# Hardware Based Frequency/Voltage Control of Voltage Frequency Island Systems*

Puru Choudhary
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
puru@cmu.edu

Diana Marculescu
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
dianam@ece.cmu.edu

## ABSTRACT

The ability to do fine grain power management via local voltage selection has shown much promise via the use of Voltage/ Frequency Islands (VFIs). VFI-based designs combine the advantages of using fine-grain speed and voltage control for reducing energy requirements, while allowing for maintaining performance constraints. We propose a hardware based technique to dynamically change the clock frequencies and potentially voltages of a VFI system driven by the dynamic workload. This technique tries to change the frequency of a synchronous island such that it will have efficient power utilization while satisfying performance constraints. We propose a hardware design that can be used to change the frequencies of various synchronous islands interconnected together by mixed-clock/mixed-voltage FIFO interfaces. Results show up to 65% power savings for the set of benchmarks considered with no loss in throughput.

**Categories and Subject Descriptors:** B.7.m [Logic Design]: Miscellaneous

**General Terms:** Performance, Design

**Keywords:** voltage-frequency islands, globally asynchronous locally synchronous, dynamic frequency and voltage scaling, mixed-clock fifos, throughput

## 1. INTRODUCTION

One of the main long-term system-level design challenges (as mentioned in the 2005 ITRS [3]), is the prohibitively costly global, on-chip synchronization due to process variability, power dissipation, and multi-cycle cross-chip signaling. Indeed, with increasing clock speeds and shrinking technologies, distributing a single global clock signal throughout a chip is becoming a difficult and challenging proposition. A Globally Asynchronous, Locally Synchronous design (GALS) is considered a promising technique for achieving low power consumption and modularity in design. As one other long-term system-level design challenge is on-chip power management, such an organization fits nicely with the concept of voltage islands, which can be effectively used as a means for achieving fine-grain system-level power management.

Voltage-Frequency Islands (VFIs) enable the design of systems that use a clock for local synchronization of data, but communication between different blocks is handled asynchronously. This not only helps to reduce the power consumed by the clock network due to reduced number of buffers that are used to meet the skew, but also helps in reducing the overall power significantly by using voltage scaling.

Most systems are overdesigned to meet the performance requirement of the worst case scenario. Such systems constantly operate at peak performance consuming peak power all the time. However, cooling and battery technology are not able to keep up and meet the power requirements of those designs. It, therefore, becomes necessary to make these systems more power and energy aware such that they use just enough power to meet the performance requirements of the given workload. Dynamic Voltage and Frequency Scaling (DVFS) schemes have become a common-place solution for adapting the power/energy consumption of a system based on a dynamically changing workload. While DVFS schemes have been applied mostly at application and system level by exploiting available slack in task scheduling for minimizing dynamic power with little or no performance hit, the case of hardware-based DVFS for VFI systems has received less attention. The goal of this paper is to provide a solution for dynamic voltage-frequency selection by using a fully hardware-based control scheme driven by the workload variations.

The rest of the paper is organized as follows: Related work and contribution of this paper are presented in Section 2. Section 3 discusses the problem formulation and assumptions made in this paper. In Section 4, we present the theoretical basis for our method and how it can be used to configure an entire system for low power. Our proposed architecture to enable DVFS in a system is discussed in Section 5. Section 6 discusses the Topology Generation Tool, while in Section 7, we provide the experimental results for software radio and MPEG-2 encoder benchmarks. Final conclusion with directions for future research are provided in Section 8.

## 2. RELATED WORK AND PAPER CONTRIBUTION

Previous approaches based on availability of channel in multiple clock systems (e.g., [4]), only gate the clock to the synchronous module. While this approach can reduce total power consumption, voltage scaling is not used as each synchronous module still operates at a fixed frequency. Also, too many pauses in the clock produce sharp variations in power consumption, potentially degrading the battery performance [14]. Our approach changes the clock frequency to minimize the idle time spent waiting for FIFOs (either for read or write of data).

There have been several proposals to implement VFIs in modern systems such as a Multiple Clock Domain processors [15][16]. Such architectures allow a system designer to implement local DVFS algorithms [17], but most of these approaches assume hardware control is done via FIFO occupancy monitoring which can provide incorrect decisions, as it will be seen in the sequel. Some of the on-line algorithms are inherently non-linear [17] requiring detailed analysis of queue behavior before an actual hardware could be implemented. Our method provides a flexible hardware platform that can be used to enable DVFS for VFI systems with simple data patterns while also providing methods to support more com-

*This research has been supported in part by Semiconductor Research Corporation contracts 2004-HJ-1189 and 2005-HJ-1314.


