Power-Aware Processors for Wireless Sensor Networks

Gürhan Küçük, Can Başaran

Department of Computer Engineering,
Yeditepe University, 34755 Istanbul, Turkey
{gkucuk, cbasaran}@cse.yeditepe.edu.tr

Abstract. Today, wireless sensor networks (WSNs) enable us to run a new range of applications from habitat monitoring, to military and medical applications. A typical WSN node is composed of several sensors, a radio communication interface, a microprocessor, and a limited power supply. In many WSN applications, such as forest fire monitoring or intruder detection, user intervention and battery replenishment is not possible. Since the battery lifetime is directly related to the amount of processing and communication involved in these nodes, optimal resource usage becomes a major issue. A typical WSN application may sense and process very close or constant data values for long durations, when the environmental conditions are stable. This is a common behavior that can be exploited to reduce the power consumption of WSN nodes. This study combines two orthogonal techniques to reduce the energy dissipation of the processor component of the sensor nodes. First, we briefly discuss silent-store filtering *MoteCache*. Second, we utilize Content-Aware Data MAnagement (CADMA) on top of MoteCache architecture to achieve further energy savings and performance improvements. The complexity increase introduced by CADMA is also compensated by further complexity reduction in MoteCache. Our optimal configuration reduces the total node energy, and hence increases the node lifetime, by 19.4% on the average across a wide variety of simulated sensor benchmarks. Our complexity-aware configuration with a minimum MoteCache size achieves not only energy savings up to 16.2% but also performance improvements up to 4.3%, on the average.

1 Introduction

Recent advances in process technologies and the shrinking sizes of radio communication devices and sensors allowed researchers to combine three operations (i.e. sensing, communication and computation) into tiny devices called wireless sensor nodes. Once these devices are scattered through the environment, they can easily construct data-oriented networks known as wireless sensor networks (WSNs). Today, there is a vast number of application scenarios involving WSNs in business, military, medical and science domain.

The lifetime, scalability, response time and effective sampling frequency are among the most critical parameters of WSNs, and they are closely related to one of the most difficult resource constraint to meet: the power consumption. The WSN nodes are designed to be battery-operated, since they may be utilized in any kind of

environment including thick forestry, volcanic mountains and oceanbeds. Consequently, everything must be designed to be power-aware in these networks.

The introduction of small-scale operating systems, such as TinyOS [1], ambientRT [2], and computation/communication-intensive applications significantly increases the energy consumption of the processor component of WSN nodes. In [3], the researchers show that the processor itself dissipates 35% of the total energy budget of the MICA2 platform while running *Surge*, a TinyOS monitoring application. In [4], the authors claim similar energy values when running a TinyDB query reporting light and accelerometer readings once every minute. In [5], the researchers find that the energy consumption of the processor/memory component for raw data compression is higher than the energy consumption of raw data transmission. Similarly, today most of the WSN applications avoid extensive computations and choose to transfer raw data to server machines to increase the lifetime of the sensor nodes. On the contrary, our proposed design encourages the WSN application developers to design less centralized applications by distributing the computation work and reducing the network traffic among the nodes.

After careful observations of the results from our departmental testbed and various simulations, we found two important characteristics of WSN data:

1) **Temporal and value locality:** Sensor network applications have periodic behavior. Especially, monitoring applications, such as *Surge*, may sense and work on constant data and constant memory locations for long durations. In [6], by caching commonly used data in a small number of latches (*MoteCache*), we showed that we considerably reduce the energy dissipation of the WSN processors. Then, we also showed that most of the store instructions in WSN applications are silent (these instructions write values that exactly match the values that are already stored at the memory address that is being written.), and we proposed to filter them by extending the MoteCache architecture.
2) **Common data values:** Further, we found that there are some common data values flowing through the datapath. In this study, we propose a technique called Content-Aware Data MAnagement (CADMA) that exploits this behavior and boosts our energy savings by reducing the read/write energy not only in SRAM but also in register file. When combined with MoteCache, we show that we can even increase the performance of the processor as much as 14%.

