Scaling, Formal Methods, MCD processors

1. INTRODUCTION

Due to the trends of increasing clock speed and increasing scale of integration, two fundamental problems are present in today's synchronous microprocessor designs. First, global clocking distribution with low clock skew is becoming increasingly difficult. Second, power consumption is becoming a limiting factor to performance. One possible solution to these problems is a Multiple Clock Domain (MCD) processor, which uses the Globally Asynchronous and Locally Synchronous (GALS) clocking style [12, 20]. An MCD processor has less clock distribution and skew burden. It is also more power efficient due to the absence of a global clock distribution tree. Another important benefit from an MCD processor is the extra flexibility in Dynamic Voltage and Frequency Scaling (DVFS), as the frequency and voltage in each function block or domain can be adjusted independently. Therefore, DVFS in an MCD processor can achieve better energy savings than DVFS in a traditional (fully) synchronous processor [12, 23].

The goal of this work is to design an *analytic online* DVFS scheme for an MCD processor. We categorize existing DVFS work (for MCD processors or other general processors) along two dimensions. The first dimension is the level of dynamism. The level of dynamism ranges from low: profile-based offline schemes, to high: online schemes driven by dynamic workloads. Most existing DVFS schemes are offline, for example [10, 16, 18, 28]. They typically use profiling to do offline analysis to obtain optimal DVFS settings for some set of assumptions about program data sets and execution model. Then either a compiler [10, 28] or a binary editor [18] is used to write the DVFS configuration into the application source. Naturally, the effectiveness of a profile-based DVFS scheme depends on how similar the run-time characteristics are to the offline analysis at profile time. Therefore, DVFS schemes in this group are best suited to special-purpose applications, such as multimedia applications, whose run-time characteristics are constrained and repeatable. In contrast, in our work we propose an online DVFS scheme which takes runtime information directly from the processor to infer characteristics of the application dynamically, and is thus suited to all kinds of applications.

The other dimension for categorizing existing DVFS schemes is the level of formalism. The level of formalism ranges from purely heuristic-based to formal analytic schemes. Although many offline schemes take mathematical optimization-based formal approaches [10, 16, 28], nearly all existing online DVFS schemes are heuristic-based [11, 19, 23]. They typically include a set of manually selected rules and threshold values. At run time, certain processor metrics, such as cache miss rate [19] or queue occupancy [11, 23], are monitored. These metrics are then compared to the threshold values and one of the rules is applied depending on the result of the comparison. The best known online DVFS scheme for an MCD processor is the AttackDecay algorithm by Semeraro *et al* [23]. However, there are significant limitations in heuristic-based schemes. First, for a result obtained from a given set of rules and parameters, it is not analytically clear how to further improve it and thus make the DVFS more effective. Second, the trial-and-error tuning process for parameters is very time consuming. Third, it is generally hard to scale the heuristics for a large system, as the number of rules and the tuning effort required can grow exponentially.

To overcome the above limitations, the online DVFS scheme in this paper takes a rigorous analytical approach. We model an MCD processor as a queue-domain network. The online DVFS problem is formulated as a feedback control problem with issue queue occupancies as feedback signals . Specifically, a stochastic model is proposed for the queue-domain dynamics. Since the queue-domain system is inherently nonlinear, we first linearize the system through an accurate *feedback linearization*. The controller is then designed and verified by stability analysis. Next, a possible hardware implementation of the controller is described. Finally the proposed online DVFS scheme is evaluated by a cycle-accurate architecture simulator with a broad set of applications selected from the MediaBench and SPEC2000 benchmark suites. Overall, we achieve a power-saving to performance degradation ratio of 6.2 (i.e. on average, 6.2% power is saved for 1% performance degradation), as compared to a ratio of 2.5 for the conventional synchronous voltage scaling.

Compared to the best known prior online DVFS approach for an MCD processor [23] which is heuristic-based, the proposed DVFS scheme has several significant advantages. First, the analytic online DVFS scheme is more effective in terms of energy saving for the same level of performance degradation. Experimental results show we achieve an energy-delay product improvement 146% higher than that of the best-known heuristic-based online scheme in [23], and 11% higher than the semi-oracle-based DVFS scheme in [24]. We attribute this promising result to the automatic frequency/voltage regulation ability in the proposed DVFS controller, which leads to a more effective and precise decision on when, where, and how much to scale. In addition, the proposed analytic online DVFS scheme requires less tuning effort than heuristic-based DVFS as the analytic DVFS chooses its control parameters through stability analysis. Furthermore, the analytic DVFS scheme is more scalable – we do not have to re-tune parameters or re-set rules in order to include new resources. Also, we can extend it to a global and centralized DVFS scheme, which will handle interactions among multiple clock domains (as will be discussed in Section 5).

Overall, we feel that the primary contribution of this work is threefold. First, our focus is on voltage/frequency control for MCD architectures, which are relatively new and have great potential in terms of energy-saving and performance improvement. Second, the proposed control-theoretic techniques applied to DVFS in MCD processors have led to a 2-3 fold increase in efficiency compared to the best-known previous heuristic-based DVFS scheme in MCD processors. In addition, our control-theoretic technique is more resilient, complete, and boundable as well. For example, we can guarantee stability, achieve significant energy savings, and offer more graceful degradation even under extreme cases. Third, this is one of the first rigorous analytic approaches to DVFS control. Previous control theoretic techniques exist [17], but only for multimedia processors with predictable workloads, while our work is for general workloads.

The rest of the paper is structured as follows. In Section 2, we give a brief review of the MCD processor design and implementation. Section 3 describes the modeling, design, and analysis of our online DVFS controller. This is followed in Section 4 by experimental results for the purpose of evaluation. In Section 5, we give a general outline of centralized DVFS design. Finally, Section 6 offers our conclusions.

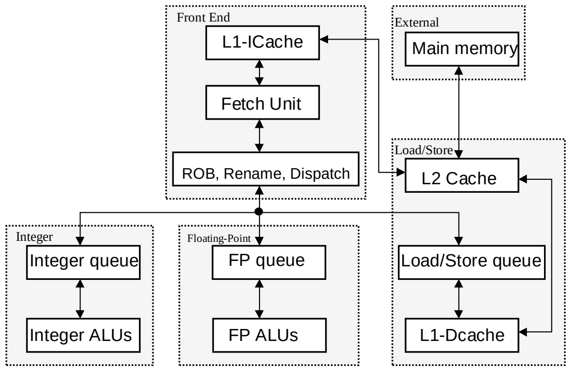

Figure 1: The clock domain partitions in an MCD processor by Semeraro *et al* [24]

2. BACKGROUND: MULTIPLE CLOCK DOMAIN MICROPROCESSORS

Some computer architects and researchers have predicted that, in order to overcome the increasing severe problems of clock distribution and power consumption, future high-performance microprocessors may need to have multiple clock domains (MCD) or use some form of asynchrony [1, 21]. MCD processors use the Globally Asynchronous Locally Synchronous (GALS) clocking style [20]. Each function block or domain operates with an independently generated clock, and synchronization circuits ensure reliable inter-domain communication.

Advantages of MCD processors include less clock distribution and skew burden, less power consumption due to the absence of a global clock tree, DVFS flexibility, and design modularity [20]. The primary disadvantage of MCD processors is the inter-domain communication and synchronization overhead. An interface circuit is needed if data passes between two domains.

One key design issue for an MCD processor is the choice of where to partition the clock domains. It is still an open research question on how to partition in order to maximize the power-performance benefit. Most existing MCD implementations use architectural functional blocks as *natural* boundaries for clock domains. For example, Figure 1 shows a 4-domain partition used by the MCD implementation by Semeraro *et al.* [24], which consists of the front end, integer processing core (INT), floating point processing core (FP), and load store unit (LS). The main memory is considered as an external separate clock domain not controlled by the processor (for more details see [24]). Another popular MCD implementation by Iyer and Marculescu [12] uses a 5-domain partition, which is similar to that in Figure 1 but with the front-end split into two clock domains.

