Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT), Palma de Mallorca, Spain, July, 2005.

Kernel Support for Energy Management in Wireless Mobile Ad-Hoc Networks *

Mauro Marinoni, Giorgio Buttazzo, Tullio Facchinetti, Gianluca Franchino
University of Pavia, Italy
{mauro.marinoni, giorgio.buttazzo, tullio.facchinetti, gianluca.franchino}@unipv.it

Abstract

Effective power management in wireless networks of mobile robots requires a proper support from the operating system, which must allow the application to dynamically configure the onboard resources to save energy consumption while guaranteeing the required real-time and performance constraints. In this paper, we present the kernel mechanisms necessary to achieve an integrated power management approach, in which energy saving is achieved at different levels of the architecture, including the processor, the communication device, and the robot peripherals, like sensors and actuators.

1. Introduction

The use of coordinated teams of small robots has several interesting applications, including monitoring, surveillance, searching, and rescuing. On the other hand, the use of small robot systems introduces several new problems that need to be solved for fully exploiting the potential benefits coming from a collaborative work. Most of the problems are due to the limited resources typically available on a small mobile robot. In fact, cost, space, weight, and energy constraints, impose the adoption of small microprocessors with limited memory and computational power. In particular, the computer architecture should be small enough to fit on the robot structure, but powerful enough to execute all the robot computational activities needed for achieving the desired level of autonomy. Moreover, since such systems are operated by batteries, they have to limit energy consumption as much as possible to prolong their lifetime.

In a wireless ad hoc network of mobile robots, energy can be saved at different architecture levels. At the operating system level, suitable scheduling and resource management algorithms can be adopted to execute tasks at the minimum speed that guarantees the required performance constraints. At the network level, the transmission power of each node can be set at the minimum level that guarantees a given degree of connectivity. At the application level, specific devices can be turned off, or configured at a proper operating low-power mode (if any), when they are not used for a sufficiently long interval of time. Also servomotors can be driven to drain less current when the robot joints are set in a configuration that does not demand high torques.

Models to describe the battery charge behavior have been proposed to help in deriving new approaches to the battery usage. Benini et al. [5] proposed a flexible discrete-time battery model that accurately describes the dynamic battery operation, allowing a careful system design accounting for realistic battery lifetime values. In a different work [4], the authors achieved a significant battery lifetime improvement by steering the power absorption in a multi-battery pack. This solution may be easily adopted when servomotors are used as actuating devices.

In the context of real-time systems, different energy-aware algorithms have been proposed to minimize energy consumption in the processor. They basically exploit voltage variable processors to minimize the speed while guaranteeing real-time constraints [15, 2, 3, 9].

In many cases, however, the approaches proposed in the literature are based on simplifying assumptions, like negligible overhead or continuous dynamic voltage scaling, which make them unusable in real applications, especially in those embedded systems based on small microcontrollers with very limited operating modes. In some cases, only two modes are available, so power management can only be achieved by switching between the two modes using suitable strategies. Recently, Bini et al. [6] proposed a method for analyzing the feasibility of real-time applications that execute by alternating two speeds, taking overheads into account.

On the network side, energy-aware algorithms have been mainly focused on the MAC level. Some others considered energy conservation in routing problems. Ye et al. [16] proposed the Sensor-MAC (SMAC) protocol, which is divided in two phases: a sleep period and an active period. In the sleep period the nodes switch their transceiver off, by putting it in sleep mode. In the active period, the nodes turn

* This work has been partially supported by the Italian Ministry of University Research under contracts 2003094275 (COFIN03) and 2004095094 (COFIN04).

its transceiver in the receiving mode to listen for incoming communications, or in transmission mode to initiate a communication. Each node can choose its sleep/active schedule, therefore sleep and active periods have to be locally synchronized between nodes. To synchronize them, nodes exchange $SYNCH$ messages, which contain the identification number of the sender and the time of its next sleep. The protocol is carrier sense multiple access with collision detection (CSMA/CD), so the synchronization does not have to be very strict. The active period is divided in two parts: the first part is used by nodes to send their $SYNCH$ messages, if any, while the second part is used for the request to send messages (RTS), if any.

