# Power Efficiency of Voltage Scaling in Multiple Clock, Multiple Voltage Cores[*]

**Anoop Iyer**[†] **Diana Marculescu**
**Department of Electrical and Computer Engineering**
**Carnegie Mellon University, Pittsburgh, PA 15213**
**Email: {aiyer,dianam}@ece.cmu.edu**

## Abstract

*Due to increasing clock speeds, increasing design sizes and shrinking technologies, it is becoming more and more challenging to distribute a single global clock throughout a chip. In this paper we study the effect of using a Globally Asynchronous Locally Synchronous (GALS) organization for a superscalar, out-of-order processor, both in terms of power and performance. To this end, we propose a novel modeling and simulation environment for multiple clock cores with static or dynamically variable voltages for each synchronous block. Using this design exploration environment we were able to assess the power/performance tradeoffs available for Multiple Clock, Single Voltage (MCSV), as well as Multiple Clock, Dynamic Voltage (MCDV) cores. Our results show that MCSV processors are 10% more power efficient when compared to single-clock single voltage designs with a performance penalty of about 10%. By exploiting the flexibility of independent dynamic voltage scaling the various clock domains, the power efficiency of GALS designs can be improved by 12% on average, and up to 20% more in select cases. The power efficiency of MCDV cores becomes comparable with the one of Single Clock, Dynamic Voltage (SCDV) cores, while being up to 8% better in some cases. Our results show that MCDV cores consume 22% less power at an average 12% performance loss.*

## 1 Introduction

Power consumption has become a critical issue in processor design, not only in embedded or portable environments, but also for high-end systems where performance has been the chief design constraint. Due to increased levels of complexity and integration, power density increases dramatically with shrinking device sizes and costly packaging and cooling techniques have to be employed to preserve the reliability and correct operation of core processors. Although performance is still the main selling factor in the high-end processor market, solutions that drastically reduce power requirements while still preserving the performance constraints are desirable. A large component of the power budget in high-end processors is the clock power consumption. Most conventional microprocessor designs are synchronous in their construction; that is, they have a global clock signal which provides a common timing reference for the operation of all the circuitry on the chip. On the other hand, fully asynchronous designs built using self-timed circuits do not have any global timing reference; examples of this design style are given in Sutherland's work on *Micropipelines* [1]. **G**lobally **A**synchronous **L**ocally **S**ynchronous systems (which we refer to as GALS systems in this paper) are an intermediate style of design between these two. GALS systems contain several independent synchronous blocks which operate with their own local clocks and communicate asynchronously with each other. The main feature of these systems is the absence of a global timing reference and the use of several distinct local clocks (or clock domains), possibly running at different frequencies.

In the microprocessor industry, global clock distribution issues are perhaps the best motivating factor for the study of GALS systems. With each technology shrink, the clock distribution network of a large chip grows rapidly in complexity and requires large design effort, power and die area. For instance in the Alpha 21264 processor, the global and major clock grids alone consume a third of its total power budget. Ronen *et al.* mention the worsening interconnect and clock distribution problem [2] and advocate global asynchrony as a possible solution. Since products in this arena have to have the highest performance at the lowest possible power cost, early design exploration tools for assessing the impact of asynchrony on both performance and power are needed.

In this paper, we describe the development of a *modeling* and *simulation* framework for GALS high end processors. Using this framework, we address the following issues:

- If we design a microprocessor in a GALS style with multiple clock domains (also called Multiple Clock, Single Voltage, or MCSV core), how much performance overhead will it incur over a fully synchronous machine?
- Will the elimination of the global clock network help in reducing the power requirements of the processor, as it has been claimed before?
- How can we exploit the extra flexibility offered by independent clock domains in a GALS processor and what is the impact of using a Multiple Clock Dynamic Voltage (MCDV) scheme on power/performance metrics?

Our analysis shows that GALS (MCSV) processors are not necessarily better in terms of *power consumption* than fully synchronous designs as it has been claimed before. Moreover, there is also a decrease in performance due to the asynchronous communication overhead and this can have adverse effects on overall energy consumption. The energy savings obtained by the elimination of the global clock are offset by the additional power consumed due to longer execution times. The use of dynamic voltage scaling in MCDV cores improves the situation for some applications by providing up to 20% additional power savings.

### 1.1 Related Work

Sutherland's paper on *Micropipelines* [1] contains the classic introduction to asynchronous design. The Amulet processor core developed at Manchester, which implements the ARM instruction set, is in its third generation and is commercially viable [3]. As Ronen *et al.* mention in

[*] This work was supported in part by IBM Corp. SUR Grant No. 4901B10170 and by SRC Grant No. 2001-HJ-898.

[†] The author is now with AMD, Austin, TX 78741 (e-mail: anoop.iyer@amd.com).

0-7803-7607-2/02/$17.00 ©2002 IEEE

[2], the worsening interconnect and clock distribution problem, as well as the burden of high clock power cost can be potentially solved by employing a global asynchronous design as a possible solution. GALS systems have been studied in detail by Chapiro in his 1984 thesis [4]. His work covers metastability issues in GALS systems and outlines a stretchable clocking strategy which provides a mechanism for asynchronous communication. Hemani *et al.* estimated in [5] the clock power savings in GALS designs compared to synchronous designs. However, their work targets a regular ASIC design flow with simpler clocking strategies rather than the aggressive clock distribution networks used in microprocessors.

Of great importance in GALS designs is the choice of the asynchronous communication mechanism between various synchronous blocks. Chelcea and Nowick propose in [6] the use of FIFOs as a low-latency asynchronous communication mechanism between synchronous blocks. Muttersbach *et al*. have implemented asynchronous wrappers around synchronous blocks [7]; they have used these wrappers along with asynchronous memory blocks to implement an ASIC and have thus proved the feasibility of GALS design in silicon. A similar system has been proposed by Moore *et al.* in [8]; pausible clocking for GALS systems has been described by Yun and Dooply in [9].