Another key design issue for an MCD processor is the synchronization interface design. A good interface design needs to have low latency, high throughput, and virtually no synchronization failure (i.e. no metastability). Nearly all of the existing MCD interface designs use some kind of queue structures for efficiency. One group of designs [6] uses token-ring based FIFOs, which have a very low

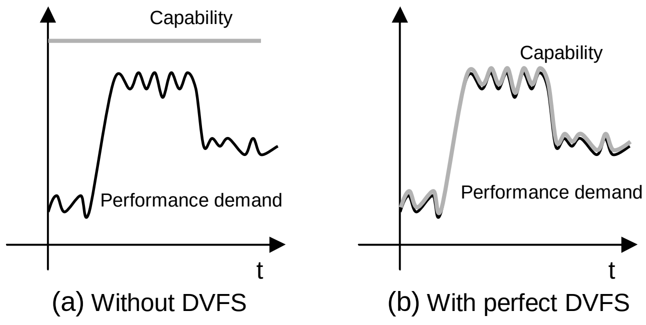

Figure 2: A conceptual illustration of a perfect online DVFS for a clock domain in an MCD processor.

latency and low synchronization overhead (there is no synchronization cost if the token-ring FIFO is neither full nor empty). Another group of designs uses arbitration-based queue structures (often with a stoppable clock) [26, 29]. The designs in this group are typically failure-free, but may need to check synchronization for each data transfer. For example, the design by Sjogren and Myers [26] includes arbitration and synchronization circuits which can detect whether the source and destination clock edge are far enough apart (i.e., greater than the so-called synchronization window size in [23]), in order for the source generated signal to be successfully accessed at the destination. This design has been used by the MCD implementation in [24]. Note that for a situation like that in Figure 1, where issue queues already exist between some domains, the interface queue structure can be integrated with the existing issue queue to form a combined issue/interface queue structure.

The MCD implementations in [12] and [24] also provide the capability of independently configuring the frequency and voltage in each clock domain. An aggressive XScale-style DVFS model [7] is assumed, in which a clock domain can execute through the DVFS transition and the penalty is negligible due to a domain being idle waiting for the PLL [7]. Also, the XScale model allows any frequency to be used within the allowable range.

The quantitative benefit/overhead studies in [12, 24] have shown that an MCD processor has the potential to achieve significant power and performance efficiency. However, many design and control issues remain open and need more investigation in order to fully take advantage of the power/performance benefits brought by the MCD processors [20]. Online DVFS is one of these issues, as we will show next.

3. ONLINE DVFS DESIGN: AN ANALYTIC APPROACH

3.1 Problem formulation

Conceptually, the online DVFS problem for an MCD processor is to scale the frequency f to match the varying performance demand in each clock domain. In other words, we want to adapt frequency to workload changes. Figure 2a depicts a typical scenario with varying demand or workload. For a clock domain, changing clock frequency will generally change the execution speed or capability of the domain. So, a perfect DVFS scheme will lead to a perfect match between the demand and domain execution capability, as illustrated in Figure 2b. In the figure, *perfect* is in the sense of no performance degradation and elimination of all energy wasted due to excessive capability slack. Thus, the goal of our online DVFS scheme is to get results as close as possible to that of perfect DVFS.

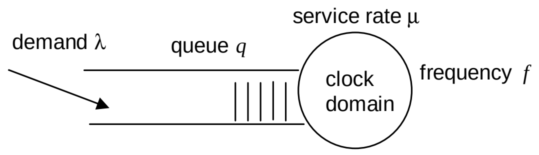

Figure 3: A single queue model for a clock domain with an input queue.

Note the excessive capability slack in Figure 2a is just the gap between the capability line and the demand line. We refer to the energy wasted due to this slack as E_{slack}. So for a *perfect* DVFS in Figure 2b, the energy savings will be all of the E_{slack}.

In addition, in Figure 2, we can continue to scale the execution capability line arbitrarily low below the performance need for non-realtime applications to get more energy savings, but that comes with a price of performance degradation, as the performance demand can not be satisfied. We call this part of energy savings *non-slack energy savings* or $E_{\text{non-slack}}$, to distinguish it from the "free" energy savings due to the elimination of E_{slack}.

In this paper, we want to make use of the queues in an MCD to guide DVFS. Recall from Section 2 that there are interface queues between clock domains for synchronization. Intuitively, these queues give clues about the speed balance between the sender domain and the receiver domain. For example, an emptying queue may mean the receiver is too fast relative to the sender; a filling queue may mean it is too slow; a stationary queue means a perfect match of receiver speed relative to the sender. This suggests a feedback control scheme for DVFS which uses the queue occupancy as a feedback signal to control the domain frequency, making its execution speed adaptive to varying demand. If the adaptiveness is fast enough, the DVFS scheme should get results close to the perfect matching in Figure 2.

There are, however, significant challenges to pursuing this idea along a rigorous analytic direction. First, to design a simple, hardware controller based on control theory, we need an analytical model for the involved queue and domain dynamics. Second, the queue-domain relationships are inherently time varying and nonlinear. So techniques may be needed to solve the nonlinearity problem accurately before we can design an effective DVFS controller.

In the rest of this section, we describe modeling and design of an online DVFS scheme for local queues and domains. That is, the DVFS controller considers each queue locally and separately, assuming the interactions between queues are weak and can be ignored (the so-called decentralized DVFS scheme). We leave the issue of centralized DVFS to Section 5.

3.2 Overview of modeling and design of DVFS controller

This subsection is intended to give an overview of the modeling and design of our online DVFS controller without going through control theoretic and mathematical details (those details will be given in subsections 3.3 - 3.6).

We use a local queue model as shown in Figure 3. The circle in that figure represents the clock domain being considered (such as a floating point functional unit). The domain has a frequency f and an execution capability or service rate μ, which is a function of f. The performance demand is denoted as λ. The queue has a finite size. (Note this model uses an input queue to a domain. For the case of an output queue, duals of the arguments in this section will hold.)

Figure 4 shows the design flow of our DVFS controller, with the block diagrams corresponding to the major steps for our methodol-

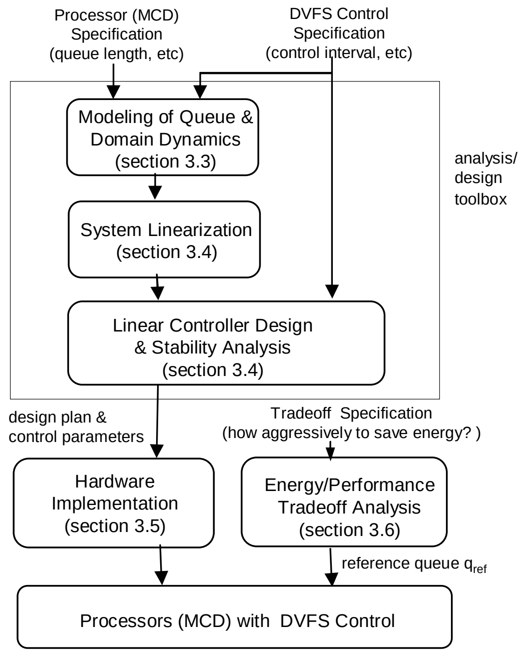
Processor (MCD) Specification (queue length, etc)
DVFS Control Specification (control interval, etc)
Modeling of Queue & Domain Dynamics (section 3.3)
System Linearization (section 3.4)
Linear Controller Design & Stability Analysis (section 3.4)
analysis/ design toolbox
design plan & control parameters
Tradeoff Specification (how aggressively to save energy?)
Hardware Implementation (section 3.5)
Energy/Performance Tradeoff Analysis (section 3.6)
reference queue q_ref
Processors (MCD) with DVFS Control

Figure 4: Design flow for our DVFS controller in MCD processors. Section numbers inside the blocks indicate the location where the details of each design step can be found.

ogy. The inputs to the modeling and design blocks are the processor (MCD) specification and the DVFS control specification. These include queue length, clock domains and their frequency ranges, control interval (i.e. the time interval for one possible change of frequency/voltage), sampling period, queue measures used as feedback signals, and other control performance requirements such as maximum percent overshoot allowed.

The first major step is the modeling of the involved queue and clock domain dynamic. (The details are described in subsection 3.3.) Based on the processor and control specifications, we derive a mathematical model for the controlled system, which is expressed as a set of difference equations. These equations describes the dynamic relationship among the feedback signal (q), the demand (λ), and the frequency (f).

The next two major steps are the system linearization and the linear controller design (both are described in subsection 3.4). Since the controlled queue/domain system is inherently nonlinear, we first linearize the system using an accurate linearization technique called *nonlinear transformation* [2]. As shown in Figure 5, this technique essentially adds a nonlinear transformer to the feedback path to compensate for the nonlinearity in the original controlled system. We then design a variation of the Proportional-Integral-Derivative (PID) controller for this linearized system, which is commonly used in industry [14, 27]. Intuitively, the PID-based controller adjusts the execution rate to adapt to the workload change. This adjustment is in proportion to the value of the workload change, the rate of the workload change, and takes into account the prior history of workload changes. Figure 5 shows the control block diagram, where q is the measured feedback signal; q_{ref} is the target or nominal queue operating point; e is the error signal and the input to the PID-based controller; μ is the obtained control signal (or actuate signal) which is proportional to the value of the error, the integral of previous errors, and the rate of the error change; f is the frequency and is obtained from μ through a nonlinear transformation; demand λ is the disturbance input which the controller is trying to adjust the execution rate to match.

