Regular Paper

Dynamic Power Management for Embedded System Idle State in the Presence of Periodic Interrupt Services

Gang Zeng,[†1] Hiroyuki Tomiyama[†1]
and Hiroaki Takada[†1]

Generally, there are periodic interrupt services such as periodic clock tick interrupts in the real-time embedded systems even though the system is in the idle state. To minimize the power consumption of idle state, power management therefore should consider the effect of periodic interrupt services. In this paper, we deal with this issue in two different cases. In case the periodic interrupt cannot be disabled, we formulate the power consumption of idle state, and propose static and dynamic approaches for the optimal frequency selection to save idle power. On the other hand, in case the periodic interrupt can be disabled, we propose the configurable clock tick to disable the interrupt service until the next task is released so that the processor can stay in the low power mode for longer time. The proposed approaches are implemented in a real-time OS; and its efficiency has been validated by theoretical calculations and actually measurements on an embedded processor.

1. Introduction

Energy consumption has become one of the major concerns in today's embedded system design especially for battery-powered devices. For the sake of safety, in real-time systems the utilization of processor is less than 100% even if all tasks run at WCET (worse case execution time). Moreover, workload of each task may vary from instance to instance, which results in the less average execution time than the WCET. All these factors lead to the system idle state in which there are no tasks needed to be scheduled. A common approach to save energy in idle state is to transition the processor into low power mode where the processor consumes the less power than run mode. However, use of low power mode is not free, in particular the transition from low power mode to run mode consumes

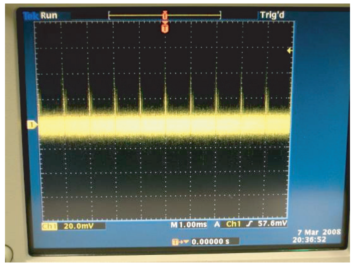


Fig. 1 Effect of periodic timer interrupt services on low power mode of M16C. (supply voltage: 3 V, current sense resistor: 2 ohm, time scale: 1 ms, voltage scale: 20 mV).

both time and energy cost. Meanwhile, a fact that may be neglected is that most real-time OS (RTOS) maintains a periodic clock interrupt to synchronize the system and trace the clock events even in the idle state. For example the uc/OS-II, eCOS, and Linux need a 10 ms clock interrupt to generate the system clock. Besides the periodic clock tick, some interrupt-driven embedded systems such as data acquisition systems also need periodic interrupts to activate the CPU from low power mode for data processing. **Figure 1** shows the captured idle state voltage waveform of a current sense resistor which is inserted between the power supply and the power pin of a M16C processor. From this figure, it is clear that the processor is periodically activated from low power mode by a 1 ms clock tick interrupt. As a result, these interrupt services offset the achieved energy savings in low power mode due to the imposed time and energy overhead for mode transitions. To tackle this problem, this paper seeks the optimal power management strategies for embedded system idle state to relieve the effects of periodic interrupt services.

Commonly, an embedded processor can provide multiple low power modes with different power consumption levels. To take advantages of these power control mechanisms, dynamic power management (DPM) tries to assign the optimal low power mode according to the predicted duration of the system idle state. As an example, **Fig. 2** shows the power mode transition graph for two typical embedded processors in high-end and low-end application, respectively. While the SA-1100 with integrated 32-bit RISC core is targeted at the high performance application, the M16C [11]) with integrated 16-bit CISC core, on-chip ROM and RAM is aimed

†1 Graduate School of Information Science, Nagoya University

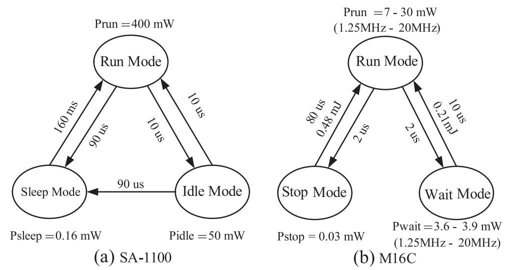


Fig. 2 Power mode transition for (a) StrongARM SA-1100 processor (b) M16C processor.

at low-end application. The SA-1100 processor provides three operation modes, i.e., run, idle, and sleep modes. The run mode is the normal operating mode with full functionalities and high power consumption. In contrast, the idle and sleep modes are low power modes with stopped CPU clock. Idle mode disables the CPU core clock but enables all peripheral clocks, thus on or off-chip interrupt service requests can quickly reactivate the CPU. To further save power, sleep mode stops both CPU and peripheral clocks. As a result, only hardware reset or special event can wakeup the CPU, which requires long transition time whenever entering or exiting the sleep mode. Similarly, the M16C also provides three power modes which have similar functionalities to that of the SA-1100 but with different names. However, the time and energy overhead of M16C for power mode transition is far less than that of SA-1100, which is benefited from its simple and single-chip architecture. Actually, only one instruction is needed to transition the processor into wait mode. In summary, we have the following observations: (1) Different low power mode has different power consumption level, which is achieved by disabling either CPU clock or both CPU and peripheral clocks. (2) The power mode transitions consume both time and energy cost which are dependent on the specified low power mode and the complexity of processors. Generally, the less power consumption the mode has, or the more complex the processor is, the more overhead it required to transition the processor back to the run mode.