*CODES+ISSS'06,* October 22–25, 2006, Seoul, Korea.

plicated workloads. The problem of voltage/speed selection in VFI systems has been addressed before [13] via providing an off-line algorithm and a dynamic on-line algorithm with limited efficiency. DVFS schemes for applications such as MPEG decoder [7] look at the overall rate and latency requirements, but do not consider the possible power/energy savings with a VFI based architecture. In our approach, the benefits of DVFS are exploited at finer granularity level, while maintaining the possibility of global configuration.

The main contributions of this paper are two-fold:

- First, it provides an *online, hardware-based* control mechanism for dynamically selecting the operating speed and voltages for individual VFIs in a VFI-based system. As opposed to existing schemes that monitor only FIFO occupancy to determine scaling factors [15][10][16], our approach takes into account the workload dynamics and relies on a combination of producer/consumer stall and FIFO occupancy monitoring. In addition, the approach is cost minimal as it relies on counters associated with stall events, as opposed to complex schemes relying on control theoretic approaches (e.g., PID controllers [17]).
- Second, we provide a framework that enables any application specified in TGFF format [1] to be automatically converted into a Verilog description of the VFI system including both computation and FIFO-based communication.

## 3. PRELIMINARIES AND ASSUMPTIONS

Without loss of generality, we consider the case of systems comprised of a number of synchronous cores, IPs or processing elements (PEs) (homogeneous or heterogeneous). In the case of VFI-based systems, PEs can only be assigned to a single VFI (in other words, cores cannot belong to more than one VFI).

A VFI might consist of a single PE or, depending on the physical or design considerations, may include a group of PEs. We assume that power in the case of VFI systems is supplied by an off- or on-chip source and can be controlled independently for a VFI. This may be achieved by using either on-chip voltage regulators or multiple power grids [2]. Since each VFI is locally synchronous, it is assumed to be clocked using a ring oscillator controlled by the intra-island supply voltage using a digital phased lock loop [12][11]. Communication is implemented via a modified version of mixed-clock FIFOs [6] that also allows for voltage level conversion. We 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. We also assume that the processes have already been scheduled on their respective processing elements. For VFI systems, a bounded number of storage cells is available in the mixed-clock FIFOs used between two communicating PEs. To this end, the system comprised of communication cores is modeled using a *component graph*. In a component graph *G(V, E)*, cores are modeled as communicating processes (nodes) that have associated communication channels between them (edges).

We will assume the following, without loss of generality:

- The component graph *G(V, E)* is characterized by the set of nodes represented as $V = \{1, 2, ..., n\}$ and edges represented as $E=\{(i,j) \mid i \text{ precedes } j\}$.
- Although the underlying component graph model may include feedback paths, in the initial theoretical treatment we restrict ourselves to directed acyclic graphs (DAGs). General graphs have been shown to be reducible to acyclic component graphs by lumping strongly connected components (SCCs) including feedback loops into supernodes [13],[8]. As shown in [8], the processing rates of these supernodes (and thus, their latencies in cycle counts) can be found by averaging across all nodes in the SCC. However, the case of feedback loops is addressed and discussed in Section 5.3.
- The component graph includes a single source node ($s$) and a single sink node ($S$). Graphs including multiple sinks or source nodes can be reduced to this case by adding dummy, zero-latency source (sink) nodes feeding into (from) the actual source (sink) nodes.

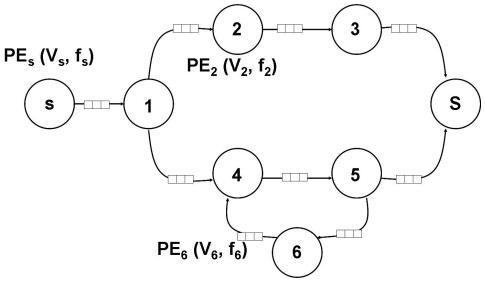


**Figure 1: A VFI-based component graph as in [13] with cores (PEs) characterized by local speeds/voltages**

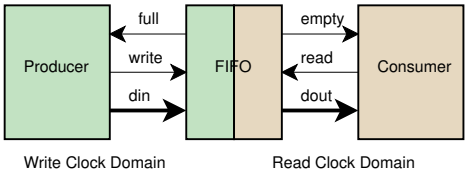


**Figure 2: The Producer Consumer model**

## 4. THE COMMUNICATION ARCHITECTURE

In this section, we describe the use of mixed-clock FIFO as a point-to-point communication architecture for connecting synchronous islands in a GALS system.

