

System-wide Energy Optimization for Multiple DVS Components and Real-time Tasks

Heechul Yun, Po-Liang Wu, Anshu Arya, Cheolgi Kim, Tarek Abdelzaher and Lui Sha

Department of Computer Science,

University of Illinois at Urbana-Champaign, Champaign, IL 61801

{heechul,wu87,arya3,cheolgi,zaher,lrs}@illinois.edu

Most dynamic voltage and frequency scaling (DVS) techniques adjust only CPU parameters; however, recent embedded systems provide multiple adjustable clocks which can be independently tuned. When considering multiple components, energy optimal frequencies depend on task set characteristics such as the number of CPU and memory access cycles. In this work, we propose a realistic energy model considering multiple components with individually adjustable frequencies such as CPU, system bus and memory, and related task set characteristics. The model is validated on a real platform and shows less than 2% relative error compared to measured values. Based on the proposed energy model, we present an optimal static frequency assignment scheme for multiple DVS components to schedule a set of periodic real-time tasks. We simulate the energy gain of the proposed scheme compared to other DVS schemes for various task and system configurations, showing up to a 20% energy reduction. We also experimentally verified the result on a real hardware platform.

Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; C.0 [**Computer Systems Organization**]: General—*Modeling of computer architecture*

General Terms: Energy Model, Embedded System, Real-time Scheduling

1. INTRODUCTION

Dynamic frequency and voltage scaling (DVS) schemes are common for reducing energy consumption, and many devices support multiple frequency/voltage levels. However, most DVS schemes only adjust CPU frequency and voltage and do not consider the energy consumption of the bus and memory. Previous studies show that bus and memory also significantly contribute to the total energy consumption [Shim et al. 2004]. Recent hardware allows these components to have their own clocks and DVS capabilities that can be tuned independently of the CPU frequency. Therefore, new DVS schemes must consider CPU, bus, and memory frequency to reduce the system-wide energy consumption.

Table I shows how a multiple component DVS (multi-DVS) scheme can save energy

Author's address: L. Lamport, System Research Center, Digital Equipment Corporation, 130 Lytton Ave., Palo Alto, CA 94301.

Start of a second footnote ...

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 1529-3785/20YY/0700-0001 \$5.00

Table I. Effect of task characteristics in energy saving measured on a real hardware platform.

Task	CPU (MHz)	Mem (MHz)	Time (s)	Energy (mJ)	Energy savings
<i>dhrystone</i>	200	100	4.26	2364	–
	200	50	4.28	2106	10%
<i>memxfer5b</i>	200	100	3.46	1690	–
	100	100	3.55	1182	30%

with a small performance penalty. We measured the energy consumption and execution time of two tasks — *dhrystone*, a CPU intensive task, and *memxfer5b*, a memory intensive task on an ARM926-ejs processor. For *dhrystone*, reducing memory frequency to half of the maximum increases the execution time by only 0.5% but reduces energy consumption by 10%. Lowering the CPU frequency to half of the maximum for *memxfer5b* results in a 2.6% increase in execution time but achieves a 30% energy reduction. The results show that a multi-DVS scheme can effectively reduce energy consumption.

In this paper, we propose a multi-DVS energy model that considers the energy consumption of CPU, bus, and memory, and considers task set characteristics such as the number of CPU and memory access cycles. We validate the model with a series of experiments on an ARM based embedded system and show that it captures real system energy consumption. Based on our energy model, we present a systematic method for assigning multiple DVS frequencies for a set of periodic real-time tasks given system and schedulability constraints. We simulate various task sets and compare several DVS schemes by exploring the effects of task and system parameters, such as idle power consumption.

In summary, we make the following contributions:

1. We propose a realistic multi-DVS energy model that considers CPU, system bus, memory, and task set characteristics, at multiple frequency settings and validate it on a real hardware platform.
2. Based on the proposed model, we present a multiple component DVS scheme for energy optimal scheduling of periodic real-time tasks.

The remainder of this paper is organized as follows: Section 2 presents the energy model and its validation on a real hardware platform. Section 4 defines and solves the frequency assignment problem for a set of real-time tasks. Section 5 compares the proposed multi-DVS scheme to other DVS schemes, and Section 6 discuss practical issues. Section 7 compares this study to related work. Section 8 conclude the paper.

2. ENERGY MODEL

Most recent ARM based systems are capable of independently tuning CPU, system bus and memory frequencies [Freescale 2008; Samsung b]. Our model incorporates independent frequency assignment and is validated on a real hardware platform. This section describes the multi-DVS energy model that considers energy consumption on a platform with multiple independently adjustable component clocks. Table II presents a summary of notation used throughout the paper.

We propose an energy model to reflect the actual characteristics of recent embedded platforms by focusing on three components: CPU, system bus, and main memory. These components are tightly integrated and contribute significantly to the total energy consump-

Table II. Summary of notation.

E	total energy consumption (mJ)
E_{comp}	pure execution block energy consumption(mJ)
E_{mem}	cache stall block energy consumption (mJ)
E_{idle}	idle block energy consumption (mJ)
e	execution time of a given task (s)
P	period(=deadline) of a given task (s)
C	CPU cycles of a given task (10^6 cycles)
M	memory cycles of a given task (10^6 cycles)
r	cache stall ratio
f_c	CPU clock (MHz)
f_b	system bus clock (MHz)
f_m	memory clock (MHz)
V_{cpu}	CPU voltage (V)
V_{bus}	system bus voltage (V)
V_{mem}	memory voltage (V)
I	idle time dynamic power consumption of CPU, bus, and memory (mW)
R	static power consumption of the system (mW)
K_{ca}	capacitance constant for active CPU (nF)
K_{cs}	capacitance constant for standby(on but idle) CPU (nF)
K_{ba}	capacitance constant for active system bus (nF)
K_{bs}	capacitance constant for standby system bus (nF)
K_{ma}	capacitance constant for active memory (nF)
K_{ms}	capacitance constant for standby memory (nF)

tion as shown in Table I. We also incorporate task set characteristics, specifically the number of CPU and memory access cycles, into the energy model.

Fig. 1 illustrates our energy model for a single task. In the model, task execution time is split into three blocks: (1) pure execution, (2) cache stall, and (3) idle. In the pure execution block, the CPU core executes instructions while the system bus and main memory are in standby. In the cache stall block, the cache fetches data from memory through the system bus while the CPU core is in standby, waiting for the data to become available in its cache. After the task finishes, all three components – CPU, system bus, and memory – are in an idle state. While actual cache stall periods are scattered throughout the entire execution, we aggregate them into a single block. This is valid since most embedded processors execute in-order and there is no overlap between execution and off-chip memory fetch operations. For out-of-order processors, there is an overlap period, but it is relatively small because off-chip memory access takes much longer than executing out-of-order instructions.

Hence, the task execution time can be expressed as

$$e = \frac{C}{f_c} + \frac{M}{f_m}, \quad (1)$$

where C is the number of CPU cycles needed to complete the task and M is the number of memory cycles for off-chip memory access during cache stall handling; f_c is CPU clock frequency and f_m is main memory frequency. Using Eq. (1), we are able to predict the execution time of a task for a specific f_c and f_m . The first term, $\frac{C}{f_c}$, is the pure execution time (the first block in Fig. 1), and $\frac{M}{f_m}$ is the cache stall time (second block in Fig. 1). Idle time is the period minus the execution time: $P - e$. We define cache stall ratio as

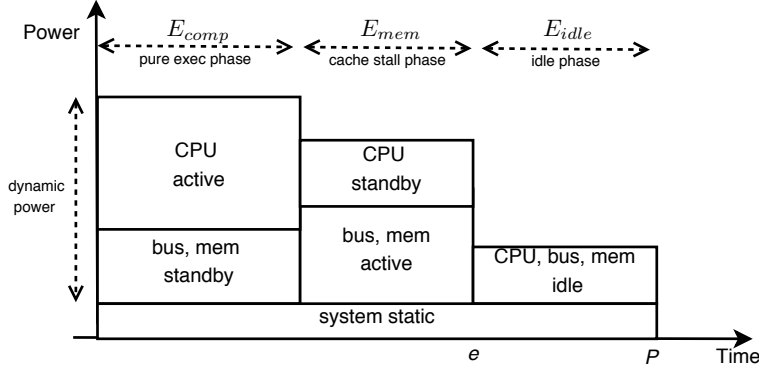


Fig. 1. Energy model for a single task with deadline (e : task finish time, P : deadline).

$$r = \frac{M}{C + M}, \quad (2)$$

describing the CPU or memory intensiveness of a task.

