Speed and Voltage Selection for GALS Systems Based on Voltage/Frequency Islands

Koushik Niyogi
Electrical and Computer Engineering
Carnegie Mellon University
Email: kniyogi@ece.cmu.edu

Diana Marculescu
Electrical and Computer Engineering
Carnegie Mellon University
Email: dianam@ece.cmu.edu

ABSTRACT

Due to increasing clock speeds and shrinking technologies, distributing a single global clock signal throughout a chip is becoming a difficult and challenging proposition. In this paper, we address the problem of energy optimal local speed and voltage selection in frequency/voltage island based systems under given performance constraints. Our results show that static voltage and speed assignment can achieve up to **42 % savings** in total energy for various media and signal processing applications, while application specific dynamic approaches provide up to **44 % energy savings** in the case of MPEG-2 encoder application, when compared to a single clocked system architecture.

1. INTRODUCTION

Achieving power efficiency has become an increasingly difficult challenge, especially in the presence of increasing die sizes, higher clock speeds and variability driven design issues. To cope with these challenges, a design style based on multiple Voltage/Frequency Islands (VFIs) has been proposed recently [1]. In addition to controlling better local clock skews and allowing for local performance optimizations, a multiple VFI design style may enable application-driven adaptation for better energy efficiency. As proposed before [1], "listening islands" may be designed for managing local speeds or voltages to match prescribed performance limits, while minimizing energy costs.

In support of a frequency island design style, a globally asynchronous, locally synchronous (GALS) approach may serve as an intermediate step between fully synchronous and fully asynchronous designs, while at the same time providing certain local adaptation capabilities. In GALS designs, the system is partitioned into synchronous blocks which are asynchronous with respect to each other. The speed and voltage of each block can be customized or chosen so as to meet the performance requirement of the logic, thereby making it an inherently energy efficient architecture.

The GALS approach is attractive because it eliminates the need for the careful design and fine tuning of a global clock distribution network, while still allowing proven synchronous design methodologies to be employed in the design of the individual modules. Designing SOCs using a voltage island based GALS paradigm also facilitates powering off an island completely, thus eliminating both leakage and active power, since each synchronous module uses a local clock which is not shared by other modules on the chip.

Contribution of this paper. This paper addresses the problem of *selecting voltage* and *speed levels* for each island in a VFI system, such that overall *dynamic* energy cost is minimized under given performance (throughput or latency) constraints. Two cases are considered: (i) *static* voltage and speed assignment; and (ii) *application adaptive, dynamic* voltage and speed assignment. Our problem formulation is applicable to general topologies with and without feedback paths and is solved as a constrained, non-linear optimization problem by using Lagrange minimization methods. We apply these methods to two real applications (software defined radio and MPEG-2 encoder) and demonstrate the energy savings which can be achieved by designing such systems as voltage island based GALS systems compared to a single clock system architecture.

The rest of the paper is structured as follows. Section 2 presents our assumptions related to the system architecture and the component graph hierarchy used to model a GALS system. The problem formulation for local voltage and speed selection is described in Section 3, while the static and dynamic cases are presented in detail in Sections 4 and 5. Section 6 shows our experimental framework, application description and results. Section 7 describes related work to our proposed speed and voltage selection methods, while Section 8 concludes the paper with some final remarks and directions for future research.

2. SYSTEM ARCHITECTURE

To this end, we consider a system comprised of a number of synchronous cores, IPs or processing elements (PEs)[1] (homogeneous or heterogeneous) that can be assigned to a single VFI (in other words, cores cannot belong to more than one VFI). A VFI can consist of a single PE or, depending on the physical or design considerations, may include a group of PEs. We assume that power is supplied to the voltage islands from an off or on chip source and can be controlled independently from any other island. This may be achieved using either on-chip voltage regulators or multiple power grids [2]. Each VFI is assumed to have a voltage level above a certain value V_{min} and, since our architecture is globally asynchronous, locally synchronous, each local module or core is assumed to be clocked using a ring oscillator controlled by the variable intra-island supply voltage using a digital phased lock loop [3][4].

Cores are assumed to communicate through mixed-clock interfaces or FIFOs [5] that enable the use of varying speeds between communicating PEs (Figure 1).

We also assume that the allocation and mapping of various processes or computational kernels of the application to PEs, as well as the number and types of the communication links and PEs have already been determined. In our system architecture, we assume that the processes have already been scheduled on their respective processing elements. Also, a bounded number of storage cells are available in the asynchronous FIFOs used between two communicating PEs.

3. PRELIMINARIES

We use a directed acyclic graph based model termed as *component graph* for the collection of communicating cores. In a component graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, synchronous cores are modelled as communicating processes that have associated communication

[1]Cores, IPs and PEs will be used interchangeably throughout the paper.

