A new memory banking system for energy-efficient wireless sensor networks

Leonardo Steinfeld*, Marcus Ritt†, Luigi Carro†, Fernando Silveira*
*Instituto de Ingenieria Electrica, Facultad de Ingenieria, Universidad de la Republica, Uruguay. {leo, silveira}@fing.edu.uy
†Instituto de Informatica, Universidade Federal do Rio Grande do Sul, Brasil. {carro, marcus.ritt @inf.ufrgs.br

***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. In this work, we propose a new strategy for memory banking, taking advantage of the software properties of WSN, and achieving aggressive power savings. We present a detailed model for the energy saving for equally sized banks with two power management schemes: a best-oracle policy and a simple greedy policy. Thanks to our modeling, at design time the optimum number of banks can be estimated, and the design can reach huge 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 application (based on TinyOS and ContikiOS). Experimental results show an energy reduction of up to 77.4% for a partition overhead of 1%.**

I. 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 to meet the demands of the ever-increasing complexity of applications. It is widely accepted that efforts toward energy reduction should target communication and processing [10].

In the last years, there has been a lot of research dealing with processing power optimization resulting in a variety of ultra-low power processors. These processors pose a primary energy limitation for SRAM, where the embedded SRAM consumes most of the total processor power [14] [7]. Partitioning a SRAM memory into multiple banks that can be independently accessed reduces the dynamic power consumption, and since only one bank is active per access, the remaining idle banks can be put into a low-leakage sleep state to also reduce the static power. However, the power and area overhead due to the extra wiring and duplication of address and control logic prohibits 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.

Memory banking has been applied for code and data using scratch-pad and cache memories in applications with high performance requirements (e.g. [6],[9]). We follow the methodology employed in [9], in which a memory access trace is used to solve an optimization problem for allocating the application memory divided in blocks to memory banks. However, to the best of our knowledge, this is the first time banked memories are considered for WSNs, leading to meaningful power savings, as it will be shown.

The main contribution of this work is to show that, thanks to our new problem formulation, one can find the optimum partitioning of memory banks in very common 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 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. We show that using our approach in a banked memory leads to aggressive (close to 80%) energy reduction in WSN applications. Our results suggest that adopting an advanced power management must be carefully evaluated, since the best-oracle is only marginally better than greedy.

The remainder of this paper is organized as follows. In Section II, we present a memory energy model, and in Section III we derive expressions for the energy savings of a banked memory. In Section IV we formulate the memory allocation and power management as an integer linear program (ILP). The experiments are presented in Section V and in Section VI we discuss the results. Finally Section VII contains concluding remarks and research directions.

II. Banked memory energy model

First, 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, in Section III, the energy consumption expressions for the different memory organizations and the different power management strategies.

A. 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

TABLE I. CURVE FITTING PARAMETERS.

	E_{acc}	E_{idl}	E_{act}	unit
a	7.95×10^{-5}	3.28×10^{-7}	1.78×10^{-6}	nJ/byte
b	0.48	1.09	0.96	-

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, considered later (Section III-C).

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

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

where n_{act}, n_{idl} and n_{slp} are the sum 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.

B. Energy variation with memory size

In this subsection the energy variation with the memory size is modeled in order to appropriately evaluate the energy saving when a banked memory is used. The energy values in Eq. (1) depend on the size of the memory, and generally the energy is considered simply proportional to it [6]. We investigated the dependence of the dynamic and static power on the memory size using the CACTI tool [13]. We obtained simulation results for a pure RAM memory, one read/write port, 65 nm technology and a high performance ITRS transistor type, varying its size from 512 B to 256 KB. CACTI outputs the dynamic and leakage energy, corresponding to the access and idle of our model. The active energy is directly computed (dynamic plus leakage). The data for the access, idle and active energy 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 are presented in Table I and Figure 1 shows the simulated data and the fitted curve.

