

# A Cooperative Management Scheme for Power Efficient Implementations of Real-Time Operating Systems on Soft Processors

Jingzhao Ou and Viktor K. Prasanna, *Fellow, IEEE*

**Abstract**—A cooperative management scheme for power efficient implementations of real-time operating systems on field-programmable gate-array (FPGA)-based soft processors is presented. Dedicated power management hardware peripherals are tightly coupled to a soft processor by utilizing its configurability. These hardware peripherals manage tasks and interrupts in cooperation with the soft processor, while retaining the real-time responsiveness of the operating system. More specifically, the hardware peripherals perform the following power management functionalities: 1) control the on-chip clock distribution network for driving the soft processor, its hardware peripherals, and the bus interfaces between them; 2) perform task and interrupt management responsibilities of the operating system when the soft processor is turned off; and 3) selectively wake up the soft processor and its hardware components, and put them into proper activation states based on the hardware resource requirements of the tasks under execution. The implementations of two popular real-time operating systems on a state-of-the-art FPGA device are presented. Measurements on an experimental board show that the proposed power management scheme can lead to significant power savings.

**Index Terms**—Field-programmable gate arrays (FPGAs), operating systems, power management, soft processors.

## I. INTRODUCTION

THE integration of various heterogeneous hardware components have made field-programmable gate arrays (FPGAs) an attractive choice for implementing many embedded systems. *Soft processors*, which are RISC processors realized using the configurable resources available on FPGA devices, are becoming popular. Examples of soft processors include Nios from Altera [1], LEON3 from Gaisler [6], and MicroBlaze from Xilinx [25]. One advantage of soft processors is their *configurability*, which allows instruction set customization and tight attachment of customized hardware peripherals for computation speed-up and efficient cooperative management.

Real-time operating systems (RTOSs) are widely used in software development, and many of them have been ported to run on soft processors. While RTOSs may encompass a wide range

of characteristics, our research focuses on the following typical features.

- *Multitasking*: The operating system can run multiple tasks simultaneously through context switch. Task scheduling is based on the priorities of the tasks ready to run. The operating system always picks tasks with the highest priority for execution.
- *Determinism*: Execution times of most OS functionalities are deterministic and do not depend on the number of tasks running in the user application. The user can calculate the time the OS takes to execute a function or a service.
- *Interrupt Management*: When an interrupt occurs, the corresponding interrupt service routine (ISR) is executed. If a higher priority task is awakened up by an interrupt, this task runs when all interrupt service routines are completed.

Power efficiency is an important performance metric in the development of many applications. Improving the power efficiency of RTOSs running on soft processors is highly desired. We consider an FPGA device configured with a soft processor and a set of hardware peripherals. Software drivers for controlling the hardware peripherals are provided. The target application running on the processor consists of a set of tasks. Each task is given as a C program. Tasks can invoke the corresponding hardware peripherals through these software drivers to perform some specific input/output (I/O) operation or computations. The execution of a task may involve only a portion of the on-chip hardware resources. For example, a task may require only the processor and the memory blocks that store the program instructions and data for this task. Its execution may not require the I/O interfacing hardware peripherals. Tasks are either executed periodically or invoked by some external interrupts.

Based on these assumptions, our objective is to customize the soft processor, and adapt it to the real-time operating system running on it, so that the power consumption of the complete system is minimized. The power reduction is achieved through the various on-chip configurable hardware resources and power management mechanisms. The required changes to the software source code of the operating system should be minimized during customization.

We propose a cooperative management scheme for power efficient implementations of real-time operating systems on soft processors. The basic idea of our approach is to integrate on-chip power management mechanisms and employ customized hardware-based cooperative management components by utilizing the *configurability* offered by soft processors. Customized hardware components are tightly coupled with the

Manuscript received May 16, 2006; revised July 23, 2007. This work was supported in part by the United States National Science Foundation (NSF) under Award CCR-0311823. An earlier version of this paper was presented at the IEEE International Symposium on Field-Programmable Custom Computing Machines, 2005.

J. Ou is with Xilinx, Inc., San Jose, CA 95124 USA (e-mail: jingzhao.ou@xilinx.com; jingzhao.ou@gmail.com).

V. K. Prasanna is with the University of Southern California, Los Angeles, CA 90089 USA (e-mail: jingzhao.ou@gmail.com).

Digital Object Identifier 10.1109/TVLSI.2007.912111

soft processor and perform the following power management functionalities: 1) manage the clock sources for driving the soft processor, its hardware peripherals, and bus interfaces; 2) perform the task and interrupt management responsibilities of the operating system when the processor is turned off; 3) selectively wake up the processor and its hardware components based on the hardware resource requirements of the tasks under execution. While there has been extensive research on performance improvement through hardware-software codesign, this paper is the first work that illustrates how hardware-software codesign can lead to power efficient RTOSs in the context of FPGAs. The implementation of two real-time operating systems based on a state-of-the-art soft processor are presented. Measurements on an FPGA experimental board show that our cooperative management scheme achieves power savings between 73%–90% for the various execution scenarios considered in our experiments. Our technique introduces negligible management overheads compared with the corresponding software implementations.

This paper is organized as follows. Section II discusses the background information and related work on real-time operating systems and the customization of FPGA-based soft processors. Section III presents our power efficient cooperative management technique. To illustrate our cooperative power management technique and demonstrate its effectiveness, Sections IV and V describe the implementations of two popular operating systems on a state-of-the-art soft processor. We conclude in Section VI.

## II. BACKGROUND AND RELATED WORK

### A. Power Efficiency of FPGAs

FPGAs provide superior performance to general-purpose processors (GPPs) and DSPs in terms of computation capability per power consumption [2], and have been deployed in many portable devices (e.g., handheld camcorders [27], software defined radio [28]), and automotive systems [29], etc. Power efficiency is a crucial performance metric in the development of these usually battery operated embedded systems due to the limited power supply.

Novel technologies have been proposed and integrated into modern FPGAs that can effectively reduce both static and dynamic power consumption. The triple oxide process technology and the six-input lookup table (LUT) architecture in the Xilinx 65-nm Virtex-5 FPGAs lead to 40% reduction in static power consumption compared with the prior generation of FPGAs [26]. The six-input LUT architecture, along with a symmetric routing structure in Virtex-5 FPGAs result in 35%–40% core power reduction compared with the prior Virtex-4 FPGAs. Dual- $V_{DD}$  low-power FPGA architectures have been proposed by both Gayasen, *et al.* [8] and Li, *et al.* [5]. An overall power reduction of 61% on a multimedia testbench is reported by employing the dual- $V_{DD}$  technology. Power efficiency placement and implementation algorithms can further reduce the FPGA power dissipation [9].

Modern FPGAs offer many on-chip power management mechanisms that can be exploited by the application developers for further power reduction. The Virtex-5 FPGAs contain up

to 12 digital clock management (DCM) units, in addition to numerous clock buffers and multiplexers, which can independently drive specific regions on the FPGA device depending on specific application requirements. The Xilinx Spartan-digital signal processing (DSP) devices provide suspend mode and hibernate mode, which yield at least 40% and 99% static power reduction, respectively, when they are not processing data.

### B. Customization of Soft Processors

There are many efforts to customize soft processors for performance optimization. Cong, *et al.* propose shadow registers to better utilize the limited data bandwidth of soft processors [3]. In their technique, the core register file of the soft processor is augmented by an extra set of shadow registers. These shadow registers are conditionally written by the soft processor in the write-back stage, and are read only by its hardware peripherals.

