Optimum design of a banked memory with power management for wireless sensor networks

Leonardo Steinfeld · Marcus Ritt · Fernando Silveira · Luigi Carro

Received: date / Accepted: date
The final publication is available at Springer via: `http://dx.doi.org/10.1007/s11276-014-0763-5`

Abstract The ever-increasing complexity of applications covered by wireless sensor networks (WSNs) demands for increasing memory size, which in turn increases the power drain. It is well known that SRAM power consumption can be reduced by employing a banked structure, where unused banks are switched into the low leakage retention mode. Although several power management strategies and algorithms for allocating the memory contents to the banks have been proposed, the energy savings limits of these techniques were not completely explored. In this work, we propose a new strategy for memory banking, taking advantage of the software properties intrinsic to WSN, and achieve aggressive power savings. We present a detailed model of the energy saving for uniform banks with two power management schemes: a best-oracle policy and a simple greedy policy. The model gives valuable insight into key factors (coming from the application, the technology, and design decisions) that are critical for reaching the maximum achievable energy saving. Using our model the optimum number of banks can be estimated at design time to reach more aggressive energy savings. The memory content allocation and the power management problem were solved by an integer linear program formulation for two real wireless sensor network applications (based on TinyOS and ContikiOS). Experimental results show memory energy reduction up to 78.3% for a partition overhead of 1%, representing an overall energy saving close to 19% in data collection WSN applications, including the communication energy and sleep power. The saving would increase to 34% in more intensive processing application.

Keywords wireless sensor network · banked memory · power management · SRAM memory · event-driven software

The authors would like to thank the financial support of ANII and CSIC-Universidad de la República.

L. Steinfeld
Julio Herrera y Reissig 565
Tel.: +598 27110974
Fax: +598 27117320
E-mail: leo@fing.edu.uy

1 Introduction

Wireless sensor networks (WSNs) embed computation and sensing in the physical world, enabling an unprecedented spectrum of applications, ranging from environmental monitoring to medicine. Nowadays, one of the major issues of WSNs is reducing the energy consumption without sacrificing the computational power. The ever-increasing complexity of applications, reflected in the software complexity, demands for increasing memory size. In some applications the code size is doubling every ten months [26]. Larger memory size requires more power, as it has been found that the memory system is responsible for a large portion of the total energy budget in SoCs [6].

The aim of this work is to reduce the energy consumption in processing, particularly in the memory subsystem. Our proposal has a major impact on applications where the processing energy is relatively important in the total budget, including the radio. This kind of more processing oriented network is increasingly spreading. Examples of such applications are: heavy processing application that transmit only final results (do not transmit raw data); network collaborative processing application, in which nodes exchange messages within their neighborhood with a very low communication workload associated to routing.

An effective approach to reduce the dynamic power is to partition a SRAM memory into multiple banks that can be independently accessed, and since only one bank is active per access, the remaining idle banks can be put into a low-leakage sleep state to further reduce the static power consumption [10]. However, the power and area overhead due to the extra wiring and additional circuits prevents an arbitrary fine partitioning into a large number of small banks. Therefore, the final number of banks should be carefully chosen at design time, taking into account this partitioning overhead. The memory organization may be limited to equally-sized banks, or it can allow any bank size. Moreover, the strategy for the bank states management may range from a greedy policy (as soon as a bank memory is not being accessed it is put into low leakage state) to the use of more sophisticated prediction algorithms [4].

The main contribution of this work is to show that, the new problem formulation proposed in this paper, one can find the optimum partitioning of memory banks in several WSN applications. We derive expressions for energy savings in the case of equally sized banks based on a detailed model for two power management strategies: best-oracle policy and a simple greedy policy. The maximum achievable energy saving is found, and the limiting factors are clearly determined. We show that it is possible to find a near optimum number of banks at design time, irrespective of the application and of the access pattern to memory, provided that the memory energy parameters are given, such as energy consumption characteristics and the partition overhead as a function of the number of banks. Experimental results show that using our approach in a banked memory leads to aggressive memory energy reduction (close to 80%), and an overall energy savings of 12%, in data collection WSN applications, to 25 % in more intensive processing applications. Our results also suggest that adopting an advanced power management must be carefully evaluated, since the best-oracle is only marginally better than a greedy strategy for a moderate wake-up energy cost.

The remainder of this paper is organized as follows. In Section 2 we give the motivation of this work, and in Section 3 we present previous related work. In Section 4, we describe the banked memory with power management and its energy model, and in Section 5 we derive expressions for the energy savings based on this model. In Section 6 we formulate the memory allocation and power management as an integer linear program (ILP). The experiments are presented in Section 7, and in Section 8 we discuss the results. Finally, Section 9 presents some concluding remarks and research directions.

2 Motivation

Reduced energy consumption is one of the major aims in WSNs, since it determines the lifetime of sensor nodes when they are powered from batteries, and dictates severe requirements on the harvesting system when the node scavenges scarce energy from the environment. In both cases, low-power techniques must be adopted.

A canonical sensor node consists of the following blocks: a processing component (usually a microcontroller) with wireless communication capabilities (RF transceiver), sensors/actuators and a power supply subsystem. The RF transceiver is the most power-consuming component of a sensor node. The instantaneous power of the radio (receive or transmit mode) exceeds the processing power (microcontroller in active mode) in around one order of magnitude. For example, in the so-called Berkeley-motes, the nominal communication to processing ratio is about ten (see [22] for this data and the evolution from earlier platforms). Prayati et al. [23] isolate and measure each contributor to the overall mote power consumption, confirming the ratio above and providing a model from which the total energy can be estimated as a function of the different power levels and the corresponding duty-cycles (i.e. the fraction of time a power contribution is present). Therefore, in the early days of WSN research the communication energy cost dominated the overall budget. Consequently, a significant research effort has been made since then to reduce this communication energy cost. The Medium Access Control (MAC) layer design is crucial, since it directly controls the transceiver determining the power profile drain. The use of advanced MAC protocols has helped in improving the energy efficiency of communication [5]. For example, ContikiMAC or TinyOS LPL reach a duty-cycle as low as 1% [16] for low data rate communication. In order to determine the processing energy we need to know the duty-cycle of the processor. This value, which is not usually published, can be readily obtained using the tools described in [8] and [11]. For simple data collection applications (e.g. MultihopOscilloscope of TinyOS and rpl-collect in ContikiOS, described in Section 7) the measured processing duty-cycle is about 3%. In this case the communication energy is just three times the processing energy. Nowadays, given the evolving scenario, efforts towards energy reduction should target both communication and processing [21].