The energy dependence on the memory size 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 dynamic energy varies approximately as the square root of the size. It could 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, 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. Hereafter, for sake of simplicity, we will work based on this approximation, that is, active energy is proportional to the memory size. However, for small footprints a exponent less than the unity should be used.

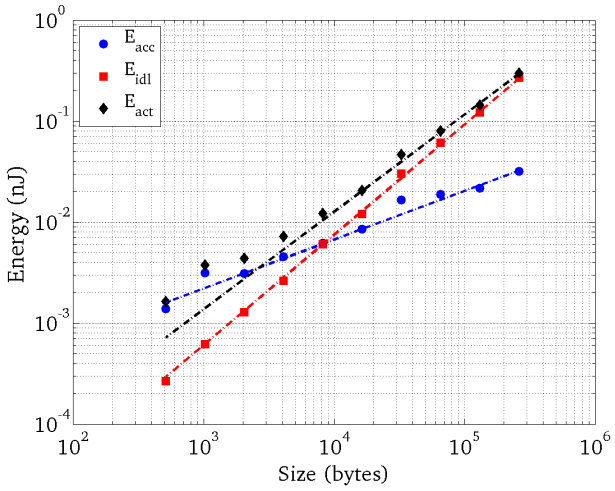

Fig. 1. Energy consumption per cycle as a function of the memory size.

Consider the remaining energy parameters in Eq. (1), sleep and wake-up energy. The energy consumed per cycle in the sleep state 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 generally accepted in the literature [12]. Finally, before a memory bank could be successfully accessed, the memory cells need to go back from the data retention voltage to the idle voltage, which involves the loading of internal capacitances. Since the involved currents in this process are similar to those in an access cycle, the associated wake-up energy cost is proportional to the access energy, ranging the proportionality constant from about 1 [2] to hundreds [8]. We adopt an intermediate value of 10.

Summarizing, the active, idle and sleep energy per cycle, and wake-up transition energy are modeled as being proportional to the memory size:

$$\mathbf{E}_k(S) = a_k S \quad (2)$$

for $k \in \{act, idl, slp, wkp\}$, where S is the memory size in bytes, and a_k is the corresponding constant of proportionality. The parameter a_k is determined using the respective values of Table I, and the factors mentioned before, that is, a factor of 0.1 of the idle energy is consumed in the sleep state and ten times the access is consumed in a wake-up transition.

III. ENERGY SAVING EXPRESSIONS

The energy savings depend on the memory organization, which may be limited to equally-sized banks, or allow any bank size. It further depends on the strategy for the bank states management that may range from a greedy policy to the use of more sophisticated prediction algorithms [3].

In this section we derive expressions for the energy savings of a memory of equally sized banks with two different management schemes: greedy and oracle. In the greedy policy as soon as a memory bank is not being accessed it is put into sleep state. Therefore, each memory bank is in one of the

following states: active or sleep. On the contrary, in the best-oracle policy a bank may remain in the idle state even if it is not beeing accessed.

A. Energy saving with a greedy policy

Using Eq. (2) the energy consumption of a bank of size s in a banked memory of total size S can be modeled as

$$\mathbf{E}_k(s) = E_k \frac{s}{S}, \tag{3}$$

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. (3) becomes

$$\mathbf{E}_k\left(\frac{S}{N}\right) = \frac{E_k}{N}. \tag{4}$$

The total energy consumption per cycle of the whole memory after n cycles is

$$\bar{E}_N = \frac{1}{N}\sum_{i=1}^{N} E_{act}\frac{n_{act_i}}{n} + E_{slp}\frac{n_{slp_i}}{n} + E_{wkp}\frac{n_{wkp_i}}{n}, \tag{5}$$

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).