Shannon, *et al.* propose a programmable controller with a systems integrating modules with predefined physical links (SIMPPL) system computing model as a flexible interface for integrating various on-chip computing elements [20]. The programmable controller allows the addition of customized hardware peripherals as computing elements. The computing elements within the controller can communicate with each other through a fixed physical interface. Their approach enables users to easily adapt the controller to the new computing requirements without redesigning other elements.

Hauck, *et al.* propose an optimization technique based on run-time reconfiguration units to adapt the processor to the ever changing computing requirements [30]. Sun, *et al.* propose a scalable synthesis methodology to customize the instruction set of the processor for a specific set of application [23]. Hubner, *et al.* have a design based on multiple soft processors for automotive applications. In their designs, each of the multiple soft processors is optimized to perform some specific management responsibilities.

### C. Real-Time Operating Systems

Several commercial real-time operating systems have been ported to soft processors, such as ThreadX [24], MicroC/OS-II [15], etc.,. In the MicroC/OS-II port on the MicroBlaze processor from Micrium [16], the operating system runs a dummy *idle* task when no useful task waits for execution and no interrupt is present. Xilinx provides *xilkernel*, a real-time operating system for MicroBlaze [25]. Aside from the various benefits offered by these real-time operating systems, none of these implementations of RTOSs makes use of the *configurability* of soft processors for power reduction.

Mooney *et al.* proposes a system-on-chip (SoC) dynamic memory management unit (SoCDMMU) for managing the global memory shared between multiple on-chip processors [21]. They have modified an existing RTOS to support the SoCDMMU unit. They extend their work and propose a more generic hardware-software RTOS framework for SoC platforms and FPGAs [17]. A user can choose to implement some of the OS functionalities in either software or hardware using their framework, and a customized operating system is generated based on the user's selection. Significant execution speed-ups are reported using their SoCDMMU unit and the codesign

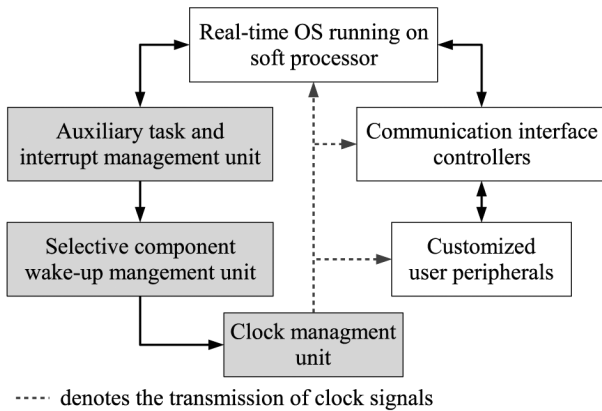


Fig. 1. Overall hardware architecture.

framework. Nevertheless, their research does not focus on improving the power efficiency of RTOSs, which is the major contribution of our work.

To the best of our knowledge, the cooperative power management scheme proposed in this paper is the first attempt to optimize the power performance of the RTOS running on soft processors by utilizing their *configurability*.

### III. OUR APPROACH

The basic idea of our cooperative management scheme is to have several tightly coupled hardware peripherals manage tasks and interrupts of the operating system in a cooperative manner with the soft processor. These hardware peripherals control the activation states of the various on-chip hardware components including the processor by utilizing on-chip power management mechanisms. They take over the task and interrupt management responsibilities of the operating systems when no task is ready for execution and/or no interrupt is present. The other hardware components, including the soft processor, are activated only when they are required to process “useful” tasks/interrupts. There are two factors in the power savings using our cooperative management scheme. One factor is that the power consumption of these attached dedicated hardware components for performing task and interrupt management are much smaller than that of the soft processor. The other factor is that we selectively wake up the hardware peripherals attached to the processor and put them into proper activation states based on the hardware resource requirements of the tasks under execution. Thus, the undesired power consumption of the hardware peripherals not required for execution can be eliminated, further reducing the power consumption of the FPGA device.

The overall hardware architecture of our cooperative power management scheme is shown in Fig. 1. Three hardware components (clock management unit, auxiliary task and interrupt management unit, and selective component wake-up unit) are attached to the soft processor for power management.

- *Clock Management Unit*: The major functionality of the clock management unit is to provide explicit control access to the clock distribution network of the FPGA device. It accepts the control signals from other hardware components and change clock sources that drive these hardware components using clock gating and dynamic clock frequency switching.

The target FPGA device is divided into clock domains driven by different clock sources. Dynamic switching between clock sources with different operating frequencies can be accomplished with a few clock cycles. For example, when the processor communicates with a low-speed hardware peripheral, instead of running the processor in dummy software loops to wait for a response, the user can switch the processor and the related hardware peripherals to a clock source with a lower frequency. This can effectively reduce the power consumption of the FPGA device. Clock gating is another important technique used in FPGA designs for power reduction. The user can dynamically change the distribution of clock signals and disable the transmission of clock signals to the hardware components that are not in use.

- *Auxiliary Task and interrupt management (ATIM) Unit*: An ATIM unit is attached to the soft processor. The OS task scheduling and interrupt management information is shared between the ATIM unit and the soft processor. When no task is ready for execution and no interrupt is present, the ATIM unit sends signals to the clock management unit to disable the clock signal transmission to the soft processor and the other hardware peripherals. The ATIM takes over the OS task and interrupt management responsibilities. When the ATIM unit determines that a task is ready for execution or any external interrupt arrives, it will wake up the processor and the related hardware peripherals and hand the task and interrupt management responsibilities back to the processor.
- *Selective Component Wake-Up and Activation State Management Unit*: Observing that the task execution may use only a portion of the device, a selective wake-up state management mechanism is developed. Using the clock management unit, the FPGA device is divided into several clock domains. Each of the clock domains can be in either “active” or “inactive” (clock gated) state. We denote a combination of the activation states of these different clock domains an *activation state* of the device. It is the user’s responsibility to assign a task an appropriate activation state of the device in which the hardware components required by the task are all active. Thus, when a task is selected by the operating system for execution, only some specific components used by the task are driven by the clock sources. The unused components are kept in the clock gated state for power reduction.

The proposed cooperative management technique incurs negligible task and interrupt management overhead. It may even reduce the context switch and interrupt handing overhead in some cases due to the parallel processing capability of the hardware peripherals. Since the proposed cooperative management scheme is transparent to the normal operations of the soft processor, it can be applied to multi-threading operating systems and hardware platforms with multiple soft processors.

### IV. IMPLEMENTATION BASED ON MICROC/OS-II

An implementation of the MicroC/OS-II operating system on the MicroBlaze processor is shown in Fig. 2. Except for the pri-

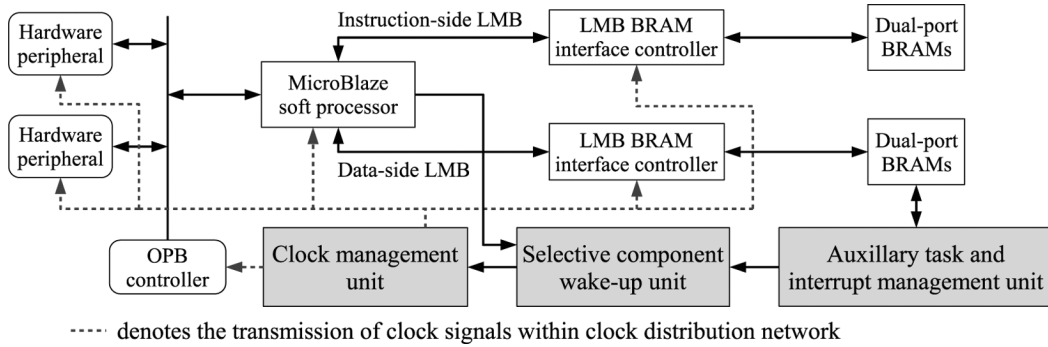


Fig. 2. Configuration of MicroBlaze soft processor with the proposed power management scheme.