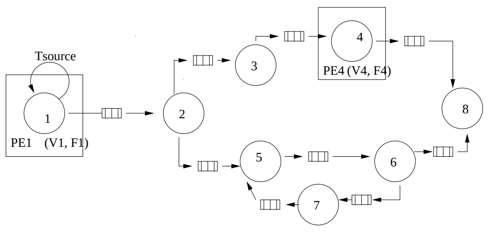

Figure 1: A VFI-based Component graph

channels between them. We consider the case of general on-chip topologies that may include directed acyclic graphs, but also graphs with feedback loops. In general, a component graph may consist of a set of Strongly Connected Components (SCCs) which essentially include feedback loops between different cores in the system. We focus mainly on periodic signal processing applications implemented as a system on a chip and having real time rate and deadline constraints.

3.1 Rate Derivation of a Component Node

Since a communication channel between two cores or PEs has a bounded capacity, it is required that the output rate of the producer and the input rate of the consumer should match. Thus assuming t_p is the number of tokens produced by the producer, t_c is the number of tokens consumed by the consumer, T_p and T_c are the time periods of the producer and consumer respectively, while the rate for each node (R_p, R_c) is the reciprocal of the period, then:

$$t_p/T_p = t_c/T_c$$

$$T_c = \frac{T_p * t_c}{t_p} \tag{1}$$

$$R_c = 1/T_c \tag{2}$$

For the sake of simplicity, we shall consider $t_p = 1$ and $t_c = 1$ henceforth in this paper. Each source process has a known input rate given by $R = 1/T_{input}$. The rate of an output node as well as all intermediate nodes can thus be derived from the rate of the source nodes. For example in Figure 1, if node 1 has a time period of 4 ms, by our above formulation, node 2 has the same time period. Since nodes 3 and 5 each consume one token from node 2, they too must have a period of 4ms.

3.2 Rate Derivation of SCCs

We denote by SCC in a component graph a strongly connected component in which every vertex has a path to all other vertices. A group of nodes which form a strongly connected component in the component graph would be represented as a single supernode. If we assume that each node in a SCC starts executing independently of each other, the time period of every process in a SCC can be shown to be the same [6] and equal to the maximum mean cycle ratio of the SCC, i.e.,

$$T_i = \max_{C \in SCC_i} \left(\frac{d(C)}{|C|}\right) \tag{3}$$

where C is a cycle in the SCC, $d(C)$ is the sum of the path delays of the cycle, $|C|$ is the number of the edges in cycle C, and T_i is the average time period of the nodes in the SCC. Let us consider the example from Figure 1. Processes 5, 6 and 7 form a SCC. Let the execution time of each node be 2ms. Since each node starts executing independently at time 0, nodes 5, 6 and 7 execute at timestamps $\{0,2,4,\cdots\}$. Here $d(C)$=2+2+2 =6ms and $|C|$=3. Thus the time period of the supernode =6/3=2ms.

Thus, given the execution cycles of each node in the SCC, one can use the above methodology to calculate the average rate of execution of each node. The SCC can then be represented as a single supernode with this execution rate.

3.3 Energy Consumption and Delay

The energy consumption per sample for every processing element in the component graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is given by:

$$E_i(V_i) = C_i * N_i * V_i^2 + c_i * n_i * V_i * exp(-V_t/k) \tag{4}$$

where the first term corresponds to dynamic power and the second term corresponds to static (leakage) power consumed while core PE_i is not actively executing a process. In more detail, C_i is proportional to the switched capacitance of PE_i, N_i is the number of active execution cycles for PE_i, c_i is proportional to the number of off-devices in PE_i, n_i is the number of idle cycles for processing a sample, k is a technology dependent constant, while V_i and V_t are the voltage supply and threshold voltage for PE_i, respectively [7]. [2]

Since we use a globally asynchronous locally synchronous architecture, a fraction of the total chip energy will be spent in communication between the PEs. In this paper we do not deal with the optimization of the communication energy consumption and treat it as a constant lumped in each PE's processing latency.[3]

The cycle time for the PE_i core in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ can be written as:

$$\tau_i(V_i) = K_i * \frac{V_i}{(V_i - V_t)^\alpha} \tag{5}$$

where K_i and α are technology dependent parameters [8]. Thus from (5), we get the worst case execution time of a process on PE_i at voltage V_i as (W_i is the worst case number of cycles for the process mapped on PE_i):

$$WCET_i(V_i) = W_i * \tau_i(V_i) = (W_i * K_i * V_i)/(V_i - V_t)^\alpha \tag{6}$$

4. STATIC VOLTAGE ASSIGNMENT

The static voltage assignment problem for a given component graph can be stated as:

Given a component graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ comprised of a set of processes mapped on a set of processing elements (PEs), find the optimal voltage and clock frequency to be assigned statically to each PE such that the energy per operation is minimized and rate and/or latency constraints are satisfied.

More formally, this can be stated as a non-linear optimization problem. Let $\mathcal{P} =< P_1, P_2, ..., P_m >$ be the set of simple forward paths from the source S of the component graph to the output or sink node T. For simplicity, we assume that the component graph has a single source node and a single sink node. However, the formulation can be easily extended to multiple source multiple sink process graphs, whereby we will have corresponding response time and rate constraints for each source and sink pair.