In this study, we mainly focus on MICA2 platform [7] which includes an AVR-RISC processor, ATmega128L, with Harvard architecture (i.e. two main memory spaces, 4Kx8 bytes of SRAM data memory and 64Kx16 bytes of flash program memory space [8]). Though, our techniques are platform-independent, and can be easily applied to other sensor platforms such as TelosB, moteiv and EYES sensor nodes. Moreover, these platforms utilize 16-bit microprocessors (compared to 8-bit microprocessor of MICA2 platform). Thus, we strongly believe that our techniques will achieve much better energy savings in these new platforms.

We begin by presenting a brief summary of MoteCache design in Section 2 followed by CADMA design in Section 3. The simulation methodology is given in Section 4, and our results are presented in Section 5. Finally, we discuss the related work in Section 6, and conclude our study in Section 7.

2 MoteCache

We now summarize the design of MoteCache (hereafter MC) which is a small SRAM buffer structure that exploits temporal locality of data in WSN applications [6]. In MC, each row consists of data buffers and an associative CAM (content-addressable memory) for holding the tags corresponding to the contents of these buffers. The idea is to cache data values of the N most recently accessed addresses in a N-byte MC [9, 10]. A read access in the MC structure proceeds as follows:

Cycle 1. MoteCache Tag Comparison: As soon as the address is computed, its tag part is compared associatively with the tag numbers associated with the contents of the MC. If there is a match (MC-hit), the scheduled readout of the memory is cancelled, potentially saving energy wasted in reading out the row from the memory.

Cycle 2. Data Steering: Incase of a MC-hit, data is read from the corresponding MC entry[1]. When the associative match using the CAM fails (MC-miss), the memory access continues in this cycle and the data is read out into a MC entry selected as a victim, based on the least recently used (LRU) replacement policy.

Writing to a MC entry has analogous steps, followed by a step that installs the update into the tag and data part of the corresponding MC entry. We study the writeback policy, since it is more suitable for our power-aware design.

2.1 The Studied MoteCache Configurations

1) Direct-Mapped MoteCache (DMMC): This configuration mimics the behavior of an ordinary direct-mapped cache. Since the SRAM of MICA2 platform contains a single byte at each line, the direct-mapped cache is organized to contain a single byte at each of its sets. The DMMC configuration has the smallest latency among the rest, since we can read data in parallel with the tag comparison as soon as the address is available.

2) Set-Associative MoteCache (SAMC): In order to reduce the possible conflict misses of DMMC configuration; we also decided to try the set associative cache configuration. This configuration is similar to a standard set-associative cache configuration, and requires the activation of more comparators in a single access.

3) Fully-Associative MoteCache (FAMC): This configuration activates all the tag/address comparators for each memory access. Therefore, it dissipates more power compared to other configurations, and its latency is not better than the DMMC configuration since it has additional MUX delay.

As locality of reference guarantees that there is a good chance of producing a MC-hit. Figure 1 shows that MC-hit rate is more than 89%, on the average, for the SAMC configuration with maximum size (8x8). The lowest hit rate is observed in the smallest DMMC configuration (4x1) as 18.4%. Please refer to Table 1 in Section 4 for the details of the simulated WSN benchmarks.

[1] To increase processor performance, data can be read from the MC at the end of the first cycle.

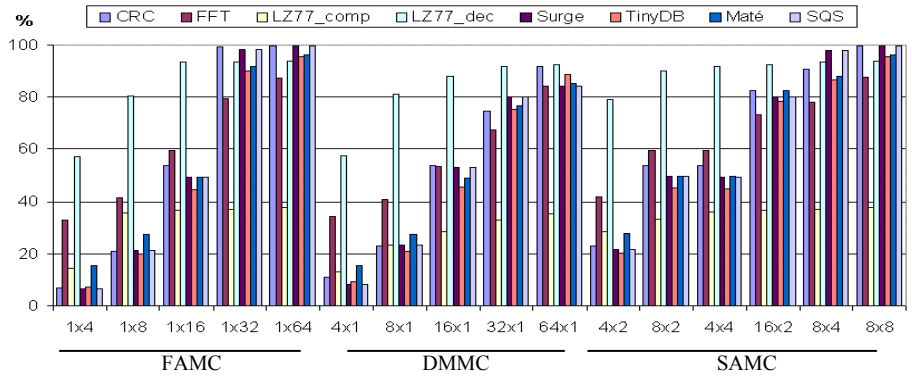


Fig. 1. Hit ratios to various MC configurations (s x a, s: set no, a: associativity)

