

COMA: A COoperative MAnagement Scheme for Energy Efficient Implementation of Real-Time Operating Systems on FPGA Based Soft Processors

Jingzhao Ou and Viktor K. Prasanna
Department of Electrical Engineering, University of Southern California
Los Angeles, California, 90089-2560 USA
Email: {ouj, prasanna}@usc.edu

Abstract

FPGA based soft processors are an attractive choice for implementing many embedded systems. As real-time operating systems are adopted in the development of many applications using soft processors, we propose COMA, a COoperative MAnagement scheme in this paper for energy efficient implementation of real-time operating systems on soft processors. By utilizing the configurability of soft processors, we tightly couple several customized energy management hardware peripherals to them. These hardware peripherals cooperatively manage tasks and interrupts together with the processor while retaining the real-time responsiveness of the operating system. More specifically, they perform the following functionalities: (1) control the clock distribution network for driving the processor, the hardware peripherals and the communication interfaces between them; (2) take over the task and interrupt management responsibility of the operating system when the processor is shut off; (3) selectively wake up the processor and the corresponding hardware components for task execution based on the configurations of the processor and the hardware resource requirements of the tasks. We implement a real-time operating system on a popular soft processor to illustrate our approach. We show the development of an embedded application on the operating system enhanced with our energy management techniques. Actual measurements on an FPGA board demonstrates that our energy management scheme leads to energy reductions ranging from 73.3% to 89.9% and 86.8% on the average for the various execution scenarios considered in this paper.

I. Introduction

There is a strong trend towards integrating FPGA with various heterogeneous hardware components, such as RISC processors, embedded multipliers, memory blocks (e.g. Block RAMs in Xilinx FPGAs), etc. Such integration leads to a reconfigurable System-on-Chip (SoC) platform, an attractive option for implementing many embedded systems. *FPGA based soft processors*, which are

RISC processors realized using the configurable resources available on FPGA devices, are also becoming popular. Examples of such processors include Nios from Altera [1], MicroBlaze and PicoBlaze from Xilinx [10]. One advantage of designing using soft processors is that they provide new design trade-offs by time sharing the limited hardware resources available on the devices. Many control and management functionalities as well as computations with tightly coupled data dependency between computation steps (e.g. many recursive algorithms) are inherently more suitable for software implementations on processors than the corresponding (parallel) hardware implementations. Their software implementations are more compact and require much smaller amount of resources. Such compact designs can effectively reduce the static energy dissipation by fitting into smaller FPGA devices [9]. Most importantly, soft processors are “configurable” by allowing the customization of the instruction set and/or the attachment of customized hardware peripherals in order to speed up the calculation of some algorithms and/or to perform some auxiliary management functionalities as described in this paper. The Nios processor allows users to customize up to five instructions. The MicroBlaze processor supports various dedicated communication interfaces for attaching customized hardware peripherals to it.

Real-time operating systems (RTOSs) provides many effective mechanisms for task and interrupt management and are adopted in the development of many embedded applications using soft processors. RTOSs simplify the application development process by splitting the application code into separate tasks. New functions can be added to the RTOSs without requiring major changes to the software. Most importantly, when adding low priority tasks, their impacts on the responsiveness of the system can be kept to a minimum extend when using real-time operating systems. For preemptive operating systems, time-critical tasks and interrupts are handled as quickly and as efficiently as possible.

Energy efficiency is a key performance metric in the design of many embedded systems. Modern FPGAs offer many on-chip energy management mechanisms to improve the energy efficiency of designs using them. Users can

dynamically stop the distribution of clock signals to some specific components when the operations of these components are not required. Many FPGA devices provide multiple clock sources with different operating frequencies and clock phases. Portions of the device can dynamically switch to operate with a slow clock frequency and thus the energy dissipation when communicating with low-speed off-chip peripherals can be reduced. Use of these on-chip energy management mechanisms is crucial for implementing real-time operating systems using FPGA based soft processors for energy efficiency. We address the following key design problem in this paper.

- *Problem definition:* The FPGA device is configured with one soft processor and several on-chip customized hardware peripherals. The processor and the hardware peripherals communicate with each other through some specific bus interfaces. There are a set of tasks to be run on the FPGA device. Each of the tasks is given as a C code. Software drivers for controlling the hardware peripherals are provided. Tasks can invoke the corresponding hardware peripherals through these software drivers in order to perform some I/O operations or to speed up the execution of some specific computations. The execution of a task may involve only a portion of on-chip hardware resources. For example, the execution of a task may require the processor, a specific hardware peripheral, and the bus interface between them while the execution of another task may only require the processor, the memory interface controllers and the memory blocks that store the programs and data for this task. Tasks are either executed periodically or invoked by some external interrupts. Tasks have different execution priorities. Tasks with higher priorities are always selected for execution and can preempt the execution of tasks with lower priorities. Besides, The rates at which the tasks are executed is considered to be “infrequent”. That is, there are intervals during which no tasks are under execution. Based on these assumptions, our objective is *to implement a real-time operating system on an FPGA based soft processor which can manage the activation status of various components of the FPGA device during its operation so that the energy dissipation for executing the tasks on the FPGA device is minimized.* We focus on the utilization of on-chip programmable resources and energy management mechanisms to achieve our objective. Also, it is desired that the required changes to the source code of the operating systems are kept minimal when applying the energy management techniques.