The performance (rate and latency) constrained energy minimization problem can thus be formulated as:

$$Minimize \sum_{i \in \mathcal{V} \ in \ \mathcal{G}(\mathcal{V},\mathcal{E})} E_i(V_i) \tag{7}$$

where $E_i(V_i)$ is the energy consumed per sample in node i of the component graph, such that the following rate and latency constraints, respectively, are satisfied:

$$WCET_i(V_i) \le T_i, \ \forall i \in \mathcal{V} \ in \ \mathcal{G}(\mathcal{V}, \mathcal{E}) \tag{8}$$

[2]We assume that all cores have been implemented in the same technology and have a fixed threshold voltage. Techniques that involve both supply voltage and threshold voltage selection are not considered in this paper.

[3]We assume that the interconnect is kept at the highest voltage available on chip and is not subject to voltage scaling.

and

$$\sum_{i \in P_j} WCET_i(V_i) \leq D,\ \forall P_j \in \mathcal{P} \tag{9}$$

In equation (8), T_i is the time period constraint of each node, while in (9), V_i is the voltage of the i^{th} node and D is the deadline of the system. Nodes which do not lie on a forward path from the source to the sink node do not affect the latency of the system.

Let us start by defining *throughput constrained* system and *latency constrained real time systems.* A *throughput constrained* system is dominated by its rate constraint as given in equation (8). Thus, as long as each node in the component graph satisfies its rate constraint, the system will always satisfy its end to end response time or latency. A *latency constrained* system, is dominated by its response time constraint as given by equation (9). In a simple *throughput constrained* system, we are given the input processing rate and/or individual rate constraints for each node in the component graph. We obtain a minimum energy configuration by setting the voltage of each node in the component graph such that it satisfies equation (8). This ensures that there is no slack for each processing element. For a simple *latency constrained* system, since the total energy of the system is minimized by "just" meeting the deadline constraint D, we can use Lagrange Multipliers to obtain a minimum energy configuration as shown in the subsection below.

4.1 Latency Constrained Systems

Let $E = \sum_{i \in \mathcal{V}\ in\ \mathcal{G}(\mathcal{V},\mathcal{E})} E_i(V_i)$, which is the total energy consumed per sample in a system of n cores each having an individual voltage V_i, be the objective function. Each path $P_j (j = 1, \cdots, m)$ in the set of simple forward paths in the graph is associated with a constraint equation as formalized in (9). Let there be m forward paths in the component graph $\mathcal{G}$. Let V_i be the voltage of node i in component graph $\mathcal{G}$ and N_j be the number of nodes in the j^{th} path of the graph. Then path constraints $C_j (j = 1, \cdots, m)$ are functions of the individual voltages of the nodes in the path. Thus:

$$C_j = (\textstyle\sum_{i \in P_j} \frac{W_i.K_i.V_i}{(V_i - V_t)^{\alpha}}) - D = 0,\ j = 1, \cdots, m \tag{10}$$

Thus, for E to be minimized, we need:

$$\partial E / \partial V_i = \sum_{j=1}^{m} \lambda_j (\partial C_j / \partial V_i),\ i = 1, \cdots, n \tag{11}$$

where the co-efficients $\lambda_1, \cdots, \lambda_m$ are called Lagrange multipliers. The above system of equations has $(n{+}m)$ variables and $(n{+}m)$ equations and can be solved using MATLAB or iterative techniques. Let us consider the example of Figure 1. We have two forward paths {1,2,3,4,8} and {1,2,5,6,8}. Node 7 does not affect the processing latency and so is not a part of the equations. Thus the constraint equations are given by :

$C_1 = WCET(V_1) + WCET(V_2) + WCET(V_3) + WCET(V_4) + WCET(V_8) - D = 0$

$C_2 = WCET(V_1) + WCET(V_2) + WCET(V_5) + WCET(V_6) + WCET(V_8) - D = 0$

The Lagrange equations are given by

$\partial E / \partial V_1 = \lambda_1 . \partial C_1 / \partial V_1 + \lambda_2 . \partial C_2 / \partial V_1$

$\vdots$

$\partial E / \partial V_8 = \lambda_1 . \partial C_1 / \partial V_8 + \lambda_2 . \partial C_2 / \partial V_8$

Substituting the values of E (equations (4) and (7)) and C_1 and C_2, we have (7+2) equations containing (7+2) variables which can be solved to get values for $V_1, \ldots, V_8$ (except V_7 as explained earlier) . Instead of a generalized component graph, if we consider only a linear pipeline with a single source-to-sink path, we get the following interesting result which we present as a lemma. We later use this result in simple pipelined applications in our experimental section.

Lemma: For the static voltage assignment for latency constrained systems, in a single source-to-sink path, to achieve minimum energy, the voltages assigned to the PEs are such that the ratios of the energy and delay of each processing node are equal.

