The Neverending Runtime: Using new Technologies for Ultra-Low Power Applications with an Unlimited Runtime

Björn Cassens
TU Braunschweig
cassens@ibr.cs.tu-bs.de

Arthur Martens
TU Braunschweig
martens@ibr.cs.tu-bs.de

Rüdiger Kapitza
TU Braunschweig
kapitza@ibr.cs.tu-bs.de

Abstract

Many wireless sensor network scenarios dictate tight node size and weight limits. Still a long, if possible infinite, mission lifetime is demanded. To fulfill these contradicting requirements, we propose ECON[1] a tailored combination of an energy harvesting device with a new *nano-power sleep mode* that requires near zero power and preserves the application state. An operating system agnostic system software layer combined with NVRAM enables lightweight mode transitions between nano-power sleep mode and normal operation. Furthermore, by introducing a new ultra-low power management circuit we can tolerate power outages that may occur when the power consumption outperforms the harvested energy. ECON has been evaluated using a Texas Instruments MSP430 node equipped with non-volatile FeRAM. Our nano-power sleep mode drains 3.7 times less power than the best onboard available sleep mode. Entering and leaving this mode is worthwhile when the microcontroller idles for more than 1.01 ms.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems; D.4.5 [**Software**]: Operating Systems—*Reliability*

General Terms

Computer Systems Organization, Software

Keywords

Energy Harvesting, Energy Aware Systems, Wearable, Firmware, Operating Systems, Power Outages

[1]**E**nergy **Con**servation Environment

International Conference on Embedded Wireless Systems and Networks (EWSN) 2016
15–17 February, Graz, Austria

ISBN: 978-0-9949886-0-7

1 Introduction

The mission of our interdisciplinary BATS[2] project is to study the social behavior of bats. In order to acquire meeting data, we equip bats with active sensor nodes. However, a bat can only carry 10% of her own body weight, while still behaving naturally [1]. Assuming the average body weight of a *Myotis myotis* bat is 20 g, the weight limit of a node (including the battery) should not exceed 2 g. Currently a runtime of eight days with a 0.6 g sensor node and a 1.2 g battery is achieved. As catching a bat causes stress to the animal resulting in a changed behavior, these events must be kept as low as possible. A frequent battery replacement or increasing the battery size is therefore infeasible. In order to still increase the runtime, decreasing the energy demand of the node is the only remaining solution.

As more than 90% of the time our sensor nodes are idling, a typical way to reduce the energy consumption is to enter a deep sleep mode. However, even in a deep sleep mode, parts of the microcontroller remain turned on, thereby still draining energy from the battery. In some Wireless Sensor Networks (WSNs) the accumulated energy consumption in sleep mode may even exceed the energy consumed during active operation [7]. Hence, we assume that decreasing the energy demand of sleep modes will be beneficial for many WSN settings. Nonetheless, the runtime remains still limited even with a sleep mode that drains near zero power.

For further extending the runtime of sensor nodes, up to an unlimited value, we propose the usage of an energy harvesting device together with a rechargeable battery. However, the amount of energy provided by the energy harvesting device can be highly unreliable as it often depends on many unpredictable factors, i.e. temperature, weather and movements. Furthermore, an energy harvesting device adds weight to the sensor node and in order to keep the weight limits, the size of the battery must be reduced. Obviously, a small battery with low capacity in combination with an unreliable energy harvesting device increases the risk of a power outage. If not prevented, a power outage will lead to loss of data and may bring a sensor node into an unrecoverable state. The challenge is to store the state of the running application and shutdown the sensor node in a controlled way before a power outage happens.

[2]Dynamically Adaptive Applications for Bat Localization using Embedded Communicating Sensor Systems, `http://www.for-bats.org`

A possible solution to preserve data on an upcoming power outage is to write the data into persistent memory. Using flash memory for saving the state of an application is infeasible because a high amount of energy is needed for writing data [6]. Furthermore, flash memory is only blockwise accessible and therefore small and fast write operations are not possible [8]. Together, this renders flash memory useless for storing data energy efficiently at high frequencies. An alternative type of persistent memory that has several advantages over flash-based memory is Non-Volatile Random Access Memory (NVRAM). This group of novel memory technologies is byte-addressable and offers low access latencies comparable to Static Random-Access Memory (SRAM). Thereby, NVRAM can be used as main memory and allows in-place saving of data with minimal overhead.