The total energy consumption shown in Fig. 1 is expressed by

$$E = E_{comp} + E_{mem} + E_{idle}, \quad (3)$$

where E_{comp} is the system-wide energy consumption during the pure execution block, E_{mem} is the consumption during the cache stall block and E_{idle} is the consumption during the idle block. The total power consumption at any given time can be expressed as the sum of each component's power consumption. The power consumption of each component can be described using a well known power equation, $K \cdot V^N \cdot f + R$, where K is half of the average capacitance, V is voltage, f is frequency of the component, N is the voltage exponent, and R is static leakage power [Brooks 2000]. Note that K is unique to each mode of operation. For example, during idle and cache stall time, the CPU consumes less power than when it is actively executing instructions, although its operating frequency and voltage may remain the same. The value of K for each operation mode may vary greatly for different processors. To generalize the model, we consider three modes – active, standby, and idle – and use multiple K values in our base equation. Other components – LCD, flash, etc. – consume static power regardless of the voltages and frequencies of the CPU, bus and memory. We do not consider a dynamic on/off strategy for those devices. In view of the above, we can write

$$E_{comp} = (K_{ca} \cdot V_{cpu}^{N_1} \cdot f_c + K_{bs} \cdot V_{bus}^{N_2} \cdot f_b + K_{ms} \cdot V_{mem}^{N_3} \cdot f_m + R) \cdot \frac{C}{f_c}. \quad (4)$$

Eq. (4) shows the energy consumption for the pure computation block. In this block, the CPU is actively executing instructions while the system bus and memory are in standby. K_{ca} is the capacitance constant for the active CPU mode. K_{bs} and K_{ms} are standby capacitance constants for system bus and memory, respectively. R represents the static power consumption of the entire system. Similarly,

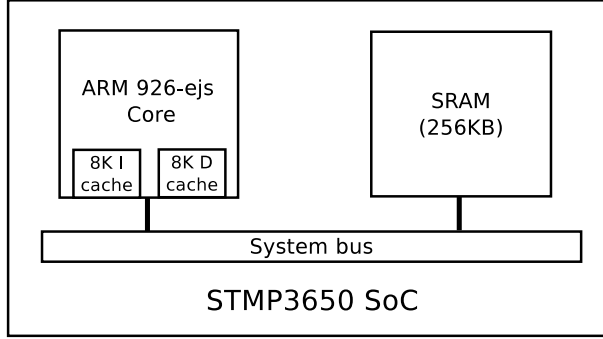


Fig. 2. Tested hardware platform. The CPU, system bus, and SRAM share a common voltage. The SRAM operates at system bus frequency.

$$E_{mem} = (K_{cs} \cdot V_{cpu}^{N_1} \cdot f_c + K_{ba} \cdot V_{bus}^{N_2} \cdot f_b + K_{ma} \cdot V_{mem}^{N_3} \cdot f_m + R) \cdot \frac{M}{f_m}. \quad (5)$$

Eq. (5) is the energy consumption during the cache stall block in which the CPU stalls the execution and waits until the data becomes available in its cache. The CPU consumes less power when it is waiting for the cache data due to clock gating technology [TI], so we introduce a constant factor, K_{cs} , for CPU standby mode. Both system bus and memory are active in this phase. K_{ba} and K_{ma} denote capacitance constants of active mode bus and memory, respectively. Finally,

$$E_{idle} = (I + R) \cdot (P - e). \quad (6)$$

Eq. (6) is the energy consumption during idle mode. Many recent embedded processors support a special idle mode which significantly reduces power consumption [Dhiman et al. 2008], so we use a separate term, I , to represent the idle mode power consumption of the CPU, system bus, and memory.

3. MODEL VALIDATION

In this section, we present validation results that demonstrate the accuracy of the model (Section 2) in predicting the energy consumption of an embedded hardware platform with an ARM926-ejs based processor [Freescale]. On the same platform, we verified two different memory configurations – internal SRAM (Section 3.1) and external DRAM (Section 5.2.2) – to show the application of the model to different devices.

3.1 SRAM Configuration

Fig. 2 shows the block diagram of the tested STMP3650 SoC. The SoC includes an ARM CPU core, L1 cache, system bus, and internal SRAM in a single package. The 256 KB internal SRAM is connected to the system bus.

While the proposed model is applicable to many embedded systems, there are several simplifications on our test platform; CPU, system bus, and internal SRAM all share the same power source ($V_{cpu} = V_{bus} = V_{mem}$) and system bus and memory operate at a same frequency ($f_b = f_m$). The resulting energy equation for our hardware platform is,

Table III. Processor specifications.

CPU clock	20 - 200 MHz (2 MHz step)
Bus clock	20 - 100 MHz (f_c/n)
Voltage	1.504 - 1.824 V (0.32 V step)
L1 cache	8 KB I, 8 KB D

$$\begin{aligned}
E = & (K_{ca} \cdot V^N \cdot f_c + K_{ms}^* \cdot V^N \cdot f_m + R) \cdot \frac{C}{f_c} \\
& + (K_{cs} \cdot V^N \cdot f_c + K_{ma}^* \cdot V^N \cdot f_m + R) \cdot \frac{M}{f_m} \\
& + (I + R) \cdot (P - e).
\end{aligned} \tag{7}$$

Because system bus and memory share a common voltage and frequency, we combine the terms for these components in Eq. (4) and (5), and use combined capacitance constants K_{ms}^* and K_{ma}^* to denote standby and active mode, respectively. These restrictions are platform and architecture specific. For example, [Intel 2004] does not share a common voltage between the CPU and system bus. Table III shows the basic specifications of the tested processor. CPU frequency, f_c , is adjustable from 20 MHz to 200 MHz, the system bus and memory clock is divided from the CPU clock, $f_b = f_m = f_c/n$ where n is an integer, and voltage can be adjusted with 0.32 V steps from 1.504 V to 1.824 V. We set the voltage proportional to the CPU clock based on the recommendation of the processor data sheet, $V = a f_c + b$. For this system $a = 0.0016$ V/MHz and $b = 1.504$ V.

In our experiments, energy consumption was measured for four synthetic tasks with different cache stall ratios, 0%, 10%, 25%, and 55%. We measured the entire board level power consumption which measures the power between the external supply and the input to the board. Our energy model requires the number of CPU cycles, C , and memory cycles, M to be known for a task. Note that M is the number of memory cycles to handle cache-misses; cache hit memory references are not included in M . In many recent processors, the cache stall cycles can be obtained by using a performance counter [Erastian]; however, the tested processor did not have a counter. Therefore, we devised a program with two loops – a loop with 100% cache misses and another with 100% cache hits – and measured their execution time. To construct a loop with 100% cache misses, we allocated an array twice the size of the cache and sequentially read words separated by one cache-line size. These reads always resulted in cache misses. By subtracting the execution time of the 100% cache-hit loop from the 100% cache-miss loop, we obtained cache stall time.

By varying the number of loop iterations, we synthesized tasks with different C and M values. The instruction code of the synthetic tasks fit into the 8 KB I-cache to avoid generating additional instruction fetch cache stalls. For each task, eight different frequency/voltage settings were tested, and for each setting we measured energy consumption and execution time. We set the voltage exponent, $N = 2$, and performed nonlinear least squares analysis on the collected data to determine the value of each parameter in Eq. (7).

Fig. 3 plots the measured energy consumption and the model predicted energy consumption. The R^2 (the coefficient of determination that estimates the validity of a model) value is 99.97% (where 100% is a perfect fit) and the maximum relative error is less than 2%, suggesting that our energy model accurately captures system behavior. Table IV shows the

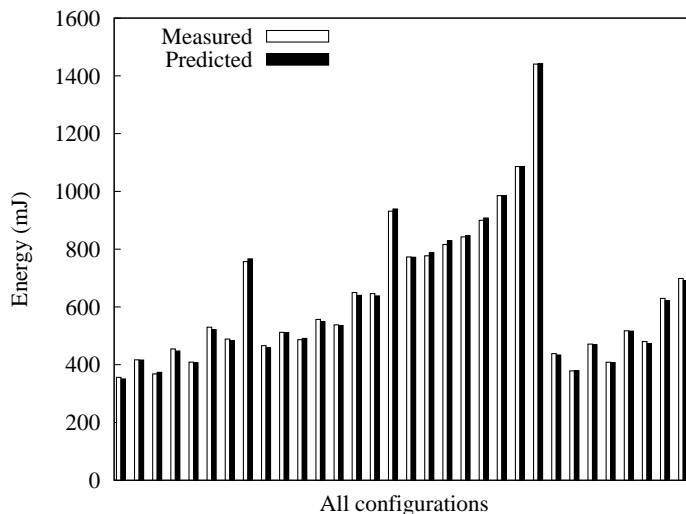


Fig. 3. Energy model fitting. Comparison of measured and model predicted energy values for 32 configurations with varying cache stall ratio and clock settings. R^2 is 99.97%.

Table IV. Model parameters for the tested hardware.

