Power Management of Voltage/Frequency Island-Based Systems Using Hardware-Based Methods

Puru Choudhary, *Student Member, IEEE*, and Diana Marculescu, *Member, IEEE*

Abstract—**Shrinking technology nodes combined with the need for higher clock speeds have made it increasingly difficult to distribute a single global clock across a chip while meeting the power requirements of the design. Globally asynchronous locally synchronous (GALS) design style can help achieve low power consumption and modularity of a design while greatly reducing the number of global interconnects. Such multiple clock domain architectures can benefit from having frequency/voltage values assigned to each domain based on workload requirements. The work presented in this paper proposes a new *hardware-based* approach to dynamically change the frequencies and potentially voltages of a voltage-frequency island (VFI) system driven by a 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. In recent years, there have been major developments, both in industry and academia, in the field of multiprocessor systems. Such multiprocessor systems are very good candidates for VFI design style implementation, where one or more processors can be part of a single VFI. To demonstrate the feasibility of our proposed method, we have implemented a multiprocessor system for a field-programmable gate array (FPGA) platform that uses independently generated clocks for each processor. The results from the FPGA platform confirm the claim that the power consumption of a system can potentially be reduced while maintaining the performance of many applications. Our work concentrates primarily on embedded systems, but the idea can be explored for general-purpose computing as well.**

Index Terms—**Dynamic voltage and frequency scaling (DVFS), globally asynchronous locally synchronous (GALS), power management, voltage-frequency islands (VFIs).**

I. INTRODUCTION

THE continuous increase in clock frequencies, along with technology scaling, has made the distribution of a single global clock to various parts of a chip increasingly difficult. The large numbers of power-hungry buffers that are needed to maintain small skew requirements elevate the power consumption of a chip significantly. Design styles based on a globally asynchronous locally synchronous (GALS) methodology alleviate the problem of clock distribution by having multiple clocks, each of which can be distributed to a relatively small portion

P. Choudhary is with Marvell Semiconductor, Inc., Santa Clara, CA 95051 USA (e-mail: puruchoudhary@gmail.com).

D. Marculescu is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: dianam@ece.cmu.edu).
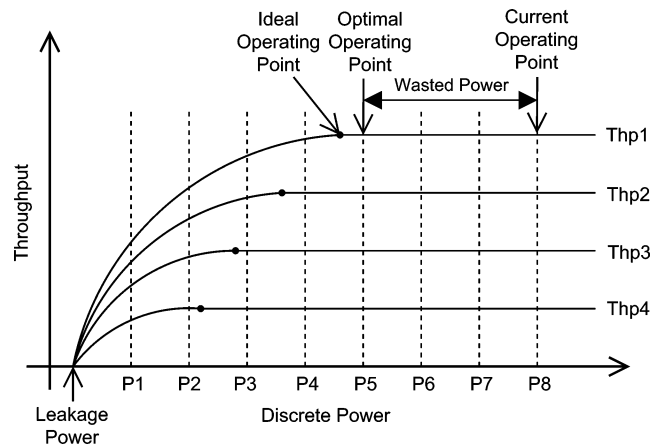
Fig. 1. Throughput versus power for a module in a system.

of the chip. The prospect of having different clock frequencies for each domain also enables design of power-aware architectures. Voltage-frequency islands (VFIs) not only enable frequency scaling, but also voltage scaling. The combined effect of frequency and voltage scaling helps to reduce the power consumption of a chip significantly. The power savings are not only in the clock distribution network, but also in the overall design. Such VFI-based architectures rely on clocks for local synchronization of data, but the communication between different blocks is handled asynchronously.

Most of the designs have irregular workloads when the actual work performed by each block in the system is compared. In general, there are a few modules that are the bottlenecks of the system while most others are idle for large periods of time. As shown in Fig. 1, in a system operating at throughput level *Thp1* and power level *P8*, there is some power wasted since the lower power level *P5* already meets the performance requirements of the system. Such slack in power of various modules can be exploited by decoupling them into independent VFIs. The finer control of frequency and voltage of these VFIs can enable conversion of slack in performance into power savings without actual loss in performance. Such a distributed approach is necessary as the global scaling of single frequency and voltage may not be able to keep up with the power/energy constraints imposed by cooling and battery technologies.

Assignment of frequency and voltage values to each of the VFIs can be done by using either offline or online methods. Offline methods can be used when the behavior of the application is very predictable for various input conditions and the worst-case

behavior is not very different from average-case behavior. However, such an approach is not very suitable for applications that show large variations in their behavior for different input conditions. For such systems, online methods are more suitable. Dynamic voltage and frequency scaling (DVFS) schemes can be used to adapt the system to meet the performance requirements of a dynamically changing workload while consuming the minimum possible power required to meet the performance targets.

A. Paper Contribution

In the first part of this paper, we present an *online, hardware-based* control mechanism for dynamically selecting the operating speed and voltages for individual VFIs in a VFI-based system. The idea behind the *hardware-based* approach is to have the necessary blocks in the system monitor the application workload at a fine-grain level. The information collected at such a fine-grain level can be used to make local, as well as global decisions about the new frequency and voltage values of various VFIs. To this end, we present a detailed architecture based on mixed-clock/mixed-voltage first-input–first-output (FIFO) to enable dynamic scaling of frequency and voltage of various VFIs. As opposed to existing schemes that monitor only FIFO occupancy to determine scaling factors [1]–[3], 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., proportional-integral-derivative (PID) controllers [4]). This approach not only enables use of local information to calculate the new frequencies/voltages of various VFIs, but also provides flexibility to take global decisions based on queue dynamics of various FIFOs in the system.

The second part of this paper discusses multiprocessor systems that have each processor assigned to an independent VFI. We consider some typical applications like JPEG, MPEG-2 Encoder, and Software Defined Radio in our approach. Each of these applications is divided into multiple tasks with each task running on a MicroBlaze processor [5]. The frequency of each of these MicroBlaze processors can be independently controlled. By implementing such a system on an FPGA platform, we demonstrate the feasibility of our approach. We use Xilinx Virtex-II Pro device on a Xilinx University Program (XUP) board for our experiments.