In this paper, we propose COMA, a cooperative management scheme for energy efficient implementation of real-time operating systems on soft processors. By utilizing the configurability of soft processors, we tightly couple several customized hardware components to them which perform the following functionalities for energy management: (1) manage the clock sources for driving the processor, the hardware peripherals and the bus in-

terfaces between them; (2) take over the task and interrupt management responsibilities of the operating system when the processor is shut off; (3) selectively wake up the processor and the corresponding hardware components for task execution based on the configurations of the processor and the hardware resource requirements of the tasks. We implement a real-time operating system on a popular soft processor to illustrate our approach. The development of an embedded application is provided in Section V to demonstrate the effectiveness of our approach. Actual measurement on an FPGA prototyping board shows that the systems implemented using our power management techniques achieve energy reductions ranging from 73.3% to 89.9% and 86.8% on the average for the different execution scenarios of the application considered in our experiments.

The paper is organized as follows. Section II discusses background information about real-time operating systems. Section III describes COMA, our approach for energy efficient implementation of real-time operating systems on soft processors. Section IV presents an implement of our approach using a popular operating system on Xilinx MicroBlaze [10] to illustrate our power management techniques. The development of an embedded FFT application are shown in Section V to demonstrate the effectiveness of our approach. Finally, we conclude in Section VI.

II. Real-Time Operating Systems

A. Background

Real-time operating systems may encompass a wide range of different characteristics. We focus on the typical characteristics listed below in this paper. We retain these important features of real-time operating systems when applying our energy management techniques to improve their energy efficiency.

- *Multitasking:* The operating system is able to run multiple tasks “simultaneously” through context switching. Each task is assigned a priority number. Task scheduling is based on the priority numbers of the tasks ready to run. The operating system always picks up the task with the highest priority for execution.

- *Preemptive:* The task under execution may be intercepted by a task with higher priority or an interrupt. A context switching is performed so that the intercepted task can resume execution later.

- *Deterministic:* Execution times for most of the functions and services provided by the operating system are deterministic and do not depend on the number of tasks running in the user application. Thus, the user should be able to calculate the amount of time the real-time OS takes to execute a function or a service.

- *Interrupt management:* When interrupts occur, the corresponding interrupt service routines (ISRs) are executed. If a higher priority task is awakened as a result of a

interrupt, this highest priority task runs as soon as all nested interrupts are completed.

B. Off-The-Shelf Systems

Several commercial real-time operating systems have been ported to soft processors. ThreadX [7], a real-time operating system from Express Logic, Inc., has been ported to both MicroBlaze and Nios. In ThreadX, only the system services used by the application are brought into the run-time image and thus the actual size of the operating system is determined by the application running on it. MicroC/OS-II is a real-time operating system that supports all the important characteristics discussed above [4]. Each task within MicroC/OS-II has a unique priority number assigned to it. The operating system can manage up to 64 tasks. There is a port of MicroC/OS-II for MicroBlaze processor. In this implementation, the operating system runs a dummy *idle* task when no useful task waits for execution and no interrupt presents. Xilinx also provides a real-time operating system for application development on MicroBlaze. Therefore, besides from the various benefits offered by these real-time operating systems, none of them addresses the energy management issue in order to improve the energy efficiency of the systems when they are running on FPGA based soft processors.

TinyOS [8] is an operating systems designed for wireless embedded sensor networks. It adopts a component-based architecture, which leads to a tighter integration between user application code and the OS kernel than traditional operating systems. Not all the features of real-time operating systems discussed in Section II-A are supported in TinyOS. For example, it may delay the processing of tasks till the arrival of next interrupt. The unpredictable and fluctuate wireless transmission makes such full support unnecessary. TinyOS turns the processor into a low-power sleep mode when no processing is required and wakes it up when interrupts occur. Lacking the customized hardware peripherals proposed in this paper to take over the management responsibilities when the processor is in sleep mode, TinyOS would result in unnecessary wake-ups of the processor when processing tasks such as the periodic management of OS clock ticks.

C. Energy Management

For the commercial and academic real-time operating systems discussed above, all the task and interrupt management responsibilities of the operating systems are implemented in “pure” software. In these implementations, the operating systems have little or no control of the status of the processor and its peripherals. As is discussed in Section II, even though there are already several real-time operating systems ported to soft processors, to our best knowledge, there have been no attempts to perform customized “configuration” of the soft processors and provide energy management capability to the operating systems in

order to improve their energy efficiency. Dynamic voltage and frequency scaling (DVFS) technique has proved to be an effective method of achieving low power consumption while meeting the performance requirements [2]. DVFS is employed by many real-time operating systems. However, DVFS cannot be realized directly on the current reconfigurable SoC platforms and requires additional and relatively sophisticated off-chip hardware support. Thus, application of DVFS is out of scope for the design problem discussed in this paper. Another effective energy management technique is to power off the processors when no tasks are ready for execution. This technique has been used in the design of operating systems such as TinyOS [8]. In these designs, the hardware component that controls the powering on/off of the processors is loosely coupled with the operating systems and share little or no responsibilities of the operating system with the processor. For example, the OS clock ticks can only be managed by the processor for the design of MicroC/OS-II shown in [4]. This would result in unnecessary waking up of the processor and undesired energy dissipation in many cases. For TinyOS, such powering on and off may also cause delays for executing tasks with high priority. Besides, selective waking up of the hardware components depending on the resource requirement of the tasks is not realized in these designs.