### 4.1 The Producer-Consumer Model

In a VFI design, a mixed-clock/mixed-voltage FIFO provides a communication channel between two VFIs. One of the VFIs (producer) writes data into the FIFO while the other one (consumer) reads data from the FIFO [6]. For proper operation of design, it is required that a producer does not write data into the FIFO if it is full. Similarly, a consumer should not read data from a FIFO if it is empty. The producer and half of the mixed-clock FIFO share a clock (producer clock) while the consumer and other half of the mixed-clock FIFO share the other clock (consumer clock). Such a clock domain partition is shown in Figure 2.

### 4.2 Rate Matching

Considering a simple producer-consumer model of a mixed-clock FIFO, the behavior for ideal frequency of operation can be derived based on the read and write data rates.

The time interval between any two write operations by the producer can be written as, $T_p = a_p/f_p$, where $a_p$ is the number of clock cycles between any two write operations by the producer and $f_p$ is the frequency of operation of the producer. Similarly, the time interval between any two read operations by the consumer can be written as, $T_c = a_c/f_c$, where $a_c$ is the number of clock cycles between any two read operations by the consumer and $f_c$ is the frequency of operation of the consumer.

If $T_p$ is equal to $T_c$, then the FIFO utilization will be constant most of the time. However, if $T_p < T_c$, the FIFO will tend to become full. Hence once the FIFO is full, the producer will have to wait until the consumer has taken at least one data item out of the FIFO. Therefore we can write,

$$T_c = T_p + T_w \quad (1)$$

where $T_w$ is the time spent by the producer waiting for an empty slot in the FIFO. To operate the system near optimal operating point, this time $T_w$ should be minimized and made zero in an ideal case. For such a case, we can write,

$$T_c = T_{pi} \Rightarrow \frac{a_c}{f_c} = \frac{a_p}{f_{pi}} \Rightarrow f_{pi} = \frac{a_p}{a_c} f_c \Rightarrow f_{pi} = \frac{a_p}{a_c} k f_p \quad (2)$$

where $T_{pi}$ is the ideal time interval between any two write operations by the producer while $f_{pi}$ is the ideal clock frequency of the producer. $k$ is the ratio of consumer clock frequency to producer clock frequency. Thus, we can also write ideal clock frequency of the producer as follows: $f_{pi} = S f_p$, where $S = (a_p/a_c)/k$ is the *Frequency Step factor* by which the producer frequency should be scaled so that the wasted power is minimized. The choice of the new clock freqeuncy should be made conservatively, such that there is no drop in overall throughput. For example, if $a_p = 2$, $a_c = 6$ and $f_p = f_c$, the ideal speed of the producer should be $f_{pi} = (1/3)f_p$. The optimal available frequency should be chosen such that it is the closest, *largest* value available such that no throughput loss is experienced. E.g., in this case, if a value of $f_{avail} = f_p/2$ is available, the producer will still be slow enough to reduce waiting time $T_w$, but fast enough to not decrease the throughput. If, however, $a_p = 2$ and $a_c = 3$, the ideal producer speed would be $f_{pi} = (2/3)f_p$ and a $f_{avail} = f_p/2$ available frequency will not guarantee the throughput constraint. Hence it is always necessary to have $f_{pi} < f_{avail}$. This analysis can be similarly applied to the case of $T_p > T_c$, where the FIFO will tend to become empty. In this case, the frequency of the consumer should be kept just enough to operate the FIFO near empty state, without having to experience any throughput reduction.

### 4.3 Problem Formulation

The goal of the work presented in this paper is to reduce the total energy consumption as well as power consumption of a system represented by a component graph *G(V, E)* subject to rate or throughput constraints.

The energy consumption per sample for every processing element in the component graph *G(V, E)* is given by:

$$E_i(V_i) = C_i * N_i * {V_i}^2 + c_i * n_i * V_i * exp(-V_t/k) \quad (3)$$

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. $C_i$ 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 [5].

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

$$\tau_i(V_i) = K_i * \frac{V_i}{(V_i - V_t)^\alpha} \quad (4)$$

where $K_i$ and $\alpha$ are design and technology dependent parameters [9]. Thus from (4), 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 $PE_i$):

$$WCET_i(V_i) = W_i * \tau_i(V_i) = (W_i * K_i * V_i)/(V_i - V_t)^\alpha \quad (5)$$

For a system to operate as per the requirements of an application workload, it is needed that,

$$WCET_i(V_i) \leq T_i \quad (6)$$

where $T_i$ is the *required* time period of every VFI core. Most of the modern systems are not only designed for worst case workload conditions, but also operate at peak performance all the time to be able to handle the worst case workload. As a result, for an average workload we get $WCET_i(V_i) << T_i$. This results in smaller $\tau_i(V_i)$ and hence larger $V_i$ which leads to higher energy consumption. To reduce the amount of the wasted energy, $WCET_i(V_i)$ should be as close as possible to $T_i$, i.e.

$$Minimize(T_i - WCET_i(V_i)) \quad (7)$$

By taking $WCET_i(V_i)$ closer to $T_i$, the amount of time wasted $T_w$ (1) waiting for the communication channel is minimized. The reverse is also true i.e. $T_w \rightarrow 0 \Rightarrow (T_i - WCET_i(V_i)) \rightarrow 0$. Operating each PE at its ideal frequency/ voltage, the amount of time wasted $T_w$ is minimized resulting in minimum energy and power consumption. However, based on the available system configuration settings of a real system (for example, number of available frequency and voltage levels), the *optimal* achievable solution will be close, but not identical to the ideal one. Our hardware based approach tries to find this optimal solution based on dynamically changing speeds/voltages driven by the workload.