Capacitance (nF)				Power (mW)	
K_{ca}	K_{cs}	K_{ma}^*	K_{ms}^*	I	R
0.51	0.22	0.54	0.21	6.57	67.43

value of each parameter in Eq. (7) for the tested hardware.

3.2 DRAM Configuration

We changed the hardware configuration of Section 3.1 to use external DRAM. Figure 4 shows the modified system. DRAM is connected to on-chip DRAM controller, which is attached to system bus. One key difference from SRAM is that DRAM does not share a common voltage with the processor core, resulting in different system energy equation.

In our platform, the DRAM [Samsung a] uses a fixed 3.0V and operates at a multiple of the system bus frequency. CPU and system bus use the same variable voltage, from 1.504V to 1.824V, as the previous section.

The energy equation of this hardware configuration is

$$\begin{aligned}
 E = & (K_{ca} \cdot V^N \cdot f_c + K_{ms}^* \cdot (V^N \cdot + 3.0^N) \cdot f_m + R) \cdot \frac{C}{f_c} \\
 & + (K_{cs} \cdot V^N \cdot f_c + K_{ma}^* \cdot (V^N \cdot + 3.0^N) \cdot f_m + R) \cdot \frac{M}{f_m} \\
 & + (I + R) \cdot (P - e).
 \end{aligned} \tag{8}$$

Because system bus and DRAM do not share a common voltage and frequency, we used separate capacitance constants K_{ba} and K_{bs} for system bus, and K_{ma} and K_{ms} for DRAM.

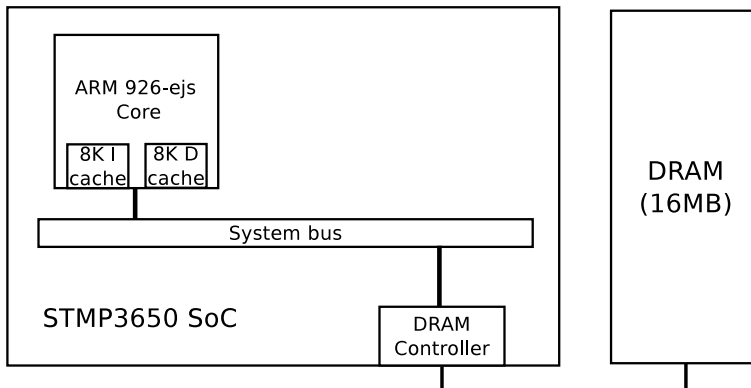


Fig. 4. Tested hardware platform with external DRAM.

Table V. Model parameters for the system with a external DRAM

Capacitance (nF)				Power (mW)	
K_{ca}	K_{cs}	K_{ma}^*	K_{ms}^*	I	R
0.52	0.30	0.18	0.05	6.52	71.18

We set $V = 0.0016f_c + 1.504$ and use $N = 2$, as in the previous section.

We measured energy consumption of four synthetic task configurations with varying cache stall ratios – 0%, 10%, 25%, and 55% – similar to the experiments in Section 3.1. However, all cache stalls are for DRAM; there are no SRAM accesses. For each task configuration, eight different clock-voltage settings were tested, and for each setting we measured the entire board level power consumption and execution time. As in the previous section, we performed a nonlinear least square analysis on the collected energy consumption data to determine the value of each parameter in Eq. 8.

Fig. 5 plots the measured energy consumption and the model predicted energy consumption. The R^2 (the coefficient of determination) value was 99.78% and the MAE was 1.25%, suggesting that our energy model accurately captures the system behavior. Table IV shows the value of each parameter in Eq. (8) for the tested hardware configuration with external DRAM, obtained from the non-linear least square analysis. We used these parameters for the evaluation of real applications in Section 5.2.2.

4. ENERGY OPTIMIZATION OF REAL-TIME TASKS

We formulate a problem to find the energy optimal frequency set, which contains the frequency assignments of multiple DVS components, to schedule periodic real-time tasks. In our solution, a single (possibly different) frequency is found for each component. All tasks that share a component, share that frequency on the component. We call such a frequency assignment static. In theory, it is possible to do better by implementing a dynamic frequency assignment, that changes the frequency of each component on each context switch. The problem of dynamic frequency assignment becomes one of finding the best frequency for each component and for each task. We do not consider the latter problem, because it is difficult to implement in practice. Nevertheless, in the evaluation section, we compare

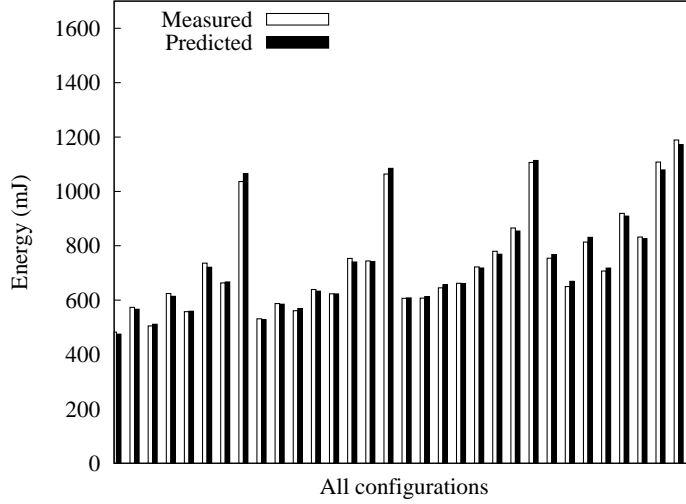


Fig. 5. Energy prediction error (residual)

the performance of our static scheme to the optimal dynamic solution (which we computed by a brute-force search for small task sets). We show that the dynamic scheme does not offer significant improvement in energy savings, further reinforcing our choice of a static (multi-DVS) frequency assignment.

4.1 Problem Definition

Given a set $T = T_1, \dots, T_n$ of n periodic real-time tasks, the period of T_i is denoted by P_i , which is equal to the deadline. The tasks are scheduled on a single processor system based on preemptive scheduling, and all tasks are assumed to be independent. In the worst case, each task invocation requires C_i CPU cycles and M_i memory cycles. The worst case execution time of task T_i is $e_i = \frac{C_i}{f_c} + \frac{M_i}{f_m}$, where f_c is CPU frequency, f_b is system bus frequency, and f_m is memory frequency (see Eq. (1)).

The energy consumption, $E_{act,i}$, of each invocation of task T_i is given by $E_{act,i} = E_{comp,i} + E_{mem,i}$ (see Eq. (4), (5)). CPU, system bus, and memory are idle if there is no task to execute, and the power consumption is $I + R$ (see Eq. (6)). The hyperperiod, H , is the least common multiple of P_1, \dots, P_n , and the total energy consumption, E , over H is $\sum_{i=1}^n \frac{H}{P_i} \cdot E_{act,i} + E_{idle}$, where $E_{idle} = (H - \sum_{i=1}^n \frac{H}{P_i} \cdot e_i) \cdot (I + R)$. A schedule of periodic tasks is *feasible* if each task, T_i , is guaranteed C_i CPU cycles and M_i memory cycles at each invocation.

The optimization formulation of the multi-DVS frequency assignment problem with EDF scheduling is

$$\text{minimize } \sum_{i=1}^n \frac{H}{P_i} E_{act,i} + E_{idle} \quad (9)$$

$$\text{subject to } \sum_{i=1}^n \frac{e_i}{P_i} \leq 1. \quad (10)$$

Eq. (9) minimizes the sum of the energy consumption of all task invocations during hyperperiod H . Eq. (10) is the necessary and sufficient schedulability constraint because the utilization of task T_i is $\left(\frac{C_i}{f_c} + \frac{M_i}{f_m}\right) / P_i$.

4.2 Static Multi-DVS Frequency Assignment

In this section, we present an optimal solution for assigning a single frequency set to a task set, T , such that all the tasks in the set run at the same CPU frequency, f_c , and the same memory frequency, f_m . The number of CPU execution cycles in a *hyperperiod*, C_H , is $\sum_{i=1}^n \left(\frac{H}{P_i} \cdot C_i\right)$, and the number of memory access cycles, M_H , is $\sum_{i=1}^n \left(\frac{H}{P_i} \cdot M_i\right)$ ¹. Total execution time in the *hyperperiod*, e_H , is $\frac{C_H}{f_c} + \frac{M_H}{f_m}$.

LEMMA 1. (9) is equivalent to the sum of the right hand side of (4)–(6) by replacing C with C_H and M with M_H .