The run-time dynamics of a real system is very complex and requires a detailed treatment. This paper proposes a simple DVFS algorithm that can be used along with our proposed hardware approach. Even though our algorithm can be configured for simple applications, it does not consider all the possible workload variations of real applications. Our work concentrates more on the hardware aspects of a DVFS system that can be used as a platform for implementation of various DVFS algorithms [6]. In addition, the hardware platform can also be improved to eliminate the need for significant offline analysis and run realtime applications with random bursts of data, different buffer size requirements, etc. In our proposed approach, the hardware overhead is a small fraction of the overall design and can be controlled during the design process.

Based on the number of frequency levels desired for the system, there will be a tradeoff between the total energy savings and the hardware overhead. There might be a point where the additional hardware to support more number of frequencies might actually degrade the total energy consumption. Finding this optimum point is outside the scope of this paper. The timing overhead in our algorithm is only a few hundred instructions and it is very small compared to the application. However, the proposed algorithm is a simple one and may not be suitable for all applications. Algorithms that suit certain applications can be used in our proposed hardware platform. Based on the closeness of actual energy consumption to the ideal one, a tradeoff between the speed of the algorithm and energy savings can be selected.

B. Paper Organization

The rest of this paper is organized as follows. Related work and contribution of this paper are presented in Section II. Section III discusses the problem formulation and assumptions made in this paper. In Section IV, 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 V. In Section VI, we provide the experimental results for software radio and MPEG-2 encoder benchmarks. Section VII discusses the issues related to implementing a synthesizable DVFS system using PicoBlaze processors. In Section VIII, we show how some of the applications can be implemented on an FPGA platform using MicroBlaze processors. Final conclusion and summary of our research are provided in Section IX.

II. RELATED WORK

Previous approaches based on availability of data channel in multiple clock systems (e.g., [7]), 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 [8]. Our approach changes the clock frequency to minimize the idle time spent waiting for FIFOs.

There have been several proposals to implement VFIs in modern systems such as multiple clock domain processors [1], [3]. Such architectures allow a system designer to implement local DVFS algorithms [4], 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 online algorithms are inherently nonlinear [4] 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 complicated workloads. The problem of voltage/speed selection in VFI systems has been addressed before [9] via providing an offline algorithm and a dynamic online algorithm with limited efficiency. In our approach, the benefits of DVFS are exploited at finer granularity level, while maintaining the possibility of global adaptation.

III. PRELIMINARIES AND ASSUMPTIONS

Without loss of generality, we consider the case of systems comprised of a number of synchronous cores, intellectual properties (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 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 [10]. 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 [11], [12]. Communication is implemented via a modified version of mixed-clock FIFOs [13] 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 $V = \{1, 2, \ldots, n\}$ 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 [9], [14]. As shown in [14], 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 V-C.
- 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.

IV. 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.

A. Producer-Consumer Model

In a VFI design, a mixed-clock/mixed-voltage FIFO provides a communication channel between two VFIs. One of the VFIs
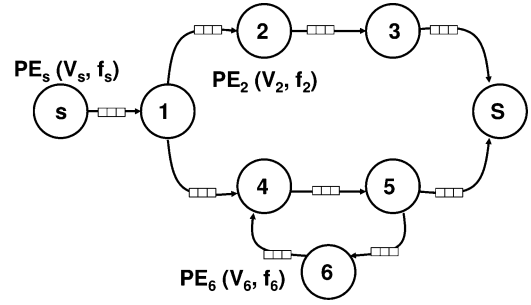

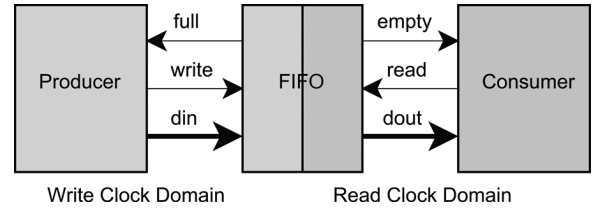Fig. 2. VFI-based component graph as in [9] with cores (PEs) characterized by local speeds/voltages.

Fig. 3. Producer consumer model. Data (din) is written into the FIFO only if the write request (write) is asserted and the FIFO is not full (full). Similarly, data ($dout$) is read from the FIFO only if the read request (read) is asserted and the FIFO is not empty (empty).

(producer) writes data into the FIFO while the other one (consumer) reads data from the FIFO [13]. For proper operation of the 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 part of the mixed-clock FIFO share a clock (producer clock) while the consumer and the other part of the mixed-clock FIFO share the other clock (consumer clock). Such a clock domain partition is shown in Fig. 3.

B. 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

$$T_c = T_p + T_w \tag{1}$$

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 frequency 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 $f_{\text{avail}} = f_p/2$ 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 $f_{\text{avail}} = f_p/2$ available frequency will not guarantee the throughput constraint. Hence, it is always necessary to have $f_{pi} < f_{\text{avail}}$. 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.

C. 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

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

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 [15].

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 α are design and technology dependent parameters [16]. 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)

$$\text{WCET}_i(V_i) = W_i * \tau_i(V_i) = \frac{(W_i * K_i * V_i)}{(V_i - V_t)^\alpha}. \quad (5)$$

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

$$\text{WCET}_i(V_i) \leq T_i \quad (6)$$

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 $\text{WCET}_i(V_i) \ll T_i$. 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, $\text{WCET}_i(V_i)$ should be as close as possible to T_i, i.e.,

$$\text{Minimize } (T_i - \text{WCET}_i(V_i)). \quad (7)$$

By taking $\text{WCET}_i(V_i)$ 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., $T_w \rightarrow 0 \Rightarrow (T_i - \text{WCET}_i(V_i)) \rightarrow 0$. 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.