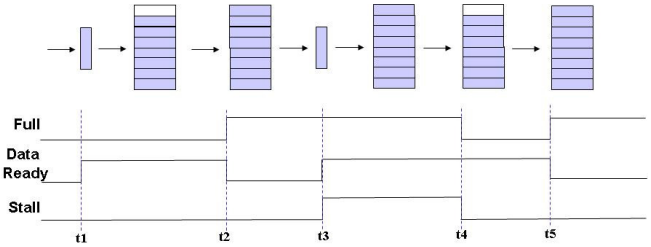


**Figure 3: Comparison between Full and Stall signal for Frequency prediction**

## 5. THE FIFO LINK ARCHITECTURE

The derivations shown in Section 4 can be used to calculate the ideal frequencies of the producer and the consumer under dynamically changing workload. However, in a complex system, the values of $a_p$ and $a_c$ are likely to change due to varying workload conditions. Also, the overhead of computations to find the value of the *Frequency Step factor* (Section 4) is likely to be significant. We propose an architecture that can predict the value of the *Frequency Step factor* (and hence the ideal frequency) on the fly.

### 5.1 Proposed Architecture

To implement such a logic for estimating the optimal operating frequency, we take advantage of the fact that when the producer (or consumer) is not operating at the ideal frequency, the FIFO will always operate near full (or empty) state. We call these *mostly full* and *mostly empty* conditions. A simple way to monitor the FIFO utilization is to check the full and empty signals and measure the amount of time they are asserted: the larger the time of assertion of any one of these signals, the greater the deviation of the frequencies of producer (or consumer) from the ideal frequency. However, full and empty signals do not accurately represent the need for scaling up or down the speed/voltage of a VFI. It can happen that even though the full signal is asserted, the producer/consumer does not have any data to write/read into/from the FIFO. Thus, taking the decision to slow down a VFI only based on the FIFO occupancy can prove to be incorrect.

Figure 3 shows an example of a producer writing data into a FIFO. For the time interval between $t1$ and $t5$, the full signal is asserted for time period $(t4 - t2)$. However, the time period where producer is actually waiting for the FIFO to have an empty slot is $(t4 - t3)$. If the *Frequency Step factor* is calculated based on the full signal alone, it is likely

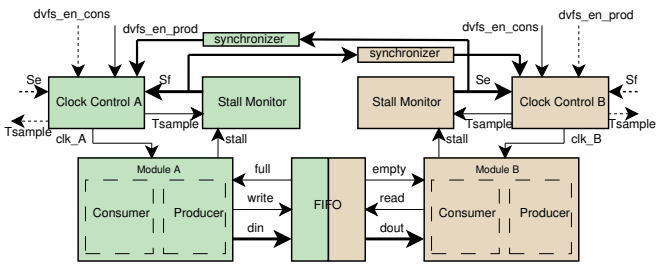


**Figure 4: Dynamic Frequency Scaling Architecture**

to overestimate the frequency decrease and can potentially reduce the throughput of the system. A similar argument applies to the empty signal. A more accurate estimation can be achieved if a signal (called *stall* signal) generated by a producer/consumer is used to estimate the ideal frequency. This signal is asserted whenever the producer/consumer has data to write/read to/from the FIFO, but the FIFO is full/empty. Figure 4 shows the architecture that can predict the ideal frequency based on this method. The stall monitors count the number of clock cycles ($S_f$-for the producer part or $S_e$-for the consumer part) the *stall* signal from producer/consumer is asserted in a sampling window $T_{sample}$. The *Frequency Step factor* can then be calculated based on the non-zero values of $S_e$ and $S_f$. While in steady-state it is impossible to have both $S_e$ and $S_f$ non-zero (i.e., both consumer and producer of a *FIFO link* stalling at the same time), when cumulative stalls are accounted for, this could happen, e.g., for bursty traffic: the producer might stall during the beginning of the sample interval $T_{sample}$, while the consumer might stall during the last part of it. In such a case, if the amount of stalling is the same on both ends, scaling the speeds of producer/consumer will not remove this problem. On the other hand, usually, in a sampling interval it is always the case that either the producer stalls due to a full FIFO or a consumer stalls due to an empty FIFO. To capture both of these cases, the *Frequency Step factor* can be calculated as $S = 1 - |S_e - S_f|/T_{sample}$. If only one of producer or consumer stalls, then the scaling factor is computed according to $S_f$ or $S_e$, respectively. If both stall at different times during the sampling interval, then the difference is used to smooth out any differences between the two rates. For a producer, if $S_f > S_e \geq 0$, then