2.2 Silent-Store Filtering MoteCache

When a store instruction writes a value that actually matches the value already stored at the memory address being written, the operation is called a *silent store* [11]. They have no effect on machine state, and, in [6], we showed that detection and removal of these instructions may considerably improve the lifetime of WSNs. Figure 2 shows the percentages of silent store instructions in our simulated benchmarks. Across all WSN benchmarks, the average percentage of silent stores is 81.6%.

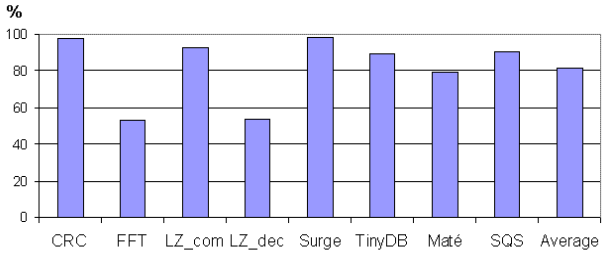


Fig. 2. Percentages of silent stores in WSN benchmarks

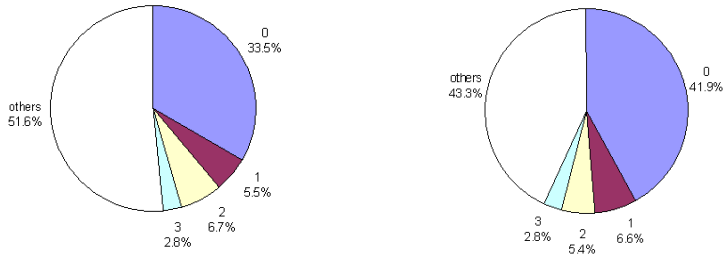
We increase the functionality of the *dirty-bit* in our writeback MC for detecting and removing silent stores. Therefore, the *dirty-bit* is renamed to *dirty&noisy-bit* (or DN-bit, in short.) A data value is written back to SRAM only when it is *dirty* (i.e. it is modified) and also *noisy* (i.e. if the new data value is different than the one stored before.) A write access in the silent-store filtering MC structure proceeds as follows:

1. MC-hit and Silent Store Detection: After a MC-hit, the data value to be written is compared with the data value of the corresponding MC line. If there is no match, the DN-bit of that MC line is set to indicate that the store instruction is *noisy*.

2. MC-miss and Writeback: After a MC-miss, we select a victim line. If the DN-bit of the victim line is set, the standard MC writeback procedure is followed. Otherwise, we just replace the MC line and cancel the rest of the writeback process, since the MC and SRAM data are already in sync.

3 Content-Aware Data Management (CADMA)

In this section, we describe the details of our proposed technique in this study: Content-Aware Data MAnagement (CADMA.) During our simulations, we observed that there are some common data values (specifically, 0, 1, 2 and 3) flowing in the datapath. Figure 3 shows that, across all simulated benchmarks, 48.4% of the SRAM data and 56.7% of the register file data are composed of these common data values.



SRAM Register File

Fig. 3. Average occurrence percentages of data values in SRAM and register file

The idea behind CADMA is to exploit this phenomenon to reduce energy dissipation of SRAM, MC and register file. For this purpose, we introduce an additional bit, CD (i.e. Common Data), to each line of these structures. The CD operations are explained as follows:

1. Data Write + CD Set: The CD bit is set when there is a register/memory write operation trying to store a common data value[2]. In that case, CADMA only writes 3 bits (one CD bit + the two least significant bits of the common data value) for that byte. Note that, for all of the common data values (0, 1, 2 and 3), there is no need for any encoding/decoding process, since they already fit into the least significant 2 bits.

