A Review of Common Belief on Power Management and Power Consumption

Daniel Schmidt, Norbert Wehn
Technische Universität Kaiserslautern,
Microelectronic Systems Design Research Group
schmidt@eit.uni-kl.de

White Paper, April 2009

Abstract

Energy efficiency is a key topic in embedded systems design. Many schemes for power management were proposed, but often they were not practically evaluated or focus on just one part of a system rather than the whole. Performing optimizations on theoretical models, however, may give misleading results. Nonetheless, some of the common beliefs about power consumption and power management are taken as facts and form the basis of many studies. In this paper we review the validity of these common beliefs and compare them to practical measurements and results form laboratory experiments. We summarize results from our previous publications to show, how overly simplified models lead to power management strategies that rather increase energy than to reduce them. We give examples of such cases for DRAM power management, dynamic voltage and frequency scaling, and communication schemes in wireless sensor networks.

1 Introduction

In the last years embedded systems have found their way into everyday-life, often in the form of mobile devices with various tasks. Due to market demands they are limited in size and energy resources. At the same time a high life time expectancy makes power and energy aware design a necessity for this class of devices. The problem gets even harder as typical tasks, as e.g. audio and video playback, communication, or a vivid user experience, pose hard or soft real time constraints. Hence, a lot of research has focused on optimizing the energy consumption in embedded systems. From a system level perspective power management is the method of choice. That means, the performance of the embedded system is adapted at runtime to the given workload, saving energy by throttling down components or even switching them in to low power modes while they are not being used.

This paper reviews some of the common beliefs on power management and power consumption and assesses their validity in a comparison with precise power measurements on an existing system. We will show that the aggressive power management schemes often proposed will not yield the expected energy savings, as the theoretic assumptions and models they are based on neglect important aspects. In some cases this even leads to the absurd situation where power management actually increases the power and energy consumption of a system.

These insights are not new and the reader can find analysis of individual aspects in different previous publications, as for example [21, 6, 15] But to the best of our knowledge for the first time they are all gathered in one paper and also introduce some newer results from our own recent publications [19, 9], as well as not yet published measurements of SDRAM power consumption [20].

The rest of the paper is organized as follows: In Section 2 we present some basic common assumptions and conclusions that are to be found as a basis for power modeling and power optimization in numerous publications. Then, we describe the experimental platform we used to assess the effects of different power management schemes by measurement in Section 3. The results of our measurement along with a comparison with common belief and an analysis is given in Section 4. Section 5 focuses on power consumption in wireless sensor networks and the question whether routing over short hops or long hops is beneficial and which consequences that has for energy optimal transmission schemes.

2 Common assumptions on power consumption

Embedded systems are as different as their respective fields of application. But at least a CPU and main memory are common to virtually all of these devices. Early research has dealt mainly with the optimization of the CPU's energy consumption, while for quite some time now also the memory has moved into the focus of research. The following Sections 2.1 and 2.2 give an overview about commonly used models and some very basic, but wide spread assumptions in these fields.

2.1 CPU

Many schemes for the dynamic adaptation of operating frequency and voltage have been proposed to reduce the energy consumption for processing a given task or set of tasks, e.g. [3]. These schemes are commonly reffered to as dynamic voltage and frequency scaling, or short DVFS or DVS. The basic idea is based on the well-known formula which relates the switching power P of a CMOS circuit to the operating voltage V and the clock frequency, according to Equation 1.

$$P \propto fV^2. \tag{1}$$

Under the additional assumption that the execution time is inversely proportional to the frequency, the total energy E for the completion of a certain task is then given by Equation 2. This assumption is identical to the assumption that the number of clock cycles C to perform a certain task is independent from the operating voltage and frequency.

$$E \propto V^2. \tag{2}$$

Here, the circuit's frequency has been ruled out. It has to be mentioned that at a reduced voltage the maximum achievable frequency is generally lower as well. Some publications assume for simplicity that the voltage and frequency of the circuit can be set to arbitrary values at no cost or at least introducing only a negligible overhead. In other works a fixed set of combinations of voltage-frequency pairs is assumed, usually with a proportional dependency $f \propto V$.

Completing a task or set of tasks at the lowest possible CPU frequency and voltage is an immediate and obvious conclusion from these assumptions, when the energy has to be minimized. So the energy question is answered early and the problem boils down to accurately predicting the runtime and throttling the CPU speed in order not to miss hard real time deadlines.