V. FIFO LINK ARCHITECTURE

The derivations shown in Section IV 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* (see Section IV) 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.

A. Proposed Architecture

To implement such a logic for estimating the optimal operating frequency, we take advantage of the fact that when the producer/consumer is not operating at the ideal frequency, the FIFO will always operate near full/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/empty signals do not accurately represent the need for scaling up or down the speed/voltage of a VFI. It
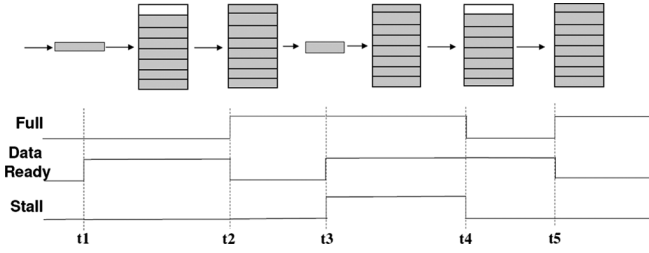
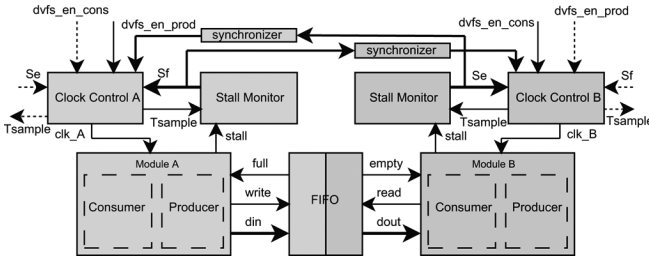Fig. 4. Comparison between full and stall signals for frequency prediction.

Fig. 5. Dynamic frequency scaling architecture.

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.

Fig. 4 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 $[t2, t4]$. However, the time period where producer is actually waiting for the FIFO to have an empty slot is $[t3, t4]$. If the *frequency step factor* is calculated based on the full signal alone, it is likely 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. Fig. 5 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 $S = 1 - |S_e - S_f|/T_{\text{sample}}$. 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

$$f_{\text{new}} = f_{\text{curr}} * S \qquad (8)$$

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

$$f_{\text{new}} = \frac{f_{\text{curr}}}{S} \qquad (9)$$

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.

B. Throughput Constraint and Scaling State

In general, throughput constrained systems require an output rate to be satisfied for correct operation. It can either be a user parameter or a system parameter. For example, in the case of the system in Fig. 2, 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 S in Fig. 2), 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 V-A. Let us consider the more common case of output rate constrained systems depicted in Fig. 6. 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 Fig. 6, 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).
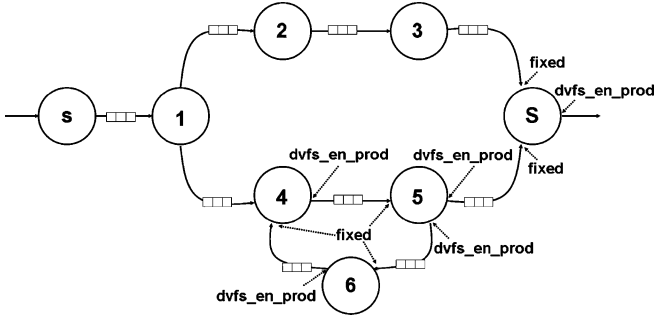
Fig. 6. VFI-based component graph with FIFO configuration.

C. 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 Fig. 5 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 [see (8) and (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. Fig. 5 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 different 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 Fig. 6, 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 (8) and (9), $f_{\text{new},5}(5, S)$ and $f_{\text{new},5}(5, 6)$ are the new potential clock speeds, the final clock speed (and associated voltage) is taken such that $f_{\text{new},5} = \max(f_{\text{new},5}(5, S), f_{\text{new},5}(5, 6))$. 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

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)

Fig. 7. Algorithm for dynamic speed/voltage selection.

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 Fig. 7.

VI. EXPERIMENTAL RESULTS

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. This program takes .tgff files [17] 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.

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 and implemented as behavioral Verilog models which were used to determine the benefits of the online voltage/frequency scaling
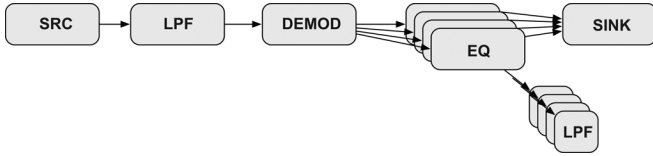
Fig. 8. Partitioned software radio.

for each module. T_{sample} was set to 5000 clock cycles for each of these benchmarks. The dynamic power is determined by a simple relative $V f^2$ comparison of various blocks. The different algorithms are compared for each block, and hence the power consumption can be compared by using only voltage and frequency without actually calculating the absolute power.

A. Software Radio

Software defined radio application can basically be partitioned into five components—namely source, low pass filter (LPF), demodulator, equalizer (EQ), and sink (see Fig. 8). 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.

A base configuration of Hitachi SH3 cores running at the clock frequency of 60 MHz and supply voltage of 3.3 V along with an offline algorithm [9] (with six levels of voltage and frequency) was used for comparison purposes. The six voltage-frequency pairs (in volts, megahertz) 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 1 kHz. As it can be seen from Fig. 9, 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, the overall improvement is still around 50% and compares well with the offline 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).