Concerning processing power optimization, there has been a large amount of research in the last years, resulting in a variety of ultra-low-power processors. In addition, it has been pointed out that the processor spends most of its energy on memory access. Dally et al. [7] provided a detailed energy breakdown of a conventional RISC processor (SPARC V8). They presented average energy values per operation disaggregated by instruction supply, data supply, arithmetic, and clock and control logic, representing 42% , 28%, 6% and 24% of the total energy consumption respectively. The presented data is quite revealing, since it shows that the processor spends most of its energy, 70% of the total processor energy, accessing to memory for supplying data and instructions. Moreover, Verma [27] illustrated that the relative SRAM memory consumption in processors increases as processors consumption decreases by applying advanced design techniques. The processor with the lowest power consumption, among the surveyed ones in [27], is a custom MSP430 processor with 16 KB SRAM cache, operating at 0.3 V [18], where the embedded SRAM consumes 69% of the total processor power. Hence, the energy consumption in ultra-low-power processors is greatly dominated by memory accesses.

In summary, reducing the power taken by naïve memory organizations enables more computationally demanding algorithms to be implemented with the extra power resources, expanding the range of WSN applications. Moreover, the communication protocols are actually restricted by the low computational capabilities and low memory footprints of current low-power processors, hence reducing the computational energy will enable to adopt more complex communications protocols leading to further optimizations to reduce the communication energy [15].

3 Related work

SRAM memory banking along with power management to put memory banks into a sleep mode is a well known technique. Ferrahi et al. [10] initially presented the memory partitioning problem to exploit the sleep mode operation to minimize power consumption, showing that it is a NP-hard problem, and that some special classes are solvable in polynomial time. Results were obtained for a set of synthetic data, randomly generated with controlled parameters, and the effectiveness of the algorithm was assessed by comparing to a random partitioning algorithm.

This idea of reducing energy consumption by increasing memory elements idleness has been applied to scratchpad and cache memories in applications with high performance requirements (see Loghi et al. [19] for a brief survey).

Focusing on SRAM memory partitioning, Benini et al. [2] applied this technique to highly data-intensive application (e.g., digital filtering, transformations, stream processing), which contain few control conditions. They proposed a recursive bi-partitioning algorithm to solve the memory partitioning problem using simulated execution traces of a set of embedded applications. Golubeva et al. [13] continue this line of investigation, considering the availability of a low-leakage sleep state for each memory block in a scratchpad memory. The partition algorithm proposed is based on a randomized search in the solution space. Finally, Loghi et al. [19] proposed an optimal partitioning algorithm based on an implicit enumeration of the partitioning solutions. They proved a theoretical property of the search space exploited to reduce the number of partition boundaries to be enumerated, making exhaustive exploration feasible. A set of applications taken from the MiBench [14] application suite were used to get execution traces. These works consider splitting the address space into multiple, contiguous memory banks. Consequently, the partition algorithm is restricted to finding the optimal boundaries between the memory banks.

Ozturk and Kandemir [20] proposed a series of techniques starting with the relocation and merge of memory blocks of the address space into memory banks, relaxing the aforementioned restriction. They formulated each of these techniques as an ILP (integer linear programming) problem, and solved them using a commercial solver. They target also data-intensive embedded applications (e.g. multimedia processing: image and video), which manipulates multidimensional arrays (with affine subscript expressions) of data using a series of nested loops (with compile time known bounds). Therefore, a static compiler analysis is used to extract data-access patterns, which in turn are the input for finding the solution. The explored techniques include nonuniform bank sizes, data migration, data compression, and data replication.

All mentioned works report energy savings in terms of a relative percentage of some baseline, i.e., the consumption of an equivalent monolithic memory. However, the presented results not only depend on the proposed technique, the memory architecture or the particular case study, but also on the selected technology. Since different technologies were chosen in each work, the comparison between them is difficult.

We follow a methodology similar to the one employed in [20], in which a memory access trace is used to solve an optimization problem for allocating the application memory divided in blocks to memory banks. Our work differs from the previous literature in three main aspects. First, we propose using banked memories with power management for code memory, in this case in WSN, exploiting the event-driven characteristics of this class of application. Second, we address the power management issue obtaining results for a greedy policy, one of the simplest possible management scheme, and an oracle policy, representing the best prediction algorithm. Therefore, we are able to assess the benefit of adopting an advanced memory bank state management. Finally, we derive expressions for energy savings based on a detailed model that favors the analysis of the different factors determining the effective energy saving.

4 Banked memory with power management

The key idea of memory banking with power management is to partition a memory in banks, which can be individually put in a low-leakage sleep state to reduce the static power. Since only one bank is active per access, the remaining idle banks can be put into the sleep state. However, the transition from sleep state to the ready state (also known as standby state) has an energy cost by itself. As a consequence, the leakage saving on the sleep state should compensate this bank wake-up cost. The overall wake-up energy cost considering all banks can be minimized by properly allocating the program code into the different banks. Highly correlated memory blocks allocated to the same bank leads to a bank access pattern with a high temporal locality, thus reducing the number of wake-ups. On the other hand, the partitioning overhead due to sleep transistors, extra wiring and duplication of address and control logic must be taken into account to find the optimum number of banks. Another concern is the memory organization that could be limited to equally-sized banks or could allow any bank size. In this work we consider only partitioning the memory in uniform banks.

In summary, the energy saving achieved using this technique highly depends on the selected number of banks, together with the allocation of memory blocks to the available banks. The proposed methodology is based on using a memory access trace to solve an optimization problem. The application binary program is divided in memory blocks, e.g. basic blocks or any other arbitrarily defined. Then, the memory access trace is obtained by simulation or execution of the program (directly from an executable format file) to get the access trace to the defined blocks. The access pattern to blocks and the memory configuration are input to the optimization solver that outputs the block-to-bank mapping that minimizes the energy consumption of the banked memory. The optimization also outputs the activation signals that control when a bank is sleeping or ready to be accessed. The memory configuration specifies the number of banks, the memory energy parameters, and the power management strategy used to control the banks states. Finally, the energy saving is obtained by simulating the application execution from the banked memory, in which the original program code was reallocated among the banks. Fig. 1 shows the described process flow.

The process may be repeated for different numbers of banks to find the optimum number by simply comparing the obtained energy saving in each step.

4.1 Memory power management

The memory bank states (sleep or ready) are defined using a given power management strategy. This strategy defines when a bank is put in sleep state and when it is woken up, and therefore includes the information if a bank remains in idle state even if it is not accessed. The basis of the chosen strategy may range from a very simple one to highly sophisticated prediction algorithms [4]. The energy savings depend much on the adopted strategy. Taking this into account, we consider two power management strategies: a simple greedy policy and the best-oracle policy. In the greedy strategy, as soon as a memory bank is not being accessed it is put into sleep state. The greedy policy is one of the simplest possible management schemes. Conversely, an oracle policy is based on the best prediction algorithm in the sense that follows. The optimization takes into account the whole access trace, including information of future access, to obtain the schedule of bank states that maximize the energy savings. In this way, we are able to assess the energy saving using two algorithms in opposite ends in terms of complexity.

Subsequently, the power management module must implement the algorithm in hardware. At runtime it must manage the bank states in accordance with the access patterns to the banks. The implementation of the greedy power management is straightforward. As soon as a new bank is accessed, it is woken up and the previously active bank is put in sleep state. The performance of the oracle policy that was obtained at the optimization stage can not be achieved at runtime, since there is no practical prediction algorithm that can beat the off-line optimum oracle strategy based on an execution trace. As a consequence, the energy saving obtained by the oracle policy represents the maximum achievable savings using any power management.