PROOF. Let W_{comp} be the power during the pure execution block and W_{mem} be the power during the cache stall block. Thus we have

$$\begin{aligned}
& \sum_{i=1}^n \frac{H}{P_i} E_{act,i} + E_{idle} \\
&= \sum_{i=1}^n \left(\frac{H}{P_i} W_{comp} \frac{C_i}{f_c} + \frac{H}{P_i} W_{mem} \frac{M_i}{f_m} \right) + E_{idle} \\
&= \frac{1}{f_c} \sum_{i=1}^n \frac{H \cdot C_i}{P_i} W_{comp} + \frac{1}{f_m} \sum_{i=1}^n \frac{H \cdot M_i}{P_i} W_{mem} + E_{idle} \\
&= W_{comp} \frac{C_H}{f_c} + W_{mem} \frac{M_H}{f_m} + E_{idle}.
\end{aligned} \tag{11}$$

□

LEMMA 2. Under EDF, if the execution time, e_H , does not exceed the hyperperiod, H , then the task set is schedulable.

PROOF. From Eq. (10),

$$\sum_{i=1}^n \frac{e_i}{P_i} = \sum_{i=1}^n \frac{H \cdot e_i}{H \cdot P_i} = \frac{1}{H} \sum_{i=1}^n \frac{H}{P_i} \cdot e_i = \frac{e_H}{H} \leq 1. \tag{12}$$

Hence,

$$e_H \leq H. \tag{13}$$

□

¹While individual M_i , cache stall cycles of task T_i , can increase when preempted by other tasks, the effect the increase is generally negligible – 0.25% of the total execution time in maximum when preempted 100 times for 3 seconds on an ARM926-ejs processor [David et al. 2007]. Nevertheless, we can measure task set cache stall cycles, M_H , including additional stall cycles caused by preemption, using performance counter.

4.2.1 *Methodology.* The energy model presented in Eq. (11) can be combined with system and deadline constraints to find the energy optimal f_c , f_b , and f_m given a task set defined with a single hyperperiod, H .

Frequency constraints on the CPU, memory, and bus arise from hardware specifications,

$$f_{c,min} \leq f_c \leq f_{c,max}, \quad (14)$$

$$f_{m,min} \leq f_m \leq f_{m,max}, \quad (15)$$

$$f_b = f_m. \quad (16)$$

The procedure for determining the energy optimal frequency assignment requires finding unconstrained energy minimizing frequency sets and finding solutions on the boundary conditions and the boundary intersections imposed by the constraints. Each frequency set must then be evaluated in the energy model and the energy minimal solution chosen.

The optimum frequency assignment is found assuming continuous variables, but real systems have discrete frequency steps. The frequency assignment for the real system can be found by testing possible frequencies that neighbor the continuous optimal frequency solution. Neighboring frequencies can be enumerated by finding the nearest higher and lower discrete frequencies. Frequency sets that violate the deadline constraint are eliminated and the remaining sets must be evaluated and compared in the energy model.

The energy model presented in Eq. (11) and the constraints in Eqs. (13)–(16) result in the following set of equations that find possible frequency assignments in the global search space and on the boundary conditions.

4.2.1.1 *Unconstrained Minima.*

$$\text{Find } f_m \text{ such that } \frac{\partial E}{\partial f_m} = 0 \quad (17)$$

$$\text{Find } f_c \text{ such that } \frac{\partial E}{\partial f_c} = 0 \quad (18)$$

Substituting f_m from Eq. (17) into Eq. (18) yields unconstrained energy minimum frequency sets.

4.2.1.2 *Minima on CPU Frequency Boundaries.*

$$\text{Find } f_m \text{ such that } \frac{\partial E}{\partial f_m} = 0 \quad (19)$$

$$f_c \in \{f_{c,min}, f_{c,max}\} \quad (20)$$

Eq. (19) solved for the scenarios where $f_c = f_{c,max}$ and $f_c = f_{c,min}$ yields the energy minimum frequency sets on the CPU frequency boundaries.

4.2.1.3 *Minima on Memory Frequency Boundaries.*

$$f_m \in \{f_{m,min}, f_{m,max}\} \quad (21)$$

$$\text{Find } f_c \text{ such that } \frac{\partial E}{\partial f_c} = 0 \quad (22)$$

Eq. (22) solved for the two scenarios where $f_m = f_{m,max}$ and $f_m = f_{m,min}$ yields the energy minimum frequency sets on the memory frequency boundaries.

4.2.1.4 *Minimum on Deadline Constraint Boundary.* In order to meet the task set deadline, H ,

$$f_m = \frac{M_H}{H - \frac{C_H}{f_c}} \quad (23)$$

$$\text{Find } f_c \text{ such that } \frac{\partial E}{\partial f_c} = 0 \quad (24)$$

Substituting f_m from Eq. (23) into Eq. (24) yields the energy minimum frequency set on the deadline constraint boundary.

4.2.1.5 *Boundary Intersections.* All combinations of maximum and minimum CPU and memory frequencies yield the frequency sets at frequency boundary intersections. Eq. (23) solved for the scenarios where $f_c = f_{c,max}$ and $f_c = f_{c,min}$, and backsolved for f_c when $f_m = f_{m,max}$ and $f_m = f_{m,min}$ yields the frequency sets on the deadline and frequency boundary intersections.

4.2.2 *Methodology Summary.* The following steps summarize the procedure for finding multiple component frequency assignments:

- 1.. Find unconstrained energy optimal frequency sets using Eqs. (17)–(18) and the energy optimal frequency sets on each boundary condition and boundary intersection using Eqs. (19)–(24).
- 2.. Eliminate results that violate any constraints from Eqs. (13)–(16).
- 3.. Evaluate and compare each frequency set in the energy model from Eq. (11), and choose the lowest energy set.
- 4.. Enumerate the frequency sets obtainable in the real system that neighbor the optimal frequency set.
- 5.. Eliminate sets that violate the deadline constraint, Eq. (13), and evaluate the remaining frequency sets in the energy model from Eq. (11), choosing the lowest energy set as the final solution.

An example of applying the procedure is presented below.

4.2.3 *Methodology Example.* Consider a system with $V_{cpu} = V_{bus} = V_{mem}$ and CPU voltage as a linear function of f_c , $V_{cpu} = A f_c + B$. Let $A = 0.0016$ V/MHz and $B = 1.504$ V. Other system parameters are given in Table IV. Let the task set be described by $C_H = 140 \cdot 10^6$ cycles, $M_H = 30 \cdot 10^6$ cycles, and hyperperiod, $H = 3$ sec.

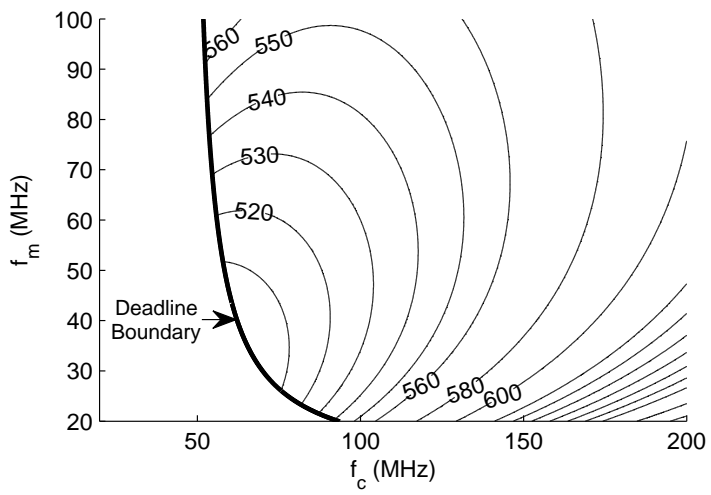
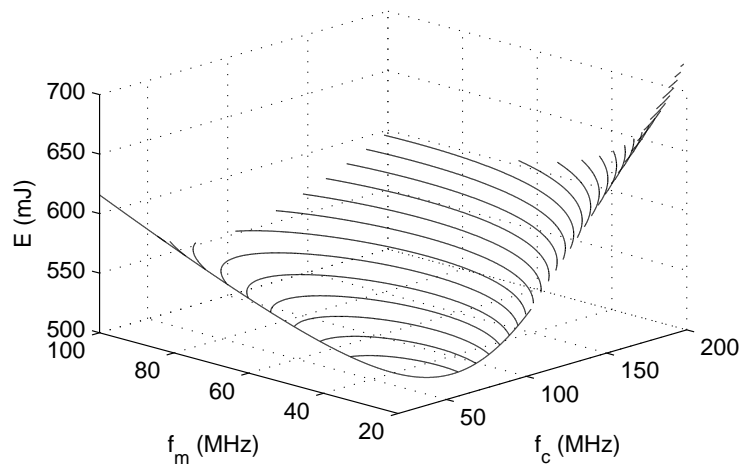


Fig. 6. Energy vs. f_c and f_m with a task set of $C_H = 140 \cdot 10^6$ cycles, $M_H = 30 \cdot 10^6$ cycles and $H = 3$ sec evaluated in the proposed energy model.