T-MAC [13], like SMAC, adopts synchronized sleep/wakeup cycles to allow nodes to operate at low duty cycles while maintaining network connectivity. In order to reduce latency, T-MAC proposes a $future - request - to - send$ (FRTS) scheme to inform a node, on the third hop, that there exist a message for it by sending a FRST packet. Hoesel and Havinga [14] proposed a MAC protocol, LMAC, based on a TDMA scheme. Time is divided into slots, whose size is sufficient to send entire messages. Each node can have only a time slot, during which the communication is collision-free. This implies that energy is not wasted for managing collisions and accessing the radio channel. The scheduling algorithm is distributed and each message is dived in two parts: a control unit and a payload unit. The control unit includes several data, such as node identifier, data size, and a sequence slot number to maintain synchronization between nodes. To save energy, each node that is not addressed for communication turns its radio off until the next slot. Moreover, two nodes switch their transceiver off when the communication between them finishes.

Yu et al. [17] proposed the Geographic and Energy Aware Routing (GEAR) algorithm, which considers energy efficiency. Based on the fact that in the sensor networks a query is often geographical, GEAR propagates a query to the appropriate geographical region using energy-aware and geographically informed neighbor selection heuristics to route a packet towards the target region. Within a region, it uses a recursive geographic forwarding technique to disseminate the packet.

What is missing in the literature, however, is an integrated framework for energy-aware control, where different strategies can be applied at different levels of the architecture, from the hardware devices to the operating system, up to the application level.

In this paper, we present a system wide approach to energy management applied to all the architecture levels and integrated with the scheduling algorithm to guarantee real-time constraints. The method is tailored for an embedded robot controller consisting of a dsPIC 30F601x family microcontroller, capable of driving more than 20 servomotors, and a wireless communication board with different power/transmission modes.

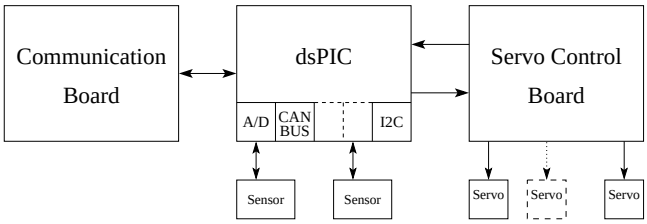

Figure 1. Block diagram of the main robot components.

The rest of the paper is organized as follows. Section 2 presents an overview of the system architecture, describing the degrees of freedom available in each component to achieve some form of power management. Section 3 illustrates the methods we propose at different architecture levels to limit energy consumption while still meeting real-time constraints. Section 4 focuses on the kernel support required to provide flexible power management services to real-time applications. Finally, Section 5 states our conclusions and future work.

2. System description

The system under consideration consists of a team of mobile robots that have to cooperate for achieving a common goal. Each robot can be either a classical wheeled vehicle or a legged walking machine actuated by servomotors, equipped with proximity and special-purpose sensors, a processing board, and a wireless communication subsystem. Hence, each robot can be seen as a mobile node of a wireless ad hoc sensor network. A block diagram of the main robot components is illustrated in Figure 1.

In the following sections we describe the characteristics of each component installed on each robot unit, focusing on the features that may enable the implementation of energy-aware control strategies.

2.1. Onboard microcontroller

The onboard processing unit is a Microchip dsPIC [10], which seamlessly integrates the control attributes of a microcontroller (MCU) with the computation and throughput capabilities of a Digital Signal Processor (DSP). It is a 16-bit microcontroller where most of the 24-bit wide instructions are executed in 1 cycle up to 30 MIPS. The model selected for prototyping includes a program memory space of 144 KBytes, a data memory space of 8 KBytes,

and a non-volatile data EEPROM of 4 KBytes. The MCU presents a full-features software stack, up to 41 interrupt sources, and 5 16-bit counters with 32-bit working mode. The DSP engine features a high speed 17-bit by 17-bit multiplier, a 40-bit ALU, two 40-bit saturating accumulators and a 40-bit bidirectional barrel shifter, and performs divisions in a 19-cycles loop. In terms of peripherals the chip supplies Capure/compare/PWM functionality, 12-bits Analog-to-Digital Converters (A/D) with 100 Ksps conversion rate and up to 16 input channels. Connectivity is provided through a full range of channels: I2C, SPI, CANbus, USART and Data Converter Interface (DCI), which supports common audio Codec protocols, as I2S and AC'97.