Dynamic voltage scaling (DVS) has been widely studied and implemented commercially. The use of multiple power grids and their implications in area overhead have been discussed in [10]. The difference in pipeline stage latencies has been explored for lowering the power requirements at system level, but for a specialized communication Myrinet GAM pipeline in a dynamic voltage scaling environment [11]. In [12], an example of adaptive dynamic voltage scaling in a system with self-timed circuits is presented. In that case, the traffic information in the communication buffers is used to adapt the supply voltage of various components. Finally, [13] presents a study of GALS processor systems and is the closest to our work. There, the authors use a communication scheme based on asynchronous FIFOs and stoppable clocks. The voltage assignment mechanism is an oracle-based scheme which finds the optimal voltages to be applied to different clock domains in a GALS processor.

The problem of power modeling at high levels of abstraction (system, software and microarchitectural level) has recently started to gain a lot of attention. Wattch [14] presents an advanced microarchitectural power simulator based on parameterized power models which are proven to be within 10% accurate when compared to three different high-end microprocessors. Wattch is based on SimpleScalar [15] which is a detailed, cycle-accurate simulator for superscalar, out-of-order cores. However, none of these simulators assume multiple clock domains or fine grain power management using dynamic adaptation of voltages and clock speeds.

### 1.2 Organization and Scope

The rest of this paper is organized as follows. Section 2 presents the motivation behind our proposed GALS architecture. In section 3, we describe some practical issues related to defining synchronous blocks, the proposed architecture and our scheme for dynamic clock management. The simulation and modeling environment is presented in section 4 while section 5 summarizes our experimental results. We conclude in section 6 with some final remarks.

## 2 Motivation

The idea of GALS system design is in itself quite old [4]. Interest in GALS design is now growing, mainly due to the global clock distribution problem. Trends of increasing die sizes and rising transistor counts may soon lead to a situation in which distributing a high-frequency global clock signal with low skew throughout a large die is prohibitively expensive in terms of design effort, die area, and power dissipation. GALS systems eliminate the need for careful design and fine-tuning of a global clock distribution network. While asynchronous systems eliminate the clock altogether, the industry is not yet ready to switch to a completely asynchronous design style mostly because design tools in this arena are not as mature as those for synchronous design.

### 2.1 Clock Power

In cutting-edge processor designs, the clock distribution network is one of the most power-hungry parts. For instance, the Alpha 21264 had a clock distribution network which consumed 24 watts out of the processor's total power budget of 72 watts. High power is not only an issue for mobile devices, but is also an issue for other chips; for instance in the first generation Alpha, the single-line driver of the clock grid led to thermal management problems since the continuous switching at the driver led to a very high temperature at the chip's center and hence a high temperature gradient [16]. Power has thus become a first-class constraint in clock system design.

### 2.2 Clock Skew

Restle *et al.* have argued in [17] that clock skew arises mainly due to process variations in the tree of buffers driving the clock. Since device geometries will continue to shrink and clock frequencies and die sizes will continue to increase, global clock skew induced by such process variations can only get worse. Hence we argue that we will reach a point where clock skew will eat up a significant proportion of the cycle time and thus, will directly affect performance. For instance, the first release of Intel's Itanium core had a clock skew that was 9% of the cycle time using traditional clock distribution techniques; using a network of active deskewing elements [18], the clock skew was reduced to 2% of the cycle time. While techniques like active deskewing help to push the envelope for clocked systems further, they come at additional cost in terms of die area and power dissipation. At some point, pushing the limits of global clock distribution networks will lead to diminishing marginal returns. At that stage, GALS design techniques will come in useful.

### 2.3 Multiple-Clock Dynamic-Voltage Designs

Most of applications running on core processors (be it high-end, embedded or application specific) exhibit a wide range of run-time profiles, both within and across applications. This is mainly manifested via non-uniform resource usage, as well as bursty communication patterns among various parts of the pipeline. One such example is the fetch stage during which the I-cache is accessed and instructions are brought into the fetch queue. While an I-cache miss is being resolved, the issue and execution stages of the pipeline may proceed at their own pace until no more instructions advance in the pipeline due to pending dependencies. A similar situation may appear in case of non-blocking D-cache misses. In this case, multiple outstanding D-cache misses are resolved while instructions may proceed normally through the pipeline. In addition, if there are instructions non-critical to the overall performance (e.g., infrequent floating point operations in integer applications), their execution may proceed at a lower speed without significantly affecting performance. While multiple clock domains offer this flexibility, they also come with additional potential for power savings. Such synchronous blocks whose speed may be gracefully scaled down, can also run at a lower voltage, thus providing additional power savings.

Since usage profiles and communication patterns among pipeline stages vary within and across applications, such MCDV cores should be able to dynamically adapt the speed and corresponding voltage for each synchronous block such that overall performance is kept within certain limits. As it will be shown later, a MCDV core can be better than a fully synchronous core as far as their power-performance trade-off is concerned given certain specific conditions.

### 2.4 Theoretical Efficiency of Voltage Scaling in MCDV Cores

In this section, we provide some theoretical results on the efficiency of using fine-grained dynamic voltage scaling in multiple-clock dynamic voltage cores. We assume that the following hold:

- We consider the case of pipelined cores in which each clock domain $i$ has $n_i$ pipe stages, with $L_i$ total load on the critical path of the clock domain.
- The switched capacitance for each clock domain is $C_i$. The voltage and clock speed associated with clock domain $i$ are $V_{dd,i}$ and $f_i$, with

$$f_i \propto \frac{(V_{dd,i} - V_t)^\alpha}{V_{dd,i}}$$

where $\alpha$ is a technology-dependent factor, which is 1.2 to 1.6 for current technologies [19] and $V_t$ is the threshold voltage.