When the CPU offers only a fixed set of frequencies, however, the problem gets already slightly more complicated as the energy-optimal frequency (according to the above model) may not be available. Let $\{f_1, ..., f_n\}$, s.t. $\forall i: f_i < f_{i+1}$ be the set of available frequencies. In this scenario there are in general two possible strategies. The first consists in completing the task at the lowest frequency f_i that meets the deadline and shutting down the CPU for the rest of the period, which is called slack time. The other possibility is to run one fraction of the task at f_i and the remainder of the task at the next lower frequency f_{i-1}. If the portions of the task to be executed at either frequency are chosen properly the slack time is being used completely and the task is finished exactly at the given deadline.

Which of the two strategies is optimal in this theoretic model depends on the assumed power consumption of the CPU in its off-mode and the overhead cost for determining the switching point between the two frequencies. In any way the theoretic model only optimizes CPU energy and does not take into account other system components, as for example the main memory, DC/DC converters, or PLL switching overheads.

2.2 Main Memory

Two very different types of memory can be distinguished: SRAM and DRAM. SRAM has many desirable properties, as for example a very simple interface and a constant access latency. DRAMs in contrast need a complex controller to take care of refresh operations as otherwise the stored data are corrupted due to leakage. Moreover, the access latencies of DRAMs are not constant at all and sometimes even hard to predict. For their smaller die area - an SRAM cell has 6 times as many transistors as a DRAM cell - and much lower cost DRAMs are nonetheless dominating in the embedded systems market. For these kinds of systems the memories' power consumption is not only non-negligible but actually in the same order as the CPU power consumption (e.g., [21]). Hence, models for their power consumption are needed, too. Aside from this, also runtime effects have to be considered, especially as modern DRAM technologies offer the possibility for advanced power management. While usually their operating frequency is fixed, i.e., it does not scale with the CPU frequency and DVFS is not applicable to them, they offer lower power states.

A large body of work on exploiting low power memory states is available. They all share a very small, common base of memory power models: either the data-sheet information published by DRAM manufacturers Rambus for RDRAM [17] or Micron [12] for their SDRAMs (including DDR varieties). Both of these sources can be interpreted as state-based models. That means that, at all times, the DRAM modules are in a certain state (e.g. idle, reading, writing, napping, etc.), consuming a certain, approximately constant amount of energy per time unit. Energy consumed during the transitions from one state to another is not reflected in the models. Both, the Micron as well as the Rambus model, can only be used, when a statistic of the device usage (e.g. number of clock cycles, number of reads, writes, precharge commandos, etc.) is known.

In [4] these statistics are generated using trace-driven simulation for a set of benchmark processes under different power management policies. Based on the simulation traces and the RDRAM power model they calculated the energy delay product, focusing entirely on the memory and completely neglecting the CPU power consumption. For the RDRAM the transition times between the lower power states and the active state, where memory words

Core Voltage (V)	CPU freq. (MHz)
1.5	333 - 733
1.32	333 - 600
1.215	333 - 400

Table 1: Possible voltage and frequency combinations

can actually be accessed, are given in the model. The authors assume the power consumption during this transition periods to be the average value of the power consumption of the lower power state and the active state. The ultimate conclusion of their study then is, that a very aggressive exploitation of low power memory states optimizes the energy delay product. They claim that DRAM energy will not benefit from more sophisticated power management policies and that they should be switched to a lower power mode immediately, whenever they become idle.

Compared to SDRAM, RDRAM offers a larger range of low-power states (with more possible trade-offs of power-saving for resynchronisation time), making it more attractive for sophisticated optimization strategies. Also, in contrast to SDRAM, RDRAM power states can be assigned on a per-bank-basis. In SDRAM the same power mode has to be used by the whole memory system. Thus, SDRAM power management has hardly raised any attention in the research community, even though SDRAM and its derivatives are far more wide spread especially in the low-cost and hand-held segment than the more expensive RDRAMs. However, there are simulators (e.g., [23] and [8]) capable of generating usage statistics and calculating the power consumption based on the Micron power models.

Few newer studies have been published, like e.g. [16], and they base on the same results and the same power models. The validity of the results depends on the validity of the models, however. Final evidence can only be achieved by measurements on actual hardware. The platform for our measurements is shortly presented in the following section. It allows us to assess the theoretic assumptions and pinpoint deviations from reality.

3 Experimental platform

A typical processor in the embedded domain has been Intel's XScale and the derived micro-controllers for many years. For the measurement results presented in this paper we used an ADI 80200 EVB evaluation board [1] equipped with the Intel 80200 XScale processor [11], 32 MByte of SDRAM from Micron (split into four 64Mbit modules [14]), and an FPGA which hosts the memory controller [10]. The CPU supports dynamic frequency scaling in steps of 66MHz and can operate from 333MHz to 733MHz. The board was extended with a small PCB connected to the peripheral bus which allows for dynamic, processor controlled voltage switching [18]. The possible combinations of CPU core voltage and CPU frequency are given in Table 1 and show that the assumption of a linear dependency of frequency and voltage is misleading. The measured switching overhead is 8μs (corresponding to 4800 operations at 600MHz) for PLL synchronization during which the operation of the CPU is stalled. This overhead is not negligible if switching occurs very often.