The microcontroller allows the application to choose among three different clock sources: an external oscillator up to 40 MHz with an internal PLL circuit to boost the frequency up to 120Mhz, an internal clock of 8 MHz, and a low power clock of 512 KHz. A postscaler can be applied to the selected source to slow down the frequency of a factor of 4, 16, or 64 to obtain the system clock. Once the clock frequency is selected, it is possible to set the supply voltage at the lowest level that supports such amount of MIPS. For example, slowing down the clock to one-third of its maximum value the power supply could be lowered to 2.5 Volts. It is also possible to create a set of frequency/voltage pairs to be used as power states in dynamic voltage scaling (DVS) algorithms. Changing the clock source is an action that is performed with a latency of 10 periods of the new clock.

The MCU has two reduced power modes, idle and sleep, which can be entered through the execution of a specific instruction. In the idle mode the CPU is disabled, but the system clock source continues to operate. Peripherals continue to operate, but can optionally be disabled. In the sleep mode, the CPU, the system clock source, and any peripherals that operate on the system clock source are disabled. This mode consumes less power, but requires a delay from 10 μs to 130 μs when exited, whereas the idle mode has no wake up delay.

A method is provided to disable a peripheral module by stopping all clock sources supplied to that module. When a peripheral is disabled with this feature, it is in a minimum power consumption state. When the command is sent, the interested module is disabled after a delay of 1 instruction cycle. Similarly, when the wake up command is given, the target module is enabled after a delay of 1 cycle.

2.2. Communication board

In the market there are many transceivers that are suitable to build small radio devices that can be used to realize sensor nodes. There are many smart features that can be exploited to design energy-aware transmission protocols. Some of them are listed below:

- RSSI (Receiving Signal Strength Indicator) is a value proportional to the strength of the received RF signal. It can give a greedy esteem of the distance from the source, if the transmission power is known.
- Different levels of transmission power. They can be exploited, in conjunction with the RSSI, to save energy, adapting the transmission power to the distance between source and sink nodes.
- Different operating modes. The most common modes are: $sleep$, $receiving$, and $transmission$. Each mode consumes different level of energy. When a transceiver is on sleep it consumes less power than in others modes, but the time to switch between modes is different. For example, switching from $sleep$ to $transmission$ takes more time than switching from $receiving$ to $transmission$. Moreover, switching between modes consumes energy too. This latter consideration is important in the communication protocols design.

The characteristics described above can be found in several devices available on the market. A couple of devices suitable for our class of embedded system are the CC1000 and the ATR86RF211.

The CC1000 is a chip produced by Chipcom, with a transmission rate of 78,5 Kbaud/sec, a variable transmission power from -20 to 10 dBm, a minimum supply voltage of 2.1V, and a RSSI output pin for signal strength acquisition. It has two operating modes: power-up and power-down. In the power-down mode, it consumes no more than 1 μA. The transceiver can be set in the ISM (Industrial, Scientific and Medical) and SRD (Short Range Device) frequency bands at 315, 433, 868 and 915 MHz, but can easily be programmed by a microcontroller to operate at other frequencies in the 300-1000 MHz range.

The ATR86RF211, produced by Atmel, operates in the ISM band (from 400 MHz to 930 MHz), with a FSK (Frequency Shift Keying) modulation, a data rate of 64 kbps, and eight digitally selectable power levels. The maximum transmitter power is 14 dBm in the 433 MHz frequency band. Its power saving features are: *power down* mode, *sleep* mode, and stand-alone *wake up* procedure. It consumes no more than 0.5 μA in power down mode and no more than 3 μA in sleep mode. It is a multi-channel transceiver with fast frequency shifts (less than 50 μs for a 100 KHz shift). This feature is suitable to implement an efficient *frequency hopping* transmission protocol. The AT86RF211 is also well adapted to battery operated systems as it can be powered with only 2.4V. It can be controlled by means of a three wire interface, either by a microcontroller or by a DSP. Finally it has an RSSI output pin in order to acquire the strength of the received signal.