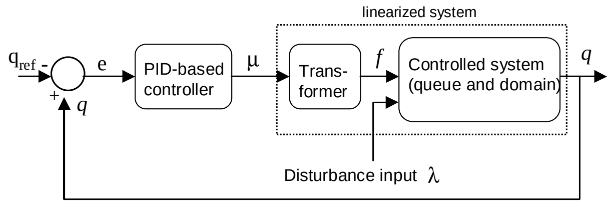

Figure 5: Control block diagram for a Proportion-Integral-Derivative (PID) controller for DVFS .

The control parameters (or control gains) for the PID-based controller are decided by the stability and control performance analysis. Note these control gains are the main output of the analysis/design toolbox. The next major step is to take the design with the control gains, and implement it in hardware (we show a possible hardware implementation in subsection 3.5). This finishes the design cycle of a DVFS controller in MCD processors.

A key design setting for the above DVFS controller is the reference queue occupancy q_{ref}. As we will show in subsection 3.6, the value of q_{ref} specifies the actual tradeoff between performance degradation and energy saving. We can increase q_{ref} to make DVFS control more aggressive in saving energy, or decrease q_{ref} to value performance more. Note that a design could make the value of q_{ref} adjustable by the OS or application software (e.g. through a special mode set instruction). This mechanism provides an opportunity for hardware/software cooperation in the sense that the hardware is responsible for implementing the fine details of speed adaptation while the OS/application software will make the overall policy decision (through q_{ref}) on how aggressively to save energy or preserve performance.

In the rest of this section, we will describe in detail the design steps in Figure 4. We will start with the modeling and linear controller design, which involves some control theoretic and derivation details. (People who are already familiar with these derivations or are not interested in these details may wish to skip subsections 3.3 and 3.4.) In practice, these control theoretic techniques/details can be incorporated into an analysis/design toolbox. A designer is able to use the toolbox to get the control parameters, and then use them directly in the DVFS policy controller design.

3.3 Analytic modeling of queue and clock domain dynamics

As mentioned earlier, we use a local queue model as shown in Figure 3. For a clock domain, the frequency and corresponding voltage cannot change instantaneously, and there is a minimum time requirement for one possible change of frequency. So we have a control interval such that the frequency is fixed inside an interval. Using T as the length of a control interval, the k_{th} control interval is just the time period $[kT, (k+1)T]$. Also we denote N as the total number of sampling periods in a control interval, and assume each sampling period has a length of Δt, so $T = N\Delta t$.

We want to first model the performance demand and service rate in a control interval. The demand $\lambda_{(t)}$ and service rate $\mu_{(t)}$ inside each control interval are modeled as an independent and stationary random process along the time axis [9] (i.e., they have identical distributions for all t inside an interval). We denote the expected values and variances (or noise levels) of $\lambda_{(t)}$, $\mu_{(t)}$ as $\bar{\lambda}$, $V_{(\lambda)}$, $\bar{\mu}$, and $V_{(\mu)}$ respectively. (From now on, for a variable x, we will use $\bar{x}$ and $V_{(x)}$ to represent the expected value and variance of x.) Also, we use a subscript k as in $\bar{\lambda}_k$, $V_{(\lambda_k)}$ to indicate these values are for the k_{th} control interval.

Consider $q(t)$ as the queue occupancy at time t. Then the basic queue equation is expressed as follows (essentially a simplified version of the Lindley equation [15]) :

$$q_{(t+\Delta t)} - q_{(t)} = (\lambda_{(t)} - \mu_{(t)})\Delta t \tag{1}$$

Intuitively, this means that queue occupancy change in a unit time is equal to the number of arrived elements minus the number of departed elements in that unit time.

Next we want to use the above basic queue equation to model the queue-domain dynamics across different control intervals. First, we need to define the feedback signal and other necessary dynamic state variables. For a control interval, there are a number of ways to measure the queue utilization and use it as a feedback signal to the controller. In this paper, we use the average queue occupancy over all sampling points in the previous interval as the feedback signal for the current interval (as was used by the heuristic-based DVFS approach in [23]). We denote this feedback signal for the k_{th} control interval as q'_k. Also, the queue occupancy at the starting point of the k_{th} interval is q_k, so we have $q_k = q(kT)$. If we express these two dynamic state variables q'_k and q_k in terms of values from the previous interval, we have

$$\begin{aligned} q'_k &= \frac{1}{N}\sum_{i=1}^{N} q_{((k-1)T+i\Delta t)} \\ q_k &= q_{((k-1)T+N\Delta t)} \end{aligned} \tag{2}$$

Next we recursively expand equations (2) using the the basic queue dynamics in (1) and take the expectation of both sides of the expanded equation. Since both λ and μ are stationary in a control interval, $\bar{\lambda}_k$ and $\bar{\mu}_k$ are independent of time t for any k. So we have

$$\begin{aligned} \bar{q}'_k &= \bar{q}_{k-1} + \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{i}(\bar{\lambda}_{k-1} - \bar{\mu}_{k-1})\Delta t \\ &= \bar{q}_{k-1} + \frac{T+\Delta t}{2}(\bar{\lambda}_{k-1} - \bar{\mu}_{k-1}) \\ \bar{q}_k &= \bar{q}_{k-1} + (\bar{\lambda}_{k-1} - \bar{\mu}_{k-1})T \end{aligned} \tag{3}$$

Intuitively, this means $\bar{q}'_k$, the average queue occupancy over a control interval, is equal to the sum of the queue occupancy at the beginning of the interval ($\bar{q}_{k-1}$) and the average queue changes due to the differences between the demand rate $\bar{\lambda}_{k-1}$ and the service rate $\bar{\mu}_{k-1}$. A similar interpretation exists for $\bar{q}_k$, the queue occupancy at the beginning of next interval, in the above equation.

The above equation describes the dynamics in the considered queue-domain system across different control intervals. The RHS of equation (3) is expressed in term of the expected value of the service rate $\bar{\mu}_k$, while the real control signal is f_k. So we need to model their relationship. We use the following model

$$\bar{\mu}_k = \frac{1}{\bar{t}_1 + \frac{\bar{C}_2}{f_k}} \tag{4}$$

The above equation comes from the fact that, in most clock domains, execution time can be separated into two parts – one that is independent of frequency and one that is dependent. For example, in a load/store domain, the time spent for accessing asynchronous memory due to a cache miss is independent of frequency, while the time for querying and accessing cache is dependent on frequency. Accordingly, in the above model, $\bar{t}_1$ is the average amount of unit time (or task step) per instruction that is independent of frequency, and $\bar{C}_2$ is the average number of frequency-dependent cycles per instruction. Parameters $\bar{t}_1$ and $\bar{C}_2$ can be estimated dynamically using techniques similar to those in [13, 28].

Putting (3) and (4) together, and dropping Δt in (3) as $T \gg \Delta t$, we have an analytic model for the considered queue-domain system as

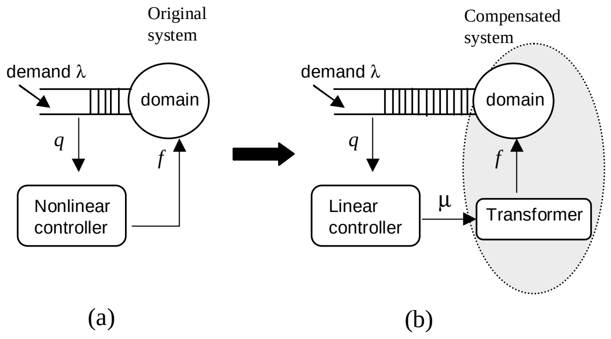

Figure 6: Linearization of the original dynamic system through a nonlinear transformation on the feedback path

$$\begin{aligned} \bar{q}'_k &= \bar{q}_{k-1} + \frac{T}{2}(\bar{\lambda}_{k-1} - \frac{1}{\bar{t}_1 + \frac{\bar{C}_2}{f_{k-1}}}) \\ \bar{q}_k &= \bar{q}_{k-1} + (\bar{\lambda}_{k-1} - \frac{1}{\bar{t}_1 + \frac{\bar{C}_2}{f_{k-1}}})T \end{aligned} \tag{5}$$

Note in the above modeling of queue dynamics, we assume the queue at the nominal operating point is partially full. The queue-domain relations when the queue is completely empty or full will be different from that in (1), which must be modeled using some discrete binary functions like those we will use in Section 5.