Together with NVRAM, energy harvester and software, ECON builds a system which is able to sustain power outages and helps to save energy via a Nano-Power Sleep (NPS) mode. Hereby, ECON consists of a hardware (Power Control Unit (PCU)) and a software counterpart, which are interacting closely. The PCU is able to control the power supply of the microcontroller and peripherals. Furthermore, to sustain power outages, the PCU is able to detect upcoming power outages early on and generates interrupts to the microcontroller accordingly. Depending on the signals of the PCU, the software counterpart stores and restores the state of peripherals and application. The same mechanisms are used to build the NPS mode, which turns off the microcontroller in order to save energy.

The paper is organized as follows. The problems arising from sustaining power outages are presented in Section 2. Next, Section 3 details related work followed by the implementation of ECON described in Section 4. The achieved results are discussed in Section 5, while Section 6 concludes our paper.

2 Problem Statement

The main challenge for the utilization of energy harvesters in wireless sensor nodes is the power outage tolerance. Power outages need to be detected in advance in order to save the state and shutdown the sensor node gracefully. For the detection of power outages and for the control of the wireless sensor node power supply we use the PCU which is described in the next chapter. In order to save the state before an immanent power outage occurs, we utilize novel NVRAM and store the state directly in-place in main memory. Together these mechanisms form our novel NPS mode which maintains the state of the sensor node but requires almost no power.

2.1 Power Outage Detection and Control

As mentioned before, the PCU controls the power supply of the microcontroller and detects power outages in advance. In our case, a power outage is not an immediate drop of the voltage to zero since a battery has a cut-off voltage. Of the-shelf Lithium-Polymer rechargeable batteries have a cut-off voltage of approximately 2.75 Volts [4]. Although most Ultra-Low Power (ULP) microcontroller can operate below this voltage, it is very dangerous to continue operation as it will drain the battery further and eventually damage the whole device due to a deep-discharged battery.

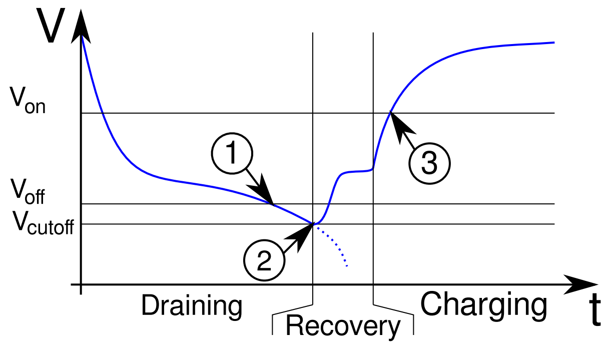

Figure 1. Voltage curves of a fictive battery while discharging, recovering and charging

As the battery has different states, which are depicted in Figure 1, different thresholds must be defined for our approach and are described in the following. While a load *drains* power from the battery the voltage decreases. If the voltage hits the cut-off voltage (V_{cutoff}) the battery must be considered as damaged due to a deep-discharge. In order to shutdown the microcontroller gracefully before a deep-discharge happens, the PCU sends a signal to the microcontroller if a threshold voltage (v_{off}) ① is reached. This threshold voltage is defined by the amount of energy needed to finish running operations on the microcontroller and to save the state of the application. Afterwards the entire sensor node is switched off until the energy harvester has sufficiently recharged the battery to continue operation. As a fall-back mechanism to prevent the deep-discharge, the PCU shuts down the entire sensor node and removes the load from the battery at (V_{cutoff}) ②.

Please note that the battery voltage will increase when the load is removed. This phenomenon is caused by chemical processes and is denoted as *Recovery* in Figure 1. To prevent an immediate start of the microcontroller after it has been turned off, a hysteresis curve must be implemented in the PCU. Therefore, only when the voltage hits a second threshold voltage (V_{on}), the microcontroller can be turned on ③. However, V_{on} must be chosen carefully to prevent the microcontroller bouncing between wakeup and shutdown states.

While designing the PCU, special care needs to be taken to get the most energy-efficient result as the PCU will always be active. Therefore, the circuit must be as small, in terms of used devices, as possible to get a small quiescence current and to prevent a high switching activity. The latter is important since the dynamic current is orders of magnitude higher than the quiescence current.