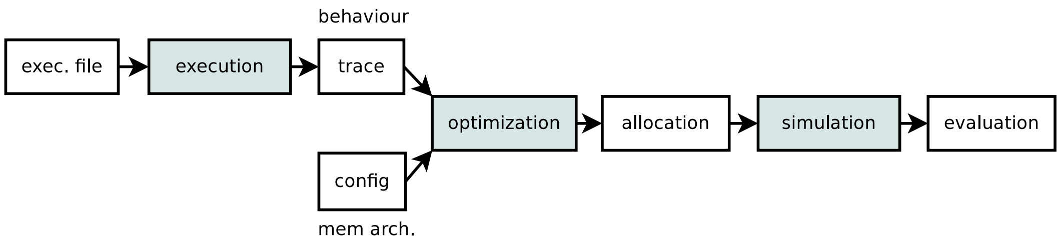

Fig. 1 Design flow.

4.2 Banked memory energy model

In this section we present a general memory energy model considering dynamic and static energy consumption. Then, the dependence of the energy on the memory size is modeled. These models are the basis for deriving the energy consumption expressions for the different memory organizations and the different power management strategies in Section 5.

4.2.1 Memory energy model

The static power consumed by a memory depends on its actual state: *ready* or *sleep*. During the ready state read or write cycles can be performed, but not in the sleep state. Since the memory remains in one of these states for a certain number of cycles, the static energy consumed can be expressed in terms of energy per cycle (E_{rdy} and E_{slp}) and the number of cycles in each state. Each memory access, performed during the ready state, consumes a certain amount of energy (E_{acc}). The ready period during which memory is accessed is usually called the active period, and the total energy spent corresponds to the sum of the access and the ready energy ($E_{act} = E_{acc} + E_{rdy}$), i.e., the dynamic and static energy. On the other hand, the ready cycles without access are called idle cycles, consuming only static energy ($E_{idl} = E_{rdy}$). Each state transition from sleep to active (i.e. the wake-up transition) has an associated energy cost (E_{wkp}) and a latency penalty, considered in Section 5.3.

Based on the parameters defined above, the total energy consumption of a memory can be defined as

$$E = E_{act}n_{act} + E_{idl}n_{idl} + E_{slp}n_{slp} + E_{wkp}n_{wkp}, \qquad (1)$$

where n_{act}, n_{idl} and n_{slp} are the number of the cycles in which the memory is in active, idle and in sleep state respectively, and n_{wkp} is the number of times the memory switches from sleep to active state. We define the ratios $r_k = n_k/n$ for $k \in \{act, idl, slp, wkp\}$ where n is the elapsed number of cycles. So the average energy per cycle is

$$\bar{E} = E_{act}r_{act} + E_{idl}r_{idl} + E_{slp}r_{slp} + E_{wkp}r_{wkp}. \qquad (2)$$

4.2.2 Energy variation with memory size

The basis of memory banking is that the energy increases with the size of the memory. Consequently, if a single bank memory is partitioned in several banks, each bank is expected to reduce its energy consumption. In order to evaluate the energy saving appropriately when a banked memory is adopted, we next model the energy consumption of a memory as a function of its size.

The energy values in Eq. (2), E_{act}, E_{idl}, E_{slp} and E_{wkp}, are generally considered simply proportional to the size of the memory [13]. We investigated the dependence of the involved parameters on the memory size, mainly using an estimation tool, CACTI [25]. However, not all parameters can be obtained from CACTI.

Table 1 lists the energy parameters and the used method to find the respective values. CACTI outputs the dynamic energy and the leakage power. The former value corresponds to the access energy of our model (E_{acc}). The leakage power is used to calculate the energy leakage per cycle, i.e., the idle energy of our model (E_{idl}), at the lowest maximum operating frequency among all memory sizes. The active energy (E_{act}) is directly computed (dynamic plus leakage).

The remaining energy parameters can not be derived directly from CACTI, so they are estimated values. The energy consumed per cycle in the sleep state (E_{slp}) is a fraction of the idle energy, since we suppose that a technique based on reducing the supply voltage is used to exponentially reduce the leakage. We assume a reduction factor of the leakage in sleep state of 0.1, which is a common value adopted in the literature [24, 19].

The wake-up transition is considered proportional to the memory size. In the literature it is reported that the proportionality constant respect to an access energy varies from about one [3] to hundreds [19]. In our case we adopt this factor respect to the active energy. We consider three different values to asses the impact of the wakeup energy cost on the energy savings, the aforementioned limits, i.e., one and one hundred, and an intermediate value of ten.

Table 2 Energy curve parameters as function of memory size.

Parameter	a (nJ/B)	b (adim.)	Model
E_{idl}	3.28×10^{-7}	1.09	linear function
E_{slp}	$E_{idl}/10$	same as E_{ilp}	linear function
E_{acc}	7.95×10^{-5}	0.48	square root function
E_{act}	1.78×10^{-6}	0.96	linear function
E_{wkp}	$K \cdot E_{act}, K = \{1, 10, 100\}$	same as E_{act}	linear function

Table 1 Energy parameters and method to find the respective values.

Parameter	Method
E_{acc}	read access: CACTI estimation
E_{idl}	leakage: CACTI estimation
E_{act}	direct calculation: $E_{acc} + E_{idl}$
E_{slp}	estimation, $E_{idl}/10$
E_{wkp}	estimation, $K \cdot E_{act}, K = \{1, 10, 100\}$

We follow the described method to get a set of parameters for each memory size, ranging from 512 B to 256 KB (in a sequence of power of two), in order to model the dependence of the memory energy with its size. The CACTI memory configuration was set for a pure RAM memory, one read/write port, 65 nm technology and a high performance ITRS transistor type. Each energy parameter was considered at a time and the values varying the memory size were fitted to a power function $\mathbf{E}(S) = aS^b$, where $\mathbf{E}(S)$ is the energy per cycle and S the memory size. The resulting fitting coefficients (the proportionality constant a and the exponent b) and the model adopted are presented in Table 2. Figure 2 shows all the energy estimated values and the fitted curves.

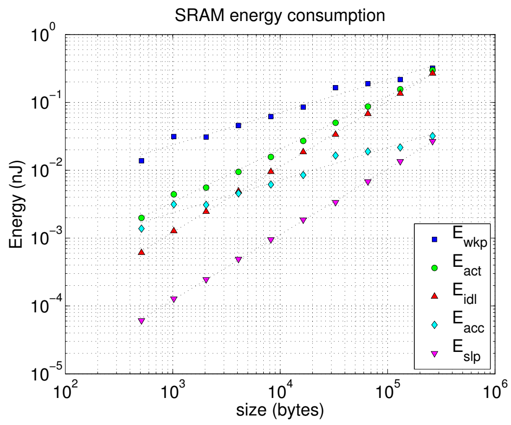

Fig. 2 Energy consumption per cycle as a function of the memory size.