Although the sleep mode of SA-1100 has the lowest power consumption, it is not suitable for the application considered in this paper. The reasons are as follows. First, the transition time overhead for returning to run mode is too large to be used in the application with short period of interrupt services. Second, the normal interrupt service requests using on-chip clock cannot work properly in the sleep mode. Therefore, we primarily consider the idle mode in our idle power management approach.

In addition to DPM, another effective technique for power reduction is dynamic voltage/frequency scaling (DVFS), since the power consumption of CMOS circuits is proportional to its clock frequency and its voltage square. The DVFS attempts to drop the clock frequency and supply voltage of the processor dynamically to the lowest possible level while meeting the deadline constraint of task. The voltage and frequency scaling are usually accomplished by controlling a DC-DC converter and PLL (phase lock loop) circuit, respectively. While many high-end processors have equipped with the DVFS capabilities, few low-end processors can dynamically change their supply voltages. However, most low-end processors can change its clock frequency easily by setting the divider registers. As a result, the clock frequency can be changed quickly for a low-end processor using divider register comparing with a high-end processor using PLL. For example, while many commercial high-end processors require the transition time ranging from 189 us to 3.3 ms for voltage and frequency scaling [10], the M16C requires negligible time for frequency change. For simplicity, we refer to DVFS hereafter whenever voltage and/or frequency are changed during execution.

The motivation of this work stems from the fact that the power consumption of processor in idle mode is not fixed but dependent on the set clock frequency before entering the idle mode [7]. In general, the higher frequency, the more power is consumed in idle mode. For example, the PXA225 processor (an upgraded product of SA-1100 series) consumes 45 mW-121 mW power in idle mode which corresponds to 100 MHz - 400 MHz frequency [7]. The reason is that although the disabled CPU cannot consume dynamic power in idle mode, the enabled peripherals still consume power which is directly dependent on the selected clock frequency [8]. To reduce the power of idle mode, we therefore expect to lower the frequency of processor. However the lowered frequency will lead to longer execution time for interrupt service routine (ISR), which may result in increased total energy. Accordingly, we need to determine the optimal frequency for the idle state with periodic interrupt services to save energy. To the best of our knowledge, this is the first work that addresses the problem of selecting the optimal frequency to save energy for idle state in the presence of periodic interrupt services. The main contribution of this work is that we proposed two DPM strategies for idle state power management with periodic interrupt services considering two different application cases. Specifically, (1) In case the periodic interrupt cannot be disabled such as the data acquisition systems, we formulate the power consumption of idle state, and propose static and dynamic approaches for the processors with large or negligible DVFS overhead, respectively. (2) In case the periodic interrupt can be disabled such as the clock tick interrupt, we propose configurable clock tick and DPM algorithm to save idle energy and keep system time synchronization simultaneously.

The rest of the paper is organized as follows. Section 2 gives related work. Section 3 presents the proposed power model and approaches. In Section 4, experimental results are described. Finally, Section 5 summarizes the paper.

2. Related Work

There have been a large number of publications so far, which employed DPM or DVFS for power savings. Most DPM literature focuses on the design of power management policies using predictive schemes or stochastic optimum control schemes [1),4)]. In these schemes, they generally assume fixed power consumption for each low power mode and their objective is to decide when and which low power mode the devices should transition into. In practice, an on-chip timer interrupt is commonly employed in embedded systems to reactivate the CPU from low power mode quickly. In this case, the on-chip clock of timer cannot be disabled, which results in varied power consumption in low power mode as mentioned in Section 1. However, this special issue has not been discussed in any previous DPM literature.

While DPM is aimed at reducing power during the long idle time by transitioning the processor into low power mode, DVFS is aimed at saving power by lowering the processor voltage and/or frequency to reclaim the runtime slack, which is generated due to the fluctuation of workload. Most DVFS algorithms assume periodic tasks with known WCET and deadline. Although the objective of DVFS is to prolong the task execution time until deadline by lowering the CPU's voltage and frequency, the slack time cannot be reclaimed completely. This is because the generated slack can only be reclaimed when there are ready tasks that can be scheduled immediately. Moreover, the discrete frequency levels of processor makes DVFS cannot utilize the generated slack completely. The above reasons result in idle state of processor even for the DVFS enabled processor. Unfortunately, most DVFS literature ignores the power management in idle state, and simply assumes to enter a low power mode with zero power [2),3)] or fixed power consumption [10)] in idle state. Moreover, in practice the time overhead for power mode transition may be too long to be applicable for some short idle duration, which results in no power reduction in this case.