**Lemma 1** *Assuming a linear pipeline organization (without feedback paths), the SCDV pipeline achieves better energy savings than the MCDV under the same slowdown factor per computation, if the switched capacitance per clock domain $C_i$ is proportional to the total load on the critical path $L_i$.*

**Proof** We assume that the SCDV base pipeline is run at a voltage $V_{dd}$ such that the latency per computation is the same as the MCDV pipeline with $k$ clock domains, each running at voltage $V_{dd,i}$ $(i = 1, 2, \ldots, k)$. We also assume that between clock domains $(i, i+1)$ there exists a load overhead of $l_i$ due to synchronization and voltage level conversion. To achieve the same latency, the following has to hold:

$$(L_1 + l_1)\frac{V_{dd,1}}{(V_{dd,1} - V_t)^\alpha} + (L_2 + l_2)\frac{V_{dd,2}}{(V_{dd,2} - V_t)^\alpha} + \ldots + L_k \frac{V_{dd,k}}{(V_{dd,k} - V_t)^\alpha} =$$

$$(L_1 + \ldots + L_k)\frac{V_{dd}}{(V_{dd} - V_t)^\alpha}$$

or

$$\sum_{i=1}^{k} \frac{L_i V_{dd,i}}{(v_{dd,i} - V_t)^\alpha} < \left(\sum_{i=1}^{k} L_i\right) \frac{V_{dd}}{(V_{dd} - V_t)^\alpha} \qquad (1)$$

Consider the function

$$f(x) = \frac{\sqrt{x}}{(\sqrt{x} - V_t)^\alpha}$$

which is convex. Thus,

$$\sum_{i=1}^{k} L_i f(V_{dd,i}^2) \geq \left(\sum_{i=1}^{k} L_i\right) f\left(\frac{\sum_{i=1}^{k} L_i V_{dd,i}^2}{\sum_{i=1}^{k} L_i}\right) \qquad (2)$$

Function $f$ is also monotonically decreasing; thus from (1) and (2) and from the proportionality of $C_j$ with $L_j$ for all clock domains $j = 1, 2, \ldots, k$, that is,

$$\frac{C_j}{\sum_{i=1}^{k} C_i} = \frac{L_j}{\sum_{i=1}^{k} L_i}$$

we get

$$\sum_{i=1}^{k} C_i V_{dd,i}^2 \geq \left(\sum_{i=1}^{k} C_i\right) V_{dd}^2$$

which shows that the SCDV pipeline performs better energy-wise than the MCDV for the same slowdown factor (or performance penalty) under the assumptions of Lemma 1.

This result is in fact a generalization of the optimal voltage scheduling problem for applications with hard real-time constraints. Ishihara and Yasuura showed in [20] that using a single voltage level in a dynamic voltage scheduling environment achieves the best energy savings under given performance constraints.

The general problem is, however, far from being that simple. In fact, in most designs, there is no relationship between the load on the critical path and the corresponding switched capacitance for that clock domain. We show in the following a necessary condition for fine-grained dynamic voltage assignment to different clock domains for achieving better energy savings for MCDV when compared to SCDV.

**Lemma 2** *In case of a 2-clock domain pipeline, if the switched capacitance per clock domain $C_i$ is **not** proportional to the load on the critical path $L_i$, MCDV cannot achieve better energy savings than SCDV for the same slowdown factor, unless the lower voltage is applied to the stage which satisfies the relation*

$$\frac{C_1}{C_1 + C_2} > \frac{L_1}{L_1 + L_2}$$

**Proof** As in Lemma 1, we have $L_1 V_{dd,1}^2 + L_2 V_{dd,2}^2 \geq (L_1 + L_2) V_{dd}^2$ if (1) is satisfied for $k = 2$. For the MCDV to be better in terms of energy than the SCDV, we need

$$\frac{L_1 V_{dd,1}^2 + L_2 V_{dd,2}^2}{L_1 + L_2} \geq V_{dd}^2 \geq \frac{C_1 V_{dd,1}^2 + C_2 V_{dd,2}^2}{C_1 + C_2}$$

From the above, we get

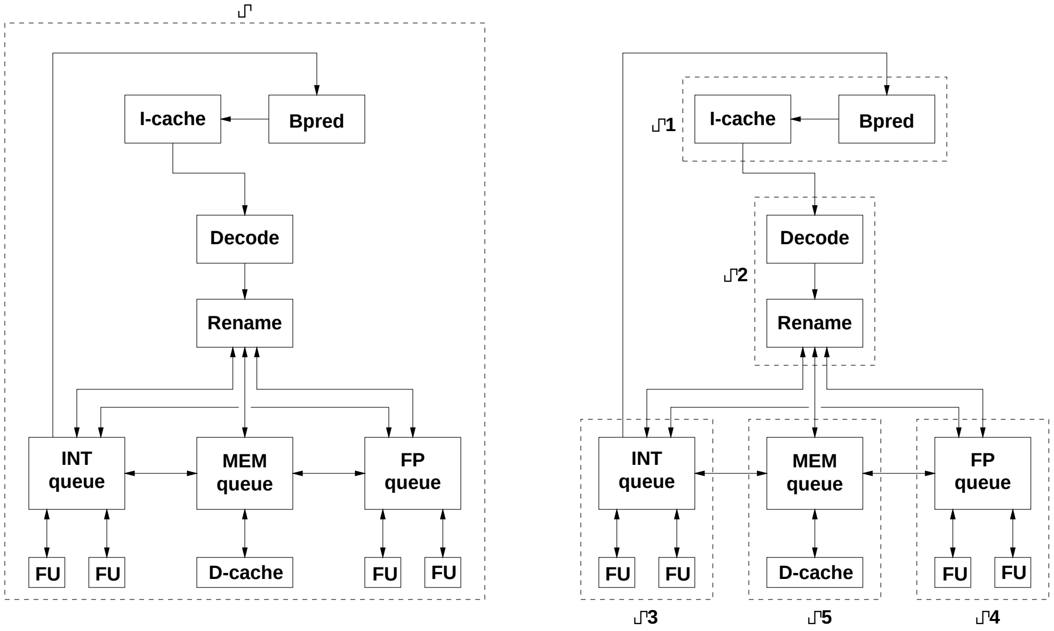
$$V_{dd,1} \leq V_{dd,2}$$

This result confirms our intuition that in order to achieve energy savings in MCDV versus the SCDV design for the same slowdown factor, lower voltages should be applied to the more power consuming clock domains if they are also contributing the least to the end-to-end latency per computation. We will consider the results provided by these two lemmas when we define the control strategy based on the application profile.