The dependence on the memory size of the energy parameters obtained directly from CACTI can be explained by examining the simulation output and analyzing the relative contribution of each memory component. The leakage energy in idle state grows nearly linearly, because the memory-cell leakage represents about 70% of the total energy and the number of memory-cells is directly proportional to the memory size. The access energy varies approximately as the square root of the size. It can be observed that between 70% and 80% of the dynamic energy come from bit-lines, sense amps, and other resource shared between memory-cells. The active energy (access plus idle, dynamic plus leakage), finally ends up varying almost linearly with size (exponent equal to one), because the leakage energy becomes more important than the dynamic energy with increasing size. However, for small footprints a exponent less than one or a polynomial model should be used.

Hereafter, for sake of simplicity, we will work based on simple models, that is the active, idle, sleep and wake-up energy are proportional to the memory size:

$$\mathbf{E}_k(S) = a_k S \tag{3}$$

for $k \in \{act, idl, slp, wkp\}$, where S is the memory size in bytes, and a_k is the corresponding constant of proportionality in Table 2.

5 Energy saving expressions

Using Eq. (3) the energy consumption of a bank of size s in a banked memory of total size S can be modeled as

$$\mathbf{E}_k(s) = E_k \frac{s}{S}, \tag{4}$$

where $E_k = a_k S$ is the corresponding energy consumption per cycle of the whole memory.

Now, considering a banked memory of N equally sized banks Eq. (4) becomes

$$\mathbf{E}_k\left(\frac{S}{N}\right) = \frac{E_k}{N}. \tag{5}$$

The total energy consumption per cycle of the whole banked memory is

$$\bar{E}_N = \sum_{i=1}^{N} \frac{E_{act}}{N} r_{act_i} + \frac{E_{slp}}{N} r_{slp_i} + \frac{E_{wkp}}{N} r_{wkp_i}, \tag{6}$$

where the first two terms of the sum represent the active and sleep energy as a function of the fraction of active and sleep cycles performed by each bank i. The last term of the sum represents the wake-up energy as a function of the average wake-up rate of each memory bank, that is, the average number of cycles elapsed between two consecutive bank transitions from sleep to active (for example, one transition in 1000 cycles).

5.1 Energy saving for the greedy policy

When a greedy strategy is used each bank is either in active or sleep state (there are no idle cycles), $r_{slp_i} = 1 - r_{act_i}$, and one obtains

$$\bar{E}_N = E_{slp} + \sum_{i=1}^{N} \frac{1}{N} \left(E_{act} - E_{slp}\right) r_{act_i} + \frac{E_{wkp}}{N} r_{wkp_i}. \quad (7)$$

Since there is only one bank active per cycle, the sum of the active cycles for all banks is the total number of cycles

$$\sum_{i=1}^{N} r_{act_i} = 1 \quad (8)$$

so Eq. (7) simplifies to

$$\bar{E}_N = \frac{E_{act}}{N} + \left(\frac{N-1}{N}\right) E_{slp} + \frac{E_{wkp}}{N} \sum_{i=1}^{N} r_{wkp_i}. \quad (9)$$

We define the energy savings of a banked memory as the relative deviation of the energy consumption of a single bank memory which is always active ($E_1 = E_{act}$)

$$\delta E = \frac{E_1 - \bar{E}_N}{E_1}. \quad (10)$$

Thus, the energy saving of a banked memory of N uniform banks is

$$\delta E_N^{greedy} = \frac{N-1}{N} \left(1 - \frac{E_{slp}}{E_{act}}\right) - \frac{1}{N} \frac{E_{wkp}}{E_{act}} \sum_{i=1}^{N} r_{wkp_i}. \quad (11)$$

The first term is related to active consumption reduction, coming from having $N-1$ banks in sleep state and only one bank in active state. The last term, which is related to the cost of wake-ups, depends on the accumulated wake-up rate and is directly proportional to the wake-up to active energy ratio, and inversely proportional to the number of banks.

In order to maximize the energy saving in a memory having N uniform banks, the optimization algorithm must minimize the accumulated wake-up rate. Note that the optimum content distribution among the banks does not depend on the wake-up cost, but rather the wake-up cost determines the final energy saving. Furthermore, note that the energy saving does not depend on the access profile among the banks, since the access to every bank costs the same as all banks have the same size. Still, the allocation of blocks to banks must consider the bank size constraint. Finally, the energy saving can be improved by increasing N and at the same time keeping the accumulated wake-up rate low. The maximum achievable saving corresponds to the sleep to active rate, which is equivalent to have the whole memory in sleep state. Even so, the partition overhead limits the maximum number of banks.

5.2 Energy saving for the oracle policy

Consider a memory with a power management, different from greedy, by means of which a bank may remain in idle state, even if it will not be immediately accessed. In this case the total number of cycles is $n = n_{act_i} + n_{idl_i} + n_{slp_i}$ for all banks, so $r_{act_i} + r_{idl_i} + r_{slp_i} = 1$. In a similar way to the greedy policy, the expression for the energy savings can be determined as:

$$\delta E_N^{oracle} = \frac{N-1}{N} \left(1 - \frac{E_{slp}}{E_{act}}\right) - \frac{1}{N} \left(\frac{E_{idl} - E_{slp}}{E_{act}}\right) \sum_{i=1}^{N} r_{idl_i} - \\ - \frac{1}{N} \frac{E_{wkp}}{E_{act}} \sum_{i=1}^{N} r_{wkp_i}. \quad (12)$$

Compared to Eq. (11), Eq. (12) has an additional term, which is related to the energy increase caused by the idle cycles. This does not necessarily imply that the energy saving is reduced, since the accumulated wake-up ratio may decrease. This expression generalizes the model for the greedy strategy, which is obtained by setting r_{idl} equal to zero for all banks.

5.3 Effective energy saving

As mentioned previously, the wake-up transition from sleep to active state of a bank memory has an associated latency. This latency forces the microprocessor to stall until the bank is ready. The microprocessor may remain idle for a few cycles each time a new bank is woken up, incrementing the energy drain. This extra microprocessor energy can be included in the bank wake-up energy and for simplicity we will not consider it explicitly. If the wake-up rate is small and the active power of the microprocessor is much higher than idle power, this overhead can be neglected. Additionally, the extra time due to the wake-up transition is not an issue in low duty-cycle applications, since it simply increases the duty-cycle slightly.

On the other hand, the partitioning overhead must be considered to determine the effective energy saving. A previous work had characterized the partitioning overhead as a function of the number of banks for a partitioned memory of arbitrary sizes [19]. In that case the hardware overhead is due to an additional decoder (to translate addresses and control signals into the multiple control and address signals), and the wiring to connect the decoder to the banks [2]. As the number of memory banks increases, the complexity of the decoder is roughly constant, but the wiring overhead increases [19]. The partition overhead is proportional to the active energy of an equivalent monolithic memory and roughly linear in the number of banks, as can be clearly seen by inspecting the data of the aforementioned work (3.5%, 5.6%, 7.3% and 9% for a 2-, 3-, 4-, and 5-bank partitions, resulting in an overhead factor of approximately 1.8% per bank).