When the data value is not one of the four common data values, on the other hand, CADMA writes 9 bits (one CD bit + the eight bits of data value) dissipating slightly more power compared to the baseline case.

2. Data Read + CD Probe: A read operation with CADMA starts by probing the CD bit. When it is set, CADMA only activates the read operation for the two least significant bitlines. By canceling the read from the other six bitlines, CADMA achieves significant power savings in all three structures. In this case, the rest of the byte is constructed by setting the six most significant bits to logic-0.

If CD bit is not set, on the other hand, the standard procedure for data read operation is followed. Then again, slightly more power is dissipated (reading 9 bits: one CD bit + the eight bits of data value) compared to the baseline case.

In this study, we integrated CADMA within MC, SRAM and register file structures. In our experiments, across all simulated benchmarks, we observed that CADMA boosts our processor energy savings as much as 10% and sensor node's total energy savings by 5% compared to the bare MC discussed in [6].

---

[2] A simple hardwired circuit, which checks if all the six most significant bits of data are zero, may easily detect a common data value.

4 Simulation Methodology

The most important requirement of this study was using a simulator which not only accurately models ATmega128L processor of MICA2 platform in a cycle-accurate way but also simulates sensor networks in various topologies. We found *Avrora*, the AVR Simulation and Analysis Framework, [12] quite suitable to our needs. For all simulated benchmarks, a three-node network topology is used. Table 1 gives the details of the benchmarks used in this study:

Table 1. WSN benchmarks details

Benchmark	Size in ROM (bytes)	Size in RAM (bytes)	Execution Time (mins)
CRC	1072	1492	5
FFT	3120	1332	5
LZ77 Compression	2700	486	5
LZ77 Decompression	1228	356	5
Surge	17120	1928	50
TinyDB	65050	3036	50
Maté	38974	3196	50
SeMA/SQS	22346	2175	50

- CRC implements a CRC32 algorithm continuously running on 448 bytes of data.
- FFT benchmark is a discrete Fourier Transform on 256 bytes of data. This transformation is executed within an infinite loop [13].
- LZ77 compression is again enclosed in an infinite loop and works on 448 bytes of data taken from the header file of an *excel* file.
- LZ77 decompression is the decompression of the compressed data obtained from the LZ77 compression. This data is 330 bytes long[3].
- Surge application is a TinyOS core application to demonstrate multihop routing.
- TinyDB application is used to execute the query "select light from sensors".
- Maté (BombillaMica) is used with a script that reads light sensor readings in every 10ms and sends them through RF interface after 10 readings [14].
- SeMA/SQS is used to execute the query "select temperature from sensors each second" [15]. The query results are sent to the base station after each reading.

4.1 Calculation of MC Energy Dissipation

We modified *Avrora* to record transitions at the level of bits for the processor/memory components. We combined the simulator results with energy per transition measurements obtained from the SPICE simulations of the actual component layouts for 0.35μ CMOS technology. We modified the energy model of Avrora to accurately model very fine-grain, instruction-level energy dissipations of the processor. For accurate energy savings estimations, we also considered detailed energy dissipations of the additional CADMA logic.

[3] CRC, FFT and LZ77 applications trigger radio communication every 2 seconds to imitate typical WSN applications.

5 Results and Discussions

Figure 4 shows our energy savings for selected silent-store filtering MC configurations. Notice that, even the smallest MC configuration (4x1) saves more than 57% of the memory access energy, on the average. The optimum configuration is found to be the 8x4 configuration, since it gives similar savings compared to the MC configurations twice of its size (80.6% of 8x4 vs. 81.4% of 8x8 – not shown in the Figure.)