In addition, in the above model, we have only considered the expected values of λ and μ, and have not considered the variance or noise level of the input $V_{(\lambda)}$, $V_{(\mu)}$. Particularly, we want to check how the input noise propagates in the feedback signal $\bar{q}'$, since the noise in the feedback signal may affect the efficiency of a controller. We compute the variance $V_{q'_k}$ in one interval, assuming the queue occupancy is known at the beginning of the interval. By expanding the first sub-equation in (2), taking the variance of both sides of the expanded equation, and following the variance calculation rules [9], we have

$$\begin{aligned} V_{(q'_k)} &= \frac{T+\Delta t}{2T}\,(V_{(\lambda_{k-1})} + V_{(\mu_{k-1})}) \\ &\approx \frac{1}{2}\,(V_{(\lambda_{k-1})} + V_{(\mu_{k-1})}) \end{aligned} \tag{6}$$

From (6), we see the variance or noise in the feedback signal q' is not amplified and stays at a roughly the same level as the input noise. This is beneficial to the controller because smaller noise in the feedback signal tends to have less negative impact on the efficiency of the controller.

3.4 DVFS controller design

A straightforward design approach for our goals would be to design a controller for f_k, as shown in Figure 6a. However, as indicated in equation (5), this control system is nonlinear, and it is generally hard to design an effective controller for a nonlinear system, as there are very limited control techniques and tools for a general nonlinear system [14]. Fortunately, the nonlinearity inside this system can be separated, so we can have a nonlinear transformation on the feedback path to compensate for the nonlinearity in the original system dynamics; in this way, the compensated system becomes a linear one. This accurate linearization technique is the so-called *feedback linearization* or *nonlinear transformation* [2], as shown in Figure 6b. Specifically, we take μ_k as the control signal for the compensated linear system. The actual or internal control signal f_k is obtained through a transformation on the feedback path.

After the linearization, we can design the controller using a rich set of linear control techniques [14]. A popular choice would be a variation on the Proportion-Integral-Derivative (PID) controller. In this work, we use a PI controller because it is relatively simple

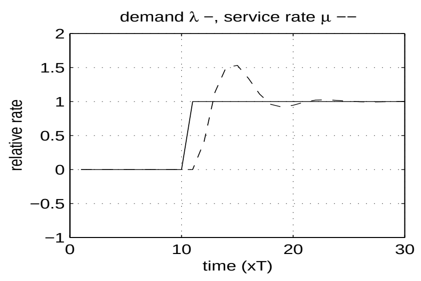

Figure 7: Step response for the system in (7) with $K'_I = 0.2$, $K'_P = 0.6$.

to design and robust in terms of steady-state control performance. The controlled system with the PI controller can be described by the following state equations.[1]

$$\begin{array}{rcl} \bar{q'}_k & = & \bar{q}_{k-1} + \frac{T}{2}(\bar{\lambda}_{k-1} - \bar{\mu}_{k-1}) \\ \bar{q}_k & = & \bar{q}_{k-1} + (\bar{\lambda}_{k-1} - \bar{\mu}_{k-1})T \\ e_k & = & \bar{q'}_k - q_{ref} \\ \bar{\mu}_k & = & \bar{\mu}_{k-1} + K_I e_k + K_P(e_k - e_{k-1}) \\ f_k & = & \frac{\bar{C}_2\bar{\mu}_k}{1-\bar{t}_1\bar{\mu}_k} \end{array} \qquad (7)$$

where the first two sub-equations are simply the analytic queue-domain model in (5); q_{ref} is the reference queue occupancy, i.e. the target or nominal operating queue point (more discussion of this in Section 3.6) ; e_k is the error signal; μ_k is the new service rate coming from the PI controller, with K_I and K_P the control parameters (or the so-called control gains); f_k is the new clock frequency obtained by solving (4) for f_k.

We then can proceed with stability and transient performance analysis of (7) and choose appropriate control gains K_I and K_P [14], as described next.

In equation (7), we define μ_k as the output signal, and λ_k as the disturbance input signal. Also we define new constants $K'_I = K_I T$, $K'_P = K_P T$. We can get the transfer function through the input-output difference equation and the *z*-transformation [14]. From the transfer function, we can get the corresponding characteristic equation as

$$2z^3 + (K'_I + K'_P - 4)z^2 + (2 + K'_P)z - K'_P \quad = \quad 0 \qquad (8)$$

From stability theory [14], the system in (7) is stable if and only if the roots of (8) are all inside the unit cycle of the *z*-plane. Assuming the roots of (8) are z_1, z_2, and z_3, we have the following relationships between the roots and the setting K'_I, K'_P, using a standard coefficient matching technique.

$$\begin{array}{rcl} K'_I & = & 2(z_1z_2 + z_2z_3 + z_3z_1) - 2 \\ K'_P & = & 2z_1z_2z_3 \\ z_3 & = & \frac{3-z_1z_2-z_1-z_2}{z_1z_2+z_2+z_3+1} \end{array} \qquad (9)$$

With equation (9), we can use the poles-placement technique [14] to choose the appropriate K'_I,K'_P setting for a given stability margin and transient performance requirement. For example, if z_1 and z_2 are chosen with a very small magnitude (e.g. $z_1 = z_2 = 0.25$, as smaller magnitudes give faster settling times), z_3 will be placed outside the unit cycle in the *z*-plane (e.g. $z_3 = 1.6$) and the system will be unstable. So to make the system stable, we need to choose z_1 and z_2 with a relatively large magnitude and sacrifice the settling time a little bit. We also may want it relatively underdamped in order to have a fast rising time, assuming we can tolerate

[1]Note these equations describe the dynamic relationship between the state and control variables. For the control relationship between these variables, please refer to the control block diagram in Figure 5 in Section 3.2.

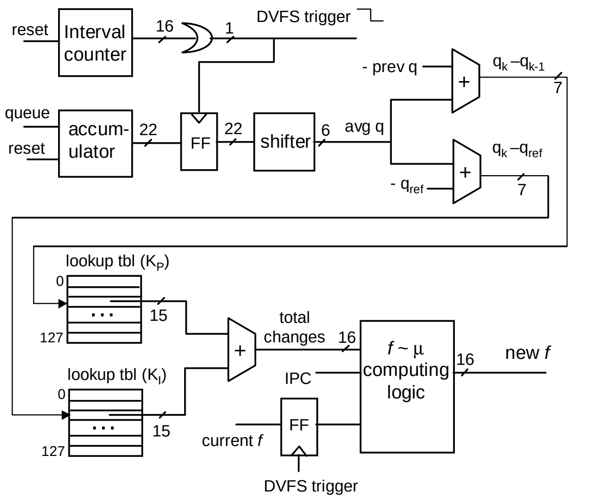

Figure 8: Block diagram for a possible hardware implementation of the DVFS controller in a clock domain.

the overshoot caused by the underdamping. Considering all of the above, one good placement is to have $z_{1,2} = 0.5 \pm 0.5i$, $z_3 = 0.6$, and the corresponding $K'_I = 0.2$, $K'_P = 0.6$. Figure 7 shows the step response of the system in (7) for this setting.

Note, in the above controller design, we assume frequency f_k can vary without bounds. In practice, f_k has an upper bound and a lower bound. Also, there is a maximum possible frequency change in one control interval for a given interval length and clock changing rate. So, in practice, the DVFS controller would need to check with these bounding values. In addition, we assume an XScale-style frequency control which allows continuous frequency changes. For processors which allow only stepwise frequency changes, the DVFS controller would need to choose a discrete step which is closest to the requested continuous value.

3.5 Hardware implementation of the DVFS controller

The most important hardware required to implement the proposed DVFS scheme are the queues. As these queues already exist as synchronization queues between clock domains, the online DVFS controller for an MCD processor uses the existing hardware in a “two for one” style.

The rest of the required hardware includes two counters, which are similar to the hardware counters in [23]. One is used to frame the control interval (a time counter or instruction counter). Since the length of a control interval is typically a few thousand cycles or instructions, a single 16-bit counter should be sufficient. The other one is a queue counter to get the cumulative queue occupancy. Given that a queue size is typically around 20 ($\ll 2^6$) and we have to total up such occupancies over $\leq 2^{16}$ cycles, a 22-bit queue counter should be sufficient. One possible implementation for computing the average queue occupancy is to choose the interval length as power of two and use a shifter as shown in Figure 8. The most complicated part is the logic required to compute the control signal (f). Since it is done only once for each control interval and is not time critical, this computation can be finished in multiple cycles. Specifically, this logic first needs to compute μ_k as

$$\mu_{k-1} + K_I(\bar{q'}_k - q_{ref}) + K_P(\bar{q'}_k - \bar{q'}_{k-1})$$

One possible implementation for this task is shown in Figure 8, which uses two adders to compute the queue differences, and two pre-computed lookup tables to get the actual control gains or changes

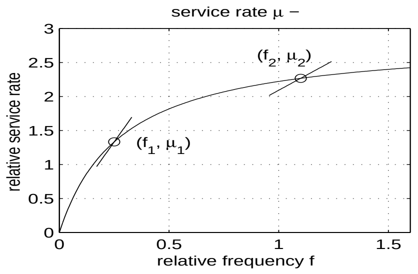

Figure 9: Piece-wise linear functions to approximate the real $\mu(f)$ function around two operating points.