Consequently, the relative overhead energy can be modeled as:

$$\delta E_N^{ovhd} = k_{ovhd} N. \tag{13}$$

In this work, the memory is partitioned into equally-sized banks. As a result the overhead is expected to decrease leading to a lower value for the overhead factor.

The maximum effective energy saving, including the partition overhead, can be expressed subtracting Eq. (13) from Eq. (11) for the greedy policy and from Eq. (12) for the oracle policy. The maximum energy savings can be found for both cases as the idle contributions (for the oracle policy) and the wake-up contributions (for both policies) tend to zero:

$$\delta E_N^{max} = \frac{N-1}{N}\left(1 - \frac{E_{slp}}{E_{act}}\right) - k_{ovhd} N. \tag{14}$$

δE_N^{max} presents a maximum for

$$N_{opt} = \sqrt{\frac{1}{k_{ovhd}}\left(1 - \frac{E_{slp}}{E_{act}}\right)}. \tag{15}$$

The memory energy model along with the partition overhead, modeled as being proportional to the number of banks, allows to find an estimated value of the optimum number of banks.

6 Problem Formulation

In this section we define an integer linear program that minimizes the energy consumption of a banked memory with power management by optimally distributing the application code divided in blocks to memory banks.

The memory has N memory banks $B = \{1, \ldots, N\}$, of equal size $s_b, b \in B$.

The application code is divided in M memory blocks $D = \{1, \ldots, M\}$ of size $s_d, d \in D$. We are further given an access pattern to these blocks over time by a_{dt}. A value of $a_{dt} = 1$ indicates that block d is accessed at time t. We want to determine an allocation of blocks to banks that respects the size constraints, and an activation schedule of the banks that minimizes total energy consumption, and such that banks that are accessed at time t are ready at time t. Let $l_{db} \in \{0,1\}$ indicate that block d is allocated to bank b, and $o_{bt} \in \{0,1\}$ that bank b is ready at time t. We define auxiliary indicator variables $a_{bt} \in \{0,1\}$ representing the access of bank b at time t, $o_{bt}^{+} \in \{0,1\}$ representing the wake-up transition of bank b at time t. Let further $T = \{1, \ldots, t\}$ be set of access times. We assume that time 0 represents the initial state where all banks are in sleep state. For a given number of banks, the partition overhead is fixed, hence the problem formulation does not need to include this term.

Now we can model the problem of finding the allocation and power management strategy by the following integer program:

$$\text{minimize} \sum_{\substack{t\in T\\ b\in B}} E_{acc} a_{bt} + E_{rdy} o_{bt} + E_{wkp} o_{bt}^{+} \tag{16}$$

subject to

$$\begin{aligned}
o_{bt}^{+} &\geq o_{bt} - o_{b,t-1} && \forall b \in B, t \in T && (17)\\
o_{bt} &\geq a_{bt} && \forall b \in B, t \in T && (18)\\
a_{bt} &= \sum_{d\in D} l_{bd} a_{dt} && \forall b \in B, t \in T && (19)\\
\sum_{b\in B} l_{db} &= 1 && \forall d \in D && (20)\\
\sum_{d\in D} l_{db} s_d &\leq s_b && \forall b \in B && (21)\\
o_{b0} &= 0 && \forall b \in B && (22)\\
l_{db} &\in \{0,1\} && d \in D, b \in B && (23)\\
o_{bt} &\in \{0,1\} && b \in B, t \in T \cup \{0\} && (24)\\
o_{bt}^{+}, a_{bt} &\in \{0,1\} && b \in B, t \in T. && (25)
\end{aligned}$$

Eq. (17) defines wake-up transitions: if some bank is ready at time t, but has not been ready at time $t-1$, a wakeup transition occurred.[1] Eq. (18) and (19) define the access pattern for a given allocation. Restriction (20) guarantees that every block has been allocated to exactly one memory bank, and restriction (21) limits the total size of the allocated blocks to the size of the bank.

[1] Since the variables involved in the inequalities are binary, $a \geq b$ corresponds to the logical implication, $a \Rightarrow b$.

The above formulation corresponds to the best-oracle strategy, since it does not limit the activation schedules. For a greedy power management the constraint (18) can be modified, so that a bank is ready only when it is accessed.

$$o_{bt} = a_{bt} \qquad \forall b \in B, t \in T. \tag{26}$$

7 Experiments

In order to assess the whole energy savings, it is needed to determine, first, the relative importance of the memory in the overall mote energy consumption, and second, the energy reduction in the memory subsystem when the proposed technique is adopted.

The benchmarks previously used for memory energy evaluation, such MiBench [14], are not adequate for evaluating our memory architecture. In most cases, each benchmark is the implementation of an algorithm compiled as an independent application. The application, executed in batches, usually reads data inputs for the algorithm form files and outputs the processing results to the console or to a file. The motes have a limited amount of memory, and usually do not have a file system, preventing using Mibench in motes as is. But the main limitation is that they do not capture the external event timing [1] needed to evaluate the memory regions that are idling. So, we inclined towards using real WSN applications for our case studies.

The criteria for selecting the cases were: public availability of source files, realistic and ready-to-use application. We intentionally left out simple code examples that do not reflect the requirements of complex real-life systems. Unfortunately, the number of cases complying with the aforementioned restrictions are scarce. We chose two data-collection applications from the standard distribution of TinyOS (version 2.1.0)[2] and ContikiOS (release 2.5)[3]. Both applications are similar, each node of the network periodically samples a sensor and the readings are transmitted to a sink node using a network collection protocol. MultihopOscilloscope (TinyOS) uses CTP (Collection Tree Protocol) [12] and rpl-collect (ContikiOS) uses RPL (IPv6 Routing Protocol for Low power and Lossy Networks) [28].

The applications were compiled for a telosb node [22] based on a MSP430 microcontroller[4]. Table 3 summarizes the section sizes of the selected applications. It can be observed that in both cases the code memory (text segment) is between five and nine times larger than the data memory (bss plus data segment). This relationship, which is typical in WSNs applications, motivates using a banked memory with power management for code memory rather than for data memory.

For the purpose of estimating the relative weight of the memory subsystem in the overall energy budget, we considered, first, that the memory energy consumption represents a fixed amount of the total processor consumption. Second, we estimated the energy contribution of the processing (including the memory energy) and the communication activities, using the Energest module on a mote operating in the field running the rpl-collect application. Energest [8], included in the ContikiOS distribution, measures the accumulated time spent in the different power levels of the microcontroller and the radio of a mote operating in the field. The power levels associated to the different power states of the microcontroller and the radio were measured in the laboratory. Finally, each average power contribution is computed as the measured duty-cycles multiplied by the corresponding power level.