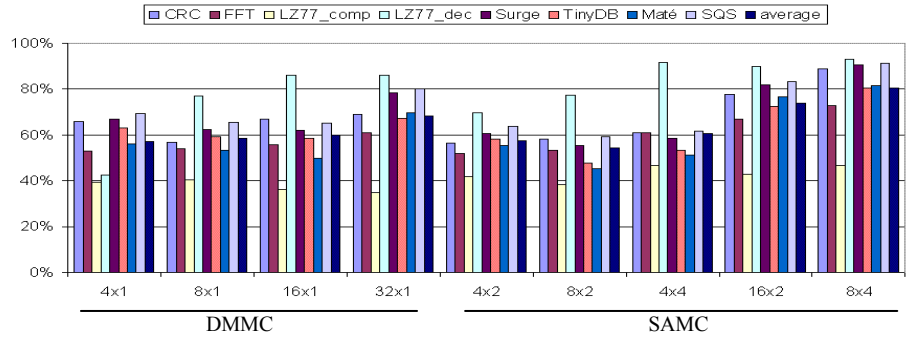


Fig. 4. Average savings on memory access energy dissipation for various MC configurations with the silent store filter (s x a, s: set no, a: associativity)

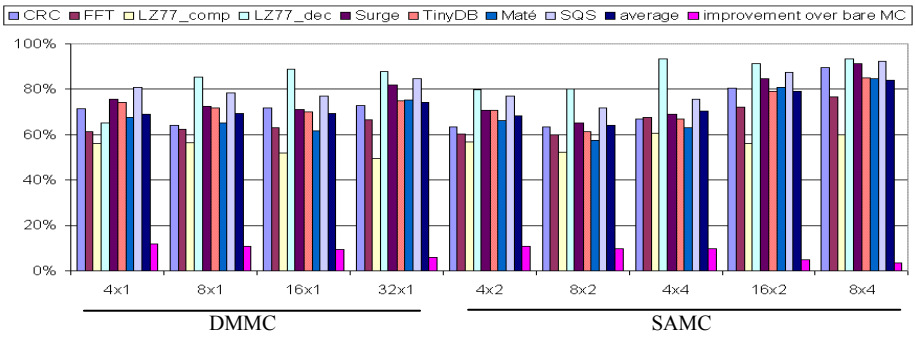


Fig. 5. Average savings on memory access energy dissipation for MC/CADMA

When we apply CADMA on top of these MC results, the energy savings gap between the minimum (DMMC 4x1) and the maximum (SAMC 8x4) configurations almost disappears (70% of 4x1 vs. 84% of 8x4). In Figure 5, we also show the percentage of improvement in the energy savings compared to the bare MC case (the rightmost bar.) Again, notice that the integration of CADMA architecture with the previously proposed MC makes the minimum MC configuration quite feasible.

We also computed the effect of CADMA on total processor energy savings. The rightmost bar in Figure 6 shows that 41% of total instructions are memory read/write instructions, on the average. When we apply these figures to our optimal configuration (SAMC 8x4), we find that the total processor energy savings are 51.3%, on the average (Figure 6). Note that CADMA extracts additional energy savings from register file (with bare MC, total processor energy savings stay at 46.5%, on the average).

When we focus on the minimum MC configuration, energy savings due to CADMA become much more clear. Total processor savings of MC/CADMA combination are 44%, on the average. Notice that, these savings are very close to the savings of bare optimum MC configuration. The energy savings of bare minimum MC configuration are as low as 34%, on the average. These results show that CADMA is a very effective technique that boosts savings of low complexity MC configurations.