Recently, a variable scheduling timeouts method is proposed for power savings in Linux systems by eliminating the useless tick interrupts during system idle state [9)]. However an existing problem needed to be solved in real-time systems is how to keep the system clock synchronization caused by tick timer reprogramming.

3. Power Model and Approaches

Consider a typical low power embedded processors, we assume that the processor can provide multiple low power modes and selectable voltage/frequency levels for power control. To simplify the calculation, we also assume that the time and power overhead for power mode transition and voltage/frequency scaling are fixed. As discussed in Section 1, we primarily consider the low power mode with enabled peripheral clocks in our approach. The proposed power management is implemented in the idle task of RTOS, which is scheduled to run when system detects the beginning of an idle state.

We deal with the power saving problem of idle state as two different cases in the following sections. While in case one the periodic interrupt cannot be disabled such as the data acquisition system, in case two the interrupt can be disabled for a specified duration such as the clock tick interrupt.

3.1 Case One: the Periodic Interrupt cannot be Disabled

Prior to formulating the power consumption of idle state, we give the following notations.

- M: selected speed ratio, i.e., $1/M$ full speed

- T_p (us): period of interrupt service
- T_h (us): execution time of interrupt service routine at full speed
- T_s (us): execution time for low power mode setting in idle task at full speed
- T_t (us): time overhead for power mode transition
- I_t (mA): average current during power mode transition
- T_v (us): time overhead for dynamic voltage/frequency scaling
- I_v (mA): average current during voltage/frequency scaling
- I_{rm} (mA): the run mode average current at $1/M$ full speed
- I_{im} (mA): the idle mode average current at $1/M$ full speed
- V_m (V): the corresponding voltage for $1/M$ full speed setting
- I_{idle} (mA): average current of idle state
- P_{idle} (mW): average power of idle state

Consider the fact that different scale processors may have different DVFS overhead as discussed in Section 1, we propose static and dynamic approaches for processors with large or negligible DVFS overhead, respectively.

On one hand, if the processor has large DVFS time overhead, a static approach is preferable, i.e., only once DVFS setting at the beginning of idle state for any continuous idle time. Specifically, the program in idle task takes corresponding actions according to the current system state. If it is the first time to enter idle state, the power management program in idle task sets the optimal speed and transitions the processor into low power mode. Otherwise, it only puts the processor into low power mode and without any speed change when the idle task is reactivated from low power mode by interrupt. The above processing procedure is illustrated in **Fig. 3**. And the corresponding notations for each stage of this procedure are denoted in **Fig. 4**. Based on the above information, the average current and power of idle state with periodic interrupt services can be calculated by the following equations:

$$I_{idle} = \{(T_h + T_s)MI_{rm} + T_tI_t + [T_p - (T_h + T_s)M - T_t]I_{im}\}/T_p \quad (1)$$

$$P_{idle} = I_{idle}V_m \quad (2)$$

In the above equations, the period of interrupt T_p and the execution time for power mode setting T_s are assumed to know in advance. Time and power overhead for power mode transition T_t, I_t, and the average current under different speed settings in run and idle mode I_{rm}, I_{im} can be obtained from processor's

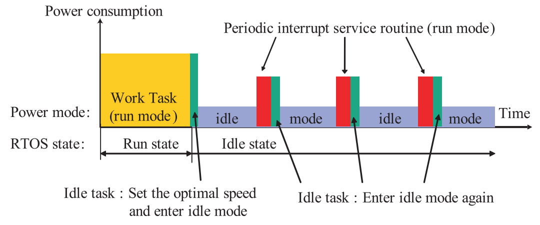

Power consumption
Periodic interrupt service routine (run mode)
Work Task (run mode)
Power mode:
idle
mode
idle
mode
Time
RTOS state:
Run state
Idle state
Idle task : Set the optimal speed and enter idle mode
Idle task : Enter idle mode again

Fig. 3 Processing procedure of idle state under power management.