The mote program memory is allocated to the banked memory by solving the aforementioned optimization problem using memory access traces. Since current sensor nodes do not support real-time execution trace generation, we simulated the network using COOJA [9]. The telosb node-level simulation relies on MSPsim, an instruction-level emulator for the MSP430 microcontroller, which also simulates hardware peripherals such as sensors, radio modules or LEDs. MSPSim is designed to be used as a COOJA plug-in, allowing to access to the MSPSim command-line client from COOJA. We modified MSPSim's code to add a new command for controlling the debug mode, so that it is possible to obtain any node execution trace from COOJA. For the experiments we set up an unique scenario based on a configuration consisting of a network composed of 25 nodes. The memory access traces were trimmed to consider 5000 cycles. The size of the blocks could be chosen regular (equally sized) or irregular, ranging from the minimum basic blocks to an arbitrary size. For the sake of simplicity, the block set was selected as those defined by the program functions and the compiler generated global symbols (user and library functions, plus those created by the compiler). The size of the blocks ranges from tens to hundreds of bytes, in accordance with the general guideline of writing short functions, considering the run-to-completion characteristic of TinyOS and any non-preemptive event-driven software architecture, such as ContikiOS. The energy parameters for the memory are as described in Table 2 in Section 4. The total memory size was considered 10% larger than the application size, to ensure the feasibility of the solution.

[2] www.tinyos.net
[3] www.contiki-os.org
[4] www.ti.com/msp430

Table 3 Application parameters (size in bytes).

OS	Application	text	bss	data	#functions
TinyOS	MultihopOscilloscope	32058	122	3534	1081
ContikiOS	rpl-collect (udp-sender)	47552	232	9250	489

For each experiment the bank memory access patterns a_{bt} have been determined using the trace a_{dt} and the allocation map l_{bd}. For the best-oracle power management the solution also outputs o_{bt}, the bank states for each cycle (i.e.,ready or sleep). The average energy consumption is calculated using the memory energy parameters and the energy saving is determined by comparing with a single bank memory with no power management. We repeated the experiment up six banks, for both power management strategies, comparing the predicted energy savings by our model to the optimal energy savings obtained for different number of banks and wake-up energy cost.

Finally, the overall mote energy savings is estimated using the memory savings achieved using the optimal number of banks, and the relative weight of the memory consumption in the total energy budget.

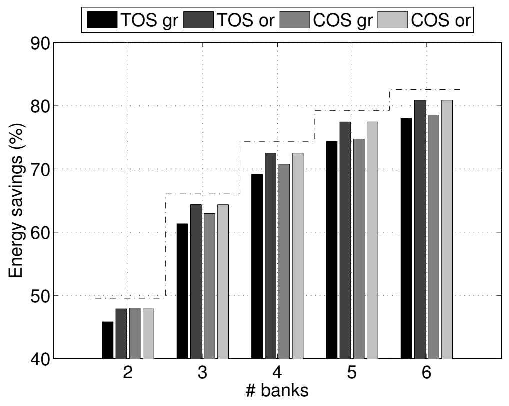

Fig. 3 Energy savings as a function of the number of banks for best-oracle and greedy policy (denoted *gr* and *or*) in TinyOS and ContikiOS applications (denoted *TOS* and *COS*) and the theoretical limit (dashed line).

8 Results and Discussion

In this section we present the results of the experiments described previously. First, we present the memory energy savings achieved using our proposal. We compare the predicted energy savings by our model to the optimal energy savings obtained by solving the ILP for different number of banks, power managements policies and wake-up energy cost. Next, the overall system energy savings are estimated for different workload scenarios.

8.1 Memory energy savings

Fig. 3 shows the energy savings for the intermediate value of wake-up energy (ten times the active energy) as a function of the number of banks for best-oracle and greedy policy in both applications (based on TinyOS and ContikiOS). As the number of banks increases, the energy savings approach the corresponding value of having all banks in sleep state. In this figure we have intentionally discarded the partition overhead, considered later. The figure shows that the oracle policy outperforms the greedy policy for both applications, as expected, and both are within 2% and 5% of the theoretical limit for the energy savings. In all cases, except for six banks, ContikiOS outperforms TinyOS by a narrow margin. The results presented hereafter are similar for both applications, and only the corresponding to TinyOS are analyzed more deeply.

Fig. 4 shows the fraction of cycles and the energy breakdown for a memory having five banks of equal size, where each contribution (i.e., access, ready, sleep, wake-up) is averaged among the different banks. The upper part clearly shows that the fraction of access cycles is equal in both cases and represents 20% of the total number of cycles, since five banks are considered (only one bank of N is active, in this case five).

For the greedy policy the number of ready cycles is equal to the access cycles, since both correspond to the active compound state. On the other hand, for the oracle policy part of the ready cycles correspond to active cycles, and the rest to idle cycles, in which the banks are ready but not accessed. Moreover, for the greedy policy 80% of the cycles are sleep cycles ($N-1$ banks are in sleep state) while for the oracle policy this percentage is slightly larger, used to reduce the wake-up cycles from 0.5% to 0.12 % (not visible in Fig. 4). The energy breakdown, Fig. 4 (lower part), shows that the difference between oracle and greedy comes mainly from the wake-up transitions. In this case study, due to its event-driven nature, the code memory access patterns are triggered by events, leading to a chain of function calls starting with the interrupt subroutine. This chain may include the execution of subsequent functions calls starting with a queued handler function called by a basic scheduler. The allocation of highly correlated func-

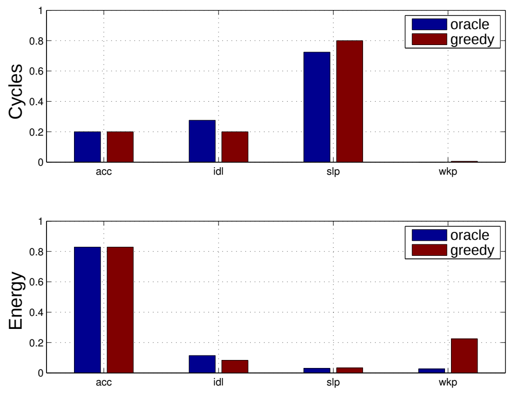

Fig. 4 Fraction of cycles and energy breakdown where each contribution is averaged among the different banks.

Table 4 Optimum number of banks as a function of partition overhead.

$k_{ovhd}(\%)$	0	1	2	3	5
N_{opt}	∞	10	7	6	4
$\delta E^{max}_{N,eff}(\%)$	98.2	78.3	70.1	63.8	53.6

tions to the same bank leads to a bank access pattern with a high temporal locality. Hence, the total wake-up fraction across the banks is very low. This explains the modest gain of applying the best-oracle policy.

The optimum number of banks estimated using Eq. (15) (after rounding) as a function of k_{ovhd} (1%, 2%, 3% and 5%) is shown in Table 4. The energy savings is limited by the partition overhead, reaching a maximum of 78.3% for an overhead of 1%. The energy saving limit, as the partition overhead tends to zero and N to infinity, is 98.2% $(1 - E_{slp}/E_{act})$.