We point out that the above results hold only for linear pipelines, without feedback paths. While this is not the case even for the simplest processor cores (which have to bypass results among different execution stages) it can be considered as such since usually some of the feedback paths are not on the critical path, and thus, they don't affect overall performance. In addition, if multiple computations are executed in parallel (as in the case of high-end processors), computations off the critical path can be executed at a lower voltage, without affecting the overall performance, for a given slowdown factor. We present in the following sections an adaptive approach for GALS processors which is able to tune the operating voltage of different clock domains, according to the usage of that clock domain.

## 3 GALS Architecture Design

We discuss below some issues involved in GALS design, specifically focusing on superscalar out-of-order microprocessors, examples of which are the Alpha and Pentium families of processors. These designs extract a high level of instruction-level parallelism from the code using out-of-order and speculative execution and have the capability of issuing and executing several instructions at the same time. Our focus is on architecture level issues. GALS design involves many other complicated issues at the circuit and system levels which we do not attempt to address, for instance:



Synchronous (base) architecture

(a)

GALS architecture

(b)

**Figure 1. Base vs. GALS machine organization**

- **Metastability resolution:** The problem of metastable signals and techniques for metastability resolution using synchronizers and arbiters has been the focus of research in asynchronous design. Our approach uses asynchronous FIFOs [6] between clock domains and this in turn relies on synchronizers. For comparison, we have also included models for FIFOs using arbiters for clock pausing whenever metastable conditions occur [9, 13].
- **Local clock generation:** Each clock domain in a GALS system needs its own local clock generator; ring oscillators have been proposed as a viable clock generation scheme [4, 7]. We assume that we can use ring oscillators in each synchronous block in the GALS processor.
- **Failure modeling:** A system with multiple clock domains is prone to synchronization failures; we do not attempt to model these since their probabilities are miniscule (but non-zero) and our work does not target mission-critical systems.

## 3.1 Defining Synchronous Blocks

Hemani *et al.* have described an automated strategy for defining locally synchronous blocks in a GALS design [5]. Starting from a hierarchical RTL description of the system, their method uses iterative refinement to get an optimal partitioning of the system into a number of synchronous blocks, using clock power as an objective function for optimization. In a custom-designed system like a high-end microprocessor core, performance requirements justify manual intervention in the partitioning phase. Since the primary motivation behind GALS design is to avoid distributing a common clock signal over large areas, the strategy for partitioning the design into synchronous blocks will largely be dictated by physical design aspects. However, since asynchrony can lead to higher latencies, it is crucial to take architecture issues into account when partitioning the design.

In the traditional superscalar out-of-order processor model shown in Figure 1 (a), the *instruction flow* consists of fetching instructions from the instruction cache, using the branch predictor for successive fetch addresses. The *register dataflow* consists of issuing instructions out of the instruction window to the integer and FP units and forwarding results to dependent instructions. The *memory dataflow* consists of issuing loads to the data cache and forwarding data to dependent instructions. Introducing high latencies in any of these three crucial flows will have an impact on the processor's performance.

The level 1 instruction cache and the branch predictor taken together are a good candidate for one synchronous block corresponding to the front-end of the pipeline. In some architectures, notably in CISC architectures like Intel's IA-32, the decode logic has large area and consists of several pipe stages; in such cases, decode would be a good candidate for another synchronous block.

Inside the out-of-order execution core, it is difficult to make generalizations and say which parts of the core may be decoupled without much overhead and which may not; such decisions are very specific to the microarchitecture and the instruction set of the processor. Area and clock distribution considerations obviously suggest a partitioning to some extent. For instance in the 21264 Alpha the 'major clocks' (tapped from the global clock and distributed locally) are defined this way, based mostly on the top-level hierarchy of the design; these clocks suggest a partitioning system for that specific implementation. The 21264 has the following major clocks [21]: (1) instruction fetch and branch predict (2) bus interface unit (3) integer issue and execution units (4) floating point issue and execution units (5) load/store unit (6) pad ring. Guided by this, we have chosen synchronous blocks for our GALS design, which we describe below.

## 3.2 The Proposed GALS Architecture

To compare GALS with synchronous microarchitecture organization, we have chosen to have five clock domains in the GALS design. This decision was driven by the functional organization of our processor and by the physical design of similarly organized cores. Figure 1 shows the

| Stage | Operation | Domains |
|---|---|---|
| 1 | Fetch from I-cache | 1 |
| 2 | Decode | 2 |
| 3 | Register rename, Regfile read | 2 |
| 4 | Dispatch into issue queue | 2, 3/4/5 |
| 5 | Issue to functional unit | 3/4/5 |
| 6 | Execute | 3/4/5 |
| 7 | Wakeup, Writeback | 3/4/5 |
| 8 | Regfile write, Commit | 3/4/5, 2 |

**Table 1. Pipeline stages in our processor models**

pipeline structure we studied. In the base (synchronous) model, all the modules run off the same clock. In the GALS model, various regions are clocked using different clock signals independent of each other. The first stage of the pipeline consists of an instruction cache and branch prediction unit (clock domain 1). The next stages are instruction decode and register rename (clock domain 2). There are three issue queues in the design: one for integer instructions (clock domain 3), one for floating-point instructions (clock domain 4) and one for loads and stores (clock domain 5). In the GALS processor, the integer ALUs and the integer issue queue are in the same clocking region. This ensures that dependent instructions within the integer issue queue can be issued back-to-back as soon as operands are available. Similarly, floating-point ALUs and the floating-point issue queue share one clock, and the data-cache and memory issue queue share one clock.