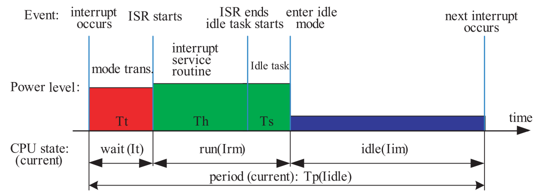


Fig. 4 Notations of idle state power management.

data manual or actual measurements. As shown in Eq. (1), the average current of idle state is a function of the selected speed M and the execution time of ISR T_h. According to this relation, the power optimization problem can be formulated as: for a specified processor and application with known T_h, T_p, T_s, T_t, I_t, I_{rm}, and I_{im}, finds the optimal M such that the average idle current is minimal. Because the relation between I_{idle} and M is linear, and the selectable speeds are limited, we can calculate all possible results of I_{idle} vs. M at given T_h. Then, the one that has the minimal average current should be the optimal speed setting.

On the other hand, if the processor has negligible DVFS time overhead, a dynamic approach may save more power at the expense of two DVFS settings for each interrupt process. The procedure is that the full speed is set at the beginning of each interrupt service, and the slowest speed is set before entering the low power mode each time. Its objective is to save more power by keeping the processor in low power mode with the minimal power consumption for longer time. In this case, the average idle current can be calculated by the following equation:

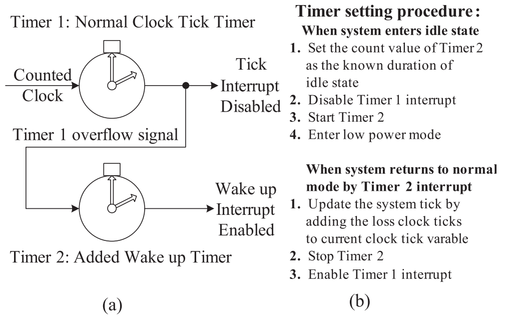
$$I_{idle} = \{[T_p - (T_h + T_s) - T_t - 2T_v]I_l + (T_h + T_s)I_h + T_tI_t + 2T_vI_v\}/T_p \quad (3)$$

where I_h represents the current of full speed running, and I_l represents the current of the slowest speed in idle mode. Note that this approach is not realistic for some high-end processors with large DVFS overhead. For example, Intel's PXA225 requires about 500 us for each DVFS setting [7]. In this case, the dynamic approach is obviously not applicable if the interrupt service is of 1 ms period.

3.2 Case Two: the Periodic Interrupt can be Disabled for a Specified Duration

We assume periodic tasks with known WCET and deadline in this case, and we only discuss how to disable clock tick interrupt by using a configurable clock tick during idle state. Under the above assumptions, whenever OS detects the beginning of an idle state, it also knows the nearest releasing time of next periodic task. As a result, the accurate duration of idle state can be calculated by OS. For this reason, OS therefore can disable the clock tick during this known idle time and transition the processor into low power mode to save more power. Note that this approach is different from general DPM in the sense that while general DPM assumes random tasks and predicts the duration of idle state by using previous idle duration information, this approach assumes periodic tasks and obtains the accurate duration of idle state in advance by calculation. Therefore the decision for power mode transition in this approach is straightforward.

When the clock tick interrupt is disabled during idle state, a problem that should be solved is how to trace the original clock tick to keep system time synchronization. To this end, another timer, as shown in **Fig. 5**, can be used to count the lost ticks during idle time when the tick interrupt is disabled. Because the original tick timer is never stopped and restarted except disabling its interrupt requests, the system time synchronization can be guaranteed easily. However, this approach is hardware-dependent since a wire connection between the output of timer 1 and the input of timer 2 is required as shown in Fig. 5 (a). The count value of timer 2 for generating the wakeup interrupt prior to the release of next task should be set to the calculated duration of idle state. The detailed timer setting procedure is listed in Fig. 5 (b). In conjunction with the configurable clock tick, the complete algorithm for idle state power management is given in **Fig. 6**. Note that this algorithm shows the most aggressive effort to save energy during idle state. It employs DVFS, DPM, as well as multiple low power modes together to achieve the maximal energy savings. In practice, only some of these potential may be exploited due to the limitations imposed by the specific processor and application. For example, if the internal clock is required to activate the processor, the deep low power mode (i.e., low power mode 1 in Fig. 6) cannot be used in this algorithm as discussed in Section 1. The efficiency of this algorithm will be evaluated in the following section with respect to several possible cases.


Timer 1: Normal Clock Tick Timer
Counted Clock
Tick Interrupt Disabled
Timer 1 overflow signal
Wake up Interrupt Enabled
Timer 2: Added Wake up Timer
(a)
(b)

Timer setting procedure:

When system enters idle state

1. Set the count value of Timer 2 as the known duration of idle state
2. Disable Timer 1 interrupt
3. Start Timer 2
4. Enter low power mode

When system returns to normal mode by Timer 2 interrupt

1. Update the system tick by adding the loss clock ticks to current clock tick varable
2. Stop Timer 2
3. Enable Timer 1 interrupt

Fig. 5 Configurable clock tick and timer setting procedure.