$$f_{new} = f_{curr} * S \tag{8}$$

where $f_{new}$ is the new frequency while $f_{curr}$ is the current frequency. However, if $S_e > S_f \geq 0$, then

$$f_{new} = f_{curr}/S \tag{9}$$

as in this case, the consumer is experiencing stalls and producer needs to increase the frequency. The reverse (i.e., changing division to multiplication and vice-versa) is true for consumer. However, for each FIFO link, only one of the producer or consumer modules will be scaled up or down to keep the throughput constraint, while minimizing wasted power during stalls. This approach is described next.

## 5.2 Throughput Constraint and Scaling State

In general, throughput constrained systems require an output rate to be satisfied for correct operation. For example, in the case of the system in Figure 1, the sink node $S$ needs to have a certain rate of generating data items. Examples of throughput constrained applications include most media processing, data communication systems, digital-to-analog converters, etc. However, many times, the constraint is given at the input - that is, the incoming data items must be processed at a certain rate to ensure correct operation. Such an example is an analog-to-digital converter. Irrespective of where the rate constraint is specified (source $s$ or sink

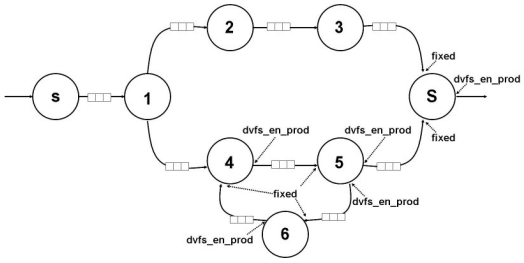


**Figure 5: A VFI-based component graph with FIFO configuration**

$S$ in Figure 1), based on it, we can determine how each producer/consumer port can be configured for possible scaling up or down of the corresponding VFI, as described in Section 5.1. Let us consider the more common case of output rate constrained systems depicted in Figure 5. For the producer port of the sink node $S$, there is no *FIFO link* associated with it, but a stall monitor can be used to determine if the data is produced at the required rate. If not, a corresponding scaling factor can be associated with the sink: $S_S = T_{S\_observed}/T_S$ where $T_{S\_observed}$ is the observed period between data items being produced and $T_S$ is the required value. For the rest of the nodes we need to consider all incoming and outgoing ports associated with each FIFO link. Intuitively, if throughput constraints are propagated from the outputs to the inputs, we need to maintain required throughput in the downstream VFIs while allowing only producers to be scaled (up or down), while the consumer port is assumed to be fixed. We call this state associated with the producer port *dvfs_en_prod*, and the one associated with the consumer *fixed* since it is not allowed to change speeds/voltages based on stall information related to that *FIFO link*.

In Figure 5, the assignment of port states for VFIs 4, 5, 6 and $S$ is shown (similar for the other nodes 1, 2, 3, and $s$) for an output rate constrained system. Similarly, for an input rate constrained system, each consumer in a *FIFO link* would be in a state of *dvfs_en_cons* (consumer is allowed to scale) and each producer would be in a *fixed* state (no scaling).

## 5.3 Functionality of Clock Control Logic

We are now ready to determine what is the correct scaling factor for each VFI, given the constraints on the output (or input) rate and given that multiple scaling factors may be determined from multiple incoming/outgoing FIFOs. We need to keep in mind that the *FIFO link* architecture depicted in Figure 4 might be replicated many times, for each producer-consumer channel. More precisely, the Clock Control Logic gets the prediction value from both stall monitors associated with the FIFO. As described previously (Eqn. 8 and Eqn. 9), in the case of the producer, the stall information from the consumer is used to increase the frequency of that domain if the current frequency is not able to meet the throughput requirements of the design (similar for the consumer).