Table 1 gives a summary of the pipeline stages in the processor models we developed for our experiments, along with a listing of the clock domains of the GALS machine which are involved in each pipe stage.

## 3.3 Asynchronous Communication Mechanisms

In the synchronous version, communication between successive logic blocks is done using regular pipeline registers. In a GALS design, the choice of an asynchronous communication mechanism is critical. Many methods have been proposed for clocking GALS systems with *stretchable clocks* [4, 7, 8]. Such clocking systems manage asynchronous communication between two clock domains by stretching one phase of both the clocks while the handshaking and data transfer takes place. This is typically done using an arbiter element inside the loop of a ring oscillator. While this mechanism provides an elegant and fail-safe method of communication, it also stalls both the synchronous blocks during the transaction.

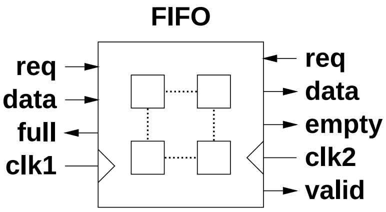


**Figure 2. Asynchronous FIFO for interfacing two clock domains**

Chelcea and Nowick have presented in [6] a design for a low-latency token-ring based FIFO which can be used for asynchronous communication between synchronous blocks. The interfaces to the FIFO are shown in Figure 2. Their design uses *full* and *empty* signals to indicate the occupancy of the FIFO. The *empty* signal is controlled by the producer of data into the FIFO and is synchronized to the consumer's clock; similarly, the *full* signal is controlled by the consumer and is synchronized into the producer's clock. A few modifications are made to the circuit to account for latencies in synchronization and to prevent deadlock. In addition to providing high throughput in the steady state, the design has low latency when compared to other methods we tried and tested. Since the focus of our work is at a higher level of abstraction, a complete description of the operation of the circuit can be found in [6]. We have included a cycle-accurate model of this FIFO in our simulation environment.

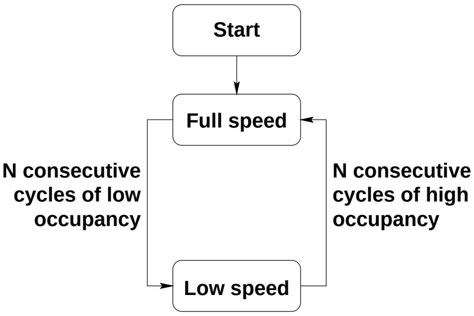

Start
Full speed
N consecutive cycles of low occupancy
N consecutive cycles of high occupancy
Low speed

**Figure 3. Outline of Dynamic Clock Management**

Semeraro *et al.* [13] have also used a FIFO-based communication mechanism for their multiple clock domain implementation. In their case, metastability is resolved by stopping clocks only when the rising clock edges of the producer and consumer clock domains are within a certain threshold. For comparison, we have also included this FIFO model in our simulation environment.

## 3.4 Dynamic Clock and Voltage Management

Since there are several independent clocks in a GALS design, we have the flexibility of designing various logic blocks to operate at different frequencies. In addition, if we have a balanced pipeline design to start with, we can slow down synchronous blocks that are off the critical path of the design while keeping the others running at full speed. The slower clock domains could also operate at a smaller supply voltage, thus producing additional power savings. The relation between supply voltage $V_{dd}$ and logic delay (or cycle time) $D$ is governed by the following equation:

$$D \propto \frac{V_{dd}}{(V_{dd} - V_t)^{\alpha}} \tag{3}$$

where $V_t$ is the threshold voltage of the transistor and $\alpha$ is 1.2 to 1.6 for the current technologies. Since energy consumption in CMOS is dependent on the *square* of the supply voltage, reducing supply voltage can lead to significant energy benefits.

We propose a multiple-clock dynamic-voltage core processor where the speed and voltage of each synchronous block are adapted dynamically based on the application profile. Our method for dynamic clock management involves changing the operating frequencies (and supply voltages) of the integer, memory and floating-point clock domains in the GALS processor (indicated as clock domains 3, 4 and 5 in Figure 1). We do not try to slow down the front-end stages since the front-end bandwidth is critical to most applications.

Figure 3 shows the outline of our method. The processor starts up with all clocks running at full speed. To minimize the performance impact of clock speed reduction, we monitor the occupancy of each issue queue and we reduce the frequency to a lower value only when the queue occupancy drops below a low threshold for $N$ consecutive cycles. Since the execution characteristics may be different even within different parts of the same application, we continue monitoring the issue queues for a number of *N* consecutive cycles. If the occupancy rises above a certain threshold for these *N* consecutive cycles, we switch the clock back to full speed. We found that 2K is a good value for *N*; using smaller values leads to many false alarms in lowering the clock frequency, while with larger values we lose out on potential power saving opportunities. This implementation is a simple hysteresis loop with only two clock speeds; we chose to keep it simple because more complex and finer-grained implementations need finer-grained voltage supply switching. Further, we assume that useful work can be done in the intermediate switching periods as shown in [22]. We also assume that level-conversion circuits are built into the drivers of the asynchronous FIFOs to interface clock domains running at different voltages.

# 4 GALS Simulation Framework

## 4.1 Event-driven Timing Simulation

We have used a general-purpose event-driven simulation engine which can be used to simulate any asynchronous system, synchronous (clocked) system, or a system which contains both asynchronous and synchronous components [23].

To set the system in motion, we need to insert one or more starting events into the event queue. The queue contains events sorted in increasing order of their scheduled times. To process the queue, we read successive events from the head of the queue and execute them by calling the appropriate action functions. To simulate systems with multiple clocks, we need to insert one event for each clock domain; for each such event, we need to specify a period of execution. When the execution engine processes such a periodic event, it schedules another instance of the same event into the queue, thus representing the next cycle of execution of the clocked system. Using this system, we can simulate any mixture of multiple clocks running at different speeds with specific or random starting phases.