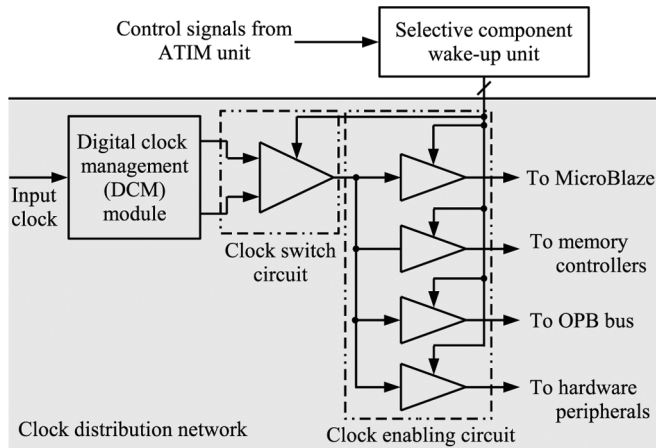


Fig. 3. Implementation of the clock management unit.

ority aliasing discussed in Section IV-D, our cooperative management technique is transparent to the operating system and minimizes the required OS kernel adaption.

#### A. Customization of the Microblaze Soft Processor

Instruction and program data of the MicroC/OS-II operating system are stored at on-chip memory blocks (BRAMs). MicroBlaze gets access to the instruction and the program data through two separate local memory bus (LMB) interfaces.

#### B. Clock Management Unit

The hardware architecture of the clock management unit is shown in Fig. 3. Xilinx Spartan-3/Virtex-II/Virtex-II Pro/Virtex FPGAs integrate on-chip digital clock management (DCM) modules. Each DCM module can provide different clock sources (CLK0, CLK2X, CLKDV, CLKFX, etc.), each of which can be used to form a clock domain and drive the hardware components within its own domain. Different on-chip hardware components can operate under different operating frequencies. For example, on Spartan-3 FPGA devices, CLKDV of the DCM module can divide the input clock by up to 16 times. For the design examples shown in Sections IV and V, when the input clock frequency is 50 MHz, the output clock frequency of CLKDV can be as low as 3.125 MHz.

The clock switch circuit is realized using multiplexers (i.e., BUFGMUXs) on the clock distribution network. It can dynamically switch the different clock sources for driving the soft pro-

cessor and its hardware peripherals. The clock enabling circuit consists of buffers with enable ports BUFGEs on the clock distribution network. It can dynamically drive hardware components only when they are required for execution to realize clock gating.

The clock management unit accepts control signals from the selective component wake-up and activation state management unit discussed in selective, and change the clock sources for driving the soft processor and other hardware components accordingly. Such changes of activation states can complete within one or two clock cycles upon the request of other management hardware components. It causes negligible time overhead compared with the software context switch overhead analyzed in Section IV-E.

#### C. Auxiliary Task and Interrupt Management (ATIM) Unit

The ATIM unit performs three major duties when the processor is turned off: ready task list management, OS clock tick management, and interrupt management.

- *Ready Task List Management:* MicroC/OS-II maintains a ready task list to monitor whether the tasks are ready for execution. Each entry on the ready task list is 1 bit that represents a task of the operating system. The list contains a fixed number of 64 entries (i.e., a total of 64 bits). Each entry is assigned with a unique number between 0 and 63. Thus, MicroC/OS-II allows up to 64 tasks to execute concurrently. When the value of a ready task list entry equals 0, the task represented by this entry is not ready for execution. Otherwise, when the value of a ready task list entry equals 1, the corresponding task is ready for execution. A task has an execution priority, equal to its entry number on the representing ready task list. A task with a smaller priority number has higher priority for execution. Whenever the ready task list indicates that there are tasks ready for execution, MicroC/OS-II always picks up the task with the smallest priority number (i.e., one with the highest priority) for execution. A context switch is performed if the task selected for execution has higher priority than the task under execution. As mentioned in Section II-C, MicroC/OS-II always runs the idle task with the lowest priority (represented by entry 63 on the ready task list) when there is no other “useful” task ready for execution.

The ready task list is stored at a specific location in the BRAMs. The MicroBlaze processor accesses through port

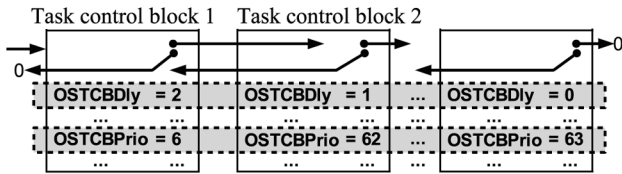


Fig. 4. Linked list of task control blocks.

A of the BRAMs, while the ATIM component accesses it through port B. The ATIM component checks the ready task list periodically. Upon detecting that only the idle task is ready for execution, it signals the clock management unit to disable the clock sources to the processor and its hardware peripherals. These disabled components are activated when there are useful tasks ready for execution and/or external interrupts present.

- *OS Clock Tick Management:* OS clock ticks are a special kind of periodic interrupts generated by a dedicated timer to keep track of the time experienced by MicroC/OS-II. Each time-out interrupt corresponds to one clock tick. MicroC/OS-II maintains an 8-bit counter for counting the number of experienced clock ticks. Upon receiving a time-out interrupt, MicroC/OS-II increases the clock tick counter by one. Clock ticks are used to keep track of execution delays and timeouts. A task can be repeatedly executed based on clock ticks. It can also stop waiting for an interrupt that fails to occur within a certain number of clock ticks. We use a special hardware component to perform the OS clock tick management and separate it from the management of interrupts for other components through the OPB bus. By doing so, frequently powering the processor and the OPB bus interface on and off to process the clock tick interrupts, which is required by normal interrupt handling and would result in unnecessary power consumption, can be avoided.

Once a task is created, MicroC/OS-II assigns a data structure (task control block) to maintain the status of the task when it is preempted. As shown in Fig. 4, the task control blocks are organized as a linked list. When the task regains controls of the CPU, the task control block allows the task to resume execution from the previous state where it has stopped. The task control block contains a field OSTCBDly, which is used when a task needs to be delayed for a certain number of clock ticks or a task needs to pend for an interrupt to occur within a timeout. In this case, OSTCBDly field contains the number of clock ticks the task is allowed to wait for the interrupt to occur. When this variable is 0, the task is not delayed or has no timeout when waiting for an interrupt. At each MicroC/OS-II clock tick, the operating system decreases OSTCBDly by 1 until OSTCBDly = 0. When OSTCBDly returns to 0, the corresponding slot on the ready task list representing this task is set accordingly, marking this task ready for execution. The OSTCBDly field for each task is stored at a specific location in the dual-port BRAMs. The MicroBlaze processor accesses to OSTCBDly through port A of the dual-port BRAMs, while the ATIM component accesses it through

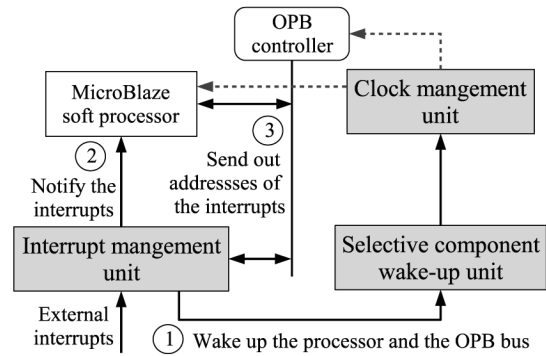


Fig. 5. Interrupt management unit.