III. Our Approach

The basic idea of COMA, our energy management approach is to tightly couple several dedicated hardware peripherals to the soft processor. These hardware peripherals cooperatively perform task and interrupt management responsibilities of the operating system together with the processor. These peripherals also control the activation states of the various on-chip hardware components including the processor by utilizing the on-chip energy management mechanisms. They take over the task and interrupt management responsibilities of the operating systems running on the soft processor when there is no task ready for execution. In this case, except for these management components, all the other hardware components including the processor can be completely shut off when they are not processing any tasks. By doing so, energy efficiency of the complete system is improved since the power consumption of these attached hardware components are much smaller than that of the processor. Besides, we selectively wake up the hardware peripherals attached to the processor based on the hardware resource usage of the tasks under execution. The undesired energy dissipation of the hardware peripherals that are not required for execution can be eliminated and the energy efficiency of the system is further improved.

The overall hardware architecture of our approach is shown in Figure 1. Three hardware components, which are

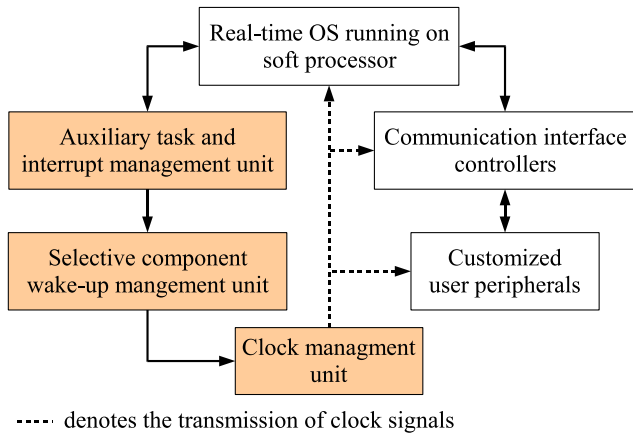


Fig. 1. Overall hardware architecture

clock management unit, auxiliary task and interrupt management unit and selective component wake-up unit, are added to the soft processor to perform energy management.

- *Clock management unit*: Many FPGA devices provide different clock sources on a single chip. The FPGA devices can then be divided into different clock domains. Each of the clock domains can be driven by different clock sources. Dynamic switching between the clock sources with different operating frequencies within a few clock cycles is also possible. For example, when the processor is communicating with a low-speed peripheral, instead of running the processor in some dummy software loops when waiting for the response from slow peripherals, the user can choose to switch the processor to work with a clock source with a slower operating frequency. This can effectively reduce the energy dissipated by the processor. Clock gating is another important technique used in FPGA designs to reduce energy dissipation. 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.

The clock management unit provides explicit control access to the clock distribution network of the FPGA device. It accepts the control signals from other components and change clock sources that drives the various hardware components using the two techniques discussed above.

- *Auxiliary task and interrupt management unit*: We attach an auxiliary task and interrupt management (ATIM) hardware peripheral to the soft processor. We let the internal data of the operating system for task scheduling and interrupt management be shared between the ATIM unit and soft processor. Hence, when the operating system determines that no task is ready for execution and no interrupt presents, the ATIM unit will send out signals to the clock management component unit and disables the transmission of clock signals to the soft processor and the other hardware peripherals attached to it. The ATIM takes over the task and interrupt management responsibilities

of the operating system. 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. Note that to retain the deterministic feature of the real-time operating system, dedicated bus interfaces with deterministic delays are used for communication between the processor and the other hardware components.

- *Selective component wake-up management unit*: Since the execution of a task uses only a portion of the device, we provide a selective wake-up state management mechanism within the OS kernel. 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 as an *activation state* of the device. It is the user’s responsibility that a task is assigned to 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 clock gated state in order to save power consumption.

IV. An Implementation

For illustration purpose, we implement a real-time operating system on the MicroBlaze soft processor configured with the proposed COMA management scheme which is shown in Figure 2. Except for the “priority aliasing” technique discussed in Section IV-D, our energy management techniques are transparent to the operating system and does not require changes to its software.

A. Configuration of MicroBlaze

The instruction and program data of the MicroC/OS-II operating system are stored at the BRAMs. The MicroBlaze processor gets access to the instruction and the program data through two LMB (Local Memory Bus) interface controllers, one for instruction side and the other for data side data access.

There are two different kinds of bus interfaces to attach customized hardware peripherals to MicroBlaze. One kind of bus interface is Faster Simplex Links (FSLs), which are dedicated interfaces for tightly coupling high speed peripherals to MicroBlaze. A 32-bit data can be sent between the processor and its hardware peripherals through FSLs in a fixed time of two clock cycles. To retain the real-time responsiveness of the operating system, we use FSLs when attaching the hardware peripherals that implement COMA to the MicroBlaze processor. Another interface is On-chip Peripheral Bus (OPB), a shared bus interface for attaching hardware peripherals that operate with a relative