For each VFI, there might be multiple producer and consumer ports as data may be coming from multiple sources or distributed to multiple sinks. In addition, for each VFI, there are as many stall monitors, associated with producer ports, as there are outgoing FIFOs, and as many stall monitors, associated with consumer ports, as there are incoming FIFOs. Figure 4 shows a single one-to-one FIFO link, hence there is only one stall monitor on each side of the FIFO. Since the Clock Control Logic module controls the frequency and voltage of a single VFI, there are as many Clock Control Logic blocks as VFIs in the system, but they will have to receive as many $S_f$ and $S_e$ signals as there are stall monitors for each *FIFO link* interface of that VFI. The decision as to what the prevailing scaling factor is for a given VFI when multiple incoming/outgoing FIFO links dictate differ-

ent scaling factors is taken conservatively. To ensure that the throughput is not reduced, the highest frequency/voltage is considered. Each VFI can have multiple producer or consumer ports, but out of these, only a subset are configured in *dvfs_en_prod* (or *dvfs_en_cons*) state. Only *these ports* and the scaling factor associated with *their* stall monitors are considered in determining the prevailing scaling factor by taking the maximum resulting speed among these. For example, in the example depicted in Figure 5, the new speed/voltage for node 5 depends on the resulting speeds/voltages determined by the FIFO links (5, $S$) and (5, 6). Assuming that based on Eqn. 8 and Eqn. 9, $f_{new,5}(5,S)$ and $f_{new,5}(5,6)$ are the new potential clock speeds, the final clock speed (and associated voltage) is taken such that $f_{new,5} = max(f_{new,5}(5,S), f_{new,5}(5,6))$. For all the other nodes (VFIs), there is only one port configured as *dvfs_en_prod*, and based on it and its associated new clock speed, the final speed/voltage is assigned. Based on these observations, the detailed algorithm for the speed/voltage selection of an output (input) rate constrained VFI system is described in Figure 6.

**Figure 6: Algorithm for Dynamic Speed/Voltage Selection**

Inputs: Component Graph $G$; Sink rate $R = 1/T_S$ or source rate $r = 1/T_s$; Discrete speed/voltage levels $(f_1, V_1), ..., (f_s, V_s)$;
Outputs: Speed/voltage assignment $(f_1, V_1), ..., (f_n, V_n)$ $\forall i \in G$ at time $t$

1. Let $(f_i, V_i) = (f, V)\ \forall i \in G$ where $f = max_i(f_i)$, $V = max_i(V_i)$
2. For all FIFO links $(i, j)$
   If system is sink constrained then
   $state\_prod(i,j) = dvfs\_en\_prod$; $state\_cons(i,j) = fixed$;
   else //source constrained
   $state\_prod(i,j) = fixed$; $state\_cons(i,j) = dvfs\_en\_cons$;
3. Repeat every $T_{sample}$ cycles
   If system is sink constrained then
   $S_S = T_{S\_observed}/T_S$; $f_S = f_S/S_S$;
   set corresponding $V_S$;
   else //source constrained
   $S_s = T_{s\_observed}/T_s$; $f_s = f_s/S_s$;
   set corresponding $V_s$;
   For all FIFO links $(i, j)$
   $S_{i,j} = 1 - |S_{e\_i,j} - S_{f\_i,j}|/T_{sample}$;
   If $S_{e\_i,j} < S_{f\_i,j}$ then $S_{i,j} = 1/S_{i,j}$;
   If system is sink constrained
   For all nodes $i$ with successors $j$
   and $state\_prod(i,j) = dvfs\_en\_prod$
   $f_i = max_j(f_i/S_{i,j})$; set corresponding $V_i$
   Else // system is source constrained
   For all nodes $j$ with predecessors $i$
   and $state\_cons(i,j) = dvfs\_en\_cons$
   $f_j = max_i(f_j * S_{i,j})$; set corresponding $V_j$
4. until (source is idle)

## 6. TOPOLOGY GENERATION TOOL

Embedded applications can be very effectively partitioned into tasks with various, but well defined functionalities. With clearly defined computational boundaries, they are very good candidates for being mapped onto a VFI system. Most of these applications can be represented as task graphs. Embedded Systems Synthesis Benchmarks Suite (E3S) based on benchmarks from The Embedded Microprocessor Benchmark Consortium contains a set of task graphs representing various applications including, but not limited to automotive, consumer, networking, etc. The task graphs available in E3S benchmark suite contain the information about the applications, constraints and various processors that can be used to map the various tasks.

We created a tool (*Topology Generation Tool*), that can convert task graphs into behavioral Verilog. As shown in Figure 7, this program takes .tgff files [1] as inputs and converts all the tasks to behavioral Verilog models of producer/consumer while all the edges are converted to FIFO links. The tool uses the processor information from the task graphs to assign the delays of each of the producer/consumer. With the help of this tool, a designer can test many types of applications just by specifying high level description in the form of task graphs. The generated Verilog can be simulated using any Verilog simulator and is also well documented to help manual changes.