Table 5 compares the energy saving results as a function of the number of banks and the partition overhead. In the upper part, the table gives the maximum achievable savings calculated using Eq. (14). It can be observed that with a partition overhead of 3% the optimum number of banks is six, whereas with 5% is four, both highlighted in gray. In the middle part of the table it can be observed that the maximum energy saving for greedy strategy with 3% and 5% of partition overhead is achieved for six and five banks respectively, different from the estimated value using the maximum achievable savings. This means that the saving loss due to wake-up transitions shifts the optimum number of banks. Similar results are obtained for the best-oracle strategy, but with higher energy savings.

Finally, Fig. 5 shows the energy savings for different wake-up energy costs: one, ten (analyzed so far) and one hundred. It can be seen that for high wake-up energy costs the oracle policy outperforms the greedy policy by ten percentage points, while for a low wake-up cost the

Table 5 Energy saving comparison: maximum, greedy and oracle.

maximum		number of banks				
		2	3	4	5	6
$k_{ovhd}(\%)$	1	47.08	62.44	69.62	73.53	75.80
	2	45.08	59.44	65.62	68.53	69.80
	3	43.08	56.44	61.62	63.53	63.80
	5	39.08	50.44	53.62	53.53	51.80
greedy		**number of banks**				
		2	3	4	5	6
$k_{ovhd}(\%)$	1	43.82	58.33	65.18	69.36	71.99
	2	41.82	55.33	61.18	64.36	65.99
	3	39.82	52.33	57.18	59.36	59.99
	5	35.82	46.33	49.18	49.36	47.99
oracle		**number of banks**				
		2	3	4	5	6
$k_{ovhd}(\%)$	1	46.40	61.88	69.12	73.07	75.41
	2	44.40	58.88	65.12	68.07	69.41
	3	42.40	55.88	61.12	63.07	63.41
	5	38.40	49.88	53.12	53.07	51.41

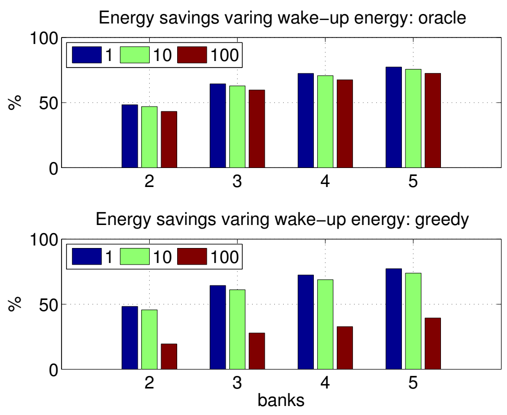

Fig. 5 Energy savings as a function of the number of banks for best-oracle and greedy policy (denoted *gr* and *or*) for increasing wake-up cost factor (TinyOS application.

difference is marginal. The energy increase due to the wake-up transitions, the last factor in Eqs. (11) and (12) is proportional to the wake-up cost. This saving lost can be reduced using the oracle policy by increasing the idle cycles with low relative energy cost.

In summary, a huge energy saving in the memory subsystem can be obtained using a banked memory, achieving close to 80% energy reduction for a partition overhead of 1% with a memory of ten banks with moderate wake-up cost.

8.2 Overall system energy savings

Table 6 shows the power reduction in a processor when a banked memory as the proposed in the present work is adopted. All values are normalized to the active power of the original processor. If the memory consumption

Table 6 Power reduction in processing due to memory optimization (relative to original processor).

Power	original	w/banked mem.
Total	1	0.46
Memory	0.69	0.15
The rest	0.31	0.31

Table 7 Average power (energy) reduction in rpl-collect application, estimated using measured duty-cycles.

	duty-cycle (%)	original		w/banking me	
		P_{inst}	P_{avg}	P_{inst}	P_{av}
Processing	3.19	1	0.0319	0.4598	0.0146
Communication	0.54	10	0.0543	10	0.0543
Sleep	96.81	0.005	0.0048	0.005	0.0048
Total			0.0910		0.0737

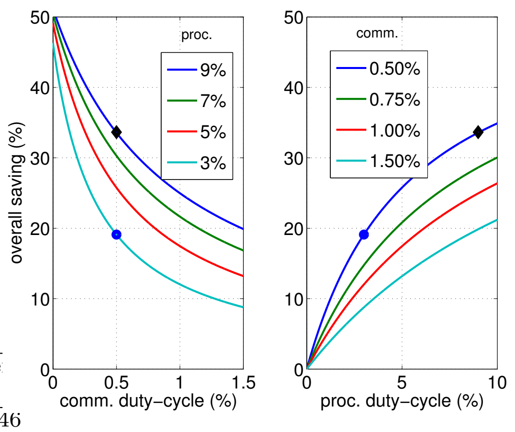

Fig. 6 Overall energy savings as a function of the processing and communication duty-cycles.

represents 69% of the total energy (as discussed in Section 2) and it is reduced by 78.3% (the maximum achievable when using ten banks with a partition overhead of 1%) the total processor consumption is reduced from 1 to 0.46, i.e. the processor consumption is reduced by 54%.

The measured duty-cycles of a node running the rpl-application are shown in Table 7. The node is processing (processor active) 3.19% of the time, communicating (radio transmitting or receiving) 0.54%, and the node is in sleep mode 96.81% of the time. The table also shows the respective instantaneous and average power (duty-cycle multiplied by the instantaneous power), and the total power. Next, it is presented the instantaneous power consumption for a node with a processor with a banked memory with power management (radio and sleep power remains unchanged), and the average power for the same set of duty-cycles. Results shows a reduction of 18.9% in the total power consumption, including radio communication consumption and sleep power.

These example application do not process the acquired data, but simply send the raw data to a sink, so that the processing workload corresponds only to protocol stack and OS housekeeping. Ko et. al. [17] review emerging wireless sensing applications that involve high-performance or high-resolution signal processing, together with their platform requirements, particularly processing power and energy consumption. For relative high-frequency sampling applications (e.g. structural health monitoring acquiring vibrations signals, some medical signal such as ECG), the processing workload would rise the duty-cycle, thus incrementing processor active power and the opportunity of energy savings.

The final overall savings depends on many factors, such as the relative importance of the memory consumption in the processor, the processing and communication requirements of protocols and applications. Fig. 6 shows the overall savings varying the processing and communication duty-cycles. The left part of the figure shows the saving as the communication duty-cycles varies from 0% to 1.5% for increasing processing duty-cycles (from 3% to 9%). For reduced communication duty-cycle the associated average power becomes increasingly negligible, and the overall saving approach to the processor saving, close to 54% (limited only by the sleep power). The right part of the figure shows the saving as the processing duty-cycles varies from 0% to 10% for different communication duty-cycles (from 0.5% to 1.5%). For high-performance processing workload the processor's memory consumption becomes more important and the savings rise. Let's consider the measured duty-cycle in the rpl-application as a point of reference, close to 0.5% for communication and 3% for processing. If the processing duty-cycle triples, reaching 9%, the overall consumption reduction rises from 19.1% (point marked as a blue dot in Fig.6) to 33.6% (point marked as a black diamond).