2.3. Servomotors

The motors considered in this work are the Hitec HS-475HB, which include an internal position control loop that allows the user to specify angular positions through a PWM input signal. This feature simplifies the external circuitry and avoids sending feedback signals to the motor control unit. The internal feedback loop imposes an angular velocity of 250 degrees per second and the motor is able to generate a maximum torque of 9.6 kg·cm with a voltage of 6 Volts. Motors are connected to a board that provides them with the required power supply and the input signals coming from the control layer. The torque applied to the motors can be estimated by monitoring the current drained by the servo. Such a current is read by using a Maxim MAX471 chip, which produces an output voltage proportional to its input current.

Monitoring the servo current absorption can also be exploited to obtain a level of feedback in servomotors control. In fact, as described in Section 3.3, the current absorption is related to the exerted torque and can be used to detect and compensate angular position errors.

3. Power management

Hardware and software components cooperate to reach the following main goals: low power consumption, onboard sensory processing, real-time computation, and communication capabilities. The robot is built with generic mechanical and electrical components, making the low-power objective more difficult to be satisfied. Nevertheless, the adoption of power-aware strategies inside the software modules significantly increased the system lifetime. To achieve significant energy saving, power management is adopted inside every system module and needs to be coordinated at the operating system level.

3.1. DVS management

Using the DVS capabilities described in Section 2.1, it is possible to implement a power-aware scheduling algorithm for a discrete set of clock frequencies. The possibility to put the processor in a sleep state can be useful for various purposes. The simplest strategy is to put the processor in this state if there is no work to be executed. The sleep mode can be exited upon the arrival of an interrupt from a peripheral or from the system timer for a task activation. The sleep state can also be considered as an actual operating mode, and the corresponding zero speed can be included in the set of available speed levels.

The simplest approach for integrating power-aware scheduling and real-time constraints is to fix the clock speed to a value computed off line to guarantee system feasibility. Due to the limited number of allowed frequencies for the system clock, this solution may cause a waste of power consumption. A better result can be achieved by alternating two clock frequencies to reach the ideal value requested by the theoretical calculation. In fact, alternating two clock states, as a PWM signal, one can approximate a speed level that is not available in the processor. In this way, an optimal speed level can be computed to guarantee real-time constraints while minimizing energy consumption, as proposed by Bini et al. [6]. Another approach is to calculate a different speed for each task and set the system clock to the appropriate value when a context switch occurs. In addition to the off-line calculation of the working frequency, it is possible to implement an on-line reclamation technique to further decrease the frequency using the unused computation time.

Another possibility for reducing energy consumption at the system level is to use the capability of the MCU to put each single device in different working modes: each one with well-known energy requirements. It is also possible to put a device in a non working state and decide whether the peripheral could work while the CPU is in the idle state.

3.2. Communication board

In a mobile node, a part from the electromechanical components, the radio module is the component that usually consumes most energy. When designing a communication protocol for such networks, one must consider the main sources of energy waste. In the following, we briefly describe some of them.

The first source is *idle listening*, which is due to the energy wasted when listening to the channel to receive possible messages. The second source is *collisions*: when two or more nodes try to send a message at the same time, some collisions are experienced and the corrupted packets have to be retransmitted, causing more energy consumption. This is particularly true in carrier sense multiple access (CSMA) systems. The third source is *overhearing*, occurring when a node picks up packets that are not sent to it. Another source is due to *protocol overhead*: simple protocols need less energy to operate. Some protocols introduce additional control packets (e.g., RTS/CTS packets in the IEEE 802.11 [1]) to solve the hidden node problem [12]. Such added control packets consume additional energy.

The energy wasted during communication can also depend on the particular approach used at the MAC-level. For example, time division multi access (TDMA) protocols are collision-free, therefore they do not have to consume energy to retransmit corrupted packets. However, they are characterized by poor scalability, high protocol overhead, and require to exchange additional information for clocks synchronization.