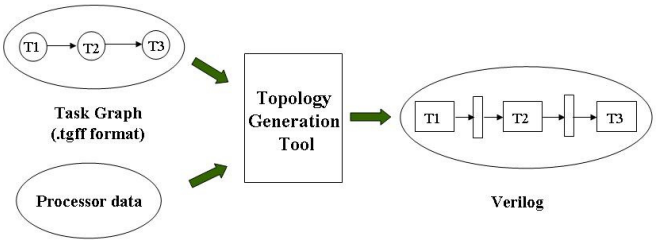


**Figure 7: Flow of Topology Generation Tool**

**Figure 8: Sample TGFF file**

```
@TASK_GRAPH 0 {
PERIOD 0.001
TASK src TYPE 45
TASK ac1 TYPE 21
TASK ce1 TYPE 24
TASK sink TYPE 45
ARC a0_0 FROM src TO ac1 TYPE 0
ARC a0_1 FROM ac1 TO sink TYPE 1
ARC a0_2 FROM ac1 TO ce1 TYPE 1
ARC a0_3 FROM ce1 TO sink TYPE 2
HARD_DEADLINE d0_0 ON sink AT 0.001
SOFT_DEADLINE d0_1 ON sink AT 0.0001
}
```

Figure 8 shows a sample TGFF file that can be used as an input to the *Topology Generation Tool*. Each of the tasks are indicated by `TASK` while the edges are indicated by `ARC`. `PERIOD`, `HARD_DEADLINE` and `SOFT_DEADLINE` are also specified in the TGFF file and are used by *Topology Generation Tool* to parameterize the FIFO system. *Topology Generation Tool* is implemented in Perl.

## 7. EXPERIMENTAL RESULTS

To test our proposed DVFS architecture of a FIFO link, we used Software Defined Radio and MPEG-2 Encoder as driver applications. These applications were represented as task graphs to be used by the *Topology Generation Tool* for generation of behavioral Verilog models which were used to determine the benefits of the online voltage/frequency scaling for each module. $T_{sample}$ was set to 5000 clock cycles for each of these benchmarks.

### 7.1 Software Radio

Software defined radio application can basically be partitioned into five components - namely source, low pass filter (LPF), demodulator, equalizer (EQ) and sink (Figure 9). Each of these nodes can be represented as a producer consumer model. Samples are generated at a fixed rate by the source which therefore defines the throughput constraint. The samples pass through various blocks finally reaching the sink node.

**Table 1: Cycles/packet for Software Defined Radio**

| LPF | Demod | Equalizer(10) | Sink |
|---|---|---|---|
| 61494 | 33086 | 463193 | 32736 |

A base configuration of Hitachi SH3 cores running at the clock frequency of $60MHz$ and supply voltage of $3.3V$ along with an off-line algorithm [13] (with six levels of voltage and frequency) was used for comparison purposes. The six voltage-frequency pairs (in $V, MHZ$) chosen were (3.3,60), (2.9,52), (2.5,45), (2.1,38), (1.7,31), and (1.3,23). The results were obtained for a required sample rate of $1kHz$. As it can be seen from Figure 10, some of the modules like Demod, Equalizer and Sink show significant savings in power, while the second instance of the pipelined LPF modules, which is the bottleneck in the system, shows no improvement at all. However,

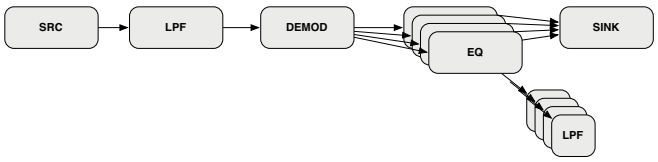


**Figure 9: Partitioned Software Radio**

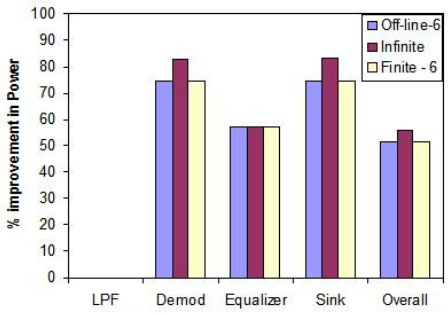


**Figure 10: Power consumption in Software Radio**

the overall improvement is still around 50% and compares well with the off-line method. When there are infinite levels of frequency and voltage levels available, the power saving are greater than those with finite levels (six frequency-voltage pairs) as expected (up to 55% power savings).