Fig. 6 shows the energy consumption of the task set as a function of f_c and f_m . No unconstrained minima are found within the limits of f_c . The lowest energy frequency set from all boundaries and boundary intersections is found on the deadline boundary at $\{f_c, f_m\} = \{65.45 \text{ MHz}, 35.35 \text{ MHz}\}$. It can be seen from Fig. 6 that this frequency assignment indeed results in the minimum energy consumption for the system. The neighboring obtainable frequency sets are $\{66,36\}$, $\{66,34\}$, $\{64,36\}$, and $\{64,34\}$ if f_c and f_m are adjustable in 2 MHz steps. Checking the frequency sets against the constraints reveals that all sets except $\{66,36\}$ violate the deadline constraint, so they are eliminated. Evaluating the remaining frequency set in the energy model in Eq. (11) determines that the set $\{f_c, f_m\} = \{66,36\}$ has an energy value of 501.3 mJ for this task set.

5. EVALUATION

In this section, we present simulation results and experimental results performed on the hardware platform to evaluate the multi-DVS scheme. In Section 5.1, we demonstrate that our multi-DVS schemes can save energy compared to traditional CPU-only DVS scheme and justify our choice of static multi-DVS over dynamic multi-DVS with simulations. In Section 5.2, we show that the multi-DVS scheme saves energy with experiments that run real applications on the previously described hardware.

5.1 Simulation

In this section we simulated the energy savings of the proposed multi-DVS scheme with other DVS schemes. We performed simulations because (1) the choice of f_m is limited in our real hardware platform— $f_m = f_c/n$ where n is an integer—but many new processors [Freescale 2008; Samsung b] do not have this constraint, and (2) we wanted to investigate the effects of varying the idle power and voltage range, which are fixed in the real hardware platform. Nevertheless, we used the parameters obtained from the real hardware platform as shown in Table IV. We use average power consumption, which is calculated by the total energy consumption divided by the *hyperperiod*, as the evaluation metric.

Five schemes are compared in our evaluation: *MAX*, *CPU-only DVS*, *Baseline multi-DVS*, *Static multi-DVS*, and *Dynamic multi-DVS*. In the *Max* scheme, tasks are executed with the maximum CPU and maximum bus/memory frequency. In the *CPU-only DVS scheme*, we set the optimal CPU frequency for each task while the system bus/memory frequency is set to the maximum value. The *Baseline multi-DVS scheme* requires that both the CPU and bus/memory frequencies be proportional to the task set CPU utilization at maximum frequency¹. The *Dynamic multi-DVS scheme* results from a brute-force search for all possible combinations of frequencies to determine the optimal frequencies for each individual task. The *Static multi-DVS scheme* is the proposed solution described in the previous section which assigns a single CPU and memory frequency to the entire task set. We normalize the average power consumption to the *Max* scheme (*i.e.*, no DVS) in every figure.

The system energy consumption was simulated by varying the task set utilization at maximum frequency, cache stall ratio, and idle power consumption while satisfying schedulability constraints for Earliest Deadline First (EDF) scheduling.

¹In the *Baseline multi-DVS* scheme, $f_c = f_{c,max} \cdot \sum_{i=1}^n u_{i,max}$, and $f_m = f_{m,max} \cdot \sum_{i=1}^n u_{i,max}$, where $u_{i,max} = \left(\frac{C_i}{f_{c,max}} + \frac{M_i}{f_{m,max}} \right) / P_i$.

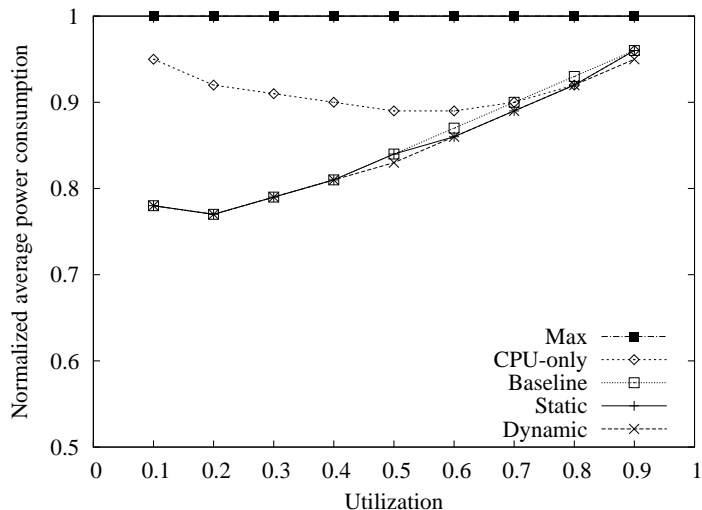


Fig. 7. Average power consumption with varying utilization and constant cache stall ratio = 0.3.

5.1.1 *Varying Task Set Utilization.* Fig. 7 shows the average power consumption of the compared DVS schemes for varying utilization. As the utilization increases, the feasible frequency scaling range decreases; as a result, the effectiveness of all the DVS schemes reduces. When the utilization is low, the *Static multi-DVS* scheme consumes less energy than the *CPU-only DVS* scheme, because it saves energy by setting a lower bus frequency without violating the deadline constraints. Although the *Baseline multi-DVS* scheme is close to *Static* and *Dynamic optimum multi-DVS* schemes in this figure, the effectiveness of the *Baseline multi-DVS* highly depends on the cache stall ratio, which we will show in the next subsection. The difference in energy consumption between the *Static multi-DVS* scheme and the *Dynamic multi-DVS* scheme is less than 1%.

5.1.2 *Varying Cache Stall Ratio.* Fig. 8 shows the average power consumption of the compared DVS schemes for varying task set cache stall ratio, $\frac{M_H}{C_H + M_H}$. The task set utilization is fixed at 50%. When the cache stall ratio is low (representing a CPU intensive workload), the *Static multi-DVS* scheme takes advantage of lowering the bus frequency without violating the deadline constraints. In contrast, when the cache stall ratio is high, the *CPU-only DVS* and the *Static multi-DVS* schemes have lower energy consumption than the *Baseline multi-DVS* scheme, because they are able to set a lower CPU frequency. When the cache stall ratio is between 0.1 and 0.2, which is common in many applications [Pellizzoni and Caccamo 2007], the *Static multi-DVS* scheme shows a clear advantage. Note that the *Static multi-DVS* scheme shows similar performance to the *Dynamic multi-DVS* scheme.

5.1.3 *Varying Cache Stall Ratio Diversity.* In the next experiment, we change the degree to which different tasks differ in their cache stall ratio. Note that, when tasks are more diverse (i.e., when a mix of CPU intensive and memory intensive tasks are present), the *Static multi-DVS* scheme is expected to perform worse than the *Dynamic multi-DVS* scheme because it cannot customize frequency settings to each task. As a proxy for task

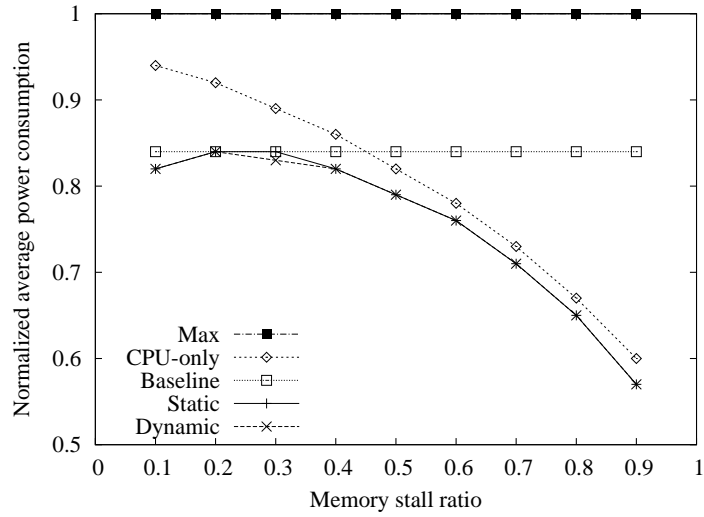


Fig. 8. Average power consumption with varying cache stall ratio and constant utilization = 0.5.