## 4.2 Performance and Power Modeling

To evaluate the architecture proposed above, we wrote models of both the synchronous and GALS processors based on the SimpleScalar toolset [15]. To simulate the GALS machine, we made use of the event-driven simulation engine described above. We have set up five clock domains in our simulator and in the first set of experiments, had all the clocks running at the same speed. The starting phase of each clock was set to a random value at runtime.

We have used the Wattch framework [14] to add power models to our processor simulation framework, including power consumption of the asynchronous FIFOs. Wattch is an activity-driven power simulator which has been shown to be within 5 to 10% accurate when compared to real processors. Wattch provides switching capacitance modeling for structures like ALUs, caches, arrays and buses in a processor. These are integrated into our base and GALS simulators to provide energy statistics. We have also modeled unused units as consuming 10% of their total power consumption to account for leakage and the overheads associated with clock gating.

In addition to modeling the switching capacitance of memories and buses inside the processor, we have also modeled the switching capacitance of clock grids. For the synchronous base processor model, we assumed a clock distribution hierarchy resembling that of the 21264 Alpha processor. We modeled one global clock grid and five local clock grids corresponding to the five clock domains discussed above. The areas and metal densities of each clock grid were approximated by the numbers published for the 21264 processor. For the GALS processor, since there is no global clock, we eliminated the switching capacitance of the global clock grid and retained the five major clock grids, corresponding to the distribution networks for each of the synchronous blocks. In all cases, we have assumed that clock gating is used to reduce the power consumption of unused modules.

| | |
|---|---|
| Fetch and decode rate | 4 inst/cycle |
| Integer issue queue size | 20 |
| FP issue queue size | 16 |
| Memory issue queue size | 16 |
| Integer registers | 72 |
| FP registers | 72 |
| L1 data cache | 16KB 4-way<br>1 cycle latency |
| L1 instruction cache | 16KB direct-mapped<br>1 cycle latency |
| L2 unified cache | 256KB 4-way<br>6 cycles latency |
| FIFO Size | 4 entries, up to 4 inst/entry |
| Integer issue queue thresholds | 7/9 |
| FP issue queue thresholds | 3/5 |
| Memory issue queue thresholds | 5/8 |
| Threshold for clock pausing | 30% of the<br>largest clock period |
| Aggressive mode | FP 2.4 slowdown<br>Mem 1.2 slowdown |
| Moderate mode | Int 1.8 slowdown<br>FP 1.2 slowdown<br>Mem 1.2 slowdown |

**Table 2. Microarchitecture organization**

Table 2 shows some details of the microarchitecture. The low and high occupancy thresholds for the three queues were chosen close to half of their maximum size, and less in the case of FP queue where a higher threshold may trigger a larger performance loss. We have used two different modes of operation: an *aggressive* mode for benchmarks that have a low number of FP operations which don't affect much performance, but do affect energy (in this case Lemma 2 is applicable); and a *moderate* mode for applications with an almost equal mix of integer, FP and memory operations. The aggressive mode was used for all benchmarks, except *swim*, *fpppp* and *epic*.

# 5 Experimental Results

To assess the performance and power of our proposed GALS processor design, we have used both the base and GALS simulators on a subset of Spec95 and Mediabench benchmarks. To this end we have performed two sets of experiments:

1. Base versus GALS (MCSV) performance and power analysis with all synchronous blocks of the MCSV system running at the same clock frequency and supply voltage. Both the synchronizer (MCSV) and pausible clock (MCSV-P) cases have been considered.
2. Base versus a multiple-clock, dynamic-voltage GALS design (MCDV), implemented as discussed in section 3.4. Both the synchronizer (MCDV) and pausible clock (MCDV-P) cases have been considered.

## 5.1 Power and Performance Analysis

On the base and MCSV models, we ran benchmarks from the Spec95 and Mediabench CPU benchmark suites to obtain indications of power and performance trends. In the first set of experiments, we kept all the local clock frequencies in the MCSV model equal. The relative stagger between the clock signals was set at runtime to random values. Not surprisingly, the MCSV processor is slowed down by asynchronous communication and does not perform as well as the synchronous processor. Figure 4 shows the relative slowdown of various benchmarks running on the MCSV processor when compared to the synchronous processor. On an average, the benchmarks we ran on MCSV were slower by 10% for MCSV and by 9% for MCSV-P. As expected, the *fppp* benchmark had the lowest performance hit. This is due to the application's exceptionally small proportion of branch instructions; on an average only one in every 67 instructions is a branch in this benchmark, while most other applications have one branch for every five or six instructions. This indicates that the asynchronous FIFO models used in our design have good throughput in the steady state when there are no branch mispredictions. This also suggests that branch mispredictions will prove more expensive in the MCSV model due to its longer recovery pipeline.

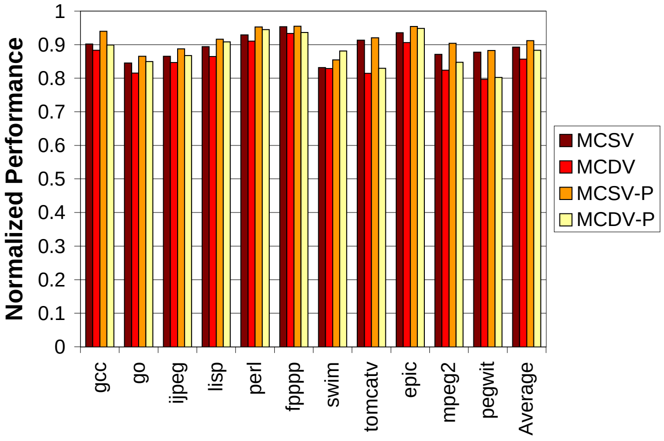


**Figure 4. Performance of the GALS model relative to the base model**

Figures 5, 6 and 7 show the relative average power and total energy consumption of the MCSV processor, normalized against the total energy and average power consumption of the synchronous processor respectively. With the removal of the global clock, we get in some cases up to 5% reduction in the total energy, whereas in some cases the power savings due to global clock grids is offset by the extra power consumption due to longer execution times. In fact, the increase in total energy in these cases (notwithstanding the reduction in average power) can be attributed to the extra time taken for program execution (e.g., *mpeg2* and *pegwit*). For the benchmarks we tested, though, the total energy is higher on average by 1% for MCSV and about the same for MCSV-P, while the average power consumption is reduced by 10% and 12%, respectively.