B. 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 cores running at a clock frequency of 133 MHz and supply voltage of 1.6 V was chosen (see Fig. 10). The same offline algorithm [9] was used for comparison purposes (with six voltage-frequency pairs). The six voltage-frequency pairs (in volts, megahertz) 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 3.5 f/s with 99 macroblocks per frame. Fig. 11 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
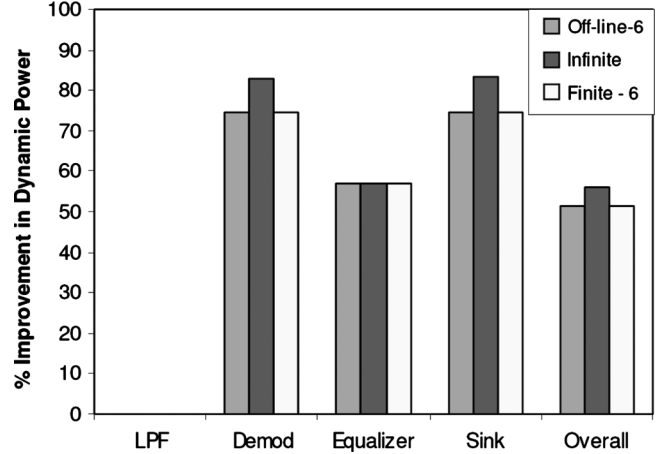

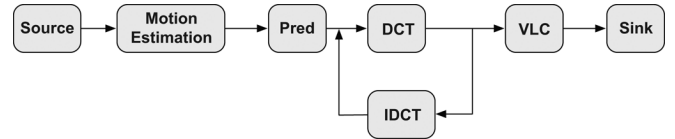Fig. 9. Dynamic power consumption in software defined radio.
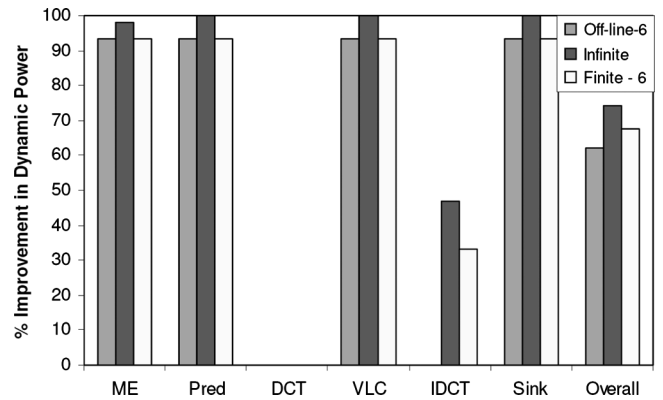

Fig. 10. Partitioned MPEG-2 encoder.

Fig. 11. Dynamic power consumption in MPEG-2 encoder.

frequency and voltage. For IDCT, our proposed method performs better than the offline method due to precise detection of workload behavior, providing additional 30%–40% power savings locally and 8% additional power savings globally. For infinite levels of voltage and frequency, the power improvement for Pred, VLC and Sink is close to 99%, even though it seems 100% in Fig. 11. The voltage and frequency values for Pred, VLC, and Sink for this case are (0.07, 6.02), (0.17, 14.36), and (0.02, 1.32), respectively. Such low values tend to give almost 100% of improvement in dynamic power. 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).

VII. VALIDATION OF SYNTHESIZABLE PRODUCER-CONSUMER SYSTEM WITH PICOBLAZE

The FIFO link architecture presented in Section V uses a behavioral model to calculate the frequencies of various VFIs.
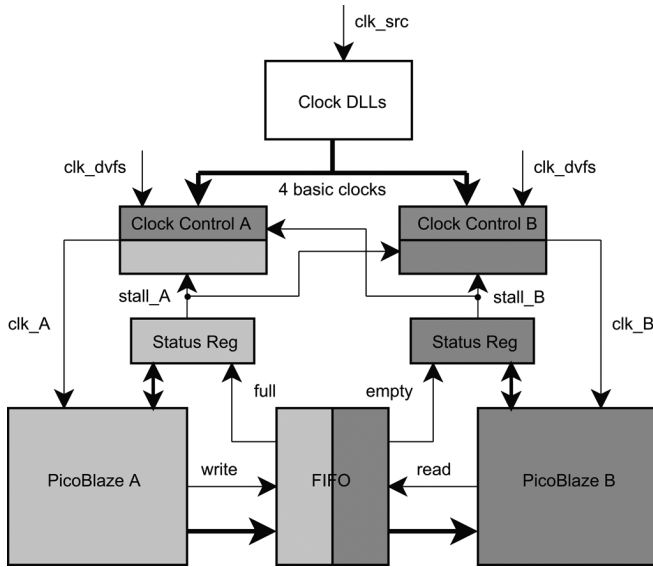
Fig. 12.   DVFS architecture with PicoBlaze processors.

Fig. 13.   Block diagram of clock control logic block.

Such a model-based approach, though useful in analyzing the performance and power consumption of system, does not consider all the issues related to synthesis of real hardware. In this section, we present an extension of previously discussed architecture and address the issues related to implementation of a *hardware-based* dynamic voltage-frequency scaling system.