9 Conclusions

We have found that aggressive energy savings in the memory subsystem can be obtained using a banked memory with up to 78.3% energy reduction for a partition overhead of 1% with a memory of ten banks. The energy savings increase as a function of the number of banks. The maximum saving is limited by the partition overhead. An estimation of the optimum number of banks can be obtained at design time, using the energy memory parameters values and the partition overhead.

We evaluated the benefits of using a partitioned memory in WSNs by simulation of two real WSN applications, one based on TinyOS and the other on ContikiOS. The energy saving is maximized by properly allocating the program memory to the banks in order to minimize the accumulated wake-up rate and the idle cycles. The optimum number of banks may differ from the estimated value, because of the saving loss due to wake-up transitions. Nevertheless, the estimated value can be used to quickly find the optimum, by restricting the search to its vicinity.

The energy saving obtained by simulations were compared with the limits given by the derived expressions, showing a good correspondence. The oracle policy outperforms the greedy policy as expected. However, the extra benefit of the oracle over the greedy policy is significant only for high wake-up energy costs. Conversely, for relative low wake-up energy costs, the difference between oracle and greedy is scarce, and the additional benefit of using an advanced algorithm to predict future access to banks must justify the increased complexity and compensate the extra energy and area cost.

The effective overall savings obtained naturally depends on the relative weight of the memory consumption within the processor (compared to the arithmetic, logic and so on), the processing workload and communication average power. Our experiments indicates that reducing the memory energy by 78.3% would reduce the overall energy in about 19%, including the radio and sleep power. Our proposal has a major impact on applications with heavy processing requirements, where the processing duty-cycle triples the measured in the considered case study, reaching energy savings close to 34%.

We are currently extending our model to support arbitrary sized banks.

References

1. Tobias Becker, Peter Jamieson, Wayne Luk, Peter Y. K. Cheung, and Tero Rissa. Towards benchmarking energy efficiency of reconfigurable architectures. In *2008 International Conference on Field Programmable Logic and Applications*, pages 691–694. IEEE, 2008.
2. L. Benini, L. Macchiarulo, A. Macii, and M. Poncino. Layout-driven memory synthesis for embedded systems-on-chip. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 10(2):96–105, April 2002.
3. A. Calimera, L. Benini, A. Macii, E. Macii, and M. Poncino. Design of a Flexible Reactivation Cell for Safe Power-Mode Transition in Power-Gated Circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(9):1979–1993, September 2009.
4. A. Calimera, A. Macii, E. Macii, and M. Poncino. Design Techniques and Architectures for Low-Leakage SRAMs. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 59(9):1992–2007, 2012.
5. C. Cano, B. Bellalta, A. Sfairopoulou, and M. Oliver. Low energy operation in WSNs: A survey of preamble sampling MAC protocols. *Computer Networks*, 55(15):3351–3363, October 2011.
6. G. Chen, F. Li, M. Kandemir, O. Ozturk, and I. Demirkiran. Compiler-Directed Management of Leakage Power in Software-Managed Memories. In *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, volume 00, pages 450–451. IEEE, 2006.
7. William J. Dally, James Chen, R. Curtis Harting, James Balfour, David Black-Shaffer, Vishal Parikh, David Sheffield, and Jongsoo Park. Efficient Embedded Computing. *Computer*, 41(7):27–32, July 2008.
8. Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based On-line Energy Estimation for Sensor Nodes. In *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, June 2007.
9. Joakim Eriksson, Fredrik Österlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Voigt, Robert Sauter, and Pedro J. Marrón. COOJA/MSPSim: interoperability testing for wireless sensor networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 1–7, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
10. Amir H. Farrahi, Gustavo E. Téllez, and Majid Sarrafzadeh. Memory segmentation to exploit sleep mode operation. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, DAC '95, pages 36–41, New York, NY, USA, 1995. ACM.
11. Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. Quanto: tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 323–338, Berkeley, CA, USA, 2008. USENIX Association.
12. Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 1–14, New York, NY, USA, 2009. ACM.
13. Olga Golubeva, Mirko Loghi, Massimo Poncino, and Enrico Macii. Architectural leakage-aware management of partitioned scratchpad memories. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 1665–1670, San Jose, CA, USA, 2007. EDA Consortium.
14. M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14. IEEE, 2001.
15. M. Hempstead, D. Brooks, and G. Wei. An Accelerator-Based Wireless Sensor Network Processor in 130 nm CMOS. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 1(2):193–202, June 2011.
16. Jeong G. Ko, Nicolas Tsiftes, Adam Dunkels, and Andreas Terzis. Pragmatic low-power interoperability: ContikiMAC vs TinyOS LPL. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th*

Annual IEEE Communications Society Conference on, pages 94–96. IEEE, June 2012.

17. JeongGil Ko, Kevin Klues, Christian Richter, Wanja Hofer, Branislav Kusy, Michael Bruenig, Thomas Schmid, Qiang Wang, Prabal Dutta, and Andreas Terzis. Low power or high performance? a tradeoff whose time has come (and nearly gone). In *Proceedings of the 9th European conference on Wireless Sensor Networks*, EWSN'12, pages 98–114, Berlin, Heidelberg, 2012. Springer-Verlag.
18. J. Kwong, Y. Ramadass, N. Verma, M. Koesler, K. Huber, H. Moormann, and A. Chandrakasan. A 65nm Sub-Vt Microcontroller with Integrated SRAM and Switched-Capacitor DC-DC Converter. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 318–616. IEEE, February 2008.
19. Mirko Loghi, Olga Golubeva, Enrico Macii, and Massimo Poncino. Architectural Leakage Power Minimization of Scratchpad Memories by Application-Driven Subbanking. *IEEE Transactions on Computers*, 59(7):891–904, July 2010.
20. Ozcan Ozturk and Mahmut Kandemir. ILP-Based energy minimization techniques for banked memories. *ACM Trans. Des. Autom. Electron. Syst.*, 13(3):1–40, July 2008.
21. M. A. Pasha, S. Derrien, and O. Sentieys. A complete design-flow for the generation of ultra low-power WSN node architectures based on micro-tasking. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 693–698. IEEE, June 2010.
22. J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369. IEEE, April 2005.
23. A. Prayati, Ch Antonopoulos, T. Stoyanova, C. Koulamas, and G. Papadopoulos. A modeling approach on the TelosB WSN platform power consumption. *Journal of Systems and Software*, 83(8):1355–1363, August 2010.
24. Jan Rabaey. Optimizing Power @ Standby Memory. In *Low Power Design Essentials*, Integrated Circuits and Systems, pages 233–248. Springer US, 2009.
25. Shyamkumar Thoziyoor, Jung H. Ahn, Matteo Monchiero, Jay B. Brockman, and Norman P. Jouppi. A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies. In *2008 International Symposium on Computer Architecture*, pages 51–62, Washington, DC, USA, June 2008. IEEE.
26. Frits Vaandrager. *Introduction*, volume 1494 of *Lecture Notes in Computer Science*, chapter 1, pages 1–3. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.
27. N. Verma. Analysis Towards Minimization of Total SRAM Energy Over Active and Idle Operating Modes. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(9):1695–1703, 2011.
28. T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), March 2012.