Assumptions and notations:

1. The time overhead for power mode transition is larger than that for DVFS.
2. T_{out1}, T_{out2}, and T_{out3} are the time thresholds for power mode control, and $T_{out1} > T_{out2} > T_{out3}$.
3. power of low power mode2 > power of low power mode1

Algorithm for idle state power management:

```
When system enters the idle state and idle task is started
 1.  Calculate the duration of idle time: $T_{idle}$
 2.  If ($T_{idle} > T_{out1}$) then
 3.       enable and set the configurable clock tick
 4.       set the slowest speed
 5.       enter the low power mode 1
 6.  else if ($T_{idle} > T_{out2}$) then
 7.       enable and set the configurable clock tick
 8.       set the slowest speed
 9.       enter the low power mode 2
10.  else if ($T_{idle} > T_{out3}$) then
11.       set the slowest speed in run mode
12.  else then
13.       maintain the previous speed without change
14.  end if
```

Fig. 6 Power management algorithm when the periodic interrupt can be disabled.

Table 1 Measured run and wait mode average current under different speed settings.

Selected Speed	Measured I (mA) at V=3 V	
(1/M full speed)	Run mode:I_{rm}	Wait mode:I_{im}
20 MHz (1/1)	10.04	1.30
10 MHz (1/2)	6.35	1.26
5 MHz (1/4)	4.35	1.24
2.5 MHz (1/8)	3.24	1.23
1.25 MHz (1/16)	2.45	1.22

4. Evaluation and Experimental Results

4.1 Experiment Setup and Measurement Environment

To validate and evaluate the proposed approach, we select the OAKS16-mini board with a M16C (M30262F8GP) embedded processor to implement the approach. As a typical embedded processor, M16C provides three power modes and can quickly change its clock frequencies by setting the divider registers, although it cannot change its supply voltage. We measure the processor current by inserting a digital multimeter between the power supply and the power pin of the processor. An oscilloscope is utilized to observe the voltage waveform of the current sense resistor which is inserted between the power supply and the power pin of the processor. The time and power overhead for power mode transition are estimated by using the captured voltage waveform through oscilloscope, which is similar to the one shown in Fig. 1. The above experiments are performed separately so that the current measurements are accurate with removed current sense resistor. The measured power consumptions of different power modes and estimated power mode transition overhead are illustrated in Fig. 2.

Our approach has been implemented in a RTOS called TOPPERS/JSP kernel [5] which is an open source RTOS in consistent with the ITRON [6] standard. The TOPPERS RTOS is targeted at real-time applications with limited resource requirement. A configurable clock tick is implemented in OS with default 1 ms interrupt period. The normal execution time of the timer ISR for system time updating is about 12 us at 20 MHz.

4.2 Evaluation of the Proposed Approach when the Periodic Interrupts cannot be Disabled

Table 1 summaries the measured run and wait mode average current under different speed settings. All these measurements are performed by executing a busy loop and the results for wait mode is measured with clock enable but without any interrupt services.

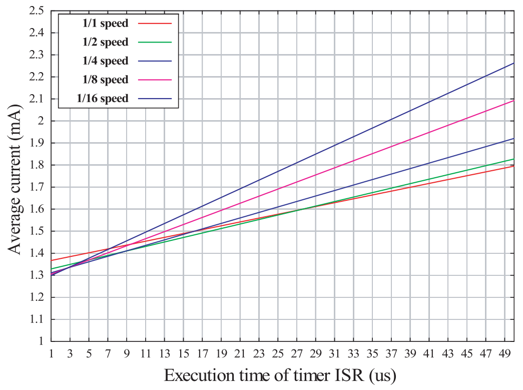


Fig. 7 Calculated average current under 1 ms interrupt period.

Based on these measured parameters, and Eq. (1), we can obtain the following current vs. execution time and speed curves under 1 ms interrupt period in **Fig. 7**. From this figure, it is clear that the optimal speed selection with the minimal power consumption is determined by the execution time of ISR. To validate the correctness of Eq. (1), we performed experiments under 4 cases with different interrupt periods and execution time of ISR. The original timer ISR of the TOPPERS is selected as baseline in case 1, and case 2, 3, and 4 are derived from the baseline by changing its period and execution time. Although the timer ISR is employed in the evaluation for the simplicity, the results can be applied to any other ISR with different parameters. The measured and calculated current results under 4 cases are summarized in **Tables 2**, **3**, **4**, and **5**, respectively. In the results, the optimal speeds leading to the minimal current are denoted with boldface. It is obvious from these results that the speed selections with the minimal measured currents are consistent with the theoretical calculated results in all 4 experiments, although there are errors between the calculated and measured values. Therefore, we can utilize the proposed idle current model in Eq. (1) to calculate the optimal speed for any ISR with known period and

Table 2 Comparison of measured and calculated average current in case 1.