(one for K_I and one for K_P). The actual size of the lookup tables depends on the queue size, the resolution of the queue values, and the control gains. For a queue size of 32, a resolution of 0.25 for queue values, and a 7-bit K_P or K_I resolution, each lookup table requires 128 entries and 15 bits per entry as illustrated in Figure 8.

This logic also needs to compute values of $\mu(f)$ with $\mu = 1/(\bar{t}_1 + \bar{C}_2/f)$. The $\bar{t}_1$ and $\bar{C}_2$ can be estimated online by some performance counters [28]. The rest can be computed by a multiplier or using again some pre-computed lookup tables. In order to reduce the hardware requirement for this part, we approximate the $\mu(f)$ function with some piece-wise linear functions. (An illustration is given in Figure 9 with $\bar{t}_1 = 0.35$, $\bar{C}_2 = 0.1$.) That is, we divide the frequency range into many small pieces, each of which is centered at an operating point. Then for f inside a small piece, μ can be computed using a linear function $\mu = IPC \cdot f$, where IPC is the effective IPC at the center operating point. For the example in Figure 9, we have $\mu = IPC_1 f$ for f inside a small piece around operating point (f_1, μ_1), and $\mu = IPC_2 f$ for the small piece around (f_2, μ_2). If the size of the frequency piece is small, this will give a quite good approximation. (In the next section we will see that because the frequency change rate is relatively slow in an MCD processor, the maximum possible frequency change in an interval is relatively small; thus the error from this piecewise approximation is relatively small.) With this approximation, the hardware required to compute $\mu \sim f$ will be reduced. We will need to estimate $IPC_{effective}$ online using a regular IPC counter, then compute the new frequency f using the current f, total control gains (changes), and IPC as shown in Figure 8. The computing logic for this part can be implemented using a pre-computed lookup table as discussed above, or using a 16-32 bit multiplier depending on the frequency resolution. As mentioned earlier, this computation is not time critical, so we can use some serial multiplication techniques to further reduce the hardware requirement. Finally, the new frequency signal f will be used by the frequency and voltage control mechanism in a clock domain.

Overall, only a modest amount of hardware support is needed to implement the online DVFS controller.

3.6 Specifying energy and performance tradeoffs with $\mathbf{q_{ref}}$

In the DVFS controller design in previous subsections, the reference queue q_{ref} specifies the nominal operating queue point. In principle, q_{ref} can be any value which is neither full nor empty (i.e $0 < q_{ref} < 1$, if expressed relative to the queue size). In this subsection, we show the position of q_{ref} specifies the actual tradeoff between performance degradation and energy efficiency.

We continue to consider the single queue model in Figure 3. Notice that, for this queue system, the steady state throughput (i.e. the steady state performance) degrades if and only if the queue is full. That is because, at that point, the performance demand from the upstream domain cannot be satisfied and the arriving process is forced to stall. Similarly, energy is wasted if and only if the queue is empty, because the domain is running idle and not doing any useful work. Note, control errors and input noise/variation are the two major causes for the queue to become full or empty.

In addition, we notice that the distance from the nominal operating point (q_{ref}) to the full-end queue point, $[q_{ref}, 1]$, reflects the relative margin for the queue to tolerate the control errors and input variation before the queue becomes full and loses performance. Similarly, the distance from q_{ref} to the empty-end queue point, $[0, q_{ref}]$, reflects the margin to tolerate errors before the queue becomes empty and wastes energy.

Therefore, if q_{ref} is increasing, then the distance $[q_{ref}, 1]$ is decreasing and the system is more likely to suffer performance degradation. On the other hand, for an increasing q_{ref}, the distance $[0, q_{ref}]$ is also increasing and the system is less likely to waste energy. Qualitatively, the bigger the q_{ref}, the more performance degradation and the more energy savings in general. We can choose how much energy performance tradeoff we want by choosing an appropriate q_{ref} in our online DVFS controller.

As mentioned in section 3.2, in our design, we can also leave the policy parameters like q_{ref} adjustable by the OS or application software. So the OS/application can direct the overall power management with a simple lever (making the overall decision on how aggressively to save energy or preserve performance) while leaving the actual implementation details of speed adaptation in hardware. This mechanism shows an example of hardware/software cooperation in DVFS control.

In the rest of this subsection, we will further give a quantitative theoretic estimation of the performance degradation and energy savings as a function of q_{ref} (later, in Section 4, we will show the actual experimental results). Denote the control error as ε_c, the input variation as ε_v, and the queue length as L. In concept, we have the following

$$\begin{aligned} \text{Performance} &= \text{PerfectP} - \text{Degradation} \\ &= \text{PerfectP} - D(q_{ref}, L, \varepsilon_c, \varepsilon_v) \\ \text{Energy Savings} &= E_{\text{slack}} - \text{Wasted} + E_{\text{non-slack}} \\ &= E_{\text{slack}} - W(q_{ref}, L, \varepsilon_c, \varepsilon_v) + E_{\text{non-slack}} \end{aligned} \tag{10}$$

where PerfectP is the perfect performance of Figure 2, D is the performance degradation, E_{slack} and $E_{\text{non-slack}}$ are the *slack energy savings* and the *non-slack energy savings* defined in Section 3.1.

From (10), we see the value of Energy Savings is mostly decided by the E_{slack} in the original program and the value of *non-slack energy savings*. The energy wasted W is affected directly by q_{ref} as we mentioned. So in general, as q_{ref} increases, W decreases, (also *non-slack energy savings* increases), and the overall Energy Savings increases.

The relative performance, on the other hand, is more closely related to q_{ref} as shown next. To get a theoretical estimation of the relative performance, we assume the program variation pattern is more or less exponentially distributed, so the demand and server process can be approximated by a Markov model [3]. The queue system considered in this section can be modeled as an M/M/1/L queue [3]. Also, assuming the controller error ε_c is 0, we have $\mu_k = \lambda_k$ and the server utilization $\rho = 1$ [3]. For a given q_{ref} and queue size L, the probability that the queue becomes full is

$$P_{loss} = \frac{1}{(1 - q_{ref})L + 1} \tag{11}$$

Using equations (10) and (11) , we have the performance as

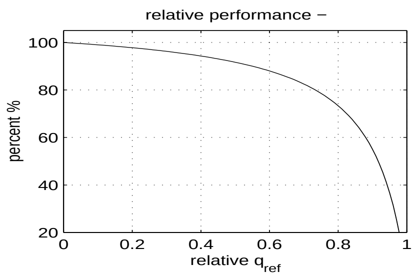

Figure 10: An analytic estimation of the relative performance of the single queue-domain system as a function of the reference queue q_{ref} using a M/M/1/L queue model.

$$\begin{aligned}\text{Performance} &= \text{PerfectP} - P_{loss}\cdot\text{PerfectP} \\ &= \left(1-\tfrac{1}{(1-q_{ref})L+1}\right)\text{PerfectP}\end{aligned} \tag{12}$$

If we use the performance at $q_{ref} = 0$ as the base performance, then we can compute the relative performance as a function of q_{ref} from (12) as

$$\text{Relative Performance} = \frac{(1-q_{ref})(L+1)}{(1-q_{ref})L+1} \tag{13}$$

Figure 10 shows an example curve from equation (13) with $L = 10$. It is observed that the relative performance degrades as q_{ref} increases, and the slope increases dramatically.[2]

The intention of the above analysis is to get some analytical insight on how the relative performance degrades with q_{ref}. In Section 4, we will give the actual experimental results on performance and energy saving as a function of q_{ref}.

4. EXPERIMENTAL RESULTS

In this section, we present experimental results to illustrate and evaluate the effectiveness of the proposed online DVFS scheme.

4.1 Simulation methodology and setup

Our simulation environment is based on the SimpleScalar toolset [5] with the Wattch [4] power estimation extension and the MCD processor extension [24]. The MCD extension by Semeraro *et al.* in [24] has 4 clock domains as shown in Figure 1. It also includes a cycle-by-cycle computation of the synchronization overhead due to independent clock frequency, phase, and clock jitter. An XScale-like dynamic voltage and frequency changing mechanism has been implemented, which allows any frequency to be used within the allowable range, as described in Section 2.

We have made two major modifications to the simulation core, in order to have a more accurate energy and performance estimation. First, the load-store queue in the MCD simulator has been split into a separate load-store issue queue and a load-store retirement buffer. Second, the energy computation in the latest MCD simulator [18] uses a formula inherited from Wattch, which computes the energy as the sum of power on a cycle-by-cycle basis. While this formula works fine for a processor with a fixed frequency, it may give overly-optimistic energy results for a processor with dynamically varying frequency. So we modified the energy computation formula to account for the varying frequency in an MCD processor with DVFS.