Since in greedy policy each bank is in active or sleep state, the total number of cycles is $n = n_{act_i} + n_{slp_i}$, then we obtain

$$\bar{E}_N = E_{slp} + \frac{1}{N}\sum_{i=1}^{N}\left(E_{act} - E_{slp}\right)\frac{n_{act_i}}{n} + E_{wkp}\frac{n_{wkp_i}}{n}. \tag{6}$$

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 ($E_1 = E_{act}$)

$$\delta E = \frac{E_1 - \bar{E}_N}{E_1}. \tag{7}$$

The energy saving of a banked memory of N uniform banks is

$$\begin{aligned}\delta E_N^{greedy} &= 1 - \frac{E_{slp}}{E_{act}} - \frac{1}{N}\sum_{i=1}^{N}\left(1 - \frac{E_{slp}}{E_{act}}\right)\frac{n_{act_i}}{n} + \\ &+ \frac{E_{wkp}}{E_{act}}\frac{n_{wkp_i}}{n}.\end{aligned} \tag{8}$$

Since there is only one bank active per cycle

$$\sum_{i=1}^{N} n_{act_i} = n \tag{9}$$

and Eq. (8) simplifies to

$$\delta E_N^{greedy} = \frac{N-1}{N}\left(1 - \frac{E_{slp}}{E_{act}}\right) - \frac{1}{N}\frac{E_{wkp}}{E_{act}}\sum_{i=1}^{N}\frac{n_{wkp_i}}{n}. \tag{10}$$

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 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 constraints of the banks size. 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.

B. Energy saving with 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. In a similar way to the greedy policy, the expression for the energy savings can be determined as:

$$\begin{aligned}\delta E_N^{oracle} &= \frac{N-1}{N}\left(1 - \frac{E_{slp}}{E_{act}}\right) - \\ &- \frac{1}{N}\left(\frac{E_{idl} - E_{slp}}{E_{act}}\right)\sum_{i=1}^{N}\frac{n_{idl_i}}{n} - \\ &- \frac{1}{N}\frac{E_{wkp}}{E_{act}}\sum_{i=1}^{N}\frac{n_{wkp_i}}{n}.\end{aligned} \tag{11}$$

Compared to Eq. (10), Eq. (11) has an additional term, which is related to the energy increase caused by the idle cycles. This does not mean that the energy saving is reduced, since the accumulated wake-up ratio may decrease. This expression is general and includes also the greedy strategy, by setting n_{idl} equal to zero for all banks.

C. 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 waken 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 simply slightly increases the duty-cycle.

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 [8]. 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 [1]. As the number of memory banks increases, the complexity of the decoder is roughly constant, but the wiring overhead increases [8]. The partition overhead is proportional to the active energy of an equivalent monolithic memory and roughly linear with 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:

$$\delta E_N^{ovhd} = k_{ovhd} N. \tag{12}$$

In this paper, 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.

D. Energy savings limits

The energy savings in the limit, as the wake-up and idle contributions tend to zero, is

$$\delta E_N^{max} = \frac{N-1}{N}\left(1 - \frac{E_{slp}}{E_{act}}\right), \tag{13}$$

valid for the oracle and greedy.

Now, considering the partition overhead, Eq. (12), the maximum effective energy saving is

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

is maximized for

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

The optimum number of memory banks can be estimated at design time, provided that the energy memory parameters are given, such as energy consumption characteristics and the partition overhead as a function of the number of banks.

IV. 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 $B = \{1, ..., N\}$, of equal size $s_b, b \in B$. The application code is divided in M memory blocks $D = \{1, ..., M\}$ 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 $T = \{1, ..., t\}$ 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:

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

subject to

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

Eq. (17) define wake-up transitions: if some bank is ready at time t, but has not been ready at time $t-1$, a wakeup transition occurred. Eq. (18) and (19) define the access pattern for a given allocation[1]. 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.

The above formulation corresponds to the best-oracle strategy, since 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.

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

V. Experiments

In this section we present experiments comparing the predicted energy savings by our model to the optimal energy savings obtained by solving the ILP.

The criteria for selecting the case study application were: public availability of source files, realistic and ready-to-use

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

TABLE II. APPLICATION PARAMETERS (SIZE IN BYTES).