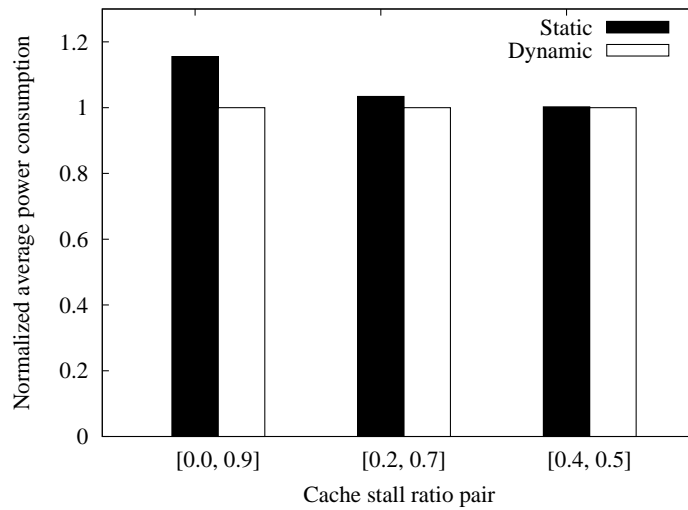


Fig. 9. Comparisons on the average power consumption with different diversity of cache stall ratio and utilization = 0.5, task set cache stall ratio = 0.45. The configuration $[min, max]$ represents a task set in which half of the tasks have a cache stall ratio of min and half have the ratio max .

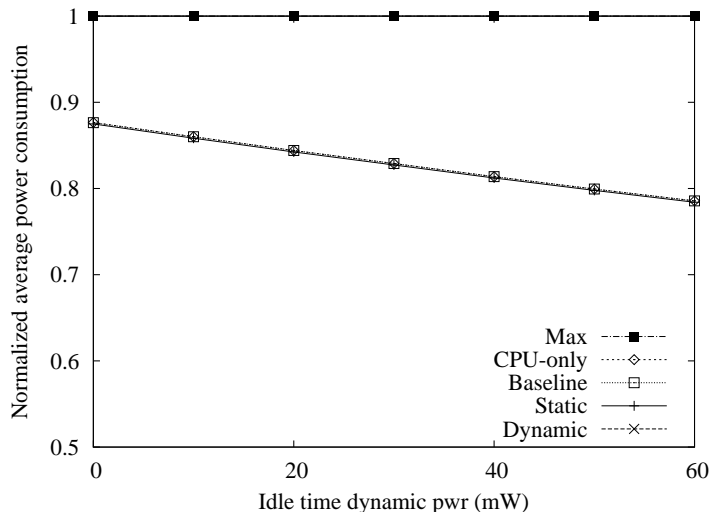


Fig. 10. Average power consumption with varying idle power and constant utilization = 0.5, and constant cache stall ratio = 0.3.

diversity, we change the variance of the memory stall ratio across the task set keeping the task set cache stall ratio fixed at 0.45. In Fig. 9, the average power consumption is plotted against the variance in the cache stall ratio.

Note that, while the *Dynamic multi-DVS* scheme is 13% better than the *Static multi-DVS* scheme for large variances, it is expected that in many realistic embedded task sets tasks are reasonably homogeneous. For example, in a process control system, where different tasks implement different controllers, it is likely that the controllers do not substantially differ in memory stall ratio. For such sets, the performance hit of the *Static multi-DVS* scheme is less than 0.5%, which we deem acceptable. Investigation of efficient dynamic multi-DVS schemes is therefore left as a topic for future work.

5.1.4 Varying Idle Power Consumption. Fig. 10 shows the average power consumption of the compared DVS schemes over varying idle power consumption of CPU, bus, and memory. When idle power is zero, only static power contributes to the total energy. When idle power is large, energy consumed during idle significantly contributes to the total energy. Therefore, all DVS schemes achieve lower energy consumption by setting a lower frequency and reducing the slack time. Moreover, all *multi-DVS* schemes outperform the *CPU-only DVS* scheme by lowering the bus frequency and further reducing the slack time.

5.1.5 Varying Voltage Scaling Range. Fig. 11 shows the average power consumption of the compared DVS schemes over varying voltage scaling range. The cache stall ratio is 10% and the task set utilization is 50%. If the system provides a larger voltage scaling range, all DVS schemes save more energy. However, all *multi-DVS* schemes are able to take greater advantage of the larger scaling range by also controlling the bus frequency.

5.1.6 Simulation Summary. Evaluation results show that cache stall ratio is one of the most important factors affecting the performance of the *Static multi-DVS* scheme. The

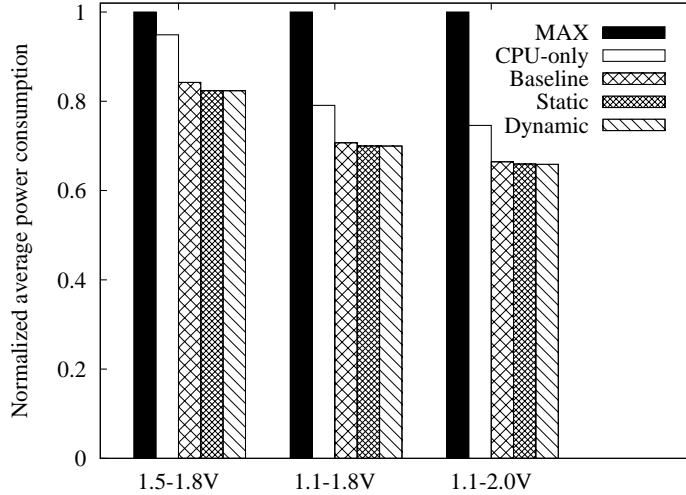


Fig. 11. Comparisons on the average power consumption with different voltage scaling range and utilization = 0.5, cache stall ratio = 0.1.

Static multi-DVS scheme is effective for workloads with low cache stall ratio, because it is able to set a low memory frequency without violating deadline constraints. Additionally, the *Static multi-DVS* scheme has lower energy consumption compared to the *CPU-only DVS* scheme when idle power is high, because it can lower both the CPU and memory frequency and make the utilization close to unity. The proposed *Static multi-DVS* scheme achieves good performance in a large range of evaluated configurations.

5.2 Experiments with Real Applications

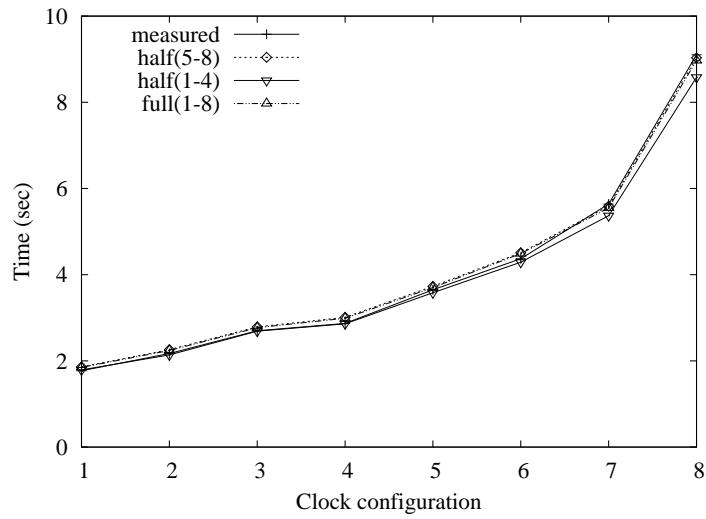
In this section, we evaluate multi-DVS scheme on a real hardware using two applications: *madplay*, a mp3 decoder, and *dhrystone*, a performance benchmark. The hardware configurations and the energy model are described in Section 5.2.2.

5.2.1 Obtaining C and M . Our energy optimization method requires knowing C and M , the CPU and memory cycles, respectively. Previously, we controlled C and M directly by changing the number of loops in our synthetic program (Section 3.1), however, there is no direct method to obtain these values for an arbitrary program without hardware support (e.g., performance counter).

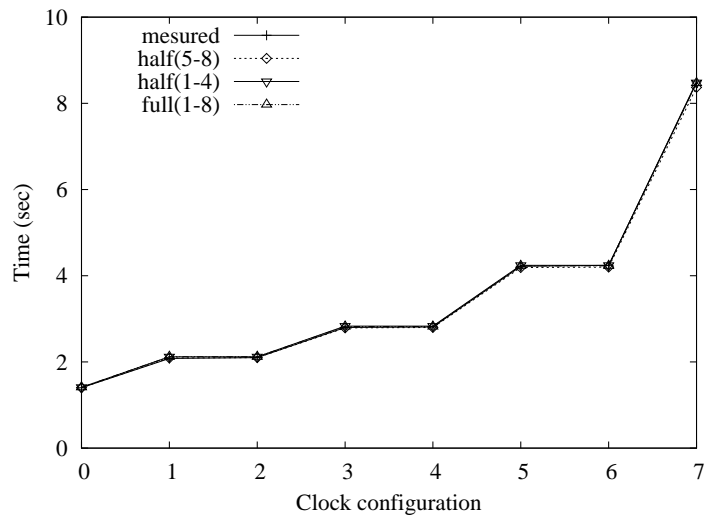
Instead, we obtained C and M values indirectly with regression analysis using the following methodology: (1) measure the execution time for various clock configurations (combinations of CPU and BUS frequencies), and then (2) perform a non-linear least square analysis for Eq. 1 using the collected execution time data.