Proof: Assume a linear pipeline of n nodes case where each node in the pipeline produces and consumes a single data item. From equation (9), we get the following inequality:

$$\sum_{i=1}^{n} WCET_i = \sum_{i=1}^{n} (W_i * K_i * V_i)/(V_i - V_t)^{\alpha}) \leq D \tag{12}$$

For latency constrained systems, we can show that the total energy is minimized when the PEs in the pipeline are designed such that the total worst case execution time of the entire pipeline equals the deadline D, i.e., $\sum_{i=1}^{n} WCET_i = D$. Applying the Lagrange minimization method, we get the following relation between the voltages of the PEs:

$$\frac{C_1.V_1.(V_1 - V_t)^{1+\alpha}}{K_1.(V_1(1-\alpha) - V_t)} = \frac{C_2.V_2.(V_2 - V_t)^{1+\alpha}}{K_2.(V_2(1-\alpha) - V_t)} = \cdots \frac{C_n.V_n.(V_n - V_t)^{1+\alpha}}{K_n.(V_n(1-\alpha) - V_t)} \tag{13}$$

If we consider $\tau_i(V_i)$ to be proportional to $V_i^{1-\alpha}$, i.e., $V_t << V_i$, we get the following relation:

$$\frac{C_1.V_1^{(1+\alpha)}}{K_1} = \frac{C_2.V_2^{(1+\alpha)}}{K_2} = \cdots = \frac{C_n.V_n^{(1+\alpha)}}{K_n} \tag{14}$$

or:

$$\frac{C_1.V_1^2}{K_1.V_1^{1-\alpha}} = \frac{C_2.V_2^2}{K_2 * V_2^{1-\alpha}} = \cdots = \frac{C_n.V_n^2}{K_n * V_n^{1-\alpha}} \tag{15}$$

This implies:

$$\frac{E_1}{D_1} = \frac{E_2}{D_2} = \cdots = \frac{E_n}{D_n} \tag{16}$$

where E_i, D_i are the energy and delay per sample for the i^{th} PE.

□

To get a simple closed form expression for each voltage value, substituting (14) in (12) we get the following assignment of voltages using $\eta_i = C_i/K_i$ and $\tau_i(V_i) \propto V_i^{(1-\alpha)}$:

$$V_i = \left(\frac{(W_1.K_1.(\eta_1/\eta_i)^{\frac{\alpha-1}{1+\alpha}} + \cdots + W_n.K_n.(\eta_n/\eta_i)^{\frac{\alpha-1}{1+\alpha}})}{D} \right)^{\frac{1}{\alpha-1}} \tag{17}$$

4.2 Static Voltage Assignment Algorithm

So far we have investigated the *throughput* and *latency* constraint cases separately. Most applications are characterized by both throughput and latency constraints. Using the above results and discussion, we present the following algorithm to statically assign an optimal voltage and frequency to the set of PEs so as to minimize the energy of the system, under both rate and latency constraints. **StaticVFAssign** is an algorithm to assign voltages to each core in our VFI islands in the presence of both throughput and deadline constraints. We design for the worst case execution time of each node in the component graph. We assume that the WCETs and switched capapcitances of the processes are known apriori through static profiling on the core. We first solve for the rate constraints as shown in steps 2 and 3. If the voltages obtained satisfy the deadline constraint, the static allocation algorithm stops. Otherwise, the system is a latency constrained system. Step 6 calculates the voltages based on the latency constraints only. If the voltages obtained for any of the nodes from this step violates the rate constraint, we ramp up its voltage (step 7). We fix the voltages of these nodes in equation (9). The paths in which, none of the nodes have their voltages ramped up, also have their node voltages fixed since they just meet the deadline D and thus minimize slack. To distribute the extra slack in the other paths, we incrementally decrease deadline D by an user defined amount δD, and

```
Algorithm StaticVFAssign
Inputs: Component Graph $\mathcal{G}$; Deadline $D$; Source rate $R$
        discrete voltage levels $V_1 \cdots V_s$; $W_i \ \forall \ i \ \in \mathcal{G}$
Outputs: Voltages $V_1, \cdots, V_n$ for cores $PE_1, \cdots, PE_n$
1. Represent each SCC in $\mathcal{G}$ as a supernode.
2. For source rate $R$, derive execution rates $R_i$ for each node $i$ in
$\mathcal{G}$. Rate of each node in SCC=derived rate of supernode
3. Solve for the rate constrained system using Eqs (7) and (8) to
get $(V_{1r}, \cdots, V_{nr})$.
4. Calculate the latency for the critical path ($T_{critical}$) in $\mathcal{G}$ using
voltages obtained from step 3.
5. If $T_{critical} > D$ goto step 6
  else
     Map $V_{ir}$ from step 3 to discrete levels $\{V_1, \cdots, V_s\}$
     return mapped voltages and exit
6. Solve for latency constrained system using Eqs (7) and (9) to
get $(V_{1l}, \cdots, V_{nl})$.
7. For all nodes $i \in \mathcal{G}$
     if $(V_{il} < V_{ir})$
          $V_{il} = V_{ir}$
          Fix the values of $V_i$s in Eq (9) with these $V_{il}$ values
     else do nothing
8. Repeat {
     $D_{new} = D - \delta D$ /* new deadline */
     Solve for latency constrained system using Eqs (7)
     and (9) and $D_{new}$ to get new $(V_{1l}, \cdots, V_{nl})$.
9. } until $\{V_{il} > V_{i,min}$ and $V_{il} < V_{i,max}\}$
10. Map $(V_{1l}, \cdots, V_{nl})$ to discrete levels $\{V_1, \cdots, V_s\}$ and return
```