Selected Speed (1/M full speed)	Idle state average I (mA) at V=3 V, T_p=1 ms, T_h=12 us	
	Measured I_{idle}	Calculated I_{idle}
20 MHz (1/1)	1.47	1.472
10 MHz (1/2)	**1.45**	**1.451**
5 MHz (1/4)	1.47	1.461
2.5 MHz (1/8)	1.50	1.498
1.25 MHz (1/16)	1.57	1.534

Table 3 Comparison of measured and calculated average current in case 2.

Selected Speed (1/M full speed)	Idle state average I (mA) at V=3 V, T_p=1 ms, T_h=7 us	
	Measured I_{idle}	Calculated I_{idle}
20 MHz (1/1)	1.40	1.419
10 MHz (1/2)	1.38	1.389
5 MHz (1/4)	**1.37**	**1.385**
2.5 MHz (1/8)	1.38	1.401
1.25 MHz (1/16)	1.42	1.416

Table 4 Comparison of measured and calculated average current in case 3.

Selected Speed (1/M full speed)	Idle state average I (mA) at V=3 V, T_p=10 ms, T_h=12 us	
	Measured I_{idle}	Calculated I_{idle}
20 MHz (1/1)	1.32	1.317
10 MHz (1/2)	1.28	1.279
5 MHz (1/4)	1.25	1.262
2.5 MHz (1/8)	1.24	1.256
1.25 MHz (1/16)	**1.24**	**1.251**

execution time, which will lead to the minimal average current.

Although only the average currents are given in the above results, the average power can be derived by multiplying the average current and the supply voltage, and energy consumption of idle state can be calculated by multiplying the average power and the duration of idle state. It is worth noting that the reduction of average power is dependent on the period and execution time of ISR, as well as the power consumption of different speeds as indicated in Eq. (1). For example, in case 3, selecting the 1.25 MHz can achieve 6% average power reduction comparing with the full speed execution, and in case 4, selecting the 10 MHz can save

Table 5 Comparison of measured and calculated average current in case 4.

Selected Speed (1/M full speed)	Idle state average I (mA) at V=3 V, T_p=10 ms, T_h=200 us	
	Measured I_{idle}	Calculated I_{idle}
20 MHz (1/1)	1.49	1.544
10 MHz (1/2)	**1.45**	**1.533**
5 MHz (1/4)	1.49	1.558
2.5 MHz (1/8)	1.52	1.621
1.25 MHz (1/16)	1.62	1.683

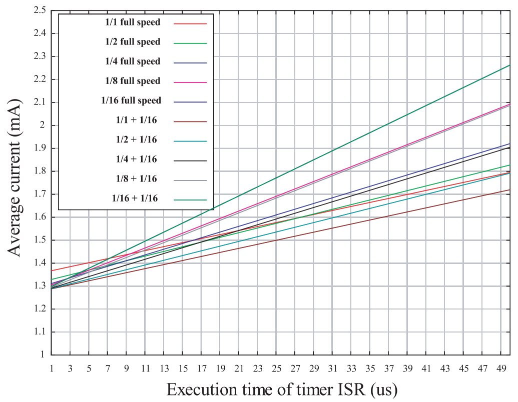


Fig. 8 Calculated average current of static and dynamic approaches under 1 ms interrupt period.

10% average power than execution with 1.25 MHz. As can be seen, selection of different speeds during idle state may result in significant difference of average power.

Experiments are also conducted to validate the proposed dynamic approach especially for the M16C with negligible DVFS overhead. The same conditions as the above case 1 are employed in these experiments. Additionally, the varied speeds are set at the beginning of ISR, and the slowest speed (1/16 full speed) is set in the idle task before entering the low power mode. The calculated results using Eq. (3) and assuming negligible DVFS overhead are depicted in **Fig. 8** where the curves for static and dynamic approaches are shown, respectively. As can be seen, the full speed setting for ISR plus the slowest speed setting (1/16) for low power mode outperforms other speed combinations in dynamic approach, as well as all speed settings in static approach. Meanwhile, the actually measured result for this case shows average current 1.39 mA which is the minimal current compared with the measured results for static approach in Table 2. The results indicate that the dynamic approach can further reduce the average power by 4.1% than the optimal static approach in case 1, and achieves the maximal 11% reduction in average power comparing with the speed selection of 1.25 MHz.