## 7.2 MPEG-2 Encoder

The MPEG-2 Encoder is broken down into six components namely the motion estimator (ME), motion predictor (Pred), DCT and quantization block, IDCT and inverse quantization block, the variable length encoding (VLC) block and the sink. For MPEG-2 Encoder, a base configuration with ARM

**Table 2: Cycles/macroblock for MPEG-2 Encoder**

| ME | Pred | DCT | VLC | IDCT | Sink |
|---|---|---|---|---|---|
| 101282 | 16722 | 370060 | 43222 | 351259 | 3188 |

cores running at a clock frequency of $133MHz$ and supply voltage of $1.6V$ was chosen. The same off-line algorithm [13] was used for comparison purposes (with six voltage-frequency pairs). The six voltage-frequency pairs (in $V, MHZ$) chosen were (1.6,133), (1.4,117), (1.2,100), (1.0,83), (0.85,70), and (0.65,54). The results were obtained for frame processing rate of $3.5f/s$ with 99 macroblocks per frame. Figure 12 shows that all blocks, except DCT and IDCT, show a large improvement in power consumption. DCT being the bottleneck of the system, operates at highest available frequency and voltage. For IDCT, our proposed method performs better than the off-line method due to precise detection of workload behavior, providing additional 30-40% power savings locally and 8% additional power savings globally. The overall savings in power are close to 65% for all the three cases with infinite frequency-voltage levels showing more improvement over the finite case (six frequency-voltage pairs).

# 8. CONCLUSION

In this paper, we proposed a hardware based architecture that can be used as a basic building block to build VFI systems and support Dynamic Voltage and Frequency Scaling schemes. The logic to predict the optimal frequency of operation is also presented. A method to propagate the throughput constraint through the entire system is also discussed. We introduced a tool to automatically generate behavioral Verilog from task graphs that can enable and automate analysis of such VFI systems. Future work in this direction can include modification of the FIFO link architecture to address latency constraints, in addition to rate constraints.

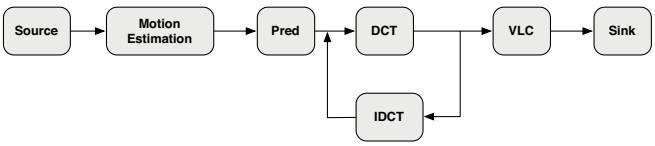


**Figure 11: Partitioned MPEG-2 Encoder**

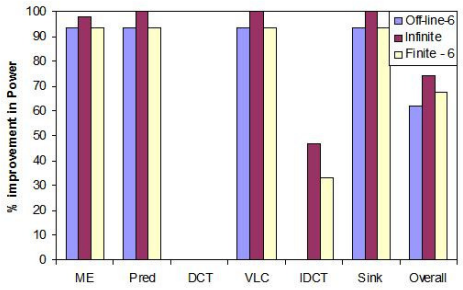


**Figure 12: Power consumption in MPEG-2 Encoder**

# 9. REFERENCES

[1] Embedded systems synthesis benchmarks suite (e3s). *http://www.ece.northwestern.edu/∼dickrp/e3s/*.

[2] Ibm blue logic cu-08 voltage islands. *http://www.ibm.com/chips/products/asics/products/v_island.html*.

[3] International technology roadmap for semiconductors. *http://public.itrs.net*.

[4] A. Agiwal and M. Singh. An architecture and wrapper synthesis for multi-clock latency-insensitive systems. *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, page 1006, November 2005.

[5] J. Butts and G. Sohi. A static power model for architects. *Proc. of International Symposium on Microarchitecture*, December 2000.

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

[7] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a mpeg decoder. *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, Nov. 2002.

[8] A. Dasdan. Rate analysis of embedded systems. *Ph.D. thesis, University of Illinois at Urbana Champagne*, 1998.

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

[10] A. Iyer and D. Marculescu. Power efficiency of multiple clock, multiple voltage cores. *Proc. of IEEE/ACM Intl. Conference on Computer-Aided Design (ICCAD) San Jose, CA*, Nov. 2002.

[11] J. Muttersbach, T. Villiger, and W. Fichtner. Practical design of globally asynchronous locally synchronous systems. *Proc. Intl Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, April 2000.

[12] L. Nielson, C. Niessen, J. Sparso, , and K. Berkel. Low-power operation using self timed circuits and adaptive scaling of the supply voltage. *IEEE Transactions on Very large Scale Integration (VLSI) Systems*, page 391397, Dec. 1994.

[13] K. Niyogi and D. Marculescu. Speed and voltage selection for gals systems based on voltage/frequency islands. *Proc. ACM/IEEE Asian-South Pacific Design Automation Conference (ASPDAC)*, January 2005.

[14] R. Rao, S. Vrudhula, and N. Chang. Battery optimization vs. energy optimization: Which to choose and when. *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, page 439, November 2005.

[15] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. *Proc. of the International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2002.

[16] E. Talpes and D. Marculescu. A critical analysis of application-adaptive multiple clock processors. *Proc. ACM/IEEE Intl. Symposium on Low Power Electronics and Design (ISLPED), Seoul, Korea*, August 2003.

[17] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2004.