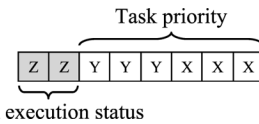


Fig. 6. Priority aliasing for selective component wake-up and activation state management.

port B. The ATIM unit takes over the management responsibilities of MicroC/OS-II when the MicroBlaze processor is turned off by the clock management unit. The ATIM unit decreases the OSTCBDly fields of the task control blocks when an OS clock tick interrupt occurs. When the OSTCBDly field of some tasks reaches zero, denoting that these tasks are ready for execution, the ATIM unit signals the clock management unit, which further brings up the processor and the related hardware components to process the tasks.

- *Interrupt Management:* The architecture of the interrupt management unit is shown in Fig. 5. When external interrupts arrive, the interrupt management checks the status of the MicroBlaze processor and the OPB bus controller. In the case that neither of them is active, the interrupt management unit performs the following operations: 1) send control signals to the selective component wake-up unit to enable the processor and the OPB controller; 2) notify the processor of the external interrupts; 3) when the processor responds and starts querying, send the interrupt vector through the OPB bus and notify the soft processor of the sources of the incoming external interrupts. If both the processor and the OPB bus controller are already in the active state, the interrupt management unit skips the wake-up operations and perform operations 2) and 3) directly.

#### D. Selective Component Wake-Up and Activation State Management Unit

To minimize the required changes to the software portion of the MicroC/OS-II operating system, we employ a technique called *priority aliasing* to realize the selective component wake-up and activation state management unit. As shown in Fig. 6, MicroC/OS-II uses an 8-bit unsigned integer to store the priority of a task. Since MicroC/OS-II supports a maximum number of 64 tasks, only the last 6 bits of the unsigned integer number are used to denote the priority of a task. When applying

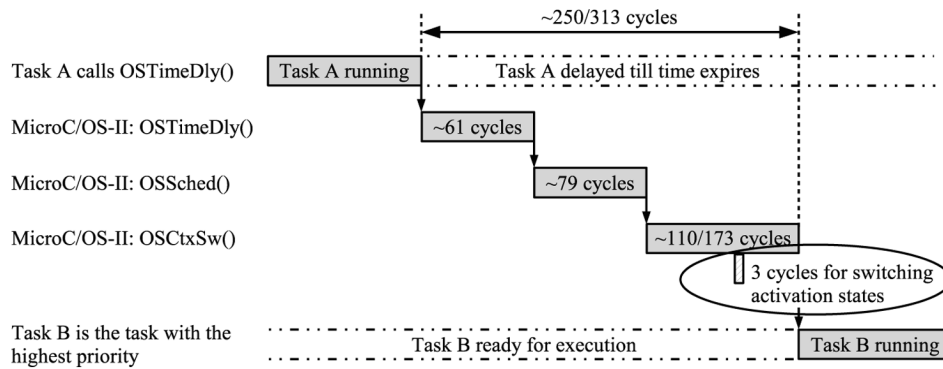


Fig. 7. Context switch overhead of MicroC/OS-II with the cooperative management scheme .

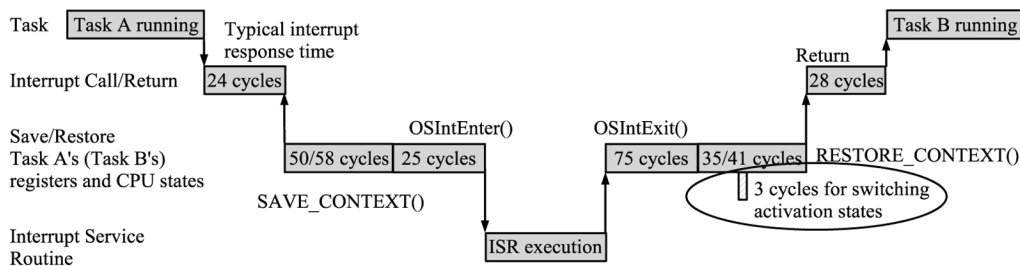


Fig. 8. Typical interrupt overhead of MicroC/OS-II with the cooperative management scheme.

the “priority aliasing” technique, we use the first 2 bits of a task’s priority to denote the four different combinations of activation states of the different hardware components. Thus, the user can assign a task with a specific priority using four different task priority numbers. Each of these four numbers corresponds to one activation state of the FPGA device. It is up to the user to assign an appropriate activation state of the device to a task by making appropriate changes to the clock management unit and the selective component wake-up management unit.

With “priority aliasing,” the processor can still change the activation state of the device during its normal operation depending on the tasks being executed. As shown in Fig. 4, OSTCBPrio in the task control blocks is stored at specific locations in the dual-port BRAMs and are accessible to both the auxiliary task and interrupt management unit, and also the MicroBlaze processor. When MicroBlaze is active, it sends the first two bits of the task priority numbers to the selective wake-up management unit and specifies the activation states of the device before executing a task.

### E. Management Overhead

Fig. 7 shows the typical context switch overhead of MicroC/OS-II. Task A notifies OSTimeDly() upon finishing execution. A context switch is then performed, which selects task B for execution. While OSCtxSw() is called during context switch to save the context of task A and restore the context of task B, MicroC/OS-II uses one clock cycle to issue a request to the selective component wake-up management unit. The request is processed by both the selective component wake-up management unit and the clock management unit to change the activation state of the FPGA device to that specified by task B. Note that the change of activation states performed by the

selective component wake-up management unit and the clock management unit is in parallel with the other context switch operations by the MicroBlaze processor. Our technique incurs only one extra management clock cycle of overhead, which is negligible considering the typical context switch overheads are between 250 to 313 clock cycles.

Fig. 8 illustrates the typical interrupt overhead of MicroC/OS-II. Task B, with a higher priority than task A, runs after the interrupt service routine (ISR). Similar to Fig. 8, our cooperative technique incurs one extra management clock cycle of overhead for interrupt handling.

### F. Illustrative Application Development

A fast Fourier transform (FFT) application is used to show the power savings yielded by the cooperative management scheme during various typical task execution and interrupt handling scenarios in many embedded systems.

- *Customization of MicroBlaze:* Consider the MicroBlaze configuration shown in Fig. 2. Two customized hardware peripherals are attached to MicroBlaze through the OPB bus interface. An opb\_gpio hardware peripheral accepts data coming from an 8-bit GPIO interface. An opb\_uartlite hardware peripheral communicates with an external computer through serial universal asynchronous receiver-transmitter (UART) protocol.

For selective component wake-up and activation state management unit, the first two bits of a task priority are used to denote four different activation states of the device. “00” represents the activation state for which only the processor and the two LMB bus controllers are active. In this case, the processor can execute tasks with instructions and data accessible through the LMB bus controllers. Access

TABLE I  
DYNAMIC POWER CONSUMPTION OF THE FPGA DEVICE

State	Dynamic power (W)	Reduction*	Note
00	0.21	57%	@ 50 MHz
01	0.46	6%	@ 50 MHz
10	0.03	94%	@ 6.25 MHz
11	0.49	—	@ 50 MHz

\*: Power reduction is compared against that of state 11.

to other peripherals is not allowed in the “00” activation state. The experimental results shown in Table I indicate that the “00” status reduces the power consumption of the device by 24% compared with activation state “11,” which indicates that all the hardware components are active. The “01” state denotes that the processor, the two memory controllers, the OPB bus controller, and the general purpose I/O hardware peripherals are active. The “10” state denotes that the processor, the two memory controllers, the OPB bus controller, and the hardware peripheral for communication through the serial port are active. Note that in states “00,” “01,” and “11,” the FPGA device is driven by CLK0 while in state “10,” it is driven by CLKDV, which is 8× slower than CLK0.