A. System Architecture

Fig. 12 shows the modified version of the dynamic frequency scaling (DFS) architecture shown in Fig. 5. As can be seen in Fig. 12, we use the PicoBlaze processor [18] to implement the producer and consumer blocks. The PicoBlaze processor is an 8-bit processor based on RISC architecture. It is a very small processor with 10-bit address and is optimized for FPGA devices. Due to the simple nature of the PicoBlaze processor, the hardware to support DVFS can be easily built around it. Such small hardware requirements make it suitable for small systems, where a simple DVFS scheme is sufficient. The PicoBlaze processor is also used in the Clock Control Logic block to allow for flexibility in implementing a DVFS algorithm. This architecture is designed taking into consideration the resources available on Xilinx FPGA devices. Most of the FPGA devices in Xilinx Virtex family have *digital locked loops (DLLs)* which can be used to divide a source clock by fractional, fixed and predetermined factor. Several such DLLs and integer dividers can produce a range of frequencies for operation of various VFIs. In our design, we use three DLLs to generate four *unfriendly*[1] frequencies from a single source clock *clk_src*. These four frequencies are then passed through a chain of integer dividers (division factor of two) to produce 22 frequencies in the *clock control logic* block. A five bit configuration value (to represent 22 frequencies) is used to select one of these frequencies by *clock control logic* state machine.
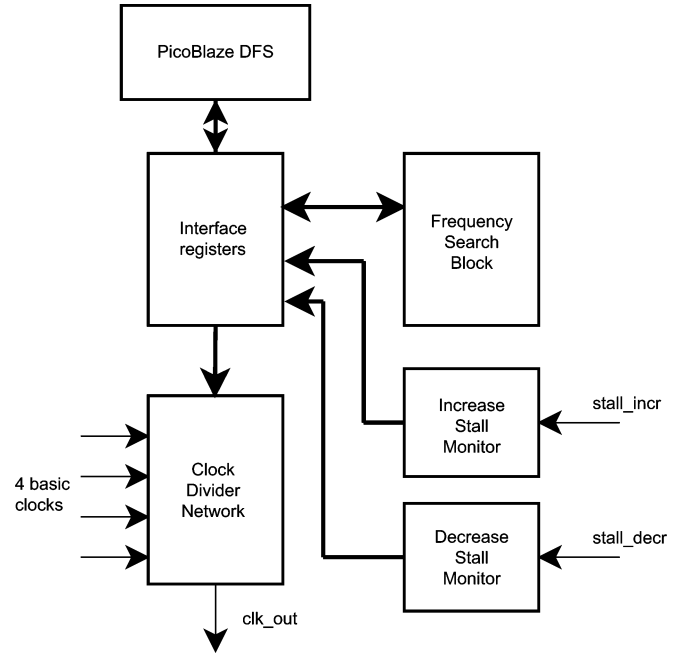
The PicoBlaze processors A and B monitor their respective *status registers* before they access the mixed-clock FIFO. If the FIFO is full, *PicoBlaze A* updates its *status register* by setting the *stall* bit high and waits for an empty slot in the FIFO. As soon as there is an empty slot available, the *stall* bit in the *status register* is cleared. A similar operation occurs in case of *PicoBlaze B* with regards to empty signal. The stall information in these *status registers* is used by Clock Control Logic blocks for calculating the new frequency.

B. Clock Control Logic Block

In Section V-C, we discussed the overall functionality of *clock control logic* block from a behavioral perspective. In this section, we discuss the architecture of this block while addressing the issues related to its implementation in hardware. The *clock control logic* block is responsible for collecting statistics about the stall information, storing the stall history, predicting the new frequency, and finally, changing the frequency of the associated VFI to the new frequency. As can be seen in Fig. 13, a PicoBlaze processor is used to implement a DFS[2] algorithm. *Interface registers* are used by PicoBlaze DFS to communicate with the other modules. Stall information from both the stall monitors is used to make predictions about the new frequency. The *decrease stall monitor* module collects statistics about the stall signal asserted by PicoBlaze processor in the same VFI as the *clock control logic* block. For example, *stall_A* is used by the *decrease stall monitor* of *Clock Control A* to gather stall information (see Fig. 12). Similarly, the *increase stall monitor* is used to collect statistics about the stall signal asserted by PicoBlaze processor in the VFI across the mixed-clock FIFO. In Fig. 12, *stall_B* is connected to *increase stall monitor* . The *clock divider network* block contains a chain

---

[1]These frequencies are not an integer multiple of each other.

[2]Since the hardware is implemented in Verilog, voltage scaling has not been taken into account. Hence, DFS and not DVFS.

NF\CF	50	33.33	25	20	16.67
50	1	0.666	0.5	0.399	0.333
33.33	0.666	1	0.75	0.599	0.5
25	0.5	0.75	1	0.799	0.666
20	0.399	0.599	0.799	1	0.833
16.67	0.333	0.5	0.666	0.833	1

Sampling Interval = 1024 clock cycles of original frequency

NF\CF	50	33.33	25	20	16.67
50	1024	682	512	409	341
33.33	682	1024	768	614	512
25	512	768	1024	819	682
20	409	614	819	1024	853
16.67	341	512	682	853	1024

☐ - Frequency Decrease

☐ - Frequency Increase

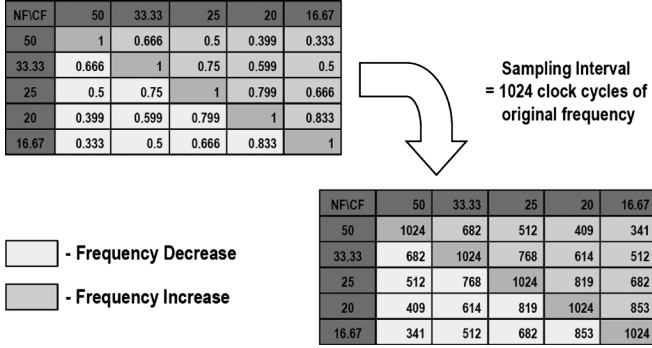Fig. 14. Frequency matrix.

Fig. 15. Stall behavior and frequency change waveforms.

of integer dividers. It uses input from *interface registers* to set the current frequency of the associated VFI.

Based on the information from the stall monitors, the DFS algorithm (implemented on *PicoBlaze DFS*) predicts the ratio between the new frequency and the current frequency. This ratio is used to search the new frequency from the set of available frequencies in the design. The list of all the available frequencies, along with the ratio between any two frequency values, is stored in a ROM in the form of *frequency matrix*. The format of the *frequency matrix* is shown in Fig. 14. The ratios between the frequencies are scaled by a factor of 1024 to enable ease of search when the sampling interval T_{sample} (see Fig. 5) is 1024. However, this factor can be chosen based on the number of available frequencies and preciseness of values required for a given application workload. A higher number of bits to represent these ratios would result in more accurate prediction of the new frequency when the requested ratio is close to the stored value. In Fig. 14, the top row represents the current frequency values, while the left-most column represents the new frequency values. Based on the direction of change (increase or decrease of frequency) desired, the appropriate section of the column (partitioned by value of 1024) associated with the current frequency is searched. If a frequency decrease is desired, the new frequency corresponding to the lowest value in the current frequency column, but higher than the requested value is selected. In this case, the search is limited to lower part of a column. For example, if the current frequency is 50 MHz and the requested value is 500, frequency value of 25 MHz is returned as it corresponds to a value of 512 in the 50 MHz column, which is lowest possible value that is higher than 500. Similar operation occurs for frequency decrease, but the search is limited to upper half of the current frequency column. The new frequency value returned by *frequency search* block is used by *PicoBlaze DFS* to set the new frequency value through the *interface registers*. The DVFS algorithm that we implemented using PicoBlaze processor takes about 250–300 instructions.