2.2 State Saving and Recovery

As mentioned before, the application state and also the states of all peripherals must be saved on an upcoming power outage. Once an upcoming power outage is detected, refer to Figure 1 ①, it has to be ensured that enough energy remains in the battery to finish running operations and to save the state. Alternatively, this procedure may also be invoked intentionally to enter our very efficient NPS mode. In this case the external peripherals should continue normal operation as long as possible as they are usually required to wakeup the microcontroller. However, if the PCU detects a power outage during NPS, an emergency wakeup needs to take place which saves the state of the external peripherals and shuts down the microcontroller again. In order to recover the state of the microcontroller on startup, we need to restore the state of the application and of the peripherals. However, external peripherals only need to be restored during the recovery from a power outage since they remain active during the NPS mode.

In terms of saving the application context, most application data is stored in the non-volatile main memory. Therefore, we only need to store the values of all registers and the stack pointer in NVRAM. The energy demand for this is driven by the processor architecture. As the number and bit-length of each register is defined by the architecture it can be assumed as constant. Saving the peripheral states, however, depends mostly on the application which may use only a subset of peripherals. Furthermore, the energy demand of external peripherals like transceivers is highly dependent on their utilization by the software. Therefore, the energy demand to store peripherals needs to be estimated in advance to define a proper threshold V_{on} for the detection of power outages.

After a wakeup of the microcontroller ③, it must be ensured that both, software and peripherals are in a coherent state. Since peripheral states usually cannot be restored at fine-grained level, they can only be set into a default state after a wakeup. In order to avoid inconsistency between the application and the peripherals after a wakeup, all operations that assume a specific state need to finish atomically before a shutdown.

Traditionally, atomicity of operation can be achieved via transactional memory [3]. In case of a power outage, a started transaction can be aborted and safely reexecuted after wakeup. However, transactional memory is not available in hardware for microcontrollers and software solutions incur a huge overhead in term of execution time and energy consumption. In order to avoid the overhead of transactional memory, we propose a different approach: Similar to transactional memory, code sections which are not allowed to be interrupted should be wrapped into atomic blocks. These blocks can only be accessed when it is ensured that the operation will finish before the sensor node runs out of energy. For this technique it is necessary to estimate the worst case energy consumption of all atomic blocks with tools like peek [5] or 0g [14] and adapt the thresholds V_{off} appropriately.

3 Related Work

In previous works, different approaches have been introduced to sustain power outages or to save energy by turning off unused devices or nodes. However, to the best of our knowledge no previous work exists which combines, resilience against power outage with energy harvesting techniques to achieve an endless runtime.

In order to prevent misbehavior of hardware and software due to voltage drops, brown out detections are commonly used [10]. Once a voltage drop has been detected, the microcontroller is restarted and brought into a well-defined state. The drawback of this approach is a complete loss of data and computational progress. ECON also detects voltage drops but maintains the state of the application. Therefore, it can be used as a very energy-efficient sleep mode.

Smart drivers [2] are a concept that utilizes NVRAM to improve energy efficiency. In order to collect data, the microcontroller performs only a partial startup into a low-power mode. After certain amount of data is acquired, a full system start is performed to process the data. While waiting for new data to arrive, the microcontroller can be turned off without any data loss. With a high performance microcontroller the authors could reach an average energy demand comparable to an ULP microcontroller. ECON, however, can also be applied to high performance microcontrollers but requires less energy than any ULP microcontrollers we know at this time. Additionally, ours is capable of energy recovery and can tolerate power outages.

In EPIC [15], Lohit Yerva et al. utilizie an energy harvester as the primary power supply of a node. To sustain power outages, the application state is stored as checkpoints in a battery backed SRAM. On a power outage or for saving energy, a hardware module, similar to our PCU, turns off the entire node. In contrast, ECON is able to turn-off the microcontroller and each of its external peripherals separately. Thus, external peripherals like a wake-up receiver can be left turned on in order to enable communication with other nodes without any complex synchronization mechanisms. Furthermore, ECON maintains the whole program context. Therefore transitions into NPS or power outages are transparent to the application and no time consuming checkpointing is required.

Similar to our approach, the microcontroller is turned off in Hypnos [7] and an energy harvester is used to refill the battery during sleep times. The main memory, however, stays active. In order to save energy, the voltage of the main memory is reduced below its specified values. The key assumption is that no power outage occurs while the energy harvester recharges the battery, which is not realistic. Moreover, an undervolted memory highly increases the probability of data corruptions. For example, a work from Zhu et al. [16], shows that undervolting increases the error rate exponentially. The authors also do not provide a solution on how to store and restore peripherals. Therefore, in our opinion the approach of Hypnos is not applicable to existing wireless sensor nodes.