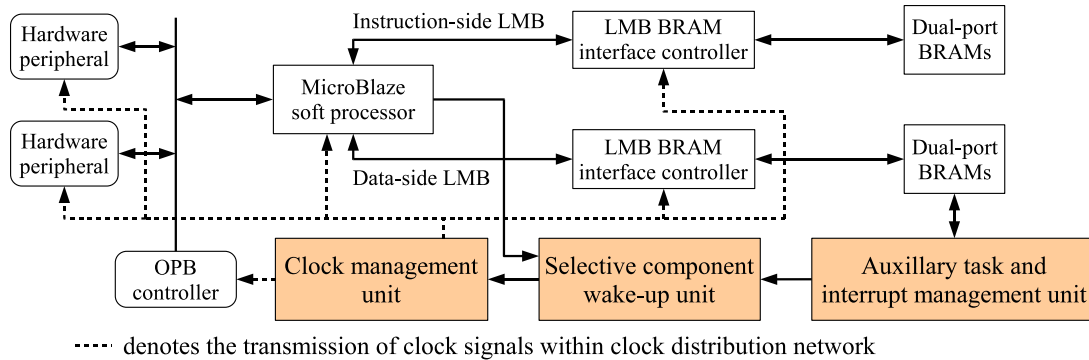


Fig. 2. Configuration of MicroBlaze soft processor with our energy management scheme

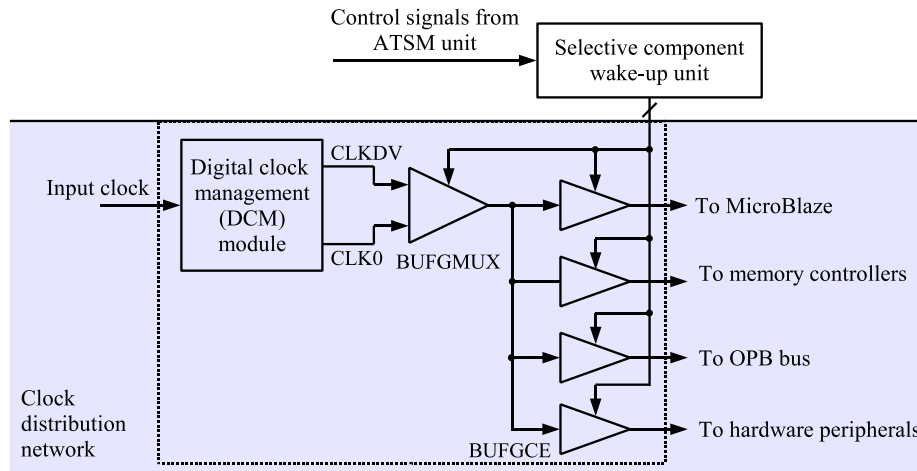


Fig. 3. Clock management unit

low frequency compared to the MicroBlaze processor. For example, peripherals for accessing general purpose input/output (GPIO) interfaces and for communication through serial ports can be attached to MicroBlaze through the OPB bus.

B. Clock Management Unit

The clock management unit is illustrated in Figure 3. Xilinx Spartan-3/Virtex-II/Virtex-II Pro/Virtex FPGAs integrate many on-chip digital clock management (DCM) modules. Each DCM module can provide different clock sources (CLK0, CLK2X, CLKDV, CLKFX, etc.). Each of these clock sources can be used to form a clock domain and drive the hardware components within its own domain. Thus, different hardware components can operate under different operating frequencies. For example, on Spartan-3, CLKDV of the DCM module can divide the input clock by up to 16. Considering the design examples in Section V, when the input clock frequency is 50 MHz, the output clock frequency of CLKDV can be as low as 6.25 MHz.

There are multiplexers (*i.e.* BUFGMUXs) within the clock distribution network. BUFGMUXs can be used to

dynamically switch the clock sources with different operating frequencies for driving the processor and some specific hardware peripherals. Besides, Xilinx FPGAs provide buffers with enable port (BUFGCEs) within their clock distribution network, which can be used to realize clock gating. These BUFGCEs be used to dynamically drive the corresponding hardware components only when these components are required for execution.

The clock management unit accepts control signals from the selective component wake-up management unit discussed in Section IV-D and change the clock sources for driving the soft processor, memory controllers, the OPB bus and the FSLs connected to the processor accordingly. The clock management unit changes the activation state of the FPGA device within one or two clock cycles and thus introduces negligible overhead to the system.

C. Auxiliary Task and Interrupt Management Unit

In order to take over the responsibilities of the operating systems when the processor is turned off, the auxiliary task and interrupt management unit performs three major functionalities: ready task list management, OS clock tick

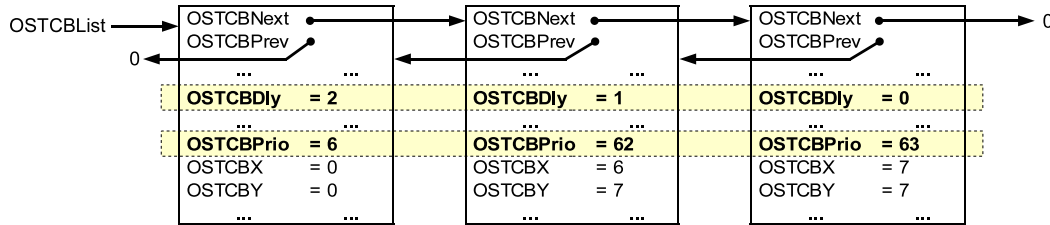


Fig. 4. Linked list of task control blocks

management, interrupt management. They are described in detail as follows.