C. Experimental Results

To demonstrate the change in frequency and the behavior of stall signals before and after the frequency change, we considered a simple system composed of one producer and one consumer, similar to the one in Fig. 12. We created a test scenario, in which the time interval between two consecutive write operations by the producer is less than the time interval between two
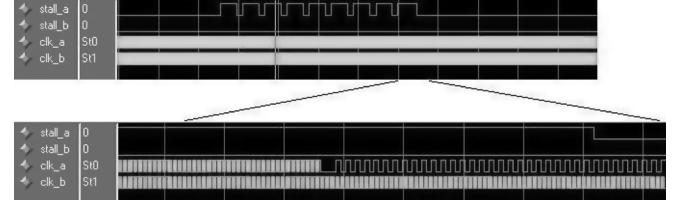
consecutive read operations by the consumer. Fig. 15 shows relevant signals, *stall_a*, *stall_b*, *clk_a*, and *clk_b*. This results in the FIFO being operated near full condition, and hence resulting in signal *stall_a* being asserted as shown in Fig. 15. To reduce the amount of stall in the producer, the DFS algorithm changes the frequency of the producer to a lower value. The change in frequency of clock *clk_a* is also shown in Fig. 15. After the frequency change, the amount of stall in the producer is reduced (to zero in this case).

VIII. MICROBLAZE-BASED SYSTEM VALIDATION USING FPGA PLATFORM

Even though the PicoBlaze processor provides the flexibility to change the DFS algorithm and FIFO access patterns of producers and consumers, the 8-bit data width and the number of instructions possible using 10-bit address limit the range of applications that can be implemented in such a system. Most modern applications use 32-bit data width with several megabytes of program memory. To enable exploration of these applications, we designed an architecture where each of the PicoBlaze processor is replaced by a MicroBlaze processor. Each of the MicroBlaze processor in such a system operates on an independent clock frequency. Xilinx Embedded Development Kit (EDK) [19] greatly simplifies the design of such systems with graphical interface that eliminates the need to write extensive code in a hardware description language. Virtex-II Pro FPGA device on Xilinx University Program board is used to implement and test all of our designs.

A. Fast Simplex Link Bus

Since all the MicroBlaze processors can potentially operate on different clock frequencies, a mechanism to enable asynchronous communication between these processors is necessary. For this purpose, we use Fast Simplex Link bus [20] as a communication medium between any two MicroBlaze processors. This "bus" consists of a mixed-clock FIFO with write and read operations occurring at different clock frequencies. The MicroBlaze processor has built-in logic to interface with this type of FIFO. Fig. 16 shows the signals associated with a *Fast Simplex Link* bus. The signals related to write operations are called *master* signals, while those associated with read operations are called *slave* signals.

B. Frequency Generation

Since all the MicroBlaze processors can potentially run on different clock frequencies, each processor requires an independent clock source capable of generating frequencies in a sufficiently large range of frequency values. *Digital Clock Manager*
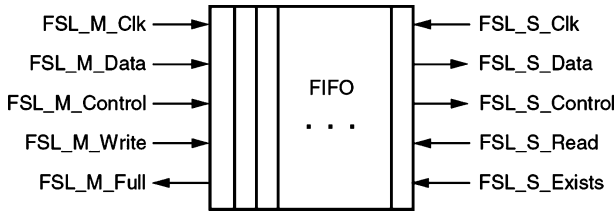
Fig. 16.   Fast simplex link bus.

Fig. 17.   System architecture using MicroBlaze processors.

(DCM) in Virtex devices is very well suited for such a purpose. Each of the eight DCMs in Virtex-II Pro device is capable of generating 13 frequencies from a clock source of 100 MHz. The various frequency values (in megahertz) that can be generated by a DCM are as follows: 100, 66.66, 50, 33.33, 28.57, 25, 22.22, 20, 18.18, 16.66, 15.38, 14.28, 13.33. These frequency values provide sufficient flexibility to experiment with workload behaviors of several applications. The major drawback of a DCM is that the generated frequency can only be statically assigned during design process and does not allow to dynamically change a frequency depending on application workload. However, as discussed in Section VII, a network of DCMs and clock dividers can be created to enable online configuration of frequency values. Our MicroBlaze-based design does not build such a network, even though it exists in PicoBlaze-based design (see Section VII).

C. System Architecture

The MicroBlaze processor uses an open peripheral bus (OBP) to connect to various peripheral devices. One such peripheral device, *Universal Asynchronous Receiver Transmitter (UART)*, can be used by MicroBlaze processor to communicate run-time information to user. It also helps in system debugging by enabling printing of statements on a terminal running on a computer. We take advantage of this feature while designing our system. Fig. 17 shows the system architecture based on MicroBlaze processor. It consists of a main processor that is used to regulate the data flow in the system. An application is represented by a task graph consisting of various tasks, each of which runs on an independent processor. The main processor generates data tokens and sends them to the source (e.g., M1) of the task graph. The data tokens travel through the task graph and reach the sink (e.g., M3). The main processor collects these data tokens and measures the performance of the system which can be represented by latency and throughput. The latency in the system is obtained by measuring the time required by a data token to traverse the task graph and reach back to main processor. On the other hand, throughput is measured by sending several data tokens into the task graph within a very short interval and then measuring the time interval between arrival of any two data tokens. The measured values of latency and throughput are reported to the user by the main processor through UART interface. The Fast Simplex Link bus allows for transfer of data as well as control information. The control flags in the link (*FSL_M_Control* and *FSL_S_Control* in Fig. 16) help to identify the control information. This can be used to send the stall
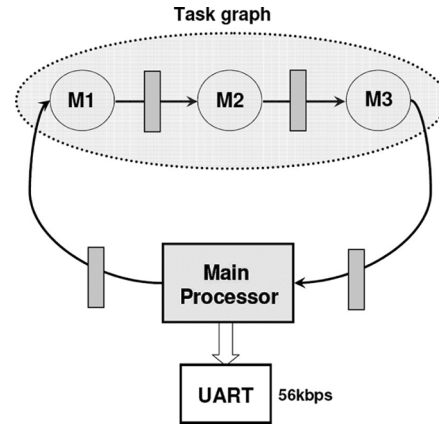
numbers to different processors as well as to the peripheral devices.