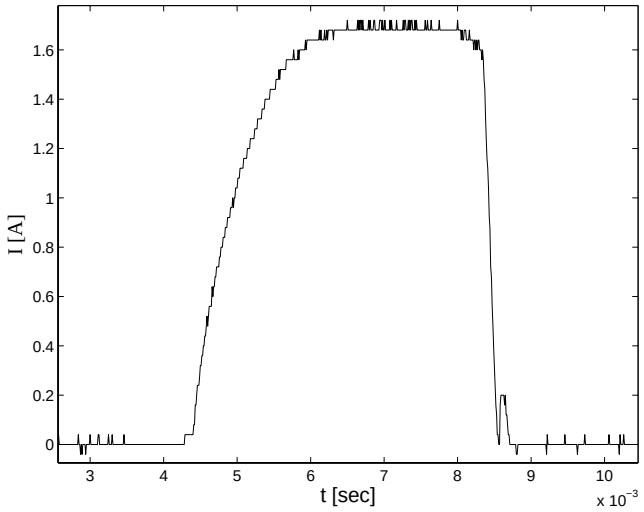

Figure 2. Current absorption per period in a servomotor.

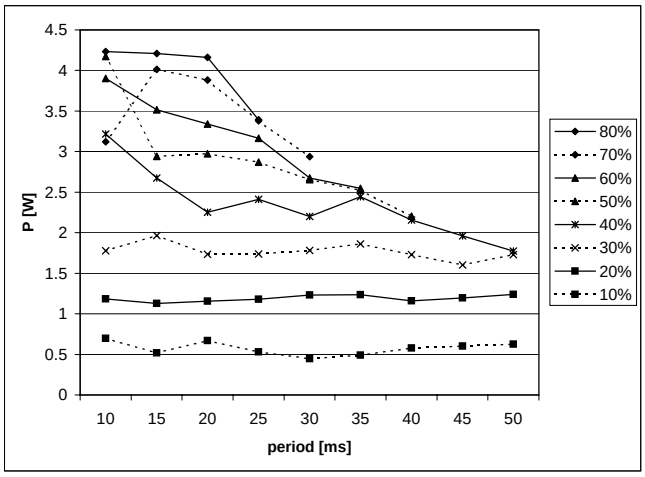

Figure 3. Power absorption per period in a servomotor with different loads percentage of the maximum load.

Transmission power control [8, 11] is useful to guarantee network connectivity, manage density and allow spatial reuse of radio channels. Moreover, minimizing transmission power can also indirectly reduce energy consumption by reducing the channel contention and collision between transmission nodes. Balancing density and connectivity networks maximize spatial reuse of the spectrum. The transmission power control is often encapsulated in the MAC or in the routing protocol.

3.3. Servomotors

In mobile robot systems, the energy consumed by motors is significantly higher than the one spent for processing and communication. Hence, a careful management of the motor power can remarkably improve the system lifetime. Servomotors are commonly adopted in robotic applications, since they are cheap and integrate reduction gears and position control to simplify their usage. A servomotor is controlled by modulating the duty cycle of a square wave signal, where the duration of the active pulse defines the angular position of the shaft. The pulse period does not influence the shaft angular position, but affects the servo current absorption, since the energy absorption starts at every period and its duration depends both on the load and the period. Figure 2 shows a typical current absorption in a servomotor within a control period.

To assess the behavior of the servo energy consumption, we performed several tests on the Hitec HS-475HB servomotor described in Section 2.3. We carried out several experiments by varying the control period and the load torque. Figure 3 shows the consumed power as a function of the control period for different loads. It is interesting to notice that for loads not exceeding 30% of the maximum value the power consumed by the servo does not depend on the control period, while for higher loads it decreases with the control period.

Figure 4 shows the angular errors between the position set-point and the real servo shaft position as a function of the control period and for different applied torques. For loads higher than 30% of the maximum torque value, the servo is not able to keep the position set point, and the error increases with the control period. However, such an error can be predicted by estimating the exerted torque through the absorbed current and can be corrected by using an external feedback loop.

On the servomotors side, power consumption can be controlled at two different levels. At the application level, the robot system can be driven to reach pre-defined postures that minimize the torques on the robot joints. This strategy can be quite effective in multi-link robots with several degrees of freedom, such a walking machines and anthropomorphic manipulators. At the signal generation level, a longer control period allows the driving hardware to maintain low-power states for longer time.