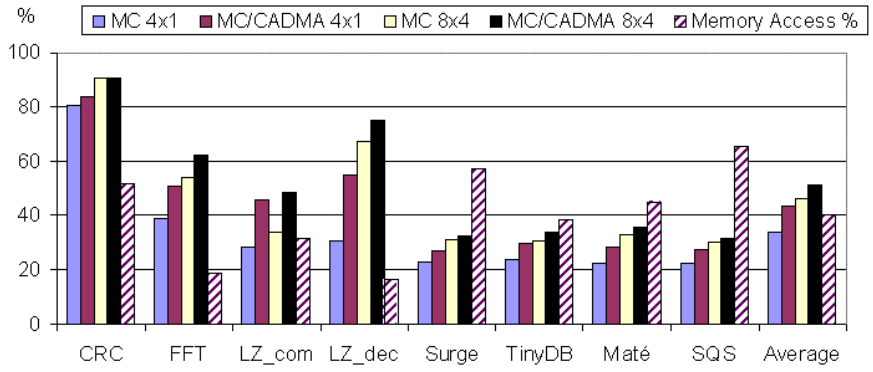


Fig. 6. Processor energy savings and percentages of memory access instructions

Next, we identify the percentage of CPU energy dissipation over total energy dissipation of a sensor node to compute the possible lifetime increase with MC/CADMA architecture. Figure 7 shows that the total energy savings are 19.3% for optimum MC/CADMA and 17% for bare optimum MC configuration. Notice that, the gap between the bare minimum MC and minimum MC/CADMA configurations in Figure 6 is still preserved in Figure 7. Again, the energy savings with minimum MC/CADMA configuration are 16.2% (very close to 17% of the bare optimum MC configuration.) The energy savings for the bare minimum MC configuration are 11.7%, on the average. These energy savings are directly related to lifetime improvement of the node running that specific application.

Finally, we also studied the possible performance improvements when we assume that we can access to a MC structure in the first cycle of a memory operation. In Figure 8, we give these results for three DMMC configurations: 4x1, 8x1 and 16x1. The minimum configuration improves the performance by 4.3%, on the average, whereas 8x1 and 16x1 improve it by 8% and 14%, respectively. These figures also indicate that when the latency of our MC/CADMA architecture and address computation cannot be squeezed into a single cycle period, we can still increase the cycle time and tolerate slight performance penalties.

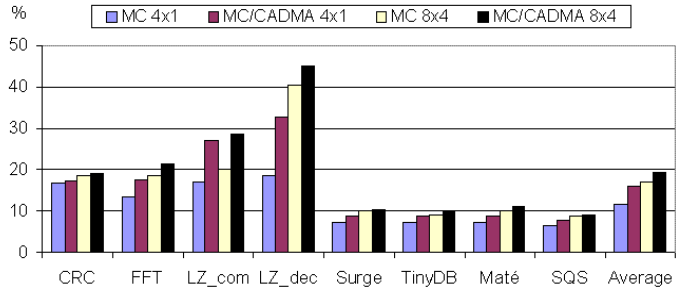


Fig. 7. Node-level energy savings and lifetime improvements

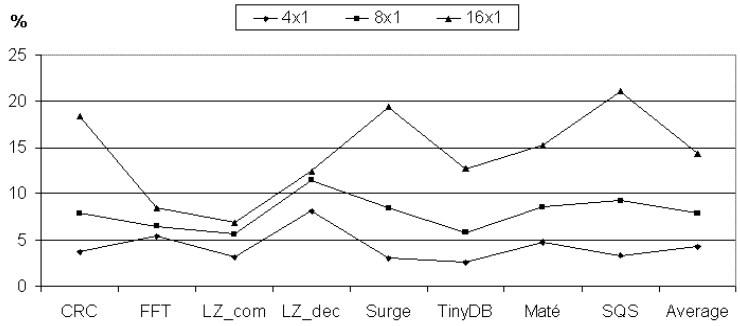


Fig. 8. Possible performance improvements with various DMMC configurations

7 Related Work

In the literature, there are various studies that target energy-efficient microprocessors for wireless sensor nodes [16, 17, 18]. In [16], the authors present SNAP/LE, an event-driven microprocessor for sensor networks. They avoid the TinyOS overhead by their asynchronous, event-driven approach. In [17], the authors seek to fully leverage the event-driven nature of applications. They study the application domain of sensor networks and they seek to replace the basic functionality of a general-purpose microcontroller with a modularized, event-driven system. In [6], we utilize a cache structure (MoteCache) to reduce the energy dissipation of the memory component of the MICA2 platform. In this study, we introduce a platform-independent technique and integrate it with the MoteCache for further reducing the energy dissipation in various processor structures of WSN nodes.