[2]Note, in reality, the applications' variation patterns are typically not strict-exponentially distributed. So the actual performance-q_{ref} curve might be slightly different from that in Figure 10. Further, different applications may have different execution variation patterns, which may lead to different performance-q_{ref} curves.

Table 1: Summary of All Simulation Parameters

Simulation Parameters	Value
Reference queue point	6 INT, 5 FP, 3 LS
Domain frequency range	250MHz – 1.0GHz
Domain voltage range	0.65V - 1.20V
Frequency/voltage change speed	73.3 ns/MHz, 171 ns/2.86mV
Control interval length	10000 instructions
Domain clock jitter	±110ps, normally distributed
Inter-domain synchro window	300ps
Branch predictor:	
2-level	L1 1024, hist 10, L2 1024
Bimodal	size 1024, BTB 4096 sets, 2-way
Combined	size 4096
Decode/Issue/Retire width	4/6/11
L1 data cache	64KB, 2-way
L1 instr cache	64KB, 2-way
L2 unified cache	1MB, direct mapped
Cache access time	2 cycle L1, 12 cycles L2
Memory access latency	80 first chunk, 2 inters
Integer ALUs	4 + 1 mult/div unit
Floating-point ALUs	2 + 1 mult/div/sqrt unit
Issue queue size	20 INT, 16 FP, 16 LS
Reorder buffer size	80
LS retire buffer size	64
Physical Register file size	72 INT, 72 FP

We implemented the online DVFS controller for local queues and domains, following the design in Section 3. Also, as in [18, 23], we made the front end domain run at a fixed maximum speed, and allowed the INT, FP, and LS domains to be controlled by the DVFS controller. We assume a performance degradation target of about 4%, which is roughly the same as that in [23]. Using the curve in Figure 10 as a general guide, we chose the reference queue point q_{ref} as roughly $\frac{1}{3}$ of the total size for INT and FP domains. (That is, q_{ref} is 6 for the INT domain, 5 for FP). Since the LS domain is relatively more critical to the overall performance as shown in [20], we chose its q_{ref} to be 3 which is roughly $\frac{1}{5}$ of the total size. For the μ and f relations in (4), we used the piece-wise linear approximation $\mu = IPC_{effective}f$ as discussed in Section 3.5. Also, we assume clock gating will be applied whenever the unit is not used. All other architecture parameters are chosen to have the same values as those in [18, 23]. A summary of all simulation parameters is in Table 1.

We want to evaluate our online DVFS scheme with a broad set of applications. To show variety, we will present results for 6 MediaBench applications, 8 SPECint applications, and 4 SPECfp applications as shown in Figure 12. We chose roughly the same subset of SPECint and SPECfp as those used in [18, 23]. We believe these programs form a representative set as they display a range of program behavior. For MediaBench benchmarks, we use the official data input set in the MediaBench web site and the whole program as the simulation window; for SPEC2000 benchmarks, we use the reference input set and choose the simulation window using the published Early SimPoint numbers [22].

As an illustrative example, in Figure 11 we show the frequency setting from the online DVFS controller for the benchmark *EPIC-Decode*. The corresponding average queue occupancy trace is also shown there. Clearly, there is a strong correlation between the

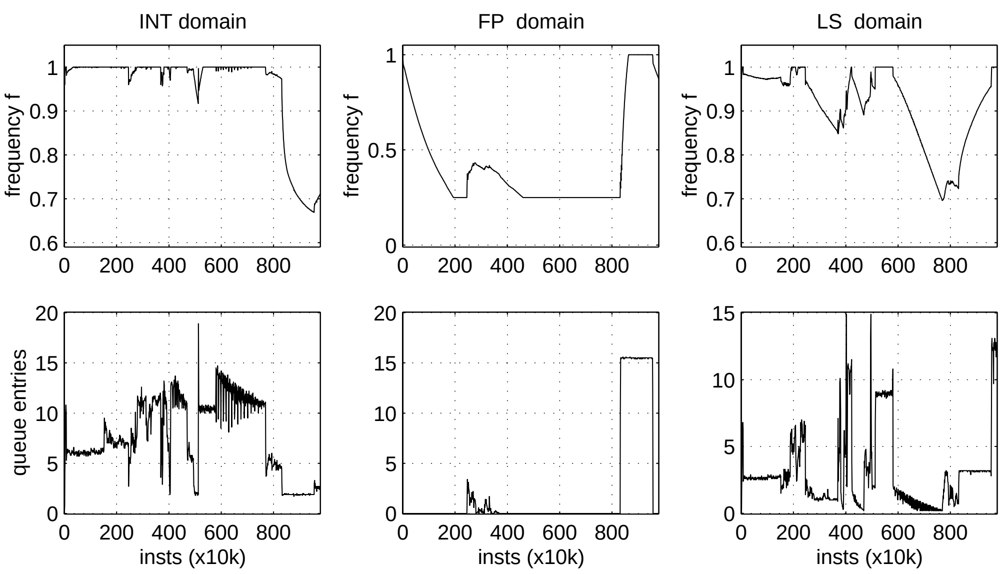

Figure 11: Frequency trace from DVFS for the benchmark EPIC-Decode in the top row, and the corresponding queue trace in the bottom row.

queue traces and the obtained DVFS frequency settings. This correlation is most obvious for the FP domain where the FP issue queue is empty except for two distinct phases. At the beginning, the DVFS controller detected the queue emptiness and gradually reduced the FP frequency to $f_{min} = 0.25GHz$. Then there was a modest frequency recovery in the first phase. During the second phase, the DVFS controller detected a dramatic increase of queue entries (i.e. increasing demand to the FP clock domain) and quickly adjusted the clock frequency to $f_{max} = 1.0GHz$. The correlations for the INT and LS domains are similar, but are less obvious as the queue traces become relatively more complicated.

Next, we will look at the energy/performance efficiency of the proposed online DVFS controller, and compare results with those from some best-known prior work.

4.2 Energy and performance results for different approaches

To compare our results to other prior DVFS approaches, we hold performance roughly the same for all approaches and look at metrics such as energy savings, energy-delay product improvement, and power/performance ratio. We define the power/performance ratio as percentage power saved per percentage performance degradation – that is, what percentage power is saved for one percent performance degradation. Note this definition is the same as that in [23].

We will present results from our analytic online DVFS scheme (denoted as *Analytic*). We compare them to those from the heuristic-based online DVFS in [23] (denoted as *Heuristic*), and those from the semi-oracle-based DVFS in [24] (denoted as *SemiOracle*). The *SemiOracle* assumes the DVFS, by oracle, has full knowledge of existing slack in a program, and uses a *Shaker* algorithm to decide DVFS settings. So its results are not realistic, but serve as a comparison. (However, as stated in [24], the *SemiOracle* result is not the upper bound for all possible DVFS results.) The parameters we use for *Heuristic* and *SemiOracle* are taken from [23] and [24], with a 5% performance degradation target for *Heuristic* and 1% for *SemiOracle* because these target values will lead to an actual performance degradation similar to *Analytic*. We also want to compare our results to those from the conventional (fully) synchronous voltage scaling (denoted as *Synchro*) which scales the frequency/voltage for the whole processor. It is set to get roughly the same performance degradation as other approaches.[3]

Table 2: Average results for different schemes, relative to a conventional (fully) synchronous processor.

Schemes	Performance degradation	Energy savings	Energy-Delay product improvement	Power/ Performance ratio
Analytic	3.6%	19.6%	16.7%	6.2
SemiOracle	3.7%	18.1%	15.0%	5.6
Heuristic	5.8%	11.9%	6.8%	3.0
Synchro	4.7%	7.6%	3.3%	2.5

The performance degradation, energy savings, and energy-delay product for each benchmark are shown in Figure 12. The results are relative to the conventional (fully) synchronous processor without voltage scaling. So all results for MCD include about 1.5% percent inherited performance and energy overhead from the baseline MCD processor, as discussed in [23]. The average results over all 18 benchmarks are summarized in Table 2.

From Table 2 and Figure 12, there are several interesting observations. First, the overall results from *Analytic* DVFS are very promising. We achieve a power/performance ratio of 6.2 on average relative to a fully synchronous processor. Second, compared to results from *Heuristic* DVFS, *Analytic* DVFS achieves far better results in terms of Energy-Delay Product (EDP) improvement and power/performance ratio. For example, the average EDP improvement by *Analytic* is 146% higher than that of *Heuristic* (16.7% vs 6.8%, with numbers relative to a synchronous processor). *Analytic* DVFS also produces an average result better than those of *SemiOracle*. For example, the average EDP improvement by *Analytic* is about 11% higher than that by *SemiOracle* (16.7% vs 15.0%)[4]

[3] We were not able to implement synchronous dynamic voltage scaling, so the *Synchro* results are for static scaling which may under-represent the benefit of synchronous voltage scaling.