4 Implementation

For our implementation of ECON we used the MSP430FR5969 microcontroller from Texas Instruments [12]. This microcontroller provides FeRAM-based NVRAM as program and main memory. This section will firstly describe the PCU and its characteristics and afterwards the software infrastructure.

4.1 Power Control Unit

As mentioned before, the PCU needs to be always active to monitor the power source. Therefore, it is crucial for this

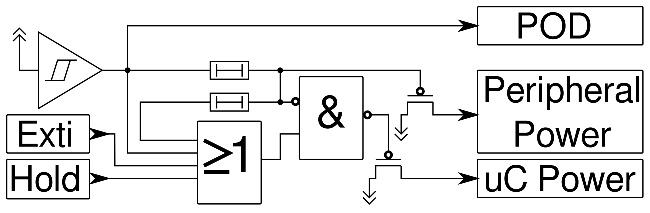

Figure 2. Power Control Unit with its internal logic including the Schmitt-trigger

device to drain as few energy as possible. In order to ensure a minimal power consumption only few parts should be used to keep the switching activity as low as possible. Furthermore, analogue parts such as analog-digital converters or operational amplifiers usually have a high power consumption. This means they cannot be used, as they consume too much power. In order to measure the voltage we propose using a Schmitt-trigger like the NSP1101 [9]. Upcoming subthreshold technologies [11] can decrease the required current for these circuits down to 4 nA. Furthermore, a digital control logic is required to generate all necessary signals for the microcontroller and also to control the power supply of the node. It is important that this circuit has a minimal power dissipation.

As shown in Figure 2, our solution requires only two gates to generate all necessary signals for the software. This includes signals for turning on and off all peripherals, the microcontroller, and the Power Outage Detection (POD) signal. If a power outage is detected the PCU generates a POD signal which in turn prompts the microcontroller to save its current state (= all registers and all peripheral states) to the NVRAM. Additionally, if the microcontroller is already in NPS mode at the time it receives the POD signal it needs to wakeup and save the state of the external peripherals. This is needed because we keep the external peripherals active during NPS. The POD signal has also a second purpose for turning off the sensor node after a predefined delay. Hereby it is ensured that, even if the microcontroller or the software misbehaves, a damage of the battery is prevented. After this delay only the PCU remains activated and drains energy from the battery. However, the energy consumption of the PCU is negligible when implemented as previously described.

In order to ensure a wakeup of the microcontroller during NPS mode, an external interrupt pin (EXTI) is implemented. Each interrupt signal of the external peripherals must be connected to this signal. A hold signal (HOLD) is implemented to prevent a shutdown of the microcontroller during critical regions. This is necessary since interrupt processing or atomic regions can reset the interrupt of an external peripheral which would cause a shutdown of the microcontroller without a hold signal.

4.2 Software Infrastructure

The software infrastructure is implemented in form of a small hardware dependent library. It interacts closely with the PCU and is responsible for saving the state of the peripherals and the application. To ensure a response to a power outage, the library requires at least one interrupt which is driven by the POD signal. Additionally, an output for the HOLD signal is required to ensure that the microcontroller stays active until the software has finished its task.

For saving the application state, only the registers and the stack pointer need to be saved. The remainder of the application state is implicitly stored in the non-volatile main memory. Saving the registers and the stack pointer can be done with only two assembly instructions (`PUSHM.W`). Hereby, saving and restoring the application state is implemented exactly like a context switch between two processes.

To save the whole state of internal peripherals to the NVRAM, specialized methods are implemented for each device. Whenever the configuration of these peripherals is altered, a new state is saved in the main memory for the recovery process. If external peripherals are used, callback functions for saving and restoring the state need to be provided by the user. This is necessary since a multitude of peripherals are available and no assumption of their protocols can be made. For saving the states persistently, the programmer can simply use global variables, as these are automatically stored in a persistent memory. For example, if a radio device is used, saving and restoring the configuration would be sufficient. Volatile data like the received bytes can be neglected since we assume that any operation depending on this data will be executed inside an atomic block and is guaranteed to finish before a system shutdown.