4.3 Evaluation of the Proposed Approach when Clock Tick Interrupts can be Disabled

To evaluate the proposed DPM algorithm, Pillai and Shin's dynamic DVFS scheduling algorithm[2)] is employed as the baseline for the comparison. The goal is to evaluate the energy saving improvement after applying our idle power management approach comparing with the original DVFS algorithm alone. This typical DVFS algorithm is designed for hard real-time embedded systems under earliest deadline first (EDF) scheduling. The algorithm is dynamic in the sense that it detects the runtime slack caused by early completion of task and attempts to lower the voltage/frequency of the next scheduled task to reclaim the slack. To implement this algorithm, we modify the scheduler of the TOPPERS to support the EDF scheduling, and add function for the measurement of task execution time. The proposed configurable clock tick and DPM algorithm are also implemented in the idle task of the TOPPERS.

As mentioned in Section 3.2, the energy saving potential of the DPM algorithm is dependent on specific processor and application. For this reason, we implement the DPM algorithm in three versions with different power saving level. The DPM 1 only sets the slowest speed at the beginning of idle state and without entering any low power mode. In contrast, the DPM 2 firstly sets the slowest speed and configurable clock tick, and then enters the wait mode. Instead of using one low power mode, DPM 3 determines to transition into the wait or stop mode according to the calculated duration of idle state. As described in Section 1, in stop mode the on-chip clock is disabled completely, thus to implement the DPM 3, a external clock source is utilized to drive the clock tick timer and activate the processor from stop mode. The time thresholds of T_{out1}, T_{out2}, and T_{out3} in the algorithm of Fig. 6 are set as 100 ms, 20 ms, and 5 ms, respectively.

Table 6 Experimental task set.

Task name	Period	WCET	Actual ET
Task 1	500–2000 (ms)	130 (ms)	28–130 (ms)
Task 2	500–3000 (ms)	245 (ms)	38–245 (ms)

It is important to note that the determination of time threshold is dependent on specific processor and OS, in particular the time and power overhead for mode transition as well as the period of clock tick. Two periodic tasks with known WCET and deadline are assumed to run on this experimental platform. **Table 6** presents the corresponding parameters of the task set, where the actual execution time of task is varied and less than the WCET, as commonly seen in most embedded systems. The task set is executed totally 5 times, each with different power management strategy as described above. Their corresponding energy results for one minute running are summarized in **Table 7**. From the results, we can derive the following observations:

- The use of DVFS cannot eliminate all idle state.
- The longer idle state, the more energy savings can be achieved.
- Combining DVFS and DPM can achieve better energy savings than using DVFS alone.
- Combining multiple low power modes can achieve more energy savings than using only one low power mode.

Specifically, in our experiments DVFS alone without any idle state power management achieves average 41% energy savings comparing with full speed execution. DVFS with the slowest speed setting during idle time achieves average 53% energy savings. Combining DVFS and DPM with wait mode obtains average 65% energy savings. DPM using both wait and stop mode accomplishes the best 67% energy savings in average. In summary, the proposed configurable clock tick and idle state power management can achieve additional 26% energy savings comparing with the original DVFS without any idle power management. Note that beside the application of combining the DVFS with the proposed DPM as done in the above experiments, the approach can also be applied alone to the periodic tasks. Furthermore, if the duration of idle state can be known in advance, it can also be applied to the aperiodic tasks.

Experiment is also conducted to verify the capability of the approach for keep-

Table 7 Evaluation of power savings for different power management strategies.

Power management strategies	Energy for one minute running under different task periods (normalized energy results)			
	P1:500 ms P2:500 ms	P1:500 ms P2:900 ms	P1:1000 ms P2:1500 ms	P1:2000 ms P2:3000 ms
Full speed	1807 mJ (1)	1807 mJ (1)	1807 mJ (1)	1807 mJ (1)
Dynamic DVFS alone	1594 mJ (0.88)	1288 mJ (0.71)	897 mJ (0.50)	468 mJ (0.26)
Dynamic DVFS+DPM1 (only the slowest speed)	1194 mJ (0.66)	1010 mJ (0.56)	752 mJ (0.42)	460 mJ (0.25)
Dynamic DVFS+DPM2 (the slowest speed+wait mode)	944 mJ (0.52)	773 mJ (0.43)	553 mJ (0.31)	282 mJ (0.16)
Dynamic DVFS+DPM3 (the slowest speed+wait mode+stop mode)	943 mJ (0.52)	747 mJ (0.41)	499 mJ (0.28)	239 mJ (0.13)

ing system time synchronization. We implement the configurable clock tick and the original clock tick in TOPPERS, respectively, and then let them run the above DVFS experiments for 30 minutes. Finally, we compare their clock tick value after running. The results show no difference between the two implementations, which indicates the configurable clock tick can trace the original clock tick precisely even if the clock tick is disabled during idle state.