- Ready Task List Management:** MicroC/OS-II maintains a ready list consisting of two variables, `OSRdyGrp` and `OSRdyTbl[]` to keep track of the task status. Each task is assigned a unique priority number between 0 and 63. As shown in Figure 5, tasks are grouped (eight tasks per group) and each group is represented by one bit in `OSRdyGrp`. `OSRdyGrp` and `OSRdyTbl[]` actually form a table. Each slot in the table represents a task with a specific priority. If the value of a slot equals to zero, it means that the task represented by this slot is not ready for execution. Otherwise, if the value of the slot is one, it means that the task represented by this slot is ready for execution. Whenever there are tasks ready for execution, the operating system searches the two variables in the ready list table to find out the task with highest priority that is ready to run. A context switch is performed if the selected task has higher priority than the task under execution. As mentioned in Section II-B, MicroC/OS-II always runs an idle task (represented by slot 63) with the lowest priority when there is no other task ready for execution.

We use some compiler construct to explicitly specify the storage location of the ready task list (*i.e.* `OSRdyGrp` and `OSRdyTbl[]`) on the dual-port BRAMs. The MicroBlaze processor can get access to `OSRdyGrp` and `OSRdyTbl[]` through the data side LMB bus controller attached to port A of the dual-port BRAMs while the ATIM component can also get access to these two variables through port B of the BRAMs. The ATIM component keeps track of the two variables of the ready list with a user-defined period. When it detects that only the idle task is ready for execution, it will signal the clock management unit to disable the clocks sent to the processor, its hardware peripherals and the bus interfaces between them. These “clock gated” components will resume their normal states when there are useful tasks become ready for execution and/or external interrupts presented.

- OS Clock Tick Management:** OS clock ticks are a special kind of interrupts that are used by MicroC/OS-II to keep track of the time experienced by the system. A dedicated timer is attached to the MicroBlaze processor and repeatedly generates time-out interrupts with a pre-defined interval. Each interrupt generated by this timer corresponds

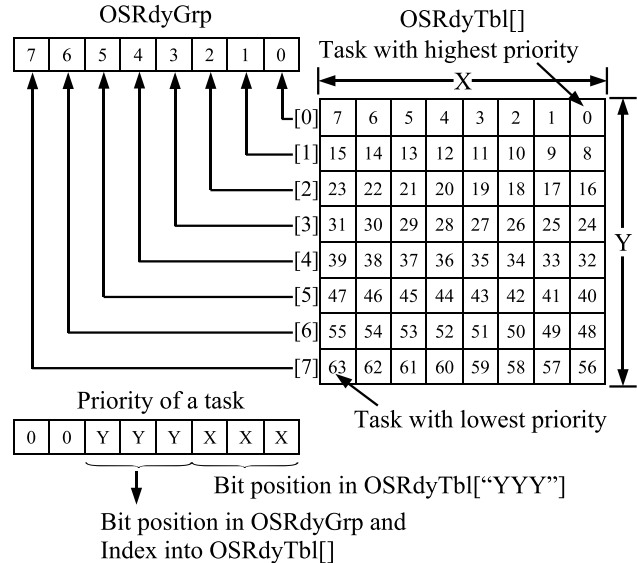


Fig. 5. Ready task list

to one clock tick. MicroC/OS-II maintains an 8-bit counter for counting the number of clock ticks experienced by the operating system. Whenever the MicroBlaze processor receives a time-out interrupt from the timer, MicroC/OS-II increases the clock tick counter by one. The counting of clock ticks is used to keep track of time delays and timeouts. For example, the period that a task is repeatedly executed is counted in clock ticks. Also, a task can stop waiting for an interrupt if the interrupt of interest fails to occur within a certain amount 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 on and off of the OPB bus controller to notify the processor of the clock tick interrupt, which is required by other interrupts and would result in unnecessary energy dissipation, can be avoided.

When a task is created, it is assigned a *task control block*, `OS_TCB`. A task control block is a data structure used by MicroC/OS-II to maintain the state of a task when it is preempted. The task control blocks are organized as a linked list, which is shown in Figure 4.

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. Especially, 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. MicroC/OS-II requires a periodic time source. Whenever the time source times out, the operating system decreases `OSTCBDly` by 1 till `OSTCBDly = 0`, which indicates that the corresponding task is ready for execution. The bits in `OSRdyGrp` and `OSRdyTbl` representing this task will then be set accordingly.

In order to apply our energy management techniques, the `OSTCBDly` field for each task is stored at a specific location of the dual-port BRAMs. The MicroBlaze processor gets access to `OSTCBDly` through the data side LMB bus controller attached to port A of the dual-port BRAMs. The ATIM component can also get access to `OSTCBDly` through port B of the BRAMs.

When the processor is turned off by the clock management unit, the ATIM unit will take over the management responsibilities of MicroC/OS-II. The ATIM unit gets access to the `OSTCBDly` fields of the task control blocks through another port of the dual-port BRAMs and decreases their values when an OS clock tick interrupt occurs. Whenever the `OSTCBDly` field of some tasks reaches zero, which means that these tasks are ready for execution, the ATIM unit will signal the clock management unit, which will bring up the processor and the corresponding hardware components to process the tasks ready for execution.

- **Interrupt Management:** The architecture of the interrupt management unit is shown in Figure 6. When external interrupts arrive, the interrupt management will check the status of the MicroBlaze processor and the OPB bus controller. If neither the processor nor the OPB bus controller are active, the interrupt management unit will perform the following operations: (1) send out 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 out the addresses of the sources of the external interrupts to the processor through the OPB bus. If both the processor and the OPB bus controller are already in active state, the interrupt management unit will skip the wake-up operation and go on directly with operation (2) and (3) described above.