- *FFT Application:* There are three tasks in the FFT application. The *data-input* task stores the input data from the 8-bit opb\_gpio hardware peripheral to BRAMs. The *data-input* task is awakened by an external interrupt generated by the general purpose I/O hardware peripheral (GPIOs) when data is available. The *FFT computation* task performs a 16-point complex number FFT computation. We consider the int data type. The  $\cos()$  and  $\sin()$  functions are realized through table look-up. When input data arrives, the MicroBlaze processor receives an interrupt from the 8-bit GPIO. MicroC/OS-II running on MicroBlaze will execute the interrupt service routine (ISR), which stores the data from GPIO and marks the FFT computation task to be ready in the task ready list. Then, after the data input ISR completes, MicroC/OS-II will begin processing the FFT computation task. The data output task is executed repeatedly with a user-defined interval. Through an opb\_uartlite hardware peripheral, the data output task sends the results of the FFT computation task to a computer through an RS-232 serial port for display. Each data sample is sent at a fixed 0.05 ms interval. MicroBlaze runs an empty *for* loop between the transmissions. We also vary the input and output data rates in our experiments and show the average power reductions in these scenarios.
- *Experimental Setup:* For the experimental setup, the MicroBlaze processor was configured on a Xilinx Spartan-3 xc3s400 FPGA [25]. The input clock frequency to the FPGA device was 50 MHz. The MicroBlaze processor, the two LMB interface controllers as well as the other hardware components shown in Fig. 2 operated at the same clock frequency. An on-chip digital clock management (DCM) module was used to generate clock sources with different operating frequencies ranging from 6.25 to 50 MHz, to drive these hardware components. Power measurements were performed using a Spartan-3 board from

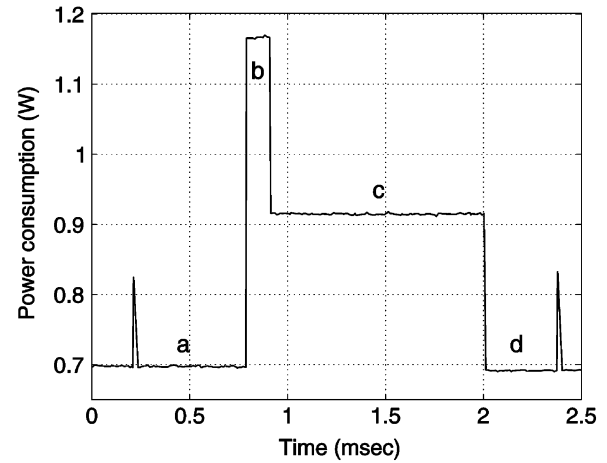


Fig. 9. Instant power consumption when processing the data-input task and the FFT computation task.

Nu Horizons [18], and a SourceMeter 2400 from Keithley [14]. We compared the differences in power consumption of the FPGA device when MicroC/OS-II is operated with different activation states. We ensured that, except for the Spartan-3 FPGA chip, all the other components on the prototyping board (e.g., the power supply indicator, the SRAM chip) were kept in the same operating state when the FPGA device executed under different activation states. Under these settings, we consider that the changes in power consumption of the FPGA prototyping board were mainly caused by the FPGA chip. Using the Keithley SourceMeter, we fixed the input voltage to the FPGA prototyping board at 6 V and measured the changes of input current to it. Dynamic power consumption of the MicroC/OS-II operating system was then calculated based on the changes in input current.

- *Experimental Results:* Table I shows the power consumption of the FPGA device in different activation states when MicroC/OS-II is processing the FFT computation task. Note that the measurement results shown in Table I account only for the differences in dynamic power consumption caused by these different activation states. Quiescent power, which is the power consumed by the FPGA device when there is no switching activity, is ignored. On the one hand, our experimental setup was unable to measure the quiescent power accurately. On the other hand, we did not consider powering on/off of the whole FPGA device, which involves reloading the configuration files. Quiescent power was fixed in our experiments. With this assumption, power reductions ranging from 6% to 95% and 53% on average were achieved by selectively waking up the various hardware components.

Fig. 9 shows the instant power consumption of the FPGA device when MicroC/OS-II was processing the data-input task and the FFT computation task. At time interval “a” only the ATIM unit was active. All the other hardware components were inactive. At time interval “b” input data was presented. The GPIO peripheral raised an interrupt to notify the ATIM unit of the availability of

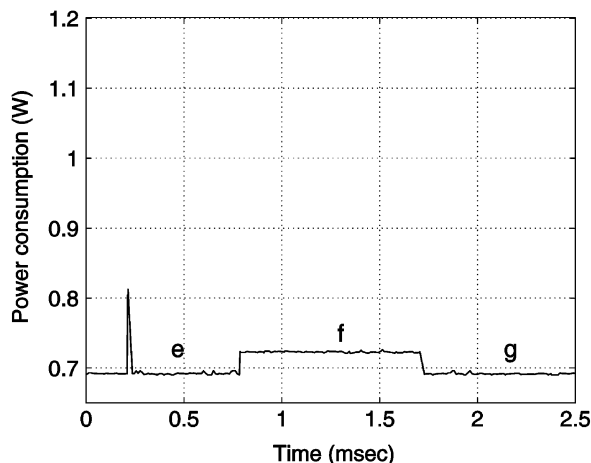


Fig. 10. Instant power consumption when processing data-output task.

input data. Upon receiving the interrupt, the ATIM unit put the FPGA device into activation state 01 and waked up the MicroBlaze processor and other hardware peripherals. MicroC/OS-II began processing the data input interrupt and stored the input data at the BRAMs through the data-side LMB bus. MicroC/OS-II also changed the ready task list and marked the FFT computation task ready for execution. Note that in order to better observe the abrupt changes of power consumption during time interval “b,” dummy software *for* loops were added to the interrupt service routine. At time interval “c,” the MicroBlaze processor sent commands to the selective wake-up management unit through an FSL channel and changed the activation state of the device to state 00. Once the FPGA device entered the desired activation state, MicroC/OS-II started executing the FFT computation task. Finally, at time interval “d,” MicroC/OS-II already finished processing the data-input interrupt and the FFT computation task. By checking the ready task list and the task control blocks, the ATIM unit detected that there are no tasks ready for execution. It then automatically disabled the clock signal transmission to all the hardware components including MicroBlaze and takes over the management responsibilities of the processor.

In Fig. 10, we show the instant power consumption of the FPGA device when MicroC/OS-II was processing the data-output task. At time interval “e” the ATIM unit was active and was managing the status of the tasks. The ATIM unit decreased the OSTCBDly fields of the task control blocks. All the other hardware components including MicroBlaze were inactive and were put into clock gated states. At time interval “f,” the ATIM unit found that the OSTCBDly field for the data-out task reached zero, which indicates that this task was ready for execution. The ATIM then marked the corresponding bit in the ready task list that represents this task to be 1. It also changed the FPGA device to activation state 10 according to the first two bits of the data-output task’s priority number. Under activation state 10, the MicroBlaze soft processor was awakened to execute the data-out task and sent the results of the FFT computation task through the `opb_uartlite` hardware peripheral. Time interval “g” is similar to time interval “d”

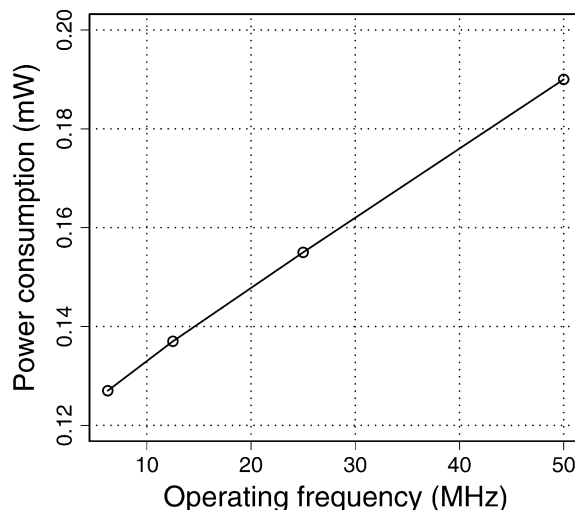


Fig. 11. Power consumption of the FPGA device for processing one instance of the data-output task with different operating frequencies.