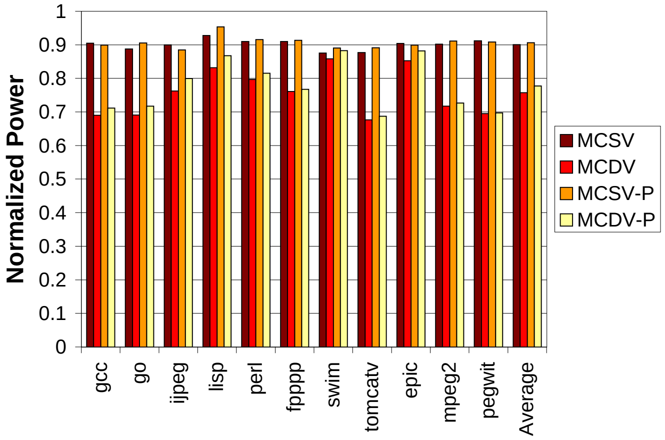


**Figure 5. Normalized power**

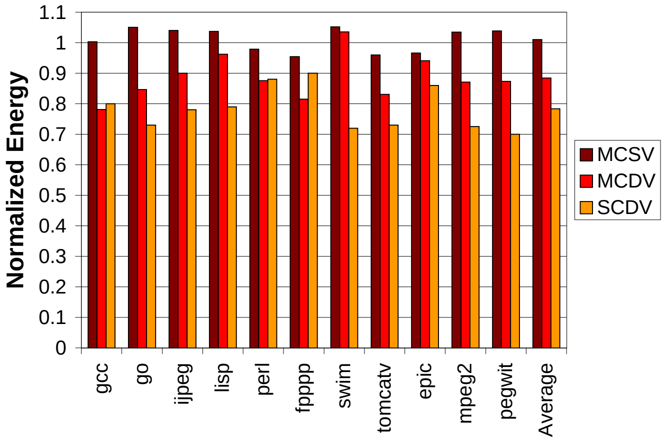


**Figure 6. Normalized energy (FIFOs with synchronizers)**

On close examination of other statistics in the processor pipeline, we can see that the introduction of asynchronous communication latencies inside the design has led to various other overheads which in some cases offset the power gains due to the absence of global clock. For instance, the the average time taken by each instruction from the fetch to the commit stage increases by 65% on average for all benchmarks in the GALS processor, out of which 25% is spent in the FIFOs and 40% is the overhead due to the longer recovery pipeline and due to slower result updates. This is because the addition of asynchronous communication channels leads to an increase in the effective length of the pipeline. This increase in pipeline length in the MCSV processor also leads to 17% higher speculative execution on the average across the benchmarks we tested. Similarly, the average number of in-flight instructions in the pipeline is higher in the GALS model; so is the average occupancy of the register allocation tables and issue queues. For instance the integer register allocation table occupancy went up from 15 in base to 23 in MCSV for the *ijpeg* benchmark. All these factors lead to higher switching activity in the MCSV machine and the power savings we get from eliminating the global clock grid may be offset by this extra switching. Hence, we conclude that eliminating the global clock and using several local clocks cannot by itself lead to dramatic power savings.

## 5.2 Analysis of MCDV cores

For this set of experiments, we have compared the base (fully synchronous) and the MCDV machine which is capable of dynamically changing the clock speeds and voltages for various synchronous blocks. In this case, we have implemented the scheme described in Section 3.4 with a value of $\alpha = 1.6$. We have considered a window size of 2K cycles for monitoring the size of the three queues. As it can be seen in Figures 4, 5, 6 and 7 (columns MCDV and MCDV-P), the average power is reduced by an additional 12% on average, with an average additional decrease in

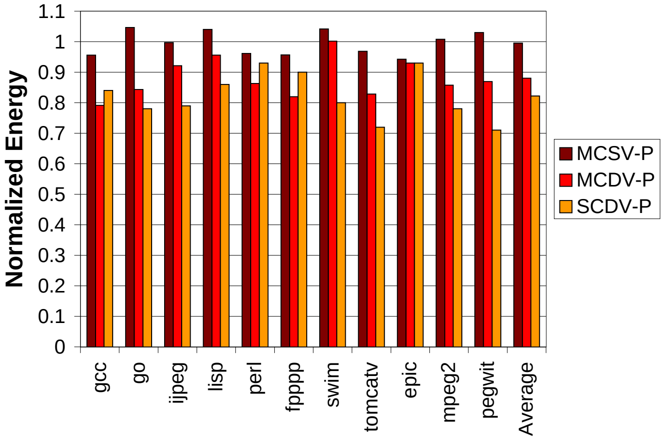


**Figure 7. Normalized energy (FIFOs with pausible clocks)**

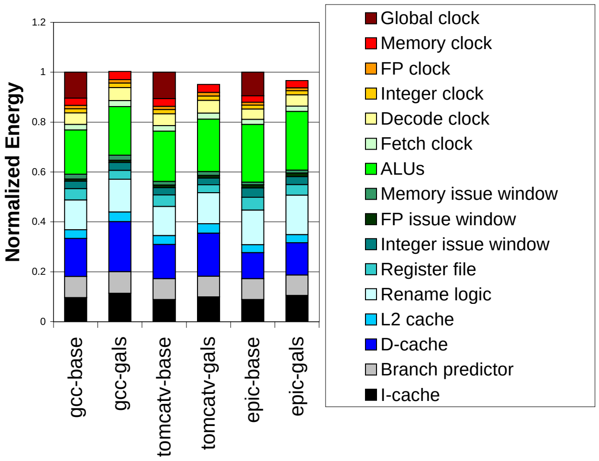


**Figure 8. Breakdown of total energy for base vs. GALS**

performance of only 2%. Total energy is also decreasing by an additional average of 12%.