D. Selective Component Wake-up Management Unit

To minimize the required changes to the MicroC/OS-II operating system, we employ a technique called “pri-

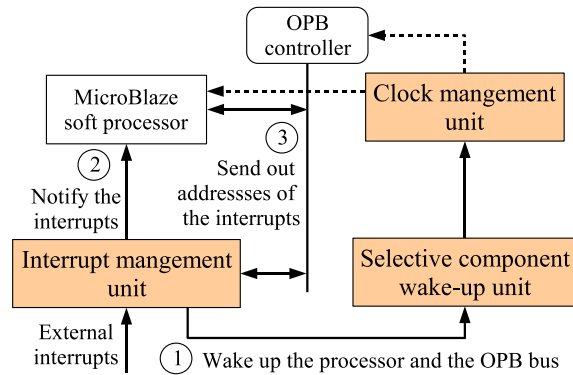


Fig. 6. Interrupt management unit

ority aliasing” to realize the selective component wake-up unit. As shown in Figure 5, 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 is used to denote the priority of a task. When applying 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 (*i.e.* an activation state of the FPGA device). 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. One possible setting of the two-bit information for the design example discussed in Section V is shown in Figure 7. The user can choose to support other activation states of the FPGA device by making appropriate changes to the clock management unit and the selective component wake-up management unit.

Note that when applying the “priority aliasing” technique, the processor can also change the activation state of the device during its normal operation depending on the tasks that are under execution. As shown in Figure 4, `OSTCBprio` in the task control blocks are stored at some specific locations on the dual-port BRAMs and are accessible to both the auxiliary task and interrupt management unit and the MicroBlaze processor through the dual-port memory blocks on the FPGA device. When MicroBlaze

Z	Z	Y	Y	Y	X	X	X
---	---	---	---	---	---	---	---

- 0 0 : processor, mem. controller
- 0 1 : processor, mem. controller, OPB, GPIO
- 1 0 : processor, mem. controller, OPB, UART
- 1 1 : all active

Fig. 7. Priority aliasing for selective component wake-up

is active, it can send the first two bits of the task priority numbers that determine the activation states of the device to the selective wake-up management unit through an FSL bus interface before it starts executing a task.

V. An Illustrative Example

In this section, we show the development of an FFT computation application using the MicroC/OS-II operating system running on a MicroBlaze processor enhanced with the proposed COMA scheme. FFT is a widely deployed applications. The scenarios of task execution and interrupt arrivals considered in our examples are typical in the design of many embedded systems such as radar and sensor network systems and software defined radio, where energy efficiency is a critical performance metric. We perform actual measurement on a commercial FPGA prototyping board to demonstrate the effectiveness of our approach.

A. Configuration of MicroBlaze

The configuration of the MicroBlaze processor is as shown in Figure 2. Two customized hardware peripherals are attached to MicroBlaze through the OPB bus. One peripheral is a `opb_gpio` peripheral that accepts data coming from an 8-bit GPIO interface. The other peripheral is a `opb_uartlite` peripheral that communicates with an external computer through serial UART (Universal Asynchronous Receiver-Transmitter) protocol.

Based on the configuration discussed above, the settings of the selective component wake-up unit is shown in Figure 7. The first two bits of a task priority denotes four different activation state of the device. “00” represents the activation state of the device that only the processor and the two LMB bus controllers are active. Under the activation state, the processor can execute tasks the instructions and data of which are stored in the BRAMs accessible through the LMB bus controllers. Access to other peripherals is not allowed in the “00” activation state. The experimental results in Section V-C show that the “00” status reduces the power consumption of the device by 23.6% compared with activation state “11” when all the hardware components are active. Other activation states of the device are: “01” stands for the state that the processor, the two memory controllers, the OPB bus controller, and the general purpose I/O hardware peripherals are active; “10” stands for the state that the processor, the two memory controllers, the OPB bus controller, and the hardware peripheral for communication through the serial port are active. Besides, as shown in Figure 3, in states “00”, “01” and “11”, the FPGA device is driven by CLK0 while in state “10”, it is driven by CLKDV, the operating frequency of which is 16 times less than that of CLK0.

B. An FFT Computation Application

The embedded application developed on MicroC/OS-II consists of one interrupt service routine for processing the input data from the 8-bit `opb_gpio` peripheral and two tasks. One task is for FFT computation while the other task is for sending out data to the external computer for displaying through the `opb_uartlite` peripheral. The FFT computation task performs a 16-point complex number FFT computation as described in [6]. We consider `int` data type. The `cos()` and `sin()` functions are realized through table look-up. When input data presents, 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 coming from the 8-bit GPIO at the BRAMs and mark 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 which controls the RS-232 serial port, the data output task sends out the results of the FFT computation task to a computer where the results get displayed. Each of the 8-bit data is sent out in a fixed 0.05 msec interval. MicroBlaze runs an empty *for* loop between the transmission intervals.

To better demonstrate the effectiveness of our energy management techniques, we generate the input data periodically and vary the input and output data rates in our experiments to show the average power reductions achieved when applying our energy management techniques.