If a code block requires atomicity, two functions are provided to mark the beginning and the end of an atomic region. Calling the function `atomic_start()` disables all interrupts. In order to finish an atomic block, the `atomic_end()` needs to be called which enables interrupts again. Please note that no additional voltage measurements are needed since the voltage threshold for the POD signal can be dimensioned appropriately by estimating the worst case energy consumption with tools like Peek [5] or 0g [14] for all atomic blocks. Therefore, even if a power outage interrupt occurs right after entering the atomic block, it is ensured that the atomic block will finish before the energy runs out. After calling `atomic_end()` the microcontroller will process any available POD signal and will shut down the sensor node gracefully.

To distinguish all possible startup routines (normal startup, wakeup from NPS and power outage) a custom startup was implemented. It maintains the HOLD signal and chooses the right startup sequence depending on an internal status variable. All data is initialized if a normal startup is performed, which is the case when the software is started the first time or when the microcontroller was not able to save all states before a power outage. This especially includes the `.data` and `.bss` sections inside the main memory and a jump into the main function of the application.

In contrast to the normal startup, the state recovery does not use any variables stored in the stack. By doing so, any data corruption of the application data stored on the stack is avoided. Afterwards, the states of internal, and if necessary external, peripherals are restored. Finally, the status variable is reset and a context switch to the point where the microcontroller entered the sleep or where the power outage occurred is performed.

In addition to the presented startup sequences, a special case must be considered. While the microcontroller is in NPS mode, a power outage can occur. Since external peripherals remain active during NPS, the microcontroller needs to wakeup and save their states on an incoming power outage. To prevent unnecessary recovery overhead, the startup function saves the external peripheral states without restoring the application states and shuts down the microcontroller again.

5 Preliminary Results

We evaluated our prototype with tools onboard the MSP-EXP430FR5969. It provides a debugger, an energy meter and a minimal environment for user interactions like LED's and switches. All measurements are based on the energy meter and the software provided by Texas Instruments. For the measurement of the energy demand we turned off any LED's and onboard peripheral devices to keep the measuring errors as low as possible. Furthermore, the microcontroller was set in a freerunning mode which prevents energy overheads caused by entering breakpoints.

In order to compare ECON to existing low power modes provided by the microcontroller, we did three evaluations. The first one is a theoretical comparison between values from a data sheet [13] and our simulation results, which are presented in Section 5.1. As our recovery of the states may add an overhead, we measured the energy demand of the state saving and recovering process and compared it to a normal startup in Section 5.2. Finally, in Section 5.3 we used the values presented in Section 5.1 and 5.2 to compare ECON to existing low power modes of the microcontroller.

5.1 Theoretical Energy Demand

The energy consumption during the NPS mode is solely driven by the PCU that has to be always active. In the BATS project we want to integrate the PCU into a custom ASIC chip, build in 150 nm technology. However, since we do not have our final hardware yet, we simulated the PCU circuit. For the simulation of the control logic we used models provided by LFoundry and the Spectre Circuit Simulator from Cadence. The stimuli data has been chosen according to the normal operational case, i.e. no power outage is detected and every second an interrupt is generated.

To get comparable values to the data sheet provided by Texas Instruments, we used the same operation voltage of 3 V. Taken from the simulation, the control logic of the PCU requires a quiescence current of 1.3 nA. Together with the Schmitt-trigger's estimated average power consumption of 4 nA [11] the PCU consumes 5.3 nA. As proposed in [13], the smallest current consumption is achieved while the microcontroller is in the Low Power Mode (LPM) 4.5. In this mode, the microcontroller typically consumes 20 nA, which is 3.7 times higher than our presented approach.

5.2 Startup and Recovery Energy Demand

Beside the quiescence current during sleep modes, another source of energy overhead is the state saving and recovery process. We compare our results with a normal startup sequence as some low power modes, and especially the LPM 4.5, trigger a reset of the microcontroller on wakeup. We conducted measurements for different amounts of global variables, since these need to be initialized on each normal

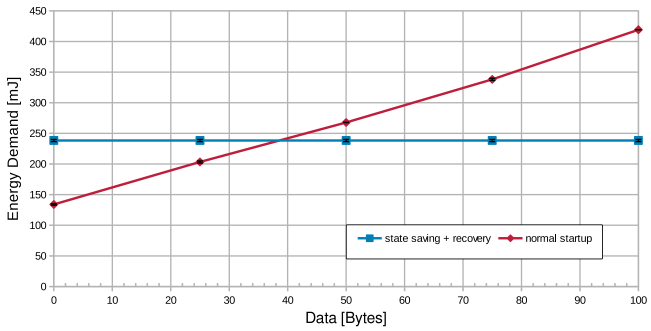

Figure 3. Measured energy demand for a normal startup compared to the combined energy demand of our state saving and restoring procedure (100,000 repetitions)