Figure 12 shows the execution time regression results for *madplay* and *dhrystone*². In addition, a partial data crosscheck was performed on the regression to verify its predictive

²For *madplay*, we played a 10 second, 1KHz sine wave mp3 file. Note that we outputted the decoded pcm data to `/dev/null` instead of `/dev/dsp`, because we did not consider the power consumption of the audio amplifier and DAC convertor chip in our energy model. For *dhrystone*, we used an input value of 100000.



(a) *madplay*



(b) *dhrystone*

data set	<i>madplay</i>	<i>dhrystone</i>
full(1-8)	2.72%	0.69%
half(1-4)	2.18%	0.79%
half(5-8)	3.08%	0.57%

(c) Mean Absolute Error (MAE)

Fig. 12. Execution time regression. (a) is *madplay*. (b) is *dhrystone*. (c) is MAE of each regression.

capability. The full(1-8) data set used all eight collected data points for the regression; half(1-4) used four data points; half(5-8) used the complement of half(1-4). Table 12(c)

	MAX	CPU-only	Baseline	Static	Dynamic
Freq. (MHz)	(200,100)	(100,100)	(40, 20)	(40,20)	[(40,20),(56,28)]
Energy(mJ)	1701.00	1588.63	1403.63	1403.63	1422.95
Saving(%)	-	6.58	17.46	17.46	16.32

Fig. 13. Comparison of actual energy saving. *madplay* and *dhrystone* both have equal deadline of 10 second. Saving was compared with MAX.

compares Mean Absolute Error (MAE) of each regression model suggesting that the predictive capability of the model is reasonable. Notice that MAE of *dhrystone* is very small indicating that this method is very effective to estimate the execution time of CPU intensive task. MAE of *madplay* is, however, relatively large – up to 3%. We conjecture that frequent I/O operations (i.e., mp3 file read) and the complexity of DRAM accesses result in this error.

5.2.2 Energy Optimal Frequency Selection. From the obtained C and M values, we performed an experiment to compare the actual energy saving of our multi-DVS scheme. We used a taskset that consisted of *madplay* and *dhrystone* with a deadline of 10 seconds in both programs. We used (C, M) values obtained from the regression: *madplay*-(137.09, 42.37), *dhrystone*-(169.37,0.00) and compared the five schemes – MAX, CPU-only DVS, Baseline multi-DVS, Static multi-DVS, and Dynamic multi-DVS – as described in Section 5.1. The experiment was performed on the hardware platform described in .

Figure 13 shows the actual energy saving for various schemes. The results show that all three multi-DVS schemes (Baseline, Static, and Dynamic) outperform CPU-only DVS scheme as we predicted from simulations and the energy saving is up to 17% compared to MAX. Interestingly, the dynamic multi-DVS scheme is slightly worse than our static scheme. This is due to the hardware limitations of selecting clock frequencies. Specifically, the calculated optimal clock configuration for *madplay* and *dhrystone* are (40,20) and (56,20) respectively. However, (56,20) was not achievable due to the hardware constraint $f_m = f_b/n$. In practice, there are various hardware constraints. Therefore, fine-grain clock settings for the dynamic scheme may not always be achievable, leading to sub-optimal energy savings.

6. DISCUSSION

In this section, we discuss the validity of our energy model calibration method described in Section 2. Figure 6 shows the main body of the synthetic program we used. The program consists of two loops: (1) the computation loop with no cache misses, and (2) the memory loop with 100% cache misses. By changing the repetition count of each loop, we synthesized multiple tasks with varying C and M values, and then used those tasks to calibrate our energy model. Notice that this program used only five instructions: MOV, ADD, CMP, BLT, and LDR. However, other instructions, for example, MUL, may have different power consumption per cycle [ARM]. Also, while we include cache-hit memory access instructions, for example, LDR on cached data, in computation cycles, it may consume more power than normal computation instructions [Wan et al. 2002].

To investigate the effect of varying power consumptions for different instructions, we compared power consumption of our synthetic program, *synthetic*, and *dhrystone*. *dhrystone* generates almost no cache misses so we configured the synthetic program to only use

```

...
// computation loop
cloop:
mov    r0, r2                // no cache access
add    r2, r2, r1
cmp    r2, r3
blt    cloop
...
// cache stall loop
mloop:
ldr    r0, [ip, r2, asl #2]  // 100% cache miss
add    r2, r2, r1           // increment pointer to next cache line.
cmp    r2, r3
blt    mloop

```

Fig. 14. Synthetic program for calibration

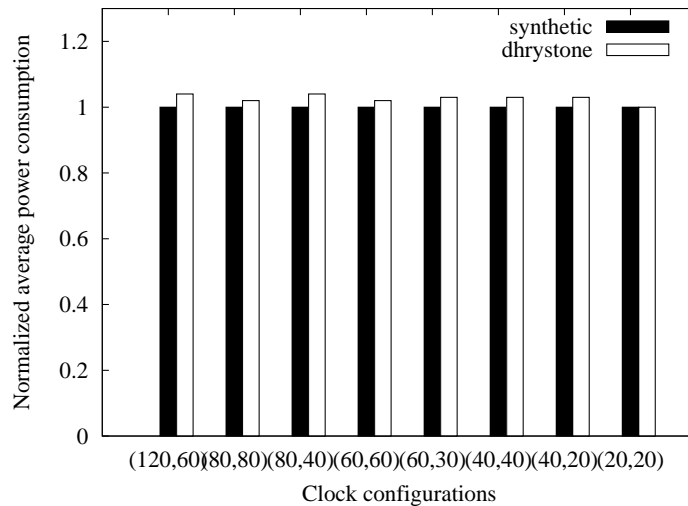


Fig. 15. Comparison of average power consumption of *synthetic* and *dhrystone*.

the computation loop for comparison. Figure 15, shows the average power consumption of the two programs at eight different clock configurations. The result is normalized to the synthetic program values. It shows that *dhrystone* consumes slightly more power (up to 4%) than *synthetic*, meaning that the model predicted power consumption, calibrated from a set of *synthetic* programs, may be different from real power consumption depending on the application's instructions and the number of cache-hit memory accesses. One way to improve the accuracy of the power model is to extend it to include instruction and cache level power consumption as in [Šimunić et al. 1999]. However, to take into account these details, the model will be significantly complicated.

Minimizing the error while maintaining the simplicity of the model is a challenging task in general. A possible method to improve accuracy is to use representative real applica-

tions for model calibration. To do this, we would need a precise method to obtain computation and memory cycles for each application, for example, using a hardware performance counter as in [Snowdon et al. 2009].

7. RELATED WORK

There is a significant amount of previous work focusing on static DVS schemes for real-time tasks under RM or EDF scheduling [Mejia-Alvarez et al. 2004; Gruian 2001; Jejurikar and Gupta 2006; Aydin et al. 2001]. [Aydin et al. 2001] formulated an energy minimization problem and proposed an algorithm to find the optimal static frequency assuming that individual tasks have different power characteristics. [Jejurikar and Gupta 2006] proposed an energy model in which the deadline was not equal to the period, and presented methods for finding weak-optimal slowdown factors for scheduling periodic tasks.

On-line slack time reclamation has been another realm of DVS research. [Pillai and Shin 2001] proposed dynamic reclamation techniques, cycle-conserving, and look-ahead, which exploited unused slack time dynamically to save energy. Gruian's dynamic algorithm also utilized slack time to lower energy usage and employed both on-line and off-line scheduling policies [Gruian 2001]. [Mejia-Alvarez et al. 2004] proposed an on-line scheduling algorithm with discrete frequency steps for DVS. Zhong *et al.* proposed an analytical model for scheduling without prior task information [Zhong and Xu 2005]. Our work focuses on static frequency scaling but is novel because we adjust multiple component frequencies, unlike previous work that focused exclusively on CPU frequency.

Several researchers account for the effects of I/O operations, such as main-memory accesses [Bini et al. 2005; 2009; Zhuo and Chakrabarti 2005; Aydin et al. 2006; Fan et al. 2002] because I/O operation time does not scale with the CPU frequency. [Bini et al. 2005; 2009] proposed an energy model that accounts for memory operations in the task execution time when only CPU frequency is adjustable. [Aydin et al. 2006] proposed a similar energy model that also considered CPU frequency independent power components. Our model incorporates CPU frequency independent components, such as bus and memory, but also allows adjusting their frequencies. It can also be extended with dynamic power management techniques such as forbidden regions [Devadas and Aydin 2008] in future work.