D. Experimental Results

To test our proposed architecture and to demonstrate the usefulness of our method, we used JPEG, MPEG-2 Encoder and Software Defined Radio as test applications. The task graph representation of these applications was implemented using a MicroBlaze processor for each task. In our experiments, software models based on the number of clock cycles required for execution of each task in the task graph of these applications is used. For JPEG application, the cycle count is based on IBM PowerPC 405 GP, while the cycle counts for MPEG-2 and software defined radio are same as in Section VI. The latency for each of these applications was calculated as an arithmetic mean of latencies for 20 data tokens. Similarly, throughput was calculated as an arithmetic mean of the time intervals between the arrival of any two consecutive data tokens for 20 data tokens sent by the main processor. A point to be noted here is that throughput is represented as the time interval between two consecutive data tokens, and not as a rate. Our experiments consisted of the following two parts.

- In the first part, all MicroBlaze processors, except the main processor, run at the maximum frequency possible (i.e., 66 MHz) when their respective DCMs are configured in *divider* mode. The main processor, however, runs at a frequency of 100 MHz. The higher frequency of the main processor is required for good accuracy of latency and throughput measurements. In this configuration of the system, latency and throughput of the application are measured. From the information about the number of clock cycles required by each task, we calculate the optimum frequency for each MicroBlaze processor using the principles explained in Section IV. Based on the list of the available frequencies, these frequency values are rounded up to nearest available frequency values.
- In the second part of the experiment, we change the clock frequencies as per the calculated values and rerun the application. The latency and throughput values are measured again and compared with the initial values. The latency values are expected to increase, but the throughput values
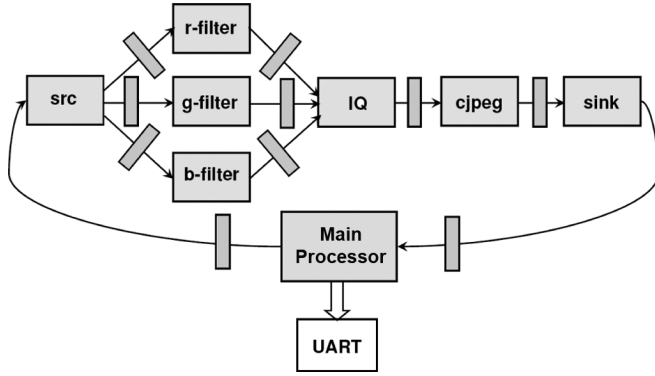
Fig. 18. Implementation of JPEG application.

TABLE I
CYCLES/PACKET FOR SOFTWARE DEFINED RADIO

LPF	Demod	Equalizer(10)	Sink
61494	33086	463193	32736

TABLE II
CYCLES/MACROBLOCK FOR MPEG-2 ENCODER

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

TABLE III
THROUGHPUT AND LATENCY MEASUREMENTS FOR JPEG APPLICATION

Task	No. of Cycles	Ini. Freq.	Ideal Freq.	Final Freq.
src	2600	66.66	0.04	13.33
r-filter	399000	66.66	6.24	13.33
g-filter	399000	66.66	6.24	13.33
b-filter	399000	66.66	6.24	13.33
iq	425600	66.66	6.66	13.33
cjpeg	4256000	66.66	66.66	66.66
sink	2600	66.66	0.04	13.33
Latency		6537905		10801520
Throughput		5472000		5472000

TABLE IV
THROUGHPUT AND LATENCY MEASUREMENTS FOR MPEG-2 ENCODER

Task	No. of Cycles	Ini. Freq.	Ideal Freq.	Final Freq.
src	3188	66.66	0.57	13.33
me	101282	66.66	18.24	20
pred	16722	66.66	3.01	13.33
dct	370060	66.66	66.66	66.66
idct	351259	66.66	63.27	66.66
vlc	43222	66.66	7.78	13.33
sink	3188	66.66	0.57	13.33
Latency		807220		1456837
Throughput		945327		945327

TABLE V
THROUGHPUT AND LATENCY MEASUREMENTS FOR SOFTWARE-DEFINED RADIO

Task	No. of Cycles	Ini. Freq.	Ideal Freq.	Final Freq.
src	32736	66.66	32.22	33.33
lpf	67494	66.66	66.66	66.66
demod	33086	66.66	32.67	33.33
eq	46319	66.66	45.74	50
sink	32736	66.66	32.33	33.33
Latency		259023		391564
Throughput		86778		86778

10, respectively. Tables IV and V show the results for these two benchmarks. Similar to JPEG application, the decrease in frequency of various processors executing certain tasks does not affect the throughput of the application. A decrease in frequency of these processors implies a potential decrease in the voltage of the associated VFIs, both of which can result in significant power savings. The final frequencies for Software-Defined Radio and MPEG-2 Encoder benchmarks match the frequencies obtained from the behavioral model explained in Section V.