In the following section we present measurement results for CPU power consumption as well as system energy consumption for various benchmarks. We use two benchmarks from the *CSiBe* [2] benchmark set (*vam*, which mostly runs from the CPUs cache, and *minigzip* with a high memory workload) to assess the validity of the simple assumptions on power assumption as described in Section 2.1.

4 Power measurements and analysis

4.1 The optimal CPU frequency

Already in the previous section we could see, that some of the theoretic assumptions do not hold in practice. For our testbed, as for basically every other platform on the market, there is no direct linear relationship between CPU voltage and maximum CPU frequency (see Table 1). And of course, not every possible combination of voltage and frequency is applicable in practice. Still, based on Equation 2, the CPU energy should be optimal at the lowest possible voltage, independent of the frequency. This implies, however, that static power consumption is negligible. The measurement of the CPU energy for the computation dominant benchmark *vam*, presented in Figure 1 show that this is not true. Here, for 333 MHz, due to the increased runtime of the benchmark, the static energy consumption is higher as for 400 MHz. This is why, even though in both cases the voltage is the same, the higher frequency is more energy efficient. Also, neglecting the static part of the power consumption, one would expect a much higher increase in energy consumption for higher frequencies. This is indicated by the uper of the two lines in the graph. These values were obtained using Equation 2, the actual CPU voltages for the different frequencies, and the measured energy consumption at 333MHz as a reference base value.

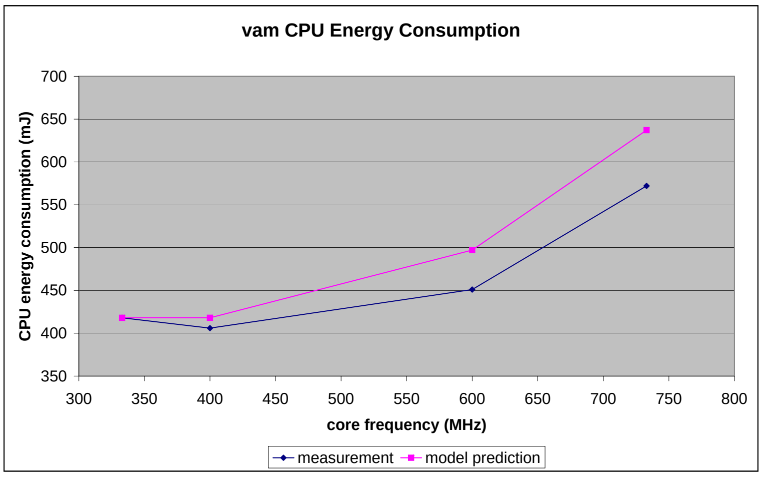

Figure 1: CPU power consumption at various frequencies for *vam* benchmark

Another flaw in the theoretic assumptions is, that program runtimes in reality do not decreases linearly with a rising CPU frequency. While this assumption holds for the simple *vam* benchmark, which fits almost completely into the caches, the runtime of other benchmarks is determined by the slower SDRAM, as can be seen in the *minigzip* benchmark. Figure 2

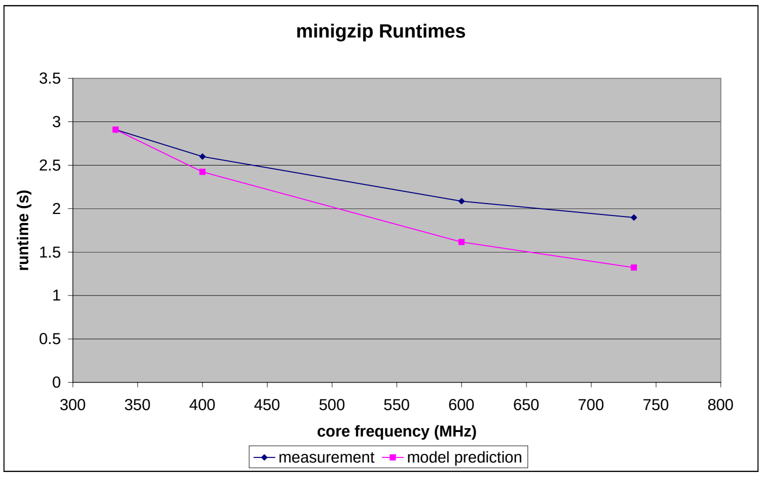

Figure 2: Runtimes of *minigzip* at various frequencies

compares the measured runtimes for various frequencies with the projected runtimes assuming a reciprocal dependency on the CPU frequency. Again, the measured value at 333MHz was taken as a reference value for this theoretical model which is implicitly included in Equation 2. At 733 MHz, the actual runtime exceeds the predicted runtime by more than 40%. The problem is that the memory is still invariably clocked at 100 MHz. The number of CPU stalls increases significantly rendering the higher frequencies less effective than predicted. On the other hand, the CPU consumes significantly less power while stalling, due to a lower internal toggling rate and internal power optimizations. Compared to the computationally intensive *vam* benchmark, the average CPU power consumption during the minigzip is almost 100 mW lower at the highest frequency.