4. Kernel Support for Energy Management

This section describes the kernel infrastructure required to support energy-aware strategies at the application level. It consists of two parts: a set of mechanisms inside the kernel that implement the methodologies described in the previous sections and a set of library functions that simplify the user interaction. In the following, the first part is re-

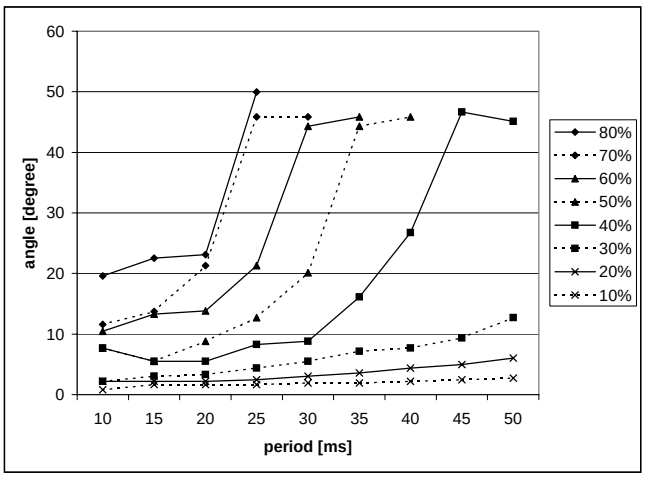

Figure 4. Shaft position errors with different periods and loads percentage of the maximum load.

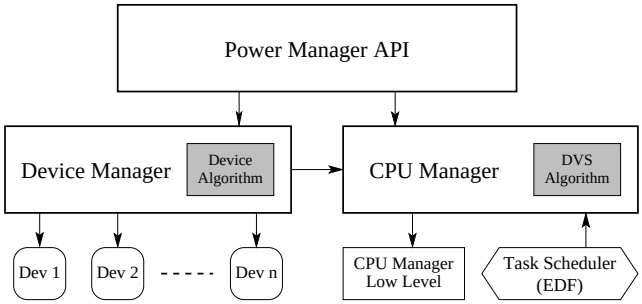

Figure 5. Block diagram of the Power Manager.

ferred to as the *Power Manager*, while the second part is referred to as the *Power Management API*. The main objective of the Power Manager is to achieve high efficiency to reduce the overhead introduced in the system, whereas the main goal of the Power Management API is to provide an abstraction layer that allows the user to control the power consumption of the different resources in a simple and uniform fashion. The Power Manager is responsible for selecting the most appropriate power states both for the CPU and the peripheral devices.

A block diagram of the power management architecture is illustrated in Figure 5. The blocks at the bottom represent the elements used for the interaction with the hardware, while the top layer provides the application programming interface (API) for interacting with the user.

- Blocks from *Dev 1* to *Dev n* represent the low-level drivers of each single peripheral that the Device Manager uses to control the power states.
- The CPU Manager at the lower level is in charge of managing the DVS capabilities of the processor working on the frequency-voltage table.
- The task scheduler interacts with the CPU Power Manager through a set of functions invoked at known scheduling events.
- The Device Manager controls the operating modes of the peripherals in order to reduce the power consumption at the system level while guaranteeing consistency of both the operating system and the application. The strategy adopted by the Device Manager is decided by a specific device algorithm, that can be changed by the user.
- The Device Manager relies on the CPU Manager to integrate the CPU with all others peripherals. It could implement different algorithms to reduce power consumption and work as a bridge between the abstraction used at the higher level and the DVS mechanism at the bottom layer.
- The set of library functions in the API layer allows the application to interact with the power-management infrastructure in a simple and uniform fashion.

4.1. CPU Power Manager

The main goal of the CPU Power Manager is to select the most appropriate voltage level and clock frequency in the processor. The clock management is split into two levels: a lower level related to the hardware and a higher level related to the DVS algorithm. The lower level is in charge of manipulating the fundamental CPU parameters, like clock frequency and voltage level, so it is architecture-dependent, while the higher level decides the time and the value of the CPU parameters, so it depends on the DVS scheduling algorithm.