OS	Application	text	bss	data
TOS	MultihopOscilloscope	32058	122	3534
COS	rpl-collect (udp-sender)	47552	232	9250

application. We chose two data-collection application 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) use CTP (Collection Tree Protocol)[5] and rpl-collect (ContikiOS) use RPL (IPv6 Routing Protocol for Low power and Lossy Networks)[15]. The applications was compiled for a telosb node [11] based on a MSP430 microcontroller[4]. Table II summarizes the section sizes of the selected applications (TOS and COS stand for TinyOS and ContikiOS respectively). It can be observed that in both cases the code memory is between five and nine times larger than the data memory. This relationship, which is typical in current WSNs applications, motivates using a banked memory with power management for code memory rather than for data memory.

Since current sensor nodes do not support real-time execution trace generation, we simulated the network using COOJA[4]. The telosb node-level simulation relies on MSPsim, an instruction-level emulator for the MSP430 microcontroller, that 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 setting the debug mode on and off, 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 trace was trimmed to consider 5000 cycles.

The size of the application blocks, s_d, could be chosen to be regular (equally sized) or irregular, ranging from the minimum basic blocks to 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, ContikiOS and any non-preemptive event-driven software architecture.

The problem of allocating the code to equally sized banks was solved for up six banks, for both power management strategies. The total memory size was considered 10% larger than the application size, to ensure the feasibility of the solution. For each experiment the bank memory access patterns a_{bt} have been determined using the trace a_{dt} and the allocation map l_{bd} (how block are allocated to banks), given by the corresponding solution. For the best-oracle power management

[2]www.tinyos.net

[3]www.contiki-os.org

[4]www.ti.com/msp430

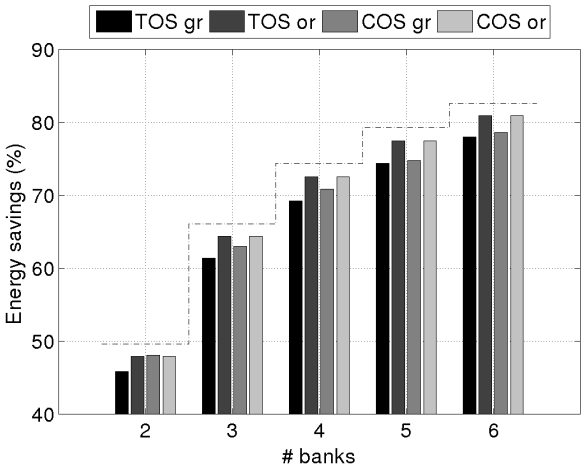

Fig. 2. Energy savings as a function of the number of banks for best-oracle and greedy policy (denoted *gr* and *or*) in TinyOS and ContikiOS applications (denoted *TOS* and *COS*) and the theoretical limit (dashed line).

the solution also outputs o_{bt}, the bank states for each cycle (i.e. ready or sleep). Finally, the average energy consumption is calculated using the memory energy parameters and the energy saving is determined comparing with a single bank memory with no power management.

VI. RESULTS AND DISCUSSION

Fig. 2 shows the energy savings as a function of the number of banks, for best-oracle and greedy policy in both applications (TinyOS and ContikiOS). As the number of banks increases, the energy savings approach to the corresponding value of having all banks in sleep state. In this figure we have intentionally discarded the partition overhead. 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.