[4] The *SemiOracle* results from our experiments are close to the pub-

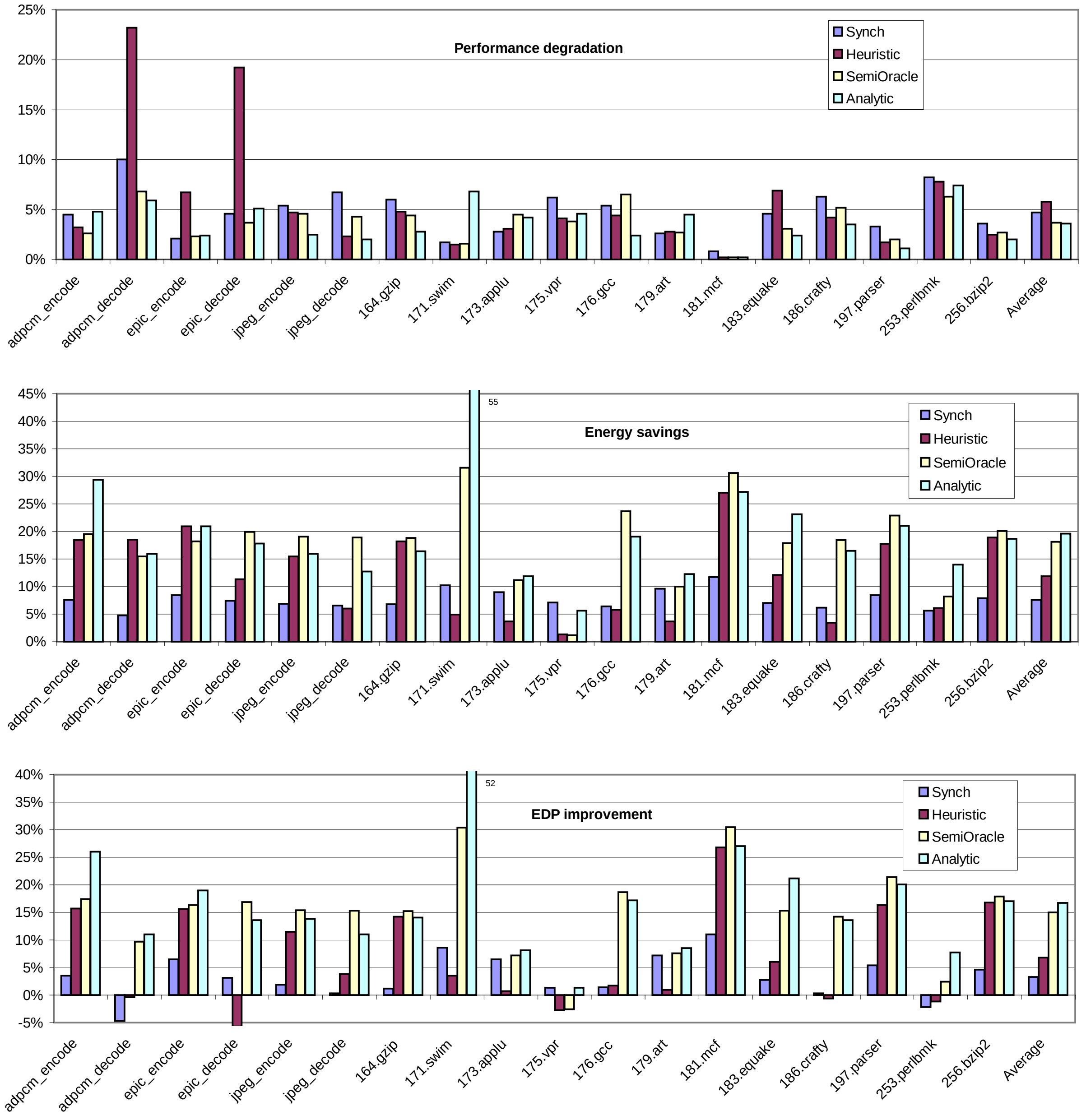

Figure 12: Performance degradation, Energy savings, and Energy-delay product improvement for each benchmark; different schemes are *Synchronous* voltage scaling, *Heuristic*-based online DVFS, *SemiOracle* DVFS, and *Analytic* online DVFS.

These results show the effectiveness of the proposed analytic online DVFS scheme due to the automatic regulation ability of a DVFS controller.

Lastly, all MCD DVFS results (both analytic and heuristic-based) are much better than those by the synchronous voltage scaling, which shows the energy savings potential of a MCD processor due to the extra flexibility in DVFS control. Note that the *Synchro* numbers from our experiments are lower than what is usually expected from the fact that performance scales linearly with f while energy scales quadratically with v. The main reason is the voltage range in our experiments is only half that of the frequency range. As stated in [23], this reflects the current trend of shrinking voltage ranges in processor designs as the supply voltage continue to scale aggressively relative to the threshold voltage.

4.3 Energy-performance tradeoffs as a function of q_{ref}

In the last subsection, the reference queue operating point q_{ref} for all benchmarks were set to the values in Table 1—roughly $\frac{1}{3}$ of the total size. Also, in Subsection 3.6, we have shown analytically how the performance and energy vary as a function of q_{ref} in general. In this subsection, we show that experimentally.

lished results (relative to a synchronous processor) in [23, 24] . We noticed, however, that the *Heuristic* results are not close to the results reported in [23]. We carefully examined the differences and consulted the authors. We found, in addition to the differences due to the energy computation formula mentioned earlier, there are other possible reasons including different simulation windows, different implementations of the voltage changing mechanism in the latest distribution of the MCD simulation toolset [25], and different ways of computing average numbers for media benchmarks.

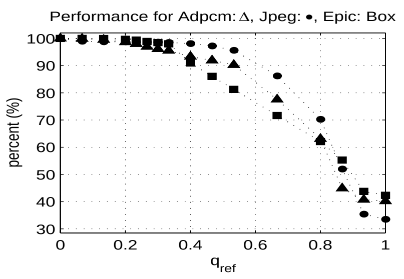

Figure 13: Relative performance degradation as a function of q_{ref} for individual benchmarks – adpcm-encode (△), epic-encode (□), and jpeg-encode (●).

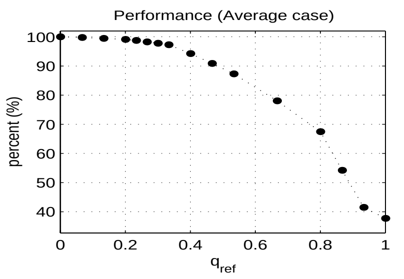

Figure 14: Average relative performance degradation over the 6 MediaBench benchmarks as a function of q_{ref}

We use the 6 MediaBench benchmarks in the last subsection for this study, as they are relatively small. For each benchmark, we apply the *Analytic* online DVFS control with a relative q_{ref} varying from 0.0 to 1.0 (the same q_{ref} value for all three INT, FP, and LS queues). We then compute the performance and energy savings relative to the values for the baseline MCD.

Figure 13 shows the relative performance as a function of q_{ref} for 3 of the MediaBench benchmarks. (Note, in our simulation, each domain has a frequency lower bound of 0.25GHz, so the lowest relative performance in Figure 13 is not 0.) From this figure we see that, due to different execution variation patterns in different applications, their performance curves have slightly different shapes and slopes. We also computed the average relative performance over all 6 MediaBench benchmarks, as shown in Figure 14, which has a shape relatively close to the analytic estimation in Figure 10.

Recall that from Section 3.6, the energy savings is also affected by q_{ref}. The general trend is that energy savings increase as q_{ref} increases. Figure 15 gives an illustration of this general trend using the average relative energy savings over the 6 MediaBench benchmarks.

Thus, we believe the results in this subsection and Section 3.6 shows both analytically and experimentally how the energy performance tradeoff is affected by the q_{ref} setting in our DVFS controller.

5. DISCUSSION OF FUTURE WORK

The online DVFS scheme studied in previous sections is decentralized. That is, it uses only local queue information and ignores interactions among multiple queues. The decentralized DVFS scheme can work fairly well in an MCD processor where frequency change in one clock domain has negligible or little impact on other domains and queues. For example, for the 4-domain MCD processor studied in Section 4 which has a fan-out structure, decentralized DVFS works quite well, as shown by the experimental results in Section 4.