These measurements show clearly, that Equations 1 and 2 do not hold even in this very simple example with just one task and no external dependencies. As it is crucial to predict the runtimes of a benchmark accurately at different frequencies, one has to take other system components into account. But they also consume energy of their own. Consequently, the goal is to optimize system energy rather than CPU energy.

4.2 Optimizing system energy

Even the most simple embedded systems consist of at least a CPU and some main memory. The main memory severely affects run times, as it does not profit from frequency scaling. We also observed a high variability of cache miss costs between 78 and 126 CPU clock cycles (at 600MHz) [8]. Because of this, a precise runtime prediction can only be done by simulation or measurements. But also the main memory's energy consumption plays a significant role. This complicates finding the optimal strategy for system energy minimization. While intuitively one expects the lowest voltage and frequency setting to optimize system energy, our measurements show a drastically different result. In Figure 3, we show the normalized energy consumption of the *complete* system as a function of the CPU frequency. In complete contradiction to the simplifying assumptions, the highest frequency setting consumes the lowest energy. This is only a surprising result at first sight. At higher frequencies the completion of the task takes less time, Thus, the energy consumed by those components of the system that do not benefit from the voltage scaling of the CPU, is reduced. These savings outweigh the penalty introduced by the higher CPU energy consumption for various frequencies larger than 333MHz.

As embedded systems usually can not be completely turned off after having completed a task, it is a more fair comparison to switch the system to idle mode after computation has finished. Assuming a periodic task with a period just equal to the execution time at 333MHz, the energy consumed in one period equals the energy consumed during the computation plus the energy consumed in idle mode, waiting for the end of the period Figure 4 shows the energy consumption during one period equal to the execution time of the task at 333 MHz. In this case, the optimal voltage and frequency setting is neither the lowest, nor the highest.

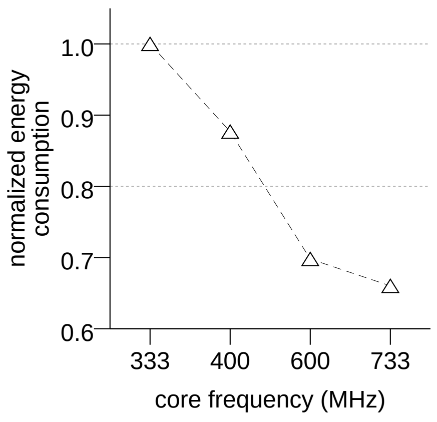

Figure 3: Measured system energy consumption of the *vam* benchmark at different frequencies

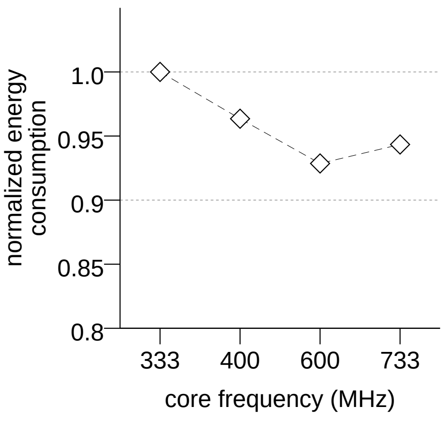

Figure 4: Measured system energy consumption, padded to equal execution time for all frequencies

It is evident that DVFS schemes based on the unrealistic assumptions of Equations 1 and 2 may lead to a higher energy consumption rather than to a lower energy consumption. In addition it is obvious that optimizing the CPU energy is different from optimizing the system energy. The optimal DVFS strategy will depend on many factors, including the type of benchmark (i.e., limited by the speed of the memory or limited by the CPU), on the number and type of system components (especially the size and type of memory, a display or other human interface devices where applicable, but also the efficiency of voltage regulators), and on the energy consumption of all components in the idle state.

4.3 Memory power management

As stated earlier already, not only the CPU, but also modern DRAMs offer lower power modes for power management. While most studies on DRAM power management focus on RDRAM modules, SDRAM is far more wide spread in embedded systems. Simulations with Micron's SDRAM power model, however, support the findings by Fan et al. [4], which we briefly summarized in Section 2.2. I.e., theoretically aggressive power management strategies should work for SDRAMs, as well.