For the experiments discussed in the paper, the MicroBlaze processor is configured on a Xilinx Spartan-3 xc3s400 FPGA [10]. The input clock frequency to the FPGA device is 50 MHz. The MicroBlaze processor, the two LMB interface controllers as well as the other hardware components shown in Figure 2 are operating at the same clock frequency. An on-chip digital clock management (DCM) module is used to generate clock sources with different operating frequencies ranging from 6.25 MHz to 50 MHz for driving these hardware components. We use EDK 6.3.02 for describing the software execution platform and for compiling the software programs. ISE 6.3.02 [10] is used for synthesis and implementation of the complete applications. Actual power consumption measurement of the applications is performed using a Spartan-3 prototyping board from Nu Horizons [5] and a SourceMeter 2400 from Keithley [3]. We compare the differences in power consumption of the FPGA device when MicroC/OS-II is operated with different activation states. We ensure 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) are kept in the same operating state when the FPGA device is executing under different activation states. Under these settings, we consider that the changes in power consumption of the

FPGA prototyping board are mainly caused by the FPGA chip. Using the Keithley SourceMeter, we fix the input voltage to the FPGA prototyping board at 6 Volts and measure the changes of input current to it. Dynamic power consumption of the MicroC/OS-II operating system is then calculated based on the changes of input current.

C. Experimental Results

Table I shows the power consumption of the FPGA device when MicroC/OS-II is processing the FFT computation task and the FPGA device is assigned to different activation states. Note that the measurement results in Table I only account 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 activities on it, is ignored. This is because quiescent power is fixed for a specific FPGA device and cannot be optimized using the techniques proposed in this paper. Power reductions ranging from 6.1% to 94.7% and 52.6% on average are achieved by selectively waking up the various hardware components through the clock management unit.

TABLE I. Dynamic power consumption of the FPGA device in different activation states

State	Power (W)	Reduction*	Note
00	0.212	57.1%	@ 50 MHz
01	0.464	6.1%	@ 50 MHz
10	0.026	94.7%	@ 6.25 MHz
11	0.494	–	@ 50 MHz

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

Figure 8 shows the instant power consumption of the FPGA device when MicroC/OS-II is processing the data-input interrupt and the FFT computation task. At time interval “a”, only the ATIM unit is active. All the other hardware components are inactive. At time interval “b”, input data is presented. The GPIO peripheral raises an interrupt to notify the ATIM unit of the incoming of the data. Upon receiving the interrupt, the ATIM unit turns the FPGA device into activation state 01 and wakes up the MicroBlaze processor and other hardware peripherals. MicroC/OS-II begins processing the data input interrupt and stores the input data at the BRAMs through the data-side LMB bus. MicroC/OS-II also changes the ready task list and marks the FFT computation task ready for execution. Note that in order to better observe the changes of power consumption during time interval “b”, some dummy loops are added to the interrupt service routine to extend its execution time. At time interval “c”, the MicroBlaze processor sends commands to the selective wake-up management unit through an FSL channel and changes the activation state of the device to state 00. Once the FPGA device enters the desired activation state, MicroC/OS-II starts executing the FFT computation task. Finally, at time interval “d”, MicroC/OS-II already finishes processing the data-input interrupt and the FFT computation task. By

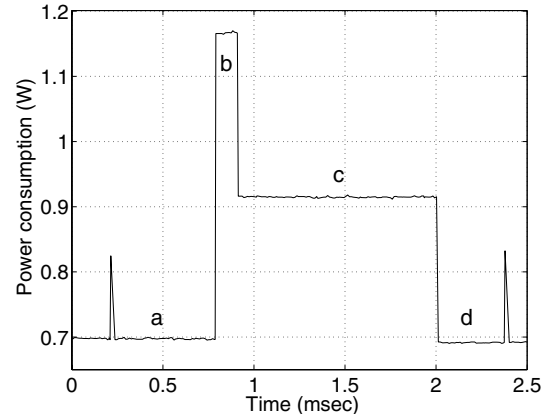


Fig. 8. Instant power consumption when processing data-input interrupt and FFT computation task

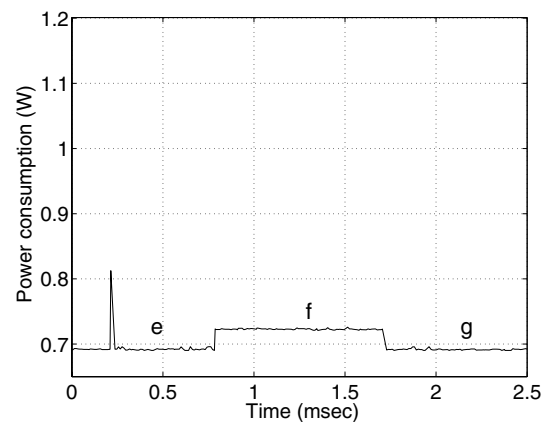


Fig. 9. Instant power consumption when processing data-output task

checking the ready task list and the task control blocks, the ATIM unit detects that there are no task ready for execution. It then automatically disables the transmission of clock signals to all the hardware components including the MicroBlaze processor and takes over the management responsibilities originally performed by MicroC/OS-II.