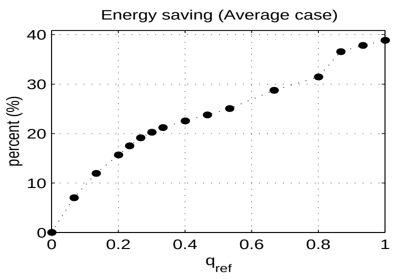

Figure 15: Average relative energy savings over the 6 MediaBench benchmarks as a function of q_{ref}

However, for an MCD processor with more elaborate domain partitions and strong interactions among multiple queues, a centralized online DVFS scheme may be needed in order to make correct and efficient scaling decisions. For example, in a system where three clock domains are in series with two queues in between, the status of first queue will not only depend on its neighboring domains, but also depend on the status of the downstream queues and domains. So, in this case, online DVFS needs to look at the status of all queues to make correct and efficient control actions. Otherwise, a clock domain can be confused by information from individual local DVFS controllers—one scenario is, when its input and output queues are both full, the local controller for its input queue will suggest a speed increase while the local controller for its output queue will suggest the opposite.

A systematic approach to design a centralized DVFS scheme needs to extend the DVFS framework in Section 3 using a global control theory. A new analytic model is first needed for all queues in an MCD processor which interact with each other. We can generalize the single queue-domain model used in previous sections into a queue-domain network. For each clock domain, its input queue can take flows (demand) from multiple sources (either a upstream domain or an external input source) through a *join* operation. Also, a clock domain can send out flows to multiple destinations (either a downstream domain or an external output sink) through a *split* operation. For this system, due to the interactions among multiple queues and domains, the actual execution speed μ and the arriving flow rate (demand) λ for a clock domain will be affected by the status of all related queues. For example, μ for a domain will be zero if its *downstream* queue is full. Similarly, λ for a domain will be zero if its *upstream* queue is empty. To analytically model these interactions, we need to use tools like an *indicate* function $\mathbf{1}()$ and introduce a *flow-matrix* A. The *indicate* function is defined as

$$\mathbf{1}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \qquad (14)$$

and the *flow-matrix* A defines how the tasks flow from one domain to another for a given queue-domain network.

With an analytical model for this system, control can be achieved by feeding back the state variables (the so-called state feedback with integral control [14]). Intuitively, this means frequency and voltage are controlled using all queue information as feedback signals. Note that, the design of an actual control law may need some linearization techniques in order to eliminate the nonlinearity due to presence of the *indicate* function.

The above extension to centralized DVFS control is one of the topics we are currently exploring. We have obtained a detailed analytic model and a practical global DVFS decision algorithm based on DVFS controllers. The details are beyond the scope of this paper.

Apart from the above extension, there are some other avenues for future exploration. In this paper, in an attempt to have a better comparison with the results from prior work, we have chosen many design options to be identical to those used earlier, such as using the average queue occupancy for a fixed interval length of $10k$ as the feedback measure. We are currently investigating how these design options affect the DVFS control and if we can choose better design options or even make them adaptive. We are also considering an analytic framework for an *event-driven* DVFS scheme, which is different from the *interval-based* DVFS approach in this paper (a simple example for an *event-driven* DVFS is in [11]).

6. CONCLUSIONS

In this paper, we have presented an analytic online scheme for dynamic voltage and frequency scaling (DVFS) in a multiple clock domain (MCD) processor. There are two salient features about the proposed DVFS scheme. First, it is online and dynamic workload driven. Second, it takes a rigorous analytic approach and is guided by control theory.

Compared to the best-known prior online DVFS, which is heuristic-based, our online DVFS scheme has achieved a 2-3 fold increase in efficiency. In addition, our control theoretic technique is more resilient and complete. For example, we can guarantee stability and achieve significant energy savings even under extreme cases. Furthermore, our scheme has the advantage of requiring less tuning effort and having better scalability.

Specifically, we model the MCD processor as a queue-domain network and the corresponding online DVFS as a feedback control problem. We have described the modeling, analysis, design, and implementation of the proposed DVFS controller. Our experimental results show that we have achieved an Energy-Delay product improvement which is 146% higher than that from the best-known heuristic-based online DVFS, and 11% higher than that from an semi-oracle-based DVFS scheme.

We believe the techniques and methodology described in this paper can be generalized for effective energy control in processors other than MCD. For example, there are many tiled CMPs in research now [8], and the ideas here would translate neatly to DVFS for them as well. Furthermore, the formal control techniques described here for DVFS can also serve as examples for applying control theory to other aspects of dynamic execution in high-performance CPUs.

Acknowledgments

We would like to thank David Albonesi and his research group at University of Rochester for their MCD simulator, which we used as a base to develop our simulation infrastructure. We also thank Greg Semeraro, Yongkang Zhu, and Diana Marculescu for their helpful discussions during the development of this work; and Youfeng Wu and the anonymous reviewers for their useful comments and suggestions on the paper. This work is supported by NSF ITR grant CCR-0086031. In addition, Martonosi's work is supported in part by Intel, IBM, and SRC.

7. REFERENCES

[1] Carl Anderson. Tuning and optimization of a 170m transistor microprocessor. In *Proceedings of the IEEE/ACM International Workshop on Timing Issue in the Specification and Synthesis of Digital System (TAU2000)*, Dec 2000.

[2] K.J. Astrom and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 1995.

[3] S. K. Bose. *An Introduction to Queueing Systems*. Kluwer Academic, 2002.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimization. In *Proc. of the ISCA-27*, June 2000.

[5] D. Burger and T. M. Austin. The SimpleScalar tool set version 2.0. Technical Report 97-1342, Department of Computer Science, University of Wisconsin-Madison, June 1997.

[6] T. Chelcea and S. M. Nowick. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In *Proc. of DAC-2001*, pages 21–26, 2001.

[7] L.T. Clark. Circuit design of XScale microprocessors. In *Proceedings of the 2001 Symposium on VLSI Circuits*, June 2001.

[8] M. Taylor et al. The RAW processor - a scalable 32-bit fabric for embedded and general purpose computing. In *Proceedings of Hot Chips XIII*, August 2001.

[9] R.V. Hogg and A.T. Craig. *Introduction to Mathematical Statistics, Fifth edition*. Prentice Hall, 1995.

[10] C-H Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proc. of PLDI-2003*, pages 38–48, June 2003.

[11] A. Iyer and D. Marculescu. Power efficiency of multiple clock multiple voltage cores. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2002.

[12] A. Iyer and D. Marculescu. Power-performance evaluation of globally asynchronous, locally synchronous processors. In *Proc. of the 26th ISCA*, May 2002.

[13] K.Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *Proceedings of DATE*, Feb 2004.

[14] B.C. Kuo. *Automatic Control Systems. , 7th edition*. Prentice Hall, 1995.

[15] D.V. Lindley. The theory of queues with a single server. In *Proceedings of the Cambridge Philosophical Society*, pages 277–289, 1952.

[16] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithm with PACE. In *Proceedings of the SIGMETRICS-2001*, pages 50–61, June 2001.

[17] Z. Lu, J.Hein, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling. In *Proc. of the Intl. Conference on Compiler, Architecture, and Synchesis for Embedded Systems (CASES)*, October 2002.

[18] G. Magklis, M.L. Scott, G. Semeraro, D.H. Albonesi, and S.Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *Proc. of the 30th ISCA*, June 2003.

[19] D. Marculescu. On the use of microarchitecture-driven dynamic voltage scaling. In *In Workshop on Complexity Effective Design, Vancouver, Canada, June 2000.*, June 2000.

[20] D. Marculescu, D.H. Albonesi, A. Buyuktosunoglu, and P. Bose. Partially asynchronous microprocessors (PAMs). In *ISCA 2003 Tutorial*, June 2003.

[21] D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, pages 37–39, Sep 1997.

[22] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *Proc. of the PACT-2003*, September 2003.

[23] G. Semeraro, D.H. Albonesi, S.G. Dropsho, G. Magklis, S. Dwarkadas, and M.L. Scott. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *Proc. of the 35th Micro*, pages 356–367, November 2002.

[24] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proc. of the 8th HPCA*, pages 29–40, February 2002.

[25] G. Semeraro, G. Magklis, and Y. Zhu. Personal communications. December 2003.

[26] A.E. Sjogren and C.J. Myers. Interfacing synchronous and asynchronous modules within a high-speeed pipeline. In *Proceedings of the 17th International Conference on Advanced Research in VLSI*, pages 47–61, Sept 1997.

[27] K. Skadron, T. Abdelzaher, and M. Stan. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *Proc. of the 8th HPCA*, February 2002.

[28] Fen Xie, Margaret Martonosi, and Sharad Malik. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proc. of 2003 PLDI*, June 2003.

[29] K.Y. Yun and A. E. Dooply. Pausible clocking based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–487, December 1999.