For our simulations we use the XScale energy simulator XEEMU [7]. This simulator includes a cycle accurate behavioral model of the SDRAM subsystem and can generate memory traces to calculate the memory energy consumption with Micron's power model. Figure 6 shows the simulation results for the *minigzip* benchmark with no memory power management and an aggressive power management scheme which switches the SDRAM to the power saving self refresh state (SREF) after 10 memory clock cycles of idleness. In the self refresh mode the SDRAM is inactive, but retains all its data. According to the Micron model, the power consumption of a single memory chip in this mode is only 3.3mW compared to 148.5mW in idle mode. The predicted reduction in average power consumption is approximately 172 mW.

For a validation of the simulated data by measurement, we modified the design of our evaluation board and of the memory controller to allow the use of the SREF state of the SDRAM. The modified memory controller can be configured by software running on the processor with a threshold value. It counts the idle cycles in which no memory access takes place. If the number of idle cycles exceeds the set threshold value, the SDRAM is transitioned to the SREF. This mode is left again when the next memory access occurs. Leaving the SREF mode incurs an overhead of 8 memory clock cycles before the memory is accessible.

Figure 5 shows the measured total power consumption of the board during the minigzip benchmark run at 600MHz for two test cases: without SDRAM power management and with a threshold of 10 clock cycles. As the simulation predicted correctly, the runtime of the benchmark increases from just above 2 seconds by roughly 20% to just above 2.4 seconds.

XEEMU does not model the energy consumption of the 7-segment LED, voltage regulators, and peripherals on the board, such as the UART or the flash ROM. But according to our measurements their power consumption can be seen as a constant offset to the sum of CPU and SDRAM power consumption. Thus, the observed power savings for simulation and measurement should ideally be the same.

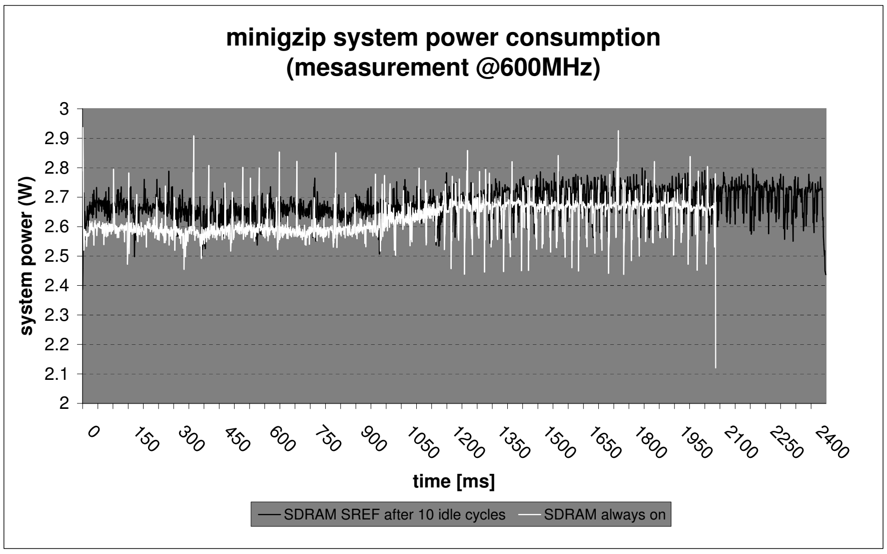

Figure 5: Measured total power consumption of the evaluation board for the *minigzip* benchmark with and without DRAM power management

However, the measurements show, that despite the memory being switched to a lower power mode, the average consumption of the board *rises* by ∼77mW, whereas the simulations