Table 1: Algorithm for Static Voltage Assignment

solve for the voltages of the remaining nodes iteratively until we reach the highest possible deadline less than D which satisfies the equations. The final voltages are mapped to the set of discrete voltage levels $(V_1, \cdots, V_s, s < n)$ assuming a finite number of voltage levels are available. This algorithm ensures a solution even though there might not be an optimal solution satisfying the exact throughput and latency constraints.

5. DYNAMIC VOLTAGE SCALING

Each individual processing element in our architecture is a locally synchronous module operating with its own clock and either being a single VFI or sharing a VFI with another synchronous module, depending on physical floorplanning of the system on a chip. This facilitates dynamic voltage/frequency scaling in each synchronous local module using a DC-DC voltage regulator and a variable delay ring oscillator maintaining the local clock of the VFI and controlled by the voltage regulator. In this section we examine a few strategies to perform dynamic voltage scaling in a network of PEs under throughput and latency constraints.

Prediction Based Intranode DVS

The results shown in the previous section for static voltage assignment given latency constraints, would hold good in the case of a system where an oracle has pre-existing knowledge of the number of run time cycles used in each PE for processing each sample of the application under consideration. In such a case, for simple forward paths, instead of the result shown in equation (17) we would need to use the actual number of cycles per sample to get the following voltage assignment for each PE, as follows:

$$V_i = \left(\frac{(A_1.K_1.(\eta_1/\eta_i)^{\frac{\alpha-1}{1+\alpha}} + \cdots + A_n.K_n.(\eta_n/\eta_i)^{\frac{\alpha-1}{1+\alpha}})}{D} \right)^{\frac{1}{\alpha-1}} \qquad (18)$$

where $A_i = actual\ number\ of\ cycles \ \leq W_i$. The variation in the number of run time cycles may occur due to the data dependent behavior of the process executing on the PE. In real life examples such as a mpeg encoder or decoder, the variability in the processing of several modules such as motion estimation, motion prediction and IDCT/DCT can be utilized to ramp down the voltage of the corresponding processing element, thereby saving energy.

However, an oracle based dynamic voltage assignment is unrealistic in all practical applications since we do not have prior knowledge of the processing time of each sample in a PE. Based on some training data such as the number of cycles required by the PE to process past samples, we may predict the workload requirement of the PE for the next sample and scale the local voltage accordingly. Such a scheme would be very useful in applications where the workload is mostly average case instead of worst case. Prediction schemes may vary from a simple moving average estimate to least mean square predictive filters [9].

Our approach to dynamic voltage scaling is a prediction based one. **DynamicVFAssign** shows our dynamic voltage scaling algorithm for real time applications with throughput and latency constraints running on a VFI based framework. The decision to change the voltage of a node is taken locally at intervals of N cycles. Each node predicts the execution cycles of the next sample based on the previous predicted and actual cycles and an user defined factor ρ $(0 < \rho <= 1)$ and scales its voltage accordingly. If there is a misprediction, we assign a *penalty factor* for the next sample and force it to run at a higher voltage to meet our constraints. Each node maps its voltage to available discrete voltage levels $(V_1, \cdots, V_s, s < n)$. Though a prediction based algorithm may create some constraint violations for very bursty traffic, these may be tolerated by a soft real time system.

In some applications such as a MPEG encoder, some of the PEs may be inactive for a significant number of cycles depending on the input data. In such cases, the power supply of the island may be switched off, saving significant static power. Static power minimization, however, is not the focus of this paper. Also, ideally in a *latency constrained* system, the assignment of optimal voltages would need a global decision strategy, as shown in equation (18). The hardware costs associated with a global controller makes it imperative to design a local/clustered solution which is the direction of future work.

```
Algorithm DynamicVFAssign
Inputs: Component Graph $\mathcal{G}$; Deadline $D$; Source rate $R$
        discrete voltage levels $V_1, \cdots, V_s$; $W_i \ \forall \ i \ \in \mathcal{G}$
Outputs: Voltages $V_1, \cdots, V_n$ at time $t$
1. Let $Pred_{i(current)}$=current predicted cycles, $A_{i(current)}$=
current actual cycles, $Pred_{i(prev)}$= previous predicted cycles,
$A_{i(prev)}$=previous actual cycles
2. For all nodes $i \in \mathcal{G}$
     $Pred_{i(current)} = W_i$
     Set initial voltage to $V_{i(static)}$ from StaticVFAssign
3. Repeat at every $N$ cycles $\forall \ i \ \in \mathcal{G}$
     $Pred_{i(current)} = \rho * Pred_{i(prev)} + (1-\rho) * A_{i(prev)}$
     Set $V_i = V_{i(static)} * (Pred_{i(current)}/W_i) *$ penalty_factor
     Map $V_i$ s.t $V_i \in \{V_1 \cdots V_s\}$
     If($Pred_{i(current)} < A_{i(current)}$)
          Set penalty_factor> 1 for next sample
          Run $(A_{i(current)} - Pred_{i(current)})$ cycles at $V_{max}$
4. until (source is idle)
```