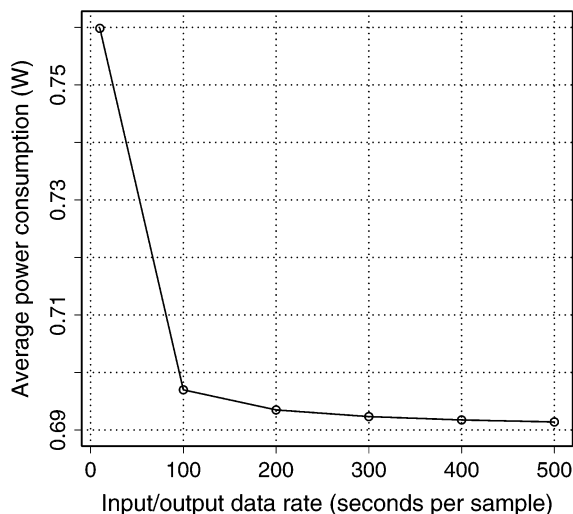


Fig. 12. Average power consumption of the FPGA device with various data I/O rates.

shown in Fig. 9. Seeing that no task was ready for execution, the ATIM unit disabled the other hardware components including MicroBlaze and resumed its management responsibility.

Fig. 11 shows the power consumption of the FPGA device when MicroC/OS-II was processing the data-out task and MicroBlaze operated with various operating frequencies. Power reductions ranging from 18% to 36% and 29% on average were achieved compared with the case when the MicroBlaze processor and its peripherals were always operating at 50 MHz. The power reductions come from the lowering of operating frequencies of the processor, rather than waiting using dummy software executions, when communicating with low-speed hardware peripherals.

Fig. 12 shows the power consumption of the FPGA device with various data input and output rates. Our power management scheme led to power reduction ranging between 37% to 43% for the different execution scenarios

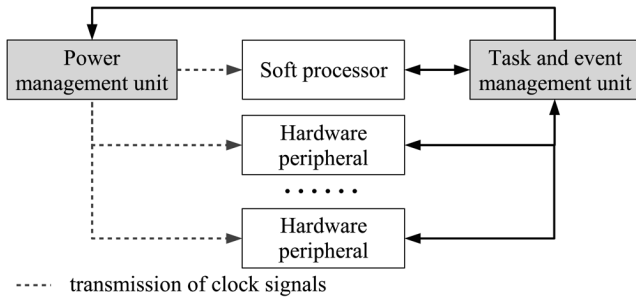


Fig. 13. Hardware architecture of *h-TinyOS*.

considered in our experiments. Lower data input and output rates led to higher power savings by our technique.

## V. IMPLEMENTATION BASED ON TINYOS

In this section, we show the development of *h-TinyOS*, an implementation of *TinyOS* with the proposed cooperative management scheme. Through the customization of the target soft processor, *h-TinyOS* has a much higher power efficiency than the “pure” software implementation *TinyOS*.

### A. Component-Based Operating Systems

Component-based operating systems provide a set of reusable system *components*. Each component performs some specific duties (e.g., OS services and FFT computation). Implementations of the components can be software programs or thin software wrappers around hardware components. An application is built by connecting the components using *wiring specifications* independent of the component implementations.

- *Task- and Event-Based Concurrency*: *nesC* is an ANSI C-based system programming language that supports the development of component-based systems. *nesC* provides two kinds of concurrency: *tasks* and *events*. Tasks are deferred executions while events signify either the completion of a split-phase operation (discussed in the following) or the occurrence of interrupts.
- *Split-Phase Operations*: Since tasks in *nesC* run nonpreemptively, the operations of the tasks with long latency are executed in a *split-phase* manner. Interfaces in *nesC* are *bidirectional*. One component can request the other component to perform an operation by issuing a *command* through an interface. For a *split-phase* operation, the commands return immediately. The component can be notified of the completion of the operation by implementing the corresponding *event* for the operation through the same interface.

See [7] and [11] for more discussions of *nesC* and *TinyOS*.

### B. Hardware Architecture

The overall hardware architecture of *h-TinyOS* is shown in Fig. 13. The duties of the customized hardware peripherals for power management are discussed in the following.

1) *Hardware-Based Task and Event Management (TEM) Unit*: The TEM unit maintains a task list. When a component posts a task, the information about the task is put into the task

list. Each task in the task list is associated with a priority number (i.e., the *pri* variable of the FFT computation task shown in Fig. 15). The management hardware peripheral always selects the task in the task list with the highest priority for execution. The TEM unit also accepts incoming events and invokes the corresponding hardware peripherals to process them. One advantage of the hardware-based TEM unit is efficient task scheduling. The management hardware peripheral can identify the next task for execution within a few clock cycles while the corresponding software implementation usually takes tens of clock cycles. Another advantage is that the soft processor can be turned off when the TEM unit is in operation, thus reducing the power consumption of the system. The soft processor is only awakened to perform useful computations. This is in contrast to the state-of-the-art operating systems including *TinyOS* [11], where the soft processor must be active and perform management duties.

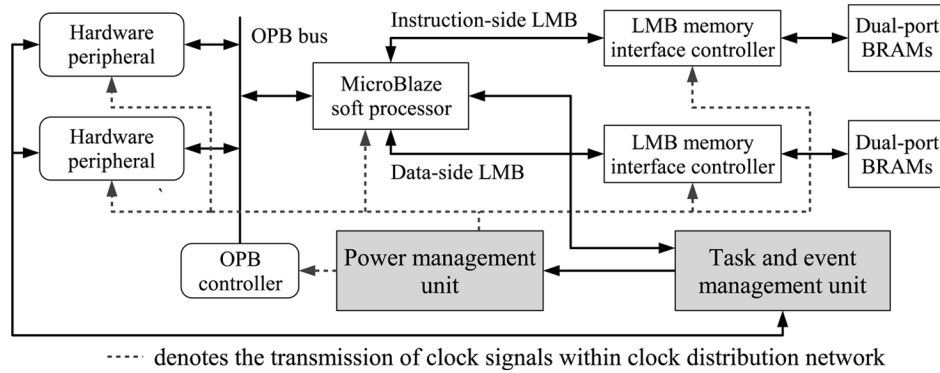
2) *Explicit Power Management Unit*: The power management unit manages the activation states of the various hardware components on the FPGA device. The power management unit provides a software interface, through which *h-TinyOS* can send out commands and explicitly control the activation status of the FPGA device. The power management unit is realized through the clock management unit discussed in Section IV-B. Similar to the selective component wake-up unit discussed in Section IV-D, each combination of the activation states of the on-chip hardware components of interest is represented by a unique state number. The application designer associates a *state* variable with a task. The *state* variable specifies the combination of the activation states of the hardware components (including the soft processor) for task execution. The *state* variable is pushed into the task list maintained by the TEM unit. Before a selected task begins execution, the TEM unit sends out the value of the *state* variable to the power management unit, which switches the FPGA device into the desired activation state before the selected task starts execution.