Since each CPU is characterized by a different set of frequencies and voltage levels, and not all possible combinations are allowed, the set of frequency-voltage pairs supported by the CPU is stored in a table, as shown in Figure 1. Then, a set of kernel primitives allows the DVS algorithm to retrieve some relevant information, such as the minimum voltage compatible with a given frequency, the maximum frequency consistent with a chosen voltage, or the normalized speed ($s = f/f_{max}$) corresponding to the selected mode. Other primitives also allow reading and writing the current frequency and voltage level, and managing the system time by acting on the system tick.

To simplify the implementation of a power management scheme for the CPU, the operating system also supplies a set of hooks for executing specific DVS functions upon the occurrence of certain events. A list of most relevant events

	f_1	f_2	f_3	f_4
v_1	1			
v_2	2	4	6	
v_3	3	5	7	8

Table 1. Table storing the frequency-voltage pairs allowed by the CPU.

that may require the intervention of the power manager is reported below:

- **System startup** - This hook allows to setup the environment of the DVS algorithm and its initial state. It can also be used to compute the working frequency based on the task set parameters.
- **Context switch** - This hook is important whenever the DVS algorithm has to modify the clock frequency as a function of the scheduled task, or perform some resource reclaiming based on the unused computation time.
- **CPU idle** - When the ready queue becomes empty, the power manager could set the CPU in a low power consumption mode until a task is activated or an interrupt is raised.
- **Wake-up** - When the system exits from a power-saving status, some action could be executed before the kernel restarts.
- **Power-Management Point** - In some cases, hooks may be explicitly inserted in the applications tasks through a proper system call, to invoke specific DVS functions. For example, some algorithms proposed in the literature [9] require to insert a function in the task body to calculate the actual computation time with respect to the worst-case one.

Finally, in some other cases, power management may need to be executed at given time instants, hence the kernel must provide a mechanism to activate a DVS function by a timer. For example, this functionality is needed by the Speed Modulation algorithm proposed by Bini et al. [6].

4.2. Device Power Manager

Reducing energy consumption at the system level is possible because most peripheral devices support various operating modes and the MCU has the capability of putting each device in a sleep state. A problem is that some peripherals trash the current job when switched in low-consumption states. For example the A/D converter loses the ongoing acquisition and the UART does not listen to incoming data. To reduce the power consumption without affecting the behavior of tasks, a system-level coordination is required.

To support power management of peripheral devices at the kernel level, a `pstate` type is defined as an array with size equal to the number of devices in the system. Each element of the array stores the power state of the peripheral, where power states are represented by integers sorted by power consumption.

Three `pstate` variables are defined in the kernel:

- W^{max} stores the maximum number of different power states each device can manage;
- W^{sys} stores the minimum power level required by each device for the correct kernel operation.
- W^{dev} stores the actual power level set by the power manager for each peripheral device.

Then, two arrays of `pstate` type are defined for each task τ_i to express its requirements:

- R_i^{run} stores the minimum power level for each device required for the correct behavior of task τ_i, while it is running;
- R_i^{idle} stores the power levels requested by τ_i, when it is not running.

It is worth observing that the proposed approach is general enough to include the CPU in the set of devices used by the task. In this case, the mapping between the power state values inside the array and the real processor behavior is performed by a function provided by the Power Manager.

The arrays defined above can be used in a static or dynamic fashion, depending on the maximum overhead that can be tolerated in the system. If the overhead has to be minimized, the static approach is more suited, where all the values in the arrays are fixed and computed in the worst case. Otherwise, each task can dynamically change these values during execution, so allowing the power manager to reduce the overall power consumption of the system. As an example of dynamic behavior, a task could request a given power level for the A/D converter only while the acquisition is in progress, and then reset the value to zero. In the static approach, the re-computation of all power levels is needed only during a context switch, while in the dynamic mode the new power state for a device has to be computed every time the running task modifies its power state requirements. If τ_i is the new active task and p is a specific device, the new value $W^{dev}[p]$ is computed as the maximum between $W^{sys}[p]$, $R_i^{run}[p]$ and the maximum value $R_{max}^{idle}(i,p)$ among all values $R_k^{idle}[p]$ for tasks different than τ_i. That is:

$$W^{dev}[p] = \max\{W^{sys}[p], R_a^{run}[p], R_{max}^{idle}(i,p)\}$$

where

$$R^{idle}_{max}(i,p) = \max_k \{R^{idle}_k[p] \mid \tau_k \neq \tau_i\}.$$

4.3. Power Management API

The interaction between the application and the power management infrastructure occurs through a set of functions that manipulate the set of `pstate` arrays. Every function is implemented to work on a single device. The most important functions perform the following tasks:

- Get the maximum power level allowed by the system to the peripheral;
- Get the minimum power level required for the operating system consistency;
- Get the power level the device is actually using;
- Get the power level required by the task while it is running;
- Require a new power level for the device while the task is in execution; the actual power level for that device will be decided by the power manager based on all task requirements.
- Get the power level required by the task while it is not running;
- Require a new power level for the device while the task is not running; the actual power level for that device will be decided by the power manager based on all task requirements.
- Get the power level required by all the other tasks while not running;

There are also five functions used to interact with the power-management algorithms.

- Two of them are used to get/set parameters of the CPU Manager to tune its behavior.
- Another function is needed to define a Power Management Point, that must be explicitly inserted by the user in the task body.
- The last two functions allow the user to get/set parameters of the Device Manager to tune its behavior.

5. Conclusions

In this paper we presented an integrated approach for achieving energy management in wireless ad hoc networks of mobile robots. We showed that significant energy saving can only be obtained by a combined effort at different architecture levels. At the operating system level, specific power-aware algorithms can be adopted to set the appropriate operational mode to minimize energy consumption while guaranteeing the timing constraints. At the network level, node transmission power can be tuned to guarantees a given degree of connectivity and, at the application level, the control strategies can trade performance with energy consumption, so that the robot can switch to a different behavior to prolong its lifetime when the batteries are low, still performing useful tasks.

We showed how the proposed techniques can be supported at the kernel level to implement flexible energy-aware strategies on the robot resources and on the network.

As a future work, we plan to implement the proposed strategies in the Erika kernel [7], that will run on the dsPIC boards embedded in all robot units.

References

[1] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*. IEEE 1999.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the Euromicro Conference on Real-Time Systems*, June 2001.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 2001.

[4] L. Benini, D. Bruni, A. Macii, E. Macii, and M. Poncino. Discharge current steering for battery lifetime optimization. *IEEE Transactions on Computers*, 52(8):985–995, August 2003.

[5] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. Discrete-time battery models for system-level low-power design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(5):630–640, October 2001.

[6] E. Bini, G. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *IEEE Proceedings of the Euromicro Conference on Real-Time Systems*, July 2005.

[7] Evidence Srl. *ERIKA Enterprise RTOS*. URL: http://www.evidence.eu.com.

[8] J. Heidemann and W. Ye. *Energy Conservation in Sensor Networks at the Link and Network Layers*. Nirupama Bulusu and Sanjay Jha (editors), 2005. Technical Report ISI-TR-2004-599, USC/Information Sciences Institute, 2004.

[9] R. Melhem, N. AbouGhazaleh, H. Aydin, and D. Mossé. *Power Management Points in Power-Aware Real-Time Systems*. R. Graybill and R. Melhem (editors), Plenum/Kluwer Publishers, 2002.

[10] Microchip Technology Inc. *dsPIC30F family reference manual (DS70046C)*, 2004. URL: http://www.microchip.com.

[11] R. Ramanathan and R. Rosales-Hain. Topology control of multihop wireless networks using transmit power adjustment. In *Proc. of the IEEE Infocom*, March 2000.

[12] F. A. Tobagi and L. Kleinrock. Packet switching in radio channels: Part ii - the hidden terminal problem in carrier sense multiple-access modes and the busy-tone solution. *IEEE Transactions on Communication*, 23(12):1417–1433, December 1975.

[13] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proc. of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 1993.

[14] L. van Hoesel and P. Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *Proc. of the 1st International Workshop on Networked Sensing Systems*, Tokyo, Japan, June 2004.

[15] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.

[16] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, June 2004.

[17] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA Computer Science Department, May 2001.