Table 2: Algorithm for Dynamic Voltage Scaling

6. EXPERIMENTAL RESULTS

In this section we describe a couple of case studies on real life applications which can be implemented as heterogenous multiprocessor system on a chip applications. To this end, we perform our experiments on Myrmigki [10], a publicly available cycle accurate processor simulator, modifed to support multiprocessor power and performance modeling, and ADS, an ARM simulator [11]. Although the presented results target a system architecture comprised of homogeneous cores, our results are equally applicable for heterogenous systems. Myrmigki currently models the Hitachi SH architecture, specifically the SH3 (60 Mhz, 3.3V). For the ARM simulator, we use an ARM7TDMI core running at 133Mhz (1.6 V).

The first application under consideration is software radio, which is partitioned into five components - source, low pass filter (LPF), demodulator (DEMOD), equalizer (EQ). Each component is implemented as a stand alone application executing on a single processor. The source state generates samples at a fixed rate which are sent to the LPF node through a point to point communication channel. In order to provide a better utilization of available cores, we have built a more refined version of the software radio in which the EQ stage is further partitioned into ten stages (Figure 2) which receive samples in round robin manner implementing a pipeline. Our baseline model is a

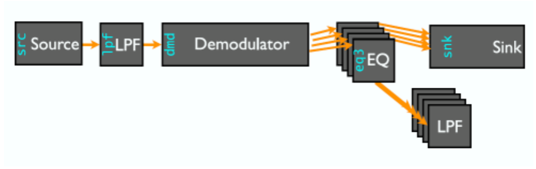

Figure 2: Software Radio with pipelined equalizer stage

single clock multiprocessor architecture having a single global clock and voltage supply. We compare this baseline case with a multiple VFI architecture in which each core running a different process is a locally synchronous module with its own voltage supply. We perform experiments with different rate and deadline constraints using our static voltage assignment algorithm. We compare the energy savings per sample against the baseline case. Since software radio does not show any data dependent processing time variations, we only perform static voltage assignment experiments to prove the viability of a multiple VFI based GALS architecture. The experimental results of running a partitioned software radio on Myrmigki are shown in Figure 3. In the baseline case, we run the modules on thirteen 60 Mhz, 3.3 V SH3 cores for an input 1 Khz signal. Applying our static voltage assignment algorithm, we determine the voltage at which each core should run with the given input rate and deadline constraints based on the worst case execution cycles of each module. Myrmigki has a built-in instruction level energy estimator which we use to calculate the switched capacitance per node. Since SH3 is a simple five stage RISC architecture and the instruction mix of each process of the software radio are almost identical, we get equal switched capacitance/cycle values for each node (0.00124 μF/cycle). We record energy savings of up to **42%** if we assume that a core can choose from an infinite number of voltage levels. Assuming six discrete voltage levels per node leads to as much as **40%** savings per sample. Our experiments (Figure 3) include a simple rate constraint (1 Khz) as well as rate *and* latency constraints (10-13 ms). Table 3 shows the worst case execution cycles and the static voltage/frequency assignments per node based on latency and deadline constraints. We observe that energy savings are less as the deadline decreases due to more stringent restrictions on a PE in the system. The strictest deadline is obtained by running each node at its maximum frequency, i.e, 60 Mhz in this case.

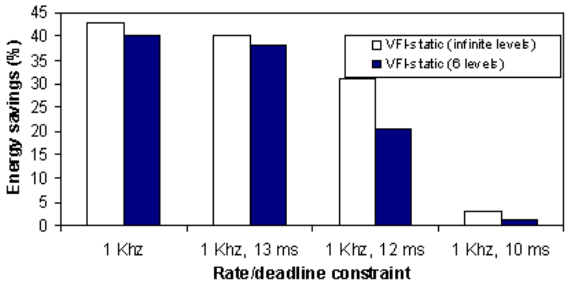

Figure 3: Software Radio static voltage allocation

Software Radio: 6 Voltage levels(3.3,2.9,2.5,2.1,1.7,1.3)(V)				
	LPF	Demod	Equalizer(10)	Sink
Cycles	61494	33086	463193	32736
1 KHz	(3.3,60)	(2.1,38)	(2.5,45)	(2.1,38)
1KHz,13ms	(3.3,60)	(2.5,45)	(2.5,45)	(2.5,45)
1KHz,12ms	(3.3,60)	(2.9,52)	(2.9,52)	(2.9,52)
1KHz,10ms	(3.3,60)	(3.3,60)	(3.3,60)	(3.3,60)