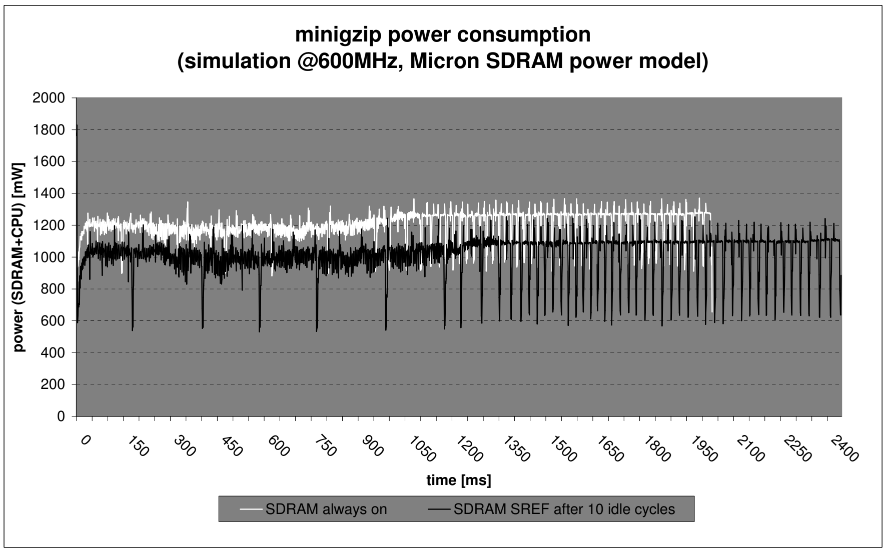

Figure 6: Simulated combined power consumption of CPU and SDRAM based on XEEMU simulator and Micron's power model for the *minigzip* benchmark with and without DRAM power management

predicted a *reduction* by ∼172mW. This means, that in fact the DRAM power management for this benchmark with a high number of memory accesses increases not only the runtime, but also the average power consumption, and in turn of course also the total energy consumption.

Figure 7 explains what happens. The dark line shows the current drawn by a single of the four memory chips on the evaluation board during a synthetic benchmark. In this benchmark the memory controller invariably toggles the SDRAM between active and SREF modes. The brighter line is the voltage at the clock enable (CKE) pin of that memory chip, which determines the operational mode (SREF or active). It can clearly be seen, that the power consumption during SREF indeed is much lower than in the normal active mode. However, this advantage is somewhat foiled by a peak that occurs whenever the SREF mode is entered. This peak stems from a refresh, which the DRAM module issues as soon as the SREF mode is entered. *This is not only true for our particular chip, but it is a general problem for all SDRAM types, including the modern DDR, DDR2, and DDR3 variants compliant with the respective JEDEC standards [13].* Thus, trying to aggressively exploit even short gaps of idleness in SDRAMs using SREF modes, is counterproductive, for all these types of memory.

Detailed measurements also show that the energy consumed in the active mode is much lower than the worst case values given in the Micron data sheet. The power consumption in the SREF mode (after the refresh finished), however, is estimated almost correctly. Thus, the absolute power savings, even when we neglect the refresh at the beginning of each self refresh cycle, are much lower than predicted. Estimating power savings using a worst case model rather than a typical model will move the break-even point, where the overhead for increased runtime outweighs the savings.

Throughout the tests the Micron power model did not only give wrong absolute values but even worse it indicated wrong trends when comparing different SDRAM power management strategies. This shows once more that optimizations based on theoretical models may not optimize energy consumption of real devices. Thus, XEEMU was extended with a precise,

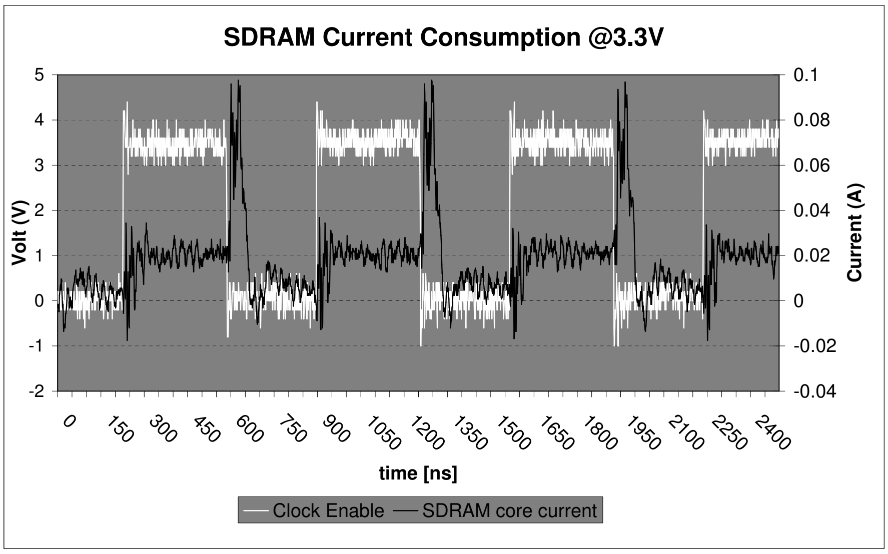

Figure 7: Clock enable signal (CKE) and power consumption of a single SDRAM chip when entering/exiting SREF mode

measurement based power model for SDRAMs [20]. This makes it a unique tool for energy optimizations on system level.