To compare the efficiency of using dynamic clock and voltage management, we report under SCDV (Single Clock, Dynamic Voltage) in Figures 6 and 7 the normalized energy values obtained for the same performance penalty as the MCDV design if the fully synchronous core were run at a lower supply voltage. As it can be seen, the SCDV core is more energy-efficient than the MCDV core for 7 of the total of 11 benchmarks tested. In case of *gcc*, *perl*, *fpppp* and *epic* MCDV and MCDV-P are either comparable, or up to 8% better than their synchronous counterpart.Overall, an average power reduction of 22% is achieved at 12% loss in performance.

Finally, we show the breakdown of average power values for a few benchmarks considered in the base (fully synchronous) and GALS (MCSV) cores. As it can be seen in Figure 8, the total clock power is reduced due to the elimination of the global clock. However, due to increased execution time and higher speculation, the power consumed by the front-end increases slightly, as does the power consumption for the D-cache and execution core.

## 6 Conclusion

In this paper, we have used a power/performance modeling and analysis framework for high-end, superscalar out-of-order processors using multiple clocks, and possibly multiple, dynamically adjustable voltages to show a direct comparison of power and performance of a GALS (MCSV) design against a comparable synchronous design. The experimental evidence shows that the overhead associated with GALS processors can sometimes offset the power savings achieved in the clock power, although in a few cases, the average power cost is decreased by up to 12% and on average by 10% for the benchmarks considered. Nevertheless, the presence of independent clocks in the design permits a fine-grained trade-off between speed and power. We have shown that MCDV designs may be up to 8% more power efficient than their fully synchronous counterpart in select cases. Overall, by using fine grain clock speed and voltage adaptation, an average power savings of 22% (up to 32%) can be achieved with 12% average loss in performance.

## References

[1] I. E. Sutherland, “Micropipelines,” *Communications of the ACM*, June 1989.

[2] R. Ronen, A. Mendelson, K. Lai, L. Shih-Lien, F. Pollack, and J. P. Shen, “Coming Challenges in Architecture and Microarchitecture,” *Proceedings of the IEEE*, March 2001.

[3] S. B. Furber, D. A. Edwards, and J. D. Garside, “AMULET3: A 100 MIPS Asynchronous Embedded Processor,” in *Proc. Intl. Conference on Computer Design (ICCD)*, 2000.

[4] D. M. Chapiro, *Globally Asynchronous Locally Synchronous Systems*. PhD thesis, Stanford University, 1984.

[5] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, and D. Lundqvist, “Lower Power Consumption in Clock By Using Globally Asynchronous Locally Synchronous Design Style,” in *Proc. Design Automation Conference (DAC)*, 1999.

[6] T. Chelcea and S. M. Nowick, “A Low-Latency FIFO for Mixed-Clock Systems,” in *Proc. IEEE Computer Society Workshop on VLSI*, 2000.

[7] J. Muttersbach, T. Villiger, and W. Fitchner, “Practical Design of Globally Asynchronous Locally Synchronous Systems,” in *Proc. Intl. Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2000.

[8] S. W. Moore, G. S. Taylor, P. A. Cunningham, R. D. Mullins, and P. Robinson, “Self Calibrating Clocks for Globally Asynchronous Locally Synchronous Circuits,” in *Proc. Intl. Conference on Computer Design (ICCD)*, 2000.

[9] K. Y. Yun and A. E. Dooply, “Pausible Clocking-Based Heterogeneous Systems,” *IEEE Transactions on VLSI Systems*, December 1999.

[10] K. Usami and M. Horowitz, “Clustered Voltage Scaling Technique for Low-Power Design,” in *Proc. Workshop on Low Power Design*, 1995.

[11] G. Qu *et al.*, “Energy Minimization of System Pipelines Using Multiple Voltages,” in *IEEE Intl. Symp. on Circuits and Systems*, 1999.

[12] L. S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, “Low-Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage,” *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, December 1994.

[13] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarakadas, and M. L. Scott, “Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling,” in *Proc. Intl. Symp. on High Performance Computer Architecture (HPCA)*, 2002.

[14] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A Framework for Architectural-level Power Analysis and Optimizations,” in *Proc. Intl Symp on Computer Architecture (ISCA)*, 2000.

[15] T. Austin, E. Larson, and D. Ernst, “SimpleScalar: an Infrastructure for Computer System Modeling,” *IEEE Computer*, February 2002.

[16] P. E. Gronowski, W. J. Bowhill, and R. P. Preston, “High-Performance Microprocessor Design,” *IEEE Journal of Solid State Circuits*, May 1998.

[17] P. J. Restle *et al.*, “A Clock Distribution Network for Microprocessors,” *IEEE Journal of Solid State Circuits (JSSC)*, May 2001.

[18] S. Tam, S. Rusu, U. N. Desai, R. Kim, J. Zhang, and I. Young, “Clock Generation and Distribution for the First IA-64 Microprocessor,” *IEEE Journal of Solid State Circuits (JSSC)*, November 2000.

[19] K. Chen and C. Hu, “Performance and Vdd Scaling in Deep Submicrometer CMOS,” *IEEE Journal of Solid State Circuits (JSSC)*, October 1998.

[20] T. Ishihara and H. Yasuura, “Voltage Scheduling Problem for Dynamically Variable Voltage Processors,” in *Proc. Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 1998.

[21] D. W. Bailey and B. J. Benschneider, “Clocking Design and Analysis for a 600-MHz Alpha Microprocessor,” *IEEE Journal of Solid State Circuits (JSSC)*, Nov 1998.

[22] T. D. Burd and R. W. Brodersen, “Design Issues for Dynamic Voltage Scaling,” in *Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2000.

[23] A. Iyer and D. Marculescu, “Power and performance evaluation of globally asynchronous, locally synchronous processors,” in *Proc. Intl. Symp. on Computer Architecture (ISCA)*, June 2002.