Table 3: Voltage-frequency(V,MHz) assignments for nodes

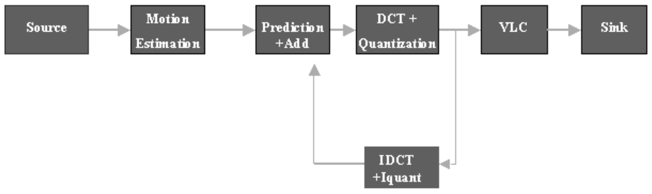

Figure 4: Partitioned MPEG-2 Encoder

As a second application, we have implemented a MPEG-2 video encoder using ARM7TDMI cores for our simulation. The encoder is broken down into six components namely the motion estimator, motion predictor, DCT and quantization block, IDCT and inverse quantization block, the variable length encoding block and the sink. Each component is modeled as a separate process on an ARM7TDMI core. We are able to pipeline the processing of macroblocks in this architecture, though at the start of a new frame, the DCT and IDCT stages have to be flushed requiring some extra buffering between the pipeline stages. This is because a macroblock in a new frame needs the processed data from the previous frame. As in the software radio case, our baseline architecture is a single clock domain architecture (133MHz, 1.6 V), which we compare against multiple voltage/frequency GALS architecture. We perform static voltage/speed assignment based on input rate and deadline constraints, and find out the energy savings per macroblock processed using static worst case time analysis. The maximum frame processing rate in this system is 3.5 frames/second because we use a slow (133MHz) core. Figure 5 shows the energy savings compared to the single clock domain architecture for a YUV movie (99 macroblocks per frame) based on both static and dynamic voltage assignment with infinite and limited (6) voltage levels. In the graph, VFI-static+dyn(INF) refers to oracle based DVS with infinite voltage levels, while VFI-pred+dyn refers to prediction based DVS using 6 discrete voltage levels. Table 4 shows the worst case execution cycles and the static voltage/frequency assignments per node based on latency and deadline constraints. Since MPEG videos show a lot of variability in processing time depending on the type of frame being processed, we also perform prediction based dynamic voltage scaling on each processor as described in Section 6 (DynamicVFAssign). The prediction decision is taken at the start of processing of a new macroblock at each node. We perform DVS over a period of 50 frames. Our experiments with 6 discrete voltage levels show no rate violations (per frame) and a maximum of 3% deadline violations per macroblock for very stringent deadline constraints (40ms/macroblock). As the results show, we get around **5-13%** savings for the static case over the baseline architecture and a further **30-35%** savings through dynamic voltage scaling. As expected, the savings decrease with stringent latency constraints. However, we see that even though our static energy savings decrease sharply, the DVS based savings remain high. This is because most

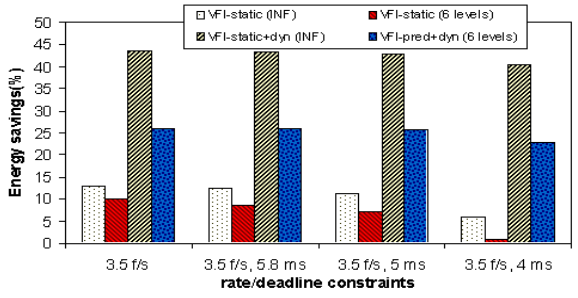

Figure 5: MPEG Encoder energy savings for carphone.yuv

MPEG Encoder: 6 Voltage levels(1.6,1.4,1.2,1.0,0.85,0.65)(V)						
	ME	Pred	DCT	VLC	IDCT	Sink
Cycles/MB	101282	16722	370060	43222	351259	3188
3.5f/s	0.65,54	0.65,54	1.6,133	0.65,54	1.6,133	0.65,54
3.5f/s,58ms	0.85,70	0.85,70	1.6,133	0.85,70	1.6,133	0.85,70
3.5f/s,50ms	1.0,83	1.0,83	1.6,133	1.0,83	1.6,133	1.0,83
3.5f/s,40ms	1.6,133	1.6,133	1.6,133	1.6,133	1.6,133	1.6,133

Table 4: Voltage-frequency(V,MHz) assignments for nodes

macroblocks require far less processing time than the worst case execution cycles based on which we do our static design.

7. RELATED WORK

GALS systems have been studied in detail by Chapiro in his thesis [12]. His work covers metastability issues in GALS systems and outlines a stretchable clocking strategy which provides a mechanism for asynchronous communication.

In a GALS based system, the synchronous modules have to communicate with each other asynchronously which may lead to metastability issues. Chelcea and Nowick [5] use mixed clock FIFOs as a low latency asynchronous communication mechanism between synchronous blocks. Mutterbasch et al. [4] have implemented asynchronous wrappers around synchronous blocks. They have used this methodology to design and fabricate an asynchronous multi-point interconnect system consisting of several GALS modules.