IX. 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 DVFS 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. To enable design of a real DVFS system, we addressed some of the issues related to synthesis and clock control using PicoBlaze-based architecture. Our MicroBlaze-based design for FPGA platform further demonstrates the feasibility of implementing real applications using VFI-based DVFS schemes.

are expected to remain unchanged. The time required by an addition operation and a conditional branch executed on a MicroBlaze processor running at a frequency of 100 MHz is used as the unit of measurement in our experiments.

From E3S benchmarks [17], we observe that a JPEG application can be divided into seven tasks, namely *src*, *r-filter*, *g-filter*, *b-filter*, *iq* (inverse quantization), *cjpeg* (jpeg compression), and *sink*. The task graph representation of JPEG, implemented as a part of our proposed architecture, is shown in Fig. 18. After running this application on the MicroBlaze platform, we measured latency and throughput values. Table III shows the initial frequency of operation, ideal frequency based on our algorithm and the final frequency for each task. Since task *cjpeg* requires maximum number of clock cycles, it limits the throughput of the system. Therefore, the frequency of processor running task *cjpeg* remains unchanged at highest possible value. We can see from the results that, even though the latency of the system increases as a result of decreasing the frequencies, the throughput of the system remains unchanged.

Similar experiments were carried out for software-defined radio and MPEG-2 Encoder benchmarks. The task graph representations of these two applications are shown in Figs. 8 and

REFERENCES

[1] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proc. Int. Symp. High Perform. Comput. Arch. (HPCA)*, Feb. 2002, p. 29.

[2] A. Iyer and D. Marculescu, "Power efficiency of multiple clock, multiple voltage cores," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, San Jose, CA, Nov. 2002, pp. 379–386.

[3] E. Talpes and D. Marculescu, "A critical analysis of application-adaptive multiple clock processors," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Des. (ISLPED)*, Seoul, Korea, Aug. 2003, pp. 278–281.

[4] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," in *Proc. Int. Conf. Arch. Support Program. Lang. Operat. Syst.*, 2004, pp. 248–259.

[5] Xilinx, San Jose, CA, "Microblaze Processor," 2007. [Online]. Available: http://www.xilinx.com/ipcenter/processor_central/microblaze/architecture.htm

[6] A. Maxiaguine, S. Chakraborty, and L. Thiele, "DVS for buffer-constrained architectures with predictable qos-energy tradeoffs," in *Proc. 3rd IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codes. Syst. Synth. (CODES + ISSS)*, 2005, pp. 111–116.

[7] A. Agiwal and M. Singh, "An architecture and wrapper synthesis for multi-clock latency-insensitive systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, Nov. 2005, pp. 1006–1013.

[8] R. Rao, S. Vrudhula, and N. Chang, "Battery optimization vs. energy optimization: Which to choose and when," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, Nov. 2005, pp. 439–445.

[9] K. Niyogi and D. Marculescu, "Speed and voltage selection for gals systems based on voltage/frequency islands," in *Proc. ACM/IEEE Asian-South Pac. Des. Autom. Conf. (ASPDAC)*, Jan. 2005, pp. 292–297.

[10] IBM, Armonk, NY, "IBM Blue Logic CU-08 Voltage Islands," 2007. [Online]. Available: http://www.ibm.com/chips/products/asics/products/v_island.html

[11] L. Nielson, C. Niessen, J. Sparso, and K. Berkel, "Low-power operation using self timed circuits and adaptive scaling of the supply voltage," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 4, pp. 391–397, Dec. 1994.

[12] J. Muttersbach, T. Villiger, and W. Fichtner, "Practical design of globally asynchronous locally synchronous systems," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, Apr. 2000, p. 52.

[13] T. Chelcea and S. Nowick, "A low latency fifo for mixed-clock systems," in *Proc. IEEE Comput. Soc. Workshop VLSI*, Apr. 2000, p. 119.

[14] A. Dasdan, "Rate analysis of embedded systems," Ph.D. dissertation, Dept. Comput. Sci., Univ. Illinois at Urbana Champagne, Urbana Champagne, 1998.

[15] J. Butts and G. Sohi, "A static power model for architects," in *Proc. Int. Symp. Microarch.*, Dec. 2000, pp. 191–201.

[16] C. Hu, *Devices and Technology Impact on Low Power Electronics, Low Power Design Methodolgies*. Norwell, MA: Kluwer, 1996.

[17] Northwestern University, Evanston, IL, "Embedded systems synthesis benchmarks suite (e3s)," 2007. [Online]. Available: http://www.ece.northwestern.edu/~dickrp/e3s/

[18] Xilinx, San Jose, CA, "Picoblaze Processor," 2007. [Online]. Available: http://www.xilinx.com/bvdocs/userguides/ug129.pdf

[19] Xilinx, San Jose, CA, "Platform studio documentation," 2007. [Online]. Available: http://www.xilinx.com/ise/embedded/edk_docs.htm

[20] Xilinx, San Jose, CA, "Fast simplex link bus," 2007. [Online]. Available: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/FSL_V20.pdf

Puru Choudhary (S'05) received the B.Tech. (Hons) degree in instrumentation engineering from the Indian Institute of Technology, Kharagpur, India, in 2002, and the M.S. degree in electrical and computer engineering from the Carnegie Mellon University, Pittsburgh, PA, in 2007.

He is currently working as a Senior Design Engineer with Marvell Semiconductor, Inc., Santa Clara, CA.

Diana Marculescu (S'94–M'98) received the M.S. degree in computer science from University Politehnica of Bucharest, Bucharest, Romania, in 1991, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1998.

She is currently an Associate Professor with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA. Her research interests include energy-aware computing, CAD tools for low-power systems, and emerging technologies (such as electronic textiles or ambient intelligent systems).

Dr. Marculescu was the recipient of a National Science Foundation Faculty Career Award (2000–2004), an ACM-SIGDA Technical Leadership Award (2003), and of the Carnegie Institute of Technology George Tallman Ladd Research Award (2004). She is an IEEE Circuits and Systems Society Distinguished Lecturer (2004–2005) and a member of Executive Board of the ACM Special Interest Group on Design Automation (SIGDA).