8 Concluding Remarks

In this study, we proposed the platform-independent MC/CADMA architecture for reducing the energy dissipation of the processor/memory component of wireless sensor nodes. We studied various MC configurations and found that 32-byte, 4-way, set-

associative, silent-store-filtering configuration shows the best energy/lifetime characteristics. Combined with CADMA, this optimal configuration reduces the node energy by 19.4%, on the average, across a variety of simulated sensor benchmarks. We also found that CADMA integration noticeably improves the results of the minimum MC configuration (4-byte, direct-mapped with silent-store filter.) We showed that this configuration achieves not only energy savings up to 16.2% but also performance improvements up to 4.3%, on the average. We strongly believe that better results can be observed, once these platform-independent techniques are implemented in newer platforms, such as TelosB, moteiv and EYES, that utilize 16-bit processors.

References

1. Hill J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K., "System Architecture Directions for Networked Sensors", in Proc. of the 9th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (2000), ACM Press, pp. 93-104
2. AmbientRT: Real-Time Operating System for embedded devices, http://www.ambient-systems.net/ambient/technology-rtos.htm.
3. Conner, W.S., Chhabra, J., Yarvis, M., Krishnamurthy, L., "Experimental Evaluation of Synchronization and Topology Control for In-Building Sensor Network Applications", in Proc. of WSNA'03, Sep. 2003
4. Madden, S., Franklin, M.J., Hellerstein, J.M., and Hong, W., "TinyDB: An Acquisitional Query Processing System for Sensor Networks", in ACM TODS, 2005
5. Polastre, J.R., "Design and Implementation of Wireless Sensor Networks for Habitat Monitoring", Research Project, University of California, Berkeley, 2003
6. Kucuk, G., Basaran, C., "Reducing Energy Dissipation of Wireless Sensor Processors Using Silent-Store Filtering MoteCache", in Proc. of PATMOS'06, Montpellier, France, 2006
7. Crossbow Technology, Inc., http://www.xbow.com
8. Lynch, C., O'Reilly, F., "Processor Choice for Wireless Sensor Networks", in Proc. of Workshop on Real-World Wireless Sensor Networks (REALWSN'05), 2005
9. Ghose, K. and Kamble, M., "Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation", in ISLPED'99, 1999, pp.70-75
10. Kucuk, G. et al, "Energy-Efficient Register Renaming", in Proc. of PATMOS'03, Torino, Italy, September 2003. Published as LNCS 2799, pp.219-228
11. Lepak, K.M., Bell, G.B., and Lipasti, M.H., "Silent Stores and Store Value Locality", in IEEE Transactions on Computers, (50) 11, Nov. 2001
12. Titzer, B. et al, "Avrora: Scalable Sensor Network Simulation with Precise Timing", In 4th International Conference on Information Processing in Sensor Networks, 2005
13. Numerical recipes in C, Cornell University, Online book, http://www.library.cornell.edu/nr/cbookcpdf.html
14. Levis, P., Culler, D. "Maté: A Tiny Virtual Machine for Sensor Networks", in Proc. of the ASPLOS X, 2002
15. Baydere, S., Ergin, M.A., "An Architectural Approach to Service Access in Wireless Ad Hoc Networks", in Proc. of Wireless and Optical Communications Conference, 2002
16. Ekanayake, V. et al, "An Ultra Low-Power Processor for Sensor Networks", in Proc. ASPLOS, Oct. 2004
17. Hempstead, M. et al, "An Ultra Low Power System Architecture for Sensor Network Applications", in the Proc. of 32nd ISCA'05, Wisconsin, USA, 2005
18. Warneke, B.A. and Pister, K.S., "An Ultra-Low Energy Microcontroller for Smart Dust Wireless Sensor Networks", in Proc. ISSCC, Jan. 2005