Dynamic Voltage Scaling schemes have been widely studied and implemented. In [3], an example of dynamic voltage scaling in a system with self-timed circuits is presented. The difference in pipeline stage latencies has been explored for lowering the power requirements at the system level [13]. In [14], a novel modelling and simulation environment is presented for multiple clock, dynamic voltage *single core* systems. Luo et al. address the problem of static and dynamic voltage scaling of multi-rate periodic task graphs and aperiodic tasks in distributed real time embedded systems in [15]. While they primarily solve a scheduling problem on multiple PEs, we focus on architecting voltage island based pre-mapped systems based on rate and deadline constraints. Finally, Chandrakasan et al. [16] present an approach to minimize the energy consumption per data sample in variable DSP systems by adaptively minimizing the supply voltage for each sample using a variable speed processor.

In [17], Peh et al. try to optimize both communication and processor energy consumption by scaling voltages of the communication links. In [18], Dhillon et al. propose an optimum methodology for assigning supply and threshold voltages to modules in a CMOS circuit to minimize energy consumption. We model our problem at the level of application specific modules mapped onto a processing element as opposed to voltage assignment to circuit level modules in this paper. Also, application level variability in processing time allows us to perform dynamic voltage/frequency scaling which is absent in the analysis in the afore mentioned paper.

8. CONCLUSIONS

This paper addresses the problem of selecting voltages and speeds for voltage/frequency islands in a globally asynchronous, locally synchronous systems. The problem of both rate and latency constrained systems is considered and a practical solution for static and application adaptive, dynamic voltage and speed scaling is provided. As a direction for future work, an interesting problem in dynamic voltage scaling of *latency constrained* systems is to partition the computation of dynamic voltages for the processing elements among several local listening islands controlling a group of processing elements. In addition, extensions that include joint supply and threshold voltage assignment can be considered and are the objective of future work.

9. REFERENCES

[1] D.E. Lackey, P.S Zuchowski, T.R Bednar, D.W. Stout, S.W Gould, and J.W Cohn. Managing power and performance for system-on-chip designs using volatge islands. *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, November 2002.

[2] IBM Blue Logic Cu-08 voltage islands. *http://www.ibm.com/chips/products/asics/products/v_island.html.*

[3] Lars S. Nielson, Cees Niessen, Jens Spars, and Kees Van Berkel. Low-power operation using self timed circuits and adaptive scaling of the supply voltage. *IEEE Transactions on Very large Scale Integration (VLSI) Systems*, 2(4):391–397, December 1994.

[4] J. Muttersbach, T. Villiger, and W. Fichtner. Practical design of globally asynchronous locally synchronous systems. *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 2000.

[5] T. Chelcea and S.M. Nowick. A low latency fifo for mixed-clock systems. *Proceedings of IEEE Computer Society Workshop on VLSI*, April 2000.

[6] Ali Dasdan. *Rate Analysis of Embedded Systems*. PhD thesis, University of Illinois at Urbana Champagne, 1998.

[7] J.Adam Butts and Gurindar Sohi. A static power model for architects. *Proceedings of International Symposium on Microarchitecture*, December 2000.

[8] C. Hu. *Devices and Technology Impact on Low Power Electronics, Low Power Design Methodolgies*. Kluwer Academic Publishers, 1996.

[9] A. Sinha and A. Chandrakasan. Dynamic voltage scaling using adaptive filtering of workload traces. *Proceedings of International Conference of VLSI Design*, January 2001.

[10] Myrmigki cycle accurate simulator. *www.myrmigki.org.*

[11] ARM Developer Suite. *http://www.arm.com/products/DevTools/ADS.html.*

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

[13] G. Qu, D. Kirovski, M. Potkonjak, and M. B. Srivastava. Energy minimization of system pipelines using multiple voltages. *Proceedings of IEEE International Symposium on Circuits and Systems*, May 1999.

[14] Anoop Iyer and Diana Marculescu. Power efficiency of voltage scaling in multiple clock, multiple voltage cores. *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, November 2002.

[15] Jiong Luo and Niraj Jha. Static and dynamic variable voltage scheduling algorithms for real time heterogenous distributed embedded systems. *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, November 2000.

[16] A. Chandrakasan, V. Gutnik, and T. Xanthopolous. Data driven signal processing: An approach for energy efficient computing. *Proceedings of International Symposium on Low Power Electronics and Design*, August 1996.

[17] LS Peh and Niraj K. Jha. Simultaneous dynamic voltage scaling of processors and communication links in real time distributed embedded systems. *Proceedings of IEEE Design Automation and Test in Europe*, March 2003.

[18] Y.V. Dhillon, A.U. Diril, A. Chatterjee, and H.S. Lee. Algorithm for achieving minimum energy consumption in cmos circuits using mutiple supply and threshold voltages at the module level. *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, November 2003.