5 Wireless Sensor Networks

Wireless sensor networks (WSN) are the key enabling technology for many applications ranging from health, environment, or traffic monitoring to industrial automation. These networks consist of nodes which are typically constrained in size and cost which, in turn, leads to a severe limitation of the available energy resources and computational power. The tasks of the individual nodes usually consist of periodic or event triggered transmission of sampled and preprocessed sensor data to a central node where the data is collected and further processed. The wireless communication usually dominates the energy consumption of each wireless sensor node and limits the maintenance-free life time of individual nodes and the whole network. The optimization focus in these networks clearly lies on the optimization of the communication.

While this problem is solved using *forward error correction* (FEC) codes in cellular networks (as for example wireless LANs or cellular telephony networks), in WSNs the majority of publications favors a different approach to limit transmission energy. Often it is argumented that error correction codes are computationally too complex for wireless sensor nodes. Instead multi hop protocols are developed to span the distance between sender and receiver. Based on the assumption that transmission energy for a given distance d scales as d^{α}, where α is the so-called path loss exponent, this sounds reasonable. In this scenario routing over many very short hops is preferable, as by dividing the distance d in to n short hops of d/n, transmission energy will ideally only increase linearly with the distance between sender and receiver.

This first order theoretical analysis is, however, highly unrealistic as it assumes that all nodes from the sender to the receiver are lined up in equal distances on a linear path. It also completely neglects the energy consumption for reception and forwarding in intermediate nodes, as well as all parts of the energy consumption that do not scale with the path loss

MICAz	current @3V [mA]		
power state	total	μ-controller	CC2420
Standby	0.02	0.01	0.01
Active	8.01	8.0	0.01
Transmit [-25dBm]	16.50	8.0	8.50
Transmit [0dBm]	25.40	8.0	17.40
Receive	26.80	8.0	18.80

Table 2: MICAz power consumption [22] in different modes of operation, separated for transceiver and μ-controller.

exponent, like bias power for example, which can be substantial.

A precise investigation of the energy consumption of the MICAz, which is one of the most wide spread wireless sensor nodes today, is presented in [22]. The model is based on measurements and data sheets and consists of a static model, modeling the nodes energy consumption in different states, and a dynamic part modeling the energy consumption and timing behavior of recurring tasks. It shows that for this node the receiver energy is not only non-negligible, but also exceeds transmission energy, even at the highest output power (see also Table 2). This identifies relaying and retransmissions due to lost frames as the dominant source of energy consumption in WSNs. As a conclusion frame losses and relaying have to be minimized to optimize energy efficiency.

Haenggi lists twelve general reasons not to use too short hops in wireless sensor networks [5], as for example end-to-end reliability, delay, and protocol overhead. In his later work [6] he even gives experimental proof in a setup with ten MICAz nodes that long hop routing can clearly perform better than short hop routing.

Zhong et al. [24] propose a radically different strategy for wireless sensor networks. In star shaped single hop networks there is no protocol overhead for route maintenance and they fit perfectly with many of the typical applications with a central data aggregator. They assume this central node has more energy resources and higher computational power which allows to exploit the asymmetric complexity of many of today's advanced FEC codes. Turbo Codes are a good example, as they are very easy to encode and only the decoding requires a computational power that exceeds the capabilities of wireless sensor nodes. The authors call these networks single hop asymmetric structures, or short SHAS.

In [19] we experimented with different forward error correction codes implemented in MICAz nodes. Two of the codes, a repetition code and a modified turbo code, were then evaluated for their energy efficiency. In a lab environment we placed three MICAz nodes using uncoded transmission, a repetition code, and a modified turbo code respectively to send known data packages to a central station. The central station acknowledged every correctly received (or correctably received) package to the sender with a special acknowledgment frame. If the sender of the frame did not receive such an ACK frame within 250ms it tried to retransmit the frame for at most four times. Then the message was considered lost.

Table 3 summarizes the results of this experiment. These results prove the practical applicability of forward error correction codes in SHAS network, and at a bit error rate of roughly 1% outperform uncoded transmission. The message error rate decreases with increasing error correction capabilities of the used code. Only 21.6% of all messages could be succesfully transmitted without coding, while the node using the TC* can deliver 96.9% of all messages.

code	avg. BER	FER	MER	$\frac{Frames}{Message}$	consumed energy [J]	$\frac{Energy}{succ.Message}[\mu J]$
uncoded**	1.23%	89.80%	78.4%	2.34	32,239	346,049
rep. 1/3	1.26%	43.30%	9.7%	1.18	6,566	16,842
TC*	1.36%	21.93%	3.1%	1.12	4,936	11,798

Table 3: Energy consumption in 120 hours (**extrapolated, battery depleted after 48.2h) [19]

This is also reflected in a very low number of ARQs. Most interesting of all is that the consumed energy during the 120 hour runtime of the experiment is lowest for the node using the modified turbo code, although the encoding and transmission of the encoded data introduces an overhead. Due to the very low number of necessary retransmissions this node can spend much more time in standby, thus effectively saving energy. Also the Rep 1/3 code performs very well at this bit error rate. Compared to the node using plain ARQ without FEC the repetition code and the turbo code consume roughly 80% and 85% less energy, respectively.