5. Concluding Remarks

Even in a DVFS enabled embedded system, there must exist idle state where no task needs to be scheduled. Moreover, a periodic interrupt services may be required to run even in the system idle state. As a common approach, the processor can be transitioned into the low power mode during idle state. However, the power consumption of low power mode is neither zero nor fixed which is dependent on the current clock frequency in low power mode. In this work we present different approaches for idle state power management in the presence of periodic interrupt services. In case the periodic interrupt cannot be disabled, we formulate the idle power consumption and propose static and dynamic methods to save energy for the processors with large or negligible DVFS overhead, respectively. In case the periodic interrupt can be disabled such as the periodic clock tick interrupt, we propose the configurable clock tick and DPM algorithm for energy savings by keeping the processor in low power mode for longer time. We have implemented the proposed approaches in a RTOS and evaluated them on a frequency scalable embedded processor. The measured results show that the maximal 11% power can be reduced in the first case, and average 26% power can be further reduced in the second case comparing with DVFS without any idle power management. As for future work, we plan to evaluate and validate the proposed approach on other embedded processors.

Acknowledgments This work is supported in part by Core Research for Evolutional Science and Technology (CREST) program from Japan Science and Technology Agency.

References

1) Benini, L., Bogliolo, A. and Micheli, G.D.: A Survey of Design Techniques for System-Level Dynamic Power Management, *IEEE Trans. on Very Large Scale Integration Systems* (*VLSI*), Vol.8, No.3, pp.299–316, (June 2000).
2) Pillai, P. and Shin, K.G.: Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems, *Proc. ACM Symposium Operating Systems Principles*, pp.89–102, (2001).
3) Kim, W., Shin, D., Yun, H., Kim, J. and Min, S.L.: Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems, *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium* (*RTAS*), pp.219–228, (2002).
4) Ren, Z., Krogh, B.H. and Marculescu, R.: Hierarchical Adaptive Dynamic Power Management, *IEEE Trans. on Computers*, Vol.54, No.4, pp.409–420, (Apr. 2005).

5) TOPPERS Project. http://www.toppers.jp/
6) ITRON Project. http://www.sakamura-lab.org/TRON/ITRON/
7) Intel, Application Note: PXA255 and PXA26x Applications Processors Power Consumption During Power-up, Sleep, and Idle (Apr. 2003).
8) Texas Instruments, Application Report: SPRA164, Calculation of TMS320LC54x Power Dissipation (June 1997).
9) Variable Scheduling Timeouts (VST) Project Page. http://tree.celinuxforum.org/CelfPubWiki/VariableSchedulingTimeouts
10) Shin, D. and Kim, J.: Intra-Task Voltage Scheduling on DVS-Enabled Hard Real-Time Systems, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.10, pp.1530–1549, (Oct. 2005).
11) Renesas Corp. http://www.renesas.com/fmwk.jspcnt=m16c-family-landing.jsp fp=/sproducts /mpumcu/m16c-family/

(Received December 25, 2007)
(Revised March 17, 2008)
(Accepted May 1, 2008)
(Released August 27, 2008)

(Recommended by Associate Editor: *Hideharu Amano*)

Gang Zeng graduated from Hunan University, China, with B.E., M.E. degrees in 1993, 2001, respectively. From 1993 to 1998, he joined Hunan University where he was a lecturer in the Institute of Electric and Information Engineering. From 2001 to 2002, he joined ZTE Corporation where he was a senior engineer. He received his Ph.D. degree in information science from Chiba University in 2006. He is currently a postdoctoral researcher in the Graduate School of Information Science of Nagoya University. His research interests include power-aware computing, embedded system design, design for testability, system-on-a-chip testing. He is a member of IEEE.

Hiroyuki Tomiyama received his Ph.D. degree in computer science from Kyushu University in 1999. From 1999 to 2001, he was a visiting postdoctoral researcher with the Center of Embedded Computer Systems, University of California, Irvine. From 2001 to 2003, he was a researcher at the Institute of Systems & Information Technologies/KYUSHU. In 2003, he joined the Graduate School of Information Science, Nagoya University, as an assistant professor, where he is now an associate professor. His research interests include system-level design automation, architectures and compilers for embedded systems and systems-on-chip. He currently serves as an editor of IPSJ Transactions on SLDM, an associate editor of ACM TODAES and an editorial board member of International Journal on Embedded Systems. He has also served on the organizing and program committees of several premier conferences including ICCAD, ASP-DAC, DATE, CODES+ISSS, and so on. He is a member of ACM, IEEE, IPSJ and IEICE.

Hiroaki Takada is a Professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in Information Science from the University of Tokyo in 1996. He was a Research Associate at the University of Tokyo from 1989 to 1997, and was an Assistant Professor and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IPSJ, IEICE, and JSSST.