Although the CPU consumes a significant amount of energy, other components such as the main memory and system bus often consume energy on a similar order of magnitude. Without considering such components, energy savings may not be maximized. Some previous work addresses power-saving with component standby modes in addition to a DVS-capable CPU. [Zhuo and Chakrabarti 2005] proposed a system-wide energy model that considered standby mode components and showed that as the number of components grew, the effectiveness of DVS schemes decreased due to increased standby power. [Zhong and Xu 2006] also presented an energy model accounting for standby modes for both periodic and sporadic tasks. [Cho and Chang 2006] proposed an energy model that considered both CPU and memory frequencies based on an idealistic energy model with no standby-energy and did not consider real-time tasks. Snowdon *et al.* also proposed an execution time and energy model [Snowdon et al. 2007; Snowdon et al. 2009] that considered multiple adjustable frequencies and task characteristics. Our model is simpler yet detailed enough to explain our system behavior.

Dynamic Power Management (DPM) is another widely adopted mechanism to reduce

power consumption by turning off the system components during idle period [Benini et al. 2002]. Several works proposed to combine DVS with DPM to achieve higher energy efficiency. [Simunic et al. 2001] proposed an algorithm to merge the DPM and the DVS approaches based on the stochastic model. [Liu et al. 2008] proposed an algorithm combining both DVS and DPM for streaming applications on embedded multiprocessors. [Cheng and Goddard] presented system-wide Energy-Aware EDF (SYS-EDF), which integrates DPM for I/O devices and DVS for the processor. [Rong and Pedram 2006] proposed a three-phase solution framework to reduce the system-wide power consumption with the consideration of task dependencies for periodic hard real-time tasks.

Recently, researchers have also considered DVS schemes for multicore processors or multiprocessors. [Chen et al. 2004] investigated energy aware scheduling for general multiprocessors, and proposed a solution in which the CPU frequency was unbounded. While previous work on multicore or multiprocessors treated cores as multiple DVS components, the energy models only considered execution time on CPU cores. However, in many systems, CPU, bus, and memory are tightly coupled and jointly affect task execution time.

Finally, our work is based on a realistic energy and execution time model which is validated on a real hardware platform. Our problem is novel because we consider multiple DVS component frequencies that jointly affect task execution time.

8. CONCLUSIONS AND FUTURE WORK

In this study, we contribute a realistic and flexible energy model for embedded systems. The model focuses on CPU, system bus, and memory and allows variable frequencies for those components. Through experiments on a real hardware platform, we showed that the model can accurately predict system-wide energy consumption. A solution was then derived to find frequency assignments for multiple components considering system constraints. Based on the model and the solution, we proposed a static multi-DVS scheme to schedule periodic real-time tasks, and we compared our multi-DVS scheme with other DVS schemes to demonstrate its effectiveness.

Future extensions of multi-DVS may include developing efficient dynamic multi-DVS solutions where component frequencies are varied for each task. Such schemes may also be integrated with DPM when individual devices have stand-by or off modes. This work will be reported in future publications.

9. ACKNOWLEDGEMENTS

Work reported in this paper was performed with partial funding from NSF grants CNS 06-15301, CNS 07-20513, and CNS 09-16028. Findings and opinions expressed in this work are those of the authors and not necessarily those of the funding agencies.

REFERENCES

- ARM. *The ARM Instruction Set*.
- AYDIN, H., DEVADAS, V., AND ZHU, D. 2006. System-Level Energy Management for Periodic Real-Time Tasks. In *Proceedings of IEEE RTSS*. IEEE Computer Society, Washington, DC, USA, 313–322.
- AYDIN, H., MELHEM, R., MOSSÉ, D., AND ALVAREZ, P. 2001. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of ECRTS*. 225–232.
- BENINI, L., BOGLIOLO, A., AND DE MICHELI, G. 2002. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 8, 3, 299–316.

- BINI, E., BUTTAZZO, G., AND LIPARI, G. 2005. Speed modulation in energy-aware real-time systems. In *Proc. of the Euromicro Conference on Real-Time Systems*.
- BINI, E., BUTTAZZO, G., AND LIPARI, G. 2009. Minimizing CPU energy in real-time systems with discrete speed management. *ACM Trans. Embed. Comput. Syst.* 8, 4, 1–23.
- BROOKS, D. M. E. 2000. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro* 20, 6, 26–44.
- CHEN, J., HSU, H., CHUANG, K., YANG, C., PANG, A., AND KUO, T. 2004. Multiprocessor energy-efficient scheduling with task migration considerations. In *Proceedings of ECRTS*. 101–108.
- CHENG, H. AND GODDARD, S. Integrated device scheduling and processor voltage scaling for system-wide energy conservation.
- CHO, Y. AND CHANG, N. 2006. Energy-Aware Clock-Frequency Assignment in Microprocessors and Memory Devices for Dynamic Voltage Scaling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 6, 1030–1040.
- DAVID, F., CARLYLE, J., AND CAMPBELL, R. 2007. Context switch overheads on mobile device platforms. In *Experimental computer science on Experimental computer science*. USENIX Association, 2.
- DEVADAS, V. AND AYDIN, H. 2008. Real-time dynamic power management through device forbidden regions. In *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium-Volume 00*. IEEE Computer Society Washington, DC, USA, 34–44.
- DHIMAN, G., PUSUKURI, K. K., AND ROSING, T. 2008. Analysis of Dynamic Voltage Scaling for System Level Energy Management. *Workshop on Power Aware Computing and Systems*.
- ERANIAN, S. *perfmon2 project website*.
- FAN, X., ELLIS, C., AND LEBECK, A. 2002. The synergy between power-aware memory systems and processor voltage scaling. *Department of Computer Science Duke University, Durham TR CS-2002-12*.
- FREESCALE. *STMP3650 Product Overview*.
- FREESCALE. 2008. *STMP3700 System-on-Chip Fact Sheet*.
- GRUIAN, F. 2001. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In *Proceedings of the 2001 International Symposium on Low power Electronics and Design*. ACM New York, NY, USA, 46–51.
- INTEL. 2004. *Low Voltage Intel Xeon Processor with 800 MHz System Bus*.
- JEJURIKAR, R. AND GUPTA, R. 2006. Optimized slowdown in real-time task systems. *IEEE Transactions on Computers* 55, 12, 1588.
- LIU, H., SHAO, Z., WANG, M., AND CHEN, P. 2008. Overhead-aware system-level joint energy and performance optimization for streaming applications on multiprocessor systems-on-chip. In *Proceedings of ECRTS*. 92–101.
- MEJIA-ALVAREZ, P., LEVNER, E., AND MOSSÉ, D. 2004. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems (TECS)* 3, 2, 306.
- PELLIZZONI, R. AND CACCAMO, M. 2007. Toward the predictable integration of real-time COTS based systems. In *Proceedings of IEEE RTSS*.
- PILLAI, P. AND SHIN, K. G. 2001. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of SOSP*. 89–102.
- RONG, P. AND PEDRAM, M. 2006. Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*. IEEE Press, 473–478.
- SAMSUNG. *K4M281633H Mobile SDRAM Datasheet*.
- SAMSUNG. *S3C2440A User's Manual*.
- SHIM, H., CHO, Y., AND CHANG, N. 2004. Power saving in hand-held multimedia systems using MPEG-21 digital item adaptation. In *Proceedings of IEEE Workshop on Embedded Systems for Real-Time Multimedia*. 13–18.
- SIMUNIC, T., BENINI, L., ACQUAVIVA, A., GLYNN, P., AND DE MICHELI, G. 2001. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the 38th annual Design Automation Conference*. ACM, 529.

- SNOWDON, D., LE SUEUR, E., PETERS, S., AND HEISER, G. 2009. Koala: A platform for OS-level power management. In *Proceedings of the fourth ACM european conference on Computer systems*. ACM New York, NY, USA, 289–302.
- SNOWDON, D., PETERS, S., AND HEISER, G. 2007. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *Proceedings of EMSOFT*. ACM, 93.
- TI. *Cortex-A8 Architecture*.
- ŠIMUNIĆ, T., BENINI, L., AND DE MICHELI, G. 1999. Cycle-accurate simulation of energy consumption in embedded systems. In *DAC '99: Proceedings of the 36th annual ACM/IEEE Design Automation Conference*. ACM, New York, NY, USA, 867–872.
- WAN, M., ICHIKAWA, Y., LIDSKY, D., AND RABAEY, J. 2002. An energy conscious methodology for early design exploration of heterogeneous DSPs. In *Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998*. IEEE, 111–117.
- ZHONG, X. AND XU, C.-Z. 2005. Energy-Aware Modeling and Scheduling of Real-Time Tasks for Dynamic Voltage Scaling. *Proceedings of IEEE RTSS*, 10 pp.–375.
- ZHONG, X. AND XU, C. Z. 2006. System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation. *The International Conference on Computer-Aided Design*, 516–521.
- ZHUO, J. AND CHAKRABARTI, C. 2005. System-level energy-efficient dynamic task scheduling. In *Proceedings of the 42nd annual Design Automation Conference*. ACM, New York, NY, USA, 628–631.