3) *Split-Phase Operations for Computations Using Hardware Peripherals*: *Split-phase operations* are used to reduce the power consumption of long computations in hardware peripherals. A component can post a task, which will issue a *command* requesting a *split-phase operation* in the hardware peripherals when selected for execution by the management hardware peripheral. The management hardware peripheral will turn off the soft processor and turn on only the hardware peripherals required for computation when the *split-phase* operation is under execution. For a component that needs to be notified of the completion of the *split-phase* operation, the application designer can wire it to the component that requests the *split-phase* operation and implement the *command* function in the wiring interface connecting the two components.

### C. Illustrative Application Development

Similar to Section IV, an FFT application is used to demonstrate the effectiveness of *h-TinyOS*. The configuration of MicroBlaze is shown in Fig. 14.

- *FFT Application*: As shown in Fig. 15, an FFT module provides a software wrapper over a customized hardware peripheral for performing the actual computation. The FFT

Fig. 14. Architecture of *h-TinyOS* on MicroBlaze.TABLE II  
VARIOUS ACTIVATION STATES FOR THE FFT APPLICATION SHOWN IN FIG. 16 AND THE POWER CONSUMPTION OF THE FPGA BOARD

Activation states	Combination of activation states	Power (W)	Reduction
0	MicroBlaze + GPIO32	1.16	3% *
1	MicroBlaze + UART (@ 6.25 MHz)	0.72	39% *
2	All On (for comparison)	1.19	—
3	FFT	0.72	32% **
4	MicroBlaze + FFT (for comparison)	1.06	—

\* : compared against state 2; \*\* compared against state 4.

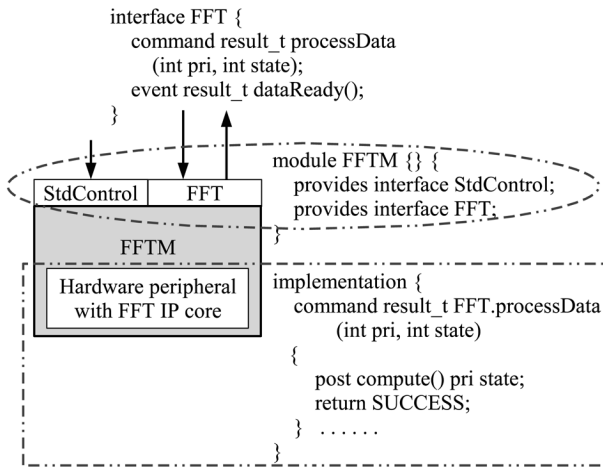
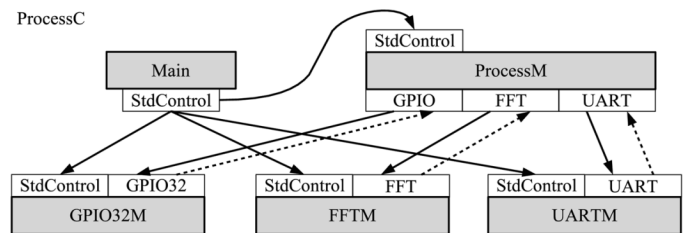


Fig. 15. Implementation of an FFT module.

module provides a *StdControl* interface and an *FFT* interface for interaction with other modules of the operating system. The *StdControl* interface is used to perform initialization during system start-up. The *FFT* interface provides command *processData()*. Invoking *processData()* posts an FFT computation task to the TEM unit. The application design associates a priority (i.e., *pri*) and an execution status (i.e., *status*) with the FFT computation task. The TEM unit determines when the FFT computation task is ready for execution based on its priority. When the FFT computation task starts execution, it invokes the wrapped hardware peripheral in the FFT module to process the data stored at the BRAMs. Before the FFT computation task starts execution, the TEM unit puts the FPGA device into the power status, which is specified by variable *status* associated with the task.

*Split-phase operations* are employed for the FFT computation task. The FFT hardware peripheral is processing

Fig. 16. Top-level configuration *ProcessC* of the FFT computation application.

the input data with MicroBlaze turned off. As shown in Table II, this reduces the power consumption of the complete system by 81%. Any modules that need to notify the completion of the FFT computation task can implement the *dataReady()* event of the FFT interface. In this case, the TEM unit will then wake up MicroBlaze to process these modules when the FFT module finishes.

The top-level configuration of the FFT application is realized as a *ProcessC* module shown in Fig. 16. It consists of five major modules: a *Main* module for performing initialization on system start-up, a *GPIOM* module for accepting the data from a 32-bit general purpose I/O (GPIO) port (i.e., the data input task), an *FFTM* module for wrapping a hardware peripheral for FFT computation (i.e., the FFT computation task), a *UARTM* module for sending the result data to a personal computer through a serial port (i.e., the data output task), and a *ProcessM* module for wiring the other four modules together.

- *Experimental Results:* The experimental setup of *h-TinyOS* was similar to Section IV-F. The power consumption of the FPGA board was shown in Table II. Activation states of 0 and 1 corresponded to the states for executing the data input and the data output tasks, respectively. With the explicit power management functionalities provided by *h-TinyOS*, power reductions of 3% and 39% for the FPGA board

were achieved for these two tasks. For activation state 2, the power management unit shown in Fig. 14 let the MicroBlaze processor and the UART hardware peripheral run with operating frequencies between 6.25 and 50 MHz. The power consumption for these different operating frequencies was shown in Fig. 11. By dynamically slowing down the operating frequency of the FPGA device when executing the data output task, power reductions up to 33% was achieved. When an application requires a fixed data output rate determined by the personal computer, the power reduction can directly lead to power reduction for the data output task.

Activation state 3 denoted that the FFT computation task was executed through *split-phase* operations. MicroBlaze was turned off while the FFT computation was performed in the hardware peripheral. Activation state 2 represented the state that was set by *TinyOS*, the “pure” software implementation, for executing the FFT computation task. Thus, for the FFT computation task, *h-TinyOS* reduced 32% of the power consumption of the FPGA board. Besides, compared with state 4, which represents that activation state that was set by the MicroC/OS-II configuration discussed in Section IV.F, *h-TinyOS* eliminated 81% of dynamic power consumption by turning off MicroBlaze.

The power reduction that can be achieved by *h-TinyOS* depends on the specific application requirements. For various data input and output rates of the FFT application, the average power consumption of *h-TinyOS* is similar to Fig. 12. Lower data input and output rates lead to higher power savings by our technique.

#### D. Management Overhead

*h-TinyOS* has a quick five to ten clock cycle event response time. Context switch is within three clock cycles: one clock cycle for notifying the completion of a task, one clock cycle for identifying the next task with the high priority for execution, and one clock cycle for changing the activation state of the FPGA device (if needed). The “pure” software *for* loop-based *TinyOS* implementation requires 20 to 40 clock cycles for context switch.

#### E. Hardware Resource Usage

For the examples discussed in Sections IV and V, the base MicroBlaze processor systems occupied around 1400 LUTs. The proposed tightly coupled power management hardware peripherals occupied between 50 to 100 LUTs. Our cooperative management technique incurred a small amount of additional hardware resource usage to achieve power reduction.

### VI. CONCLUSION

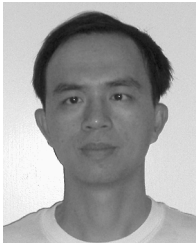
We proposed a cooperative management technique for power efficient implementation of real-time operating systems on FPGAs. Two real-time operating systems were used to demonstrate the effectiveness of the proposed technique. Hard processor cores embedded in configurable logic are also becoming popular. The PowerPC processor core on Xilinx Virtex-4 FPGAs allows configurable logic to be involved in the instruction decoding process [25]. The proposed cooperative

management technique can be used to improve the power efficiency of real-time operating systems running on them.