startup. To get comparable results, no peripherals other than the in- and output ports that are needed by ECON were initialized. As the energy demand of the startup and saving procedures is extremely low, we repeated these procedures 100,000 times each and measured the overall energy demand. Additionally, to prevent any influence by transient effects we repeated each measurement five times. As Figure 3 shows, the energy demand of a normal startup highly depends on the amount of initialized global variables. ECON on the contray, has to restore the application state by writing all processor register values back which has a fixed energy demand of 238.14 mJ for 100,000 subsequent startups. At less than 40 Bytes of global variables, a normal startup consumes less energy than our solution. Typical applications, however, require much more than 40 Bytes of global variables and would benefit from ECON. For example a simple `Hello-World` application written in C requires 1312 Bytes stored in global variables. In this case, our recovery outperforms the normal startup that requires 1348.27 mJ for 100,000 startups by a factor of 5.6.

5.3 Sleep Mode and NPS comparison

Since the NPS adds an overhead for saving and restoring the states, we investigated the minimal duration at which the NPS mode would save more energy compared to existing low power modes of the microcontroller. By using the results from Section 5.2 (238.14 mJ for 100,000 iterations) the average energy demand to enter and leave the NPS is 2.38 μJ. Furthermore, we have to add the power consumption of the PCU, as this device remains always active. Since we do not have the PCU in hardware yet, we use the simulation numbers from Section 5.1. By multiplying the voltage and current, we get a power consumption of $P = U \cdot I = 3\ V \cdot 5.3\ nA = 15.9\ nW$ for the PCU. Based on the formula $E = P \cdot t$ the energy demand of ECON is $2.38\ \mu J + 15.9\ nW \cdot t_{sleep}$, where t_{sleep} is the time spend in sleep. We compare these numbers with all low power modes that do not require an entire restart of the microcontroller. For this evaluation, we measured the average power consumption during 60 seconds in each low power mode. The results of this experiment are shown in Table 1 in the *Power* column. As can be seen, the power draw during the LPM modes is orders of magnitude higher compared to the

Table 1. Measured values for each sleep mode (Power) and computed times (Time) in which ECON would achieve higher energy savings

Sleep Mode	Power [mW]	Time [us]
LPM 0	2.7191	875.29
LPM 1	2.6434	900.36
LPM 2	2.4941	954.26
LPM 3	2.4673	965.62
LPM 4	2.3550	1010.62

PCU. Therefore, the power consumption of the PCU is negligible and the energy overhead of the NPS is mainly driven by the constant energy demand that is needed to enter and leave the NPS.

With the known power draw of the NPS and the LPM we can now compute the minimal amount of time needed to be spent in NPS to become more energy-efficient than any given LPM. Again the formula $E = P \cdot t$ is used here. As can be seen in the *time* column in Table 1, the NPS becomes more energy-efficient than every other LPM when the microcontroller remains for approximately 1.01 ms in this mode. Furthermore, as the highest low power mode (LPM 4.5) typically consumes more energy during startup and has a higher quiescence current, our NPS renders this mode useless. Since our measured values for the LPMs differ orders of magnitude from the data sheet, we tested the accuracy of the onboard measurement device. Measuring the current through high precision resistors of known size we noticed a maximum measurement error of 1.6%. Therefore, we assume that our measured values are realistic and the bulk of the current is consumed by external circuits which are required by the microcontroller.

6 Conclusions and Future Work

Wireless sensor nodes are commonly restricted by weight and size limits, yet always higher runtimes are desired. ECON tackles this contradicting requirements by using an energy harvester in combination with a novel Nano-Power Sleep (NPS). The NPS deactivates the sensor node without loosing the application state and can therefore tolerate power outages that are caused by an unreliable power supply like an energy harvester. ECON achieves this by utilizing NVRAM technology in combination with system software to store the entire application state in-place in main memory. Additionally, we introduce a novel Power Control Unit (PCU) that is responsible for preliminary detection of power outages and for the power control. As the PCU is the sole component that remains active during NPS, we show how to implement this part in a very energy efficient way. To prove the feasibility of ECON we implemented a prototype based on a microcontroller with non-volatile FeRAM. Our evaluation results show that the PCU implementation can achieve a very low power drain of 15.9 nW. Together with our lightweight save and restore mechanism for the application state the NPS outperforms any available sleep mode on our hardware