Contrary to what one could expect, the extra benefit of oracle over greedy policy is scarce. Fig. 3 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 are equal in both cases and represent 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. While, 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

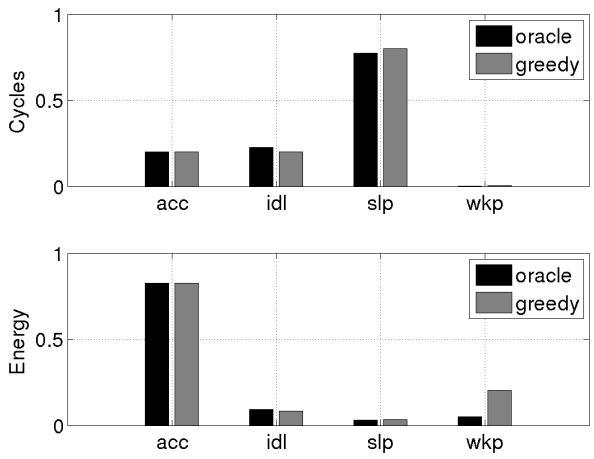

Fig. 3. Fraction of cycles and energy breakdown where each contribution is averaged among the different banks.

TABLE III. OPTIMUM NUMBER OF BANKS AS A FUNCTION OF PARTITION OVERHEAD.

$k_{ovhd}(\%)$	0	1	2	3	5
N_{opt}	∞	10	7	6	4
$\delta E^{max}_{N,eff}(\%)$	97.1	77.4	69.2	62.9	52.8

Fig. 3). The energy breakdown, Fig. 3 (lower part), shows that the difference between oracle and greedy comes mainly from the wake-up transitions. In this case study, due to its even-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 functions 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 explain 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 III. The energy savings is limited by the partition overhead, reaching a maximum of 77.4% for an overhead of 1%. The energy saving limit, as the partition overhead tends to zero and N to infinity, is 97.1% $(1 - E_{slp}/E_{act})$.

Table IV 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 background.

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 what arises in the previous limit case. This means that the saving lost due to wake-up transitions shifts the optimum number of banks. Finally, similar results are obtained for the best-oracle strategy, but with higher energy savings.

TABLE IV. ENERGY SAVING COMPARISON: MAXIMUM, GREEDY AND ORACLE.

maximum		number of banks				
		2	3	4	5	6
$k_{ovhd}(\%)$	1	47.55	63.06	70.32	74.27	76.58
	2	45.55	60.06	66.32	69.27	70.58
	3	43.55	57.06	62.32	64.27	64.58
	5	39.55	51.06	54.32	54.27	52.58
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

VII. CONCLUSIONS

We have found that aggressive energy savings can be obtained using a banked memory, up to 77.4% 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. Thanks to our modeling, at design time the optimum number of banks can be estimated, provided that the energy memory parameters are given, such as energy consumption characteristics and the partition overhead as a function of the number of banks.

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, due to the saving lost due to wake-up transitions. However 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, but contrary to what is expected, the extra benefit of the oracle over the greedy policy is scarce. The additional benefit of using an advanced algorithm to predict future access to banks must justify the increasing complexity and compensate the extra energy and area cost.

Future research includes extending our model to support arbitrary sized banks, and the evaluation of the effective savings when the access pattern is different from the one used for the off-line optimization. Preliminary results suggest a small degradation.

REFERENCES

[1] 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, Apr. 2002.

[2] 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, Sept. 2009.

[3] 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.

[4] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. 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, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[5] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. 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.

[6] O. Golubeva, M. Loghi, M. Poncino, and E. 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.

[7] J. Kwong and A. P. Chandrakasan. An Energy-Efficient Biomedical Signal Processing Platform. *Solid-State Circuits, IEEE Journal of*, 46(7):1742–1753, July 2011.

[8] M. Loghi, O. Golubeva, E. Macii, and M. Poncino. Architectural Leakage Power Minimization of Scratchpad Memories by Application-Driven Sub-Banking. *IEEE Transactions on Computers*, 2010.

[9] O. Ozturk and M. Kandemir. ILP-Based energy minimization techniques for banked memories. *ACM Trans. Des. Autom. Electron. Syst.*, 13(3):1–40, July 2008.

[10] 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.

[11] 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, Apr. 2005.

[12] J. Rabaey. *Low power design essentials*. Springer Verlag, 2009.

[13] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. 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.

[14] 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.

[15] 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), Mar. 2012.