### REFERENCES

- [1] Altera, Inc., San Jose, CA, “Altera homepage,” (2007). [Online]. Available: <http://www.altera.com>
- [2] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, “Energy-efficient signal processing using FPGAs,” in *Proc. ACM Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2003, pp. 225–234.
- [3] J. Cong, Y. Fan, G. Han, A. Jagannathan, G. Reinman, and Z. Zhang, “Instruction set extension with shadow registers for configurable processors,” in *Proc. Int. Conf. Field-Program. Gate Arrays (FPGA)*, 2005, pp. 99–106.
- [4] Crossbow Technology, Inc., San Jose, CA, “Motes, smart dust sensors, wireless sensor networks,” (2007). [Online]. Available: <http://www.xbow.com/>
- [5] F. Li, Y. Lin, and L. He, “FPGA power reduction using configurable dual-Vdd,” in *Proc. IEEE/ACM Des. Autom. Conf. (DAC)*, 2004, pp. 735–740.
- [6] Gaisler Research, Inc., Goteborg, Sweden, “LEON3 user manual,” (2007). [Online]. Available: <http://www.gaisler.com/>
- [7] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, “The nesC language: A holistic approach to networked embedded systems,” in *Proc. Int. Conf. Program. Lang. Des. Implementation (PLDI)*, 2003, pp. 1–11.
- [8] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, “A Dual-VDD low power FPGA architecture,” in *Proc. Int. Conf. Field Program. Logic its Appl. (FPL)*, 2004, pp. 145–157.
- [9] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, “Reducing leakage energy in FPGAs using region-constrained placement,” in *Proc. Int. Conf. Field-Program. Gate Arrays (FPGA)*, 2004, pp. 51–58.
- [10] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1995.
- [11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, “System architecture directions for networked sensors,” in *Proc. Int. Conf. Arch. Support Program. Lang. OSs*, 2000, pp. 93–104.
- [12] M. Horowitz, T. Indermaur, and R. Gonzalez, “Low-power digital design,” in *Proc. IEEE Int. Symp. Low Power Electron. Des. (ISLPED)*, 1994, pp. 8–11.
- [13] M. Hubner, K. Paulsson, and J. Becker, “Parallel and flexible multiprocessor system-on-chip for adaptive automotive applications based on Xilinx microblaze soft-cores,” in *Proc. Reconfigurable Arch. Workshop (RAW)*, 2005, p. 149.
- [14] Keithley Instruments, Inc., Cleveland, OH, “Keithley homepage,” (2007). [Online]. Available: <http://www.keithley.com/>
- [15] J. J. Larbrosse, *MicroC/OS-II The Real-Time Kernel*, 2nd ed. Gilroy, CA: CMP Books, 2002.
- [16] Micrium, Inc., Weston, FL, “Micrium, Inc. homepage,” (2007). [Online]. Available: <http://www.micrium.com/>
- [17] V. J. Mooney and D. M. Blough, “A hardware-software real-time operating system framework for SoCs,” *IEEE Des. Test Comput.*, vol. 19, no. 6, pp. 44–51, Nov./Dec. 2002.
- [18] Nu Horizons Electronics Corp., Melville, NY, “Nu Horizons Electronics Corp. homepage,” (2007). [Online]. Available: <http://www.nuhorizons.com/>
- [19] J. Ou and V. K. Prasanna, “COMA: A cooperative management scheme for energy efficient implementation of real-time operating systems on FPGA based soft processors,” in *Proc. IEEE Intel. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2005, pp. 139–148.
- [20] L. Shannon and P. Chow, “Simplifying the integration of processing elements in computing systems using a programmable controller,” in *Proc. IEEE Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2005, pp. 63–72.
- [21] M. Shalan and V. J. Mooney, “Hardware support for real-time embedded multiprocessor system-on-a-chip memory management,” in *Proc. Int. Symp. Hardw./Softw. Codes.*, 2002, pp. 79–84.
- [22] V. Subramonian, H.-M. Huang, and S. Datar, “Priority scheduling in TinyOS: A case study,” Washington Univ., St. Louis, MO, 2003.
- [23] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha, “A scalable application-specific processor synthesis methodology,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, 2003, pp. 283–290.
- [24] Express Logic, Inc., San Diego, CA, “ThreadX user guide,” (2007). [Online]. Available: <http://www.expresslogic.com/>
- [25] Xilinx, Inc., San Jose, CA, “Xilinx, Inc. homepage,” (2007). [Online]. Available: <http://www.xilinx.com/>

- [26] Xilinx, Inc., San Jose, CA, "Power analysis of 65 nm Virtex-5 FPGAs," (2007). [Online]. Available: <http://www.xilinx.com/bvdocs/whitepapers/wp246.pdf>
- [27] Xilinx, Inc., San Jose, CA, "Xilinx Spartan-3E FPGAs enable JVC's GY-HD250," (2006). [Online]. Available: [http://digitalcontentproducer.com/hdhdv/prods/xilinx\\_hdv\\_11212006/](http://digitalcontentproducer.com/hdhdv/prods/xilinx_hdv_11212006/)
- [28] Xilinx, Inc., San Jose, CA, "Xilinx SDR radio kit wins 2006 portable design editor's choice award," (2007). [Online]. Available: [http://www.xilinx.com/prs\\_rls/2007/xil\\_corp/0733\\_pdawards.htm](http://www.xilinx.com/prs_rls/2007/xil_corp/0733_pdawards.htm)
- [29] Xilinx, Inc., San Jose, CA, "Xilinx demonstrates industry's first scalable 3-D graphics hardware accelerator for automotive applications," (2007). [Online]. Available: [http://www.xilinx.com/prs\\_rls/2007/end\\_markets/0703\\_xylon3dCES.htm](http://www.xilinx.com/prs_rls/2007/end_markets/0703_xylon3dCES.htm)
- [30] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: A high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Proc. Int. Symp. Comput. Arch. (ISCA)*, 2000, pp. 225–235.



**Jingzhao Ou** received the B.S. and M.S. degrees in electrical engineering from the South China University of Technology, Guangzhou, China, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles.

He is currently with the DSP Design Tools and Methodologies Group, Xilinx, Inc., San Jose, CA. His main research interests include hardware-software codesign and energy efficient application development using reconfigurable hardware.



**Viktor K. Prasanna** (F'96) is Charles Lee Powell Professor of Engineering with the Ming Hsieh Department of Electrical Engineering and Professor with the Computer Science Department, the University of Southern California (USC), Los Angeles. He served as the Division Director for the Computer Engineering Division during 1994–1998. His research interests include parallel and distributed systems including networked sensor systems, embedded systems, configurable architectures, and high performance computing. He has published extensively and consulted for industries in these areas.

Dr. Prasanna was a recipient of the 2005 Okawa Foundation Grant. He is an associate member of the Center for Applied Mathematical Sciences (CAMS), USC. He has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He is the Steering Co-chair of the International Parallel and Distributed Processing Symposium [merged IEEE International Parallel Processing Symposium (IPPS) and the Symposium on Parallel and Distributed Processing (SPDP)] and is the Steering Chair of the International Conference on High Performance Computing (HiPC). He has served on the editorial boards of the *Journal of Parallel and Distributed Computing*, *PROCEEDINGS OF THE IEEE*, *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, and *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*. He served as the Editor-In-Chief of the *IEEE TRANSACTIONS ON COMPUTERS* during 2003–2006. He was the founding Chair of the IEEE Computer Society Technical Committee on Parallel Processing. (See <http://ceng.usc.edu/~prasanna> for further details.)