Taking into account the high number of messages that could not be delivered within four attempts and calculating the energy per successful message transmission the picture gets even worse for the pure ARQ scheme.

6 Conclusion

Without sound models of the power consumption of the underlying hardware and without taking the full system with all its relevant components into account power aware design is nothing less than impossible. We could show that often the reality is in plain contradiction to common belief, intuition, and simple theoretical power models. This is due to non negligible transition times and energy overheads, leakage and other static power consumption. It is imperative to investigate and quantify these effects on system performance when designing power management strategies.

References

[1] ADI Engineering. *80200EVB Evaluation Board Hardware - Users Guide*, Dec. 2001. Revision 1.1.

[2] Department of Software Engineering, University of Szeged. GCC code-size benchmark environment (CSiBE). http://www.csibe.org/.

[3] G. Dhiman and T. S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*, pages 207–212, New York, NY, USA, 2007. ACM.

[4] X. Fan, C. S. Ellis, and A. R. Lebeck. Memory controller policies for DRAM power management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 129–134, 2001.

[5] M. Haenggi. Twelve reasons not to route over many short hops. *IEEE VTC '04 Fall, Los Angelos, CA*, pages 3130–3134, September 2004.

[6] M. Haenggi and D. Puccinelli. Routing in ad hoc networks: A case for long hops. *IEEE Communications Magazine*, pages 93–101, October 2005.

[7] Z. Herczeg, Ákos Kiss, and D. Schmidt. XEEMU homepage.
http://www.inf.u-szeged.hu/xeemu/.

[8] Z. Herczeg, Ákos Kiss, D. Schmidt, N. Wehn, and T. Gyimóthy. XEEMU: An improved XScale power simulator. In N. Azémard and L. J. Svensson, editors, *PATMOS*, volume 4644 of *Lecture Notes in Computer Science*, pages 300–309. Springer, 2007.

[9] Z. Herczeg, Ákos Kiss, D. Schmidt, N. Wehn, and T. Gyimóthy. Energy simulation of embedded XScale system with XEEMU. *Journal of Embedded Computing*, 3(3), 2009.

[10] Intel corporation. *High Performance Memory Controller for the Intel 80200 Processor*, Mar. 2001. Order Number: 273494-001.

[11] Intel corporation. *Intel 80200 Processor based on Intel XScale Microarchitecture: Developer's Manual*, Mar. 2003. Order Number: 273411-003.

[12] J. Janzen. The micron system-power calculator. www.micron.com/support/part_info/powercalc.

[13] JEDEC solid state association. JEDEC standard. www.jedec.org/download/search/JESD208.pdf, accessed March 2009.

[14] Micron. Synchronous DRAM datasheet. http://download.micron.com/pdf/datasheets/dram/sdram/64MSDRAM.pdf.

[15] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, pages 35–44, 2002.

[16] J. Park and Y. Chu. Finite state machine-based DRAM power management with early resynchronization. *IET Computers & Digital Techniques*, 1(4):434–442, July 2007.

[17] Rambus inc. http://www.rambus.com.

[18] D. Schmidt. A PCB for processor controlled voltage switching on ADI 80200 EVB, 2007. internal report, University of Kaiserslautern.

[19] D. Schmidt, M. Berning, and N. Wehn. Error correction in single-hop wireless sensor networks – a case study. In *Proceedings of the IEEE Conference on Design, Automation, and Test in Europe (DATE'09)*, pages 1296–1301, 2009.

[20] D. Schmidt and N. Wehn. DRAM power management and energy consumption: a critcal assessment. In *SBCCI'09, Natal, Brazil, accepted for publication*, 2009.

[21] D. C. Snowdon, S. Ruocco, and G. Heiser. Power management and dynamic voltage scaling: Myths and facts. In *PARC '05: Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, Sept. 2005.

[22] F. Walther. Energy modelling of MICAz: A low power wireless sensor node. Technical Report, University of Kaiserslautern, www.eit.uni-kl.de/wehn/files/reports/micaz_power_model.pdf, February 2006.

[23] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMsim: a memory system simulator. *SIGARCH Computer Architecture News*, 33(4):100–107, 2005.

[24] L. C. Zhong, J. M. Rabaey, and A. Wolisz. Does proper coding make single hop wireless sensor networks reality: The power consumption perspective. *IEEE Wireless Communications and Networking Conference*, 2:664–669, March 2005.