In Figure 9, we show the instant power consumption of the FPGA device when MicroC/OS-II is processing the data-output task. At time interval “e”, the ATIM unit is active and is managing the status of the tasks, which includes decreasing the OSTCBDly field of the task control blocks. All the other hardware components including MicroBlaze are inactive. At time interval “f”, the ATIM unit finds out that the OSTCBDly field for the data-out task reaches zero. It then marks this task ready for execution and changes the FPGA device to activation state 10 according to the first two bits of the data-output task’s priority number. The MicroBlaze processor is then woken up to execute the data-out task and send out the results of the FFT computation task through the `opb_uartlite`

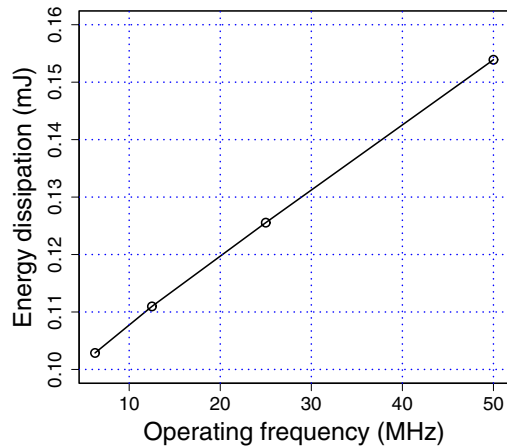


Fig. 10. Energy dissipation of the FPGA device for processing one instance of the data-out task with different operating frequencies

peripheral. Time interval “g” is similar to time interval “d” shown in Figure 8. As no tasks are ready for execution during this interval, the ATIM unit disables the other hardware components including the MicroBlaze processor and resumes its management responsibility in representation of MicroC/OS-II.

Besides, the arrows shown in Figure 8 and Figure 9 identify the spikes in power consumption. These spikes are caused by the ATIM unit when it is processing the interrupts for managing OS clock ticks. Limited by the maximum sampling rate that can be supported by the Keithley SourceMeter, we are unable to observe all the spikes caused by the OS clock tick management.

Figure 10 shows the power consumption of the FPGA device when MicroC/OS-II is processing the data-out task and the MicroBlaze processor is operating with different operating frequencies. Energy reduction ranging from 18.4% to 36.3% and 29.0% on average can be achieved by lowering the clock frequencies when communicating with the low-speed hardware peripherals.

Figure 11 shows the energy dissipation of the FPGA device with different data input and output rates. Our energy management scheme leads to energy reduction ranging from 73.3% to 89.9% and 86.8% on average for the different execution scenarios considered in our experiments. A lower data input and output rates, which imply a longer system idle time between task execution, lead to more energy savings when our energy management techniques is applied.

VI. Conclusion

COMA, a cooperative management technique for energy efficient implementation of real-time operating systems on FPGAs, is proposed in this paper. An implemen-

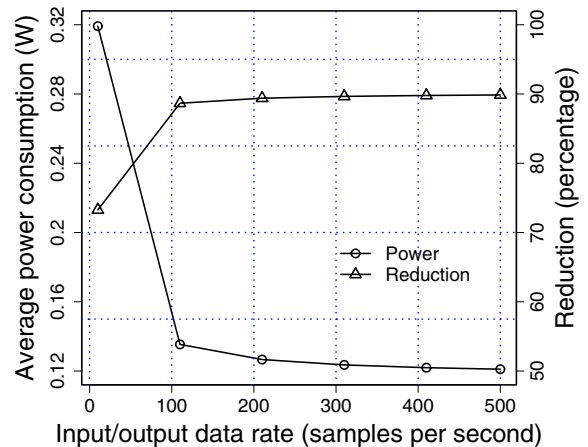


Fig. 11. Average power consumption of the FPGA device with different data input/output rates

tation of COMA using Xilinx FPGAs and the development of an embedded application are used to demonstrate the effectiveness of our approach.

In addition to soft processors, hard processor cores tightly coupled with the surrounding configurable logic are also becoming popular on reconfigurable SoC platforms. The PowerPC processor core available on Xilinx Virtex-4 FPGAs even provides dedicated interface through which configurable logic can get involved in the instruction decoding process [10]. We are currently working on applying our energy management techniques for developing real-time operating systems using these embedded hard processor cores.

VII. Acknowledgment

This work is supported by the United States National Science Foundation (NSF) under award No. CCR-0311823. The authors thank Aditya Kwatra for his editorial assistance.

References

- [1] Altera, Inc., <http://www.altera.com>.
- [2] M. Horowitz, T. Indermaur, and R. Gonzalez, “Low-Power Digital Design,” *IEEE Symp. on Low Power Electronics*, 1994.
- [3] Keithley Instruments, Inc., www.keithley.com.
- [4] J. J. Larbrosse, “MicroC/OS-II The Real-Time Kernel (2nd Edition),” *CMP Books*, 2002.
- [5] Nu Horizons Electronics Corp., www.nuhorizons.com.
- [6] W. Press, B. Flannery, S. Teukolsky, W. Vetterling, “Numerical Recipes in C: The Art of Scientific Computing (Second Edition),” *Cambridge University Press*, 2002.
- [7] “ThreadX User Guide,” *Express Logic, Inc.*, available online at <http://www.expresslogic.com>.
- [8] “TinyOS Documentation,” available online at <http://www.tinyos.net/tinyos-1.x/doc/>.
- [9] T. Tuan and B. Lai, “Leakage Power Analysis of A 90nm FPGA,” *IEEE Custom Integrated Circuits Conf. (CICC)*, 2003.
- [10] Xilinx, Inc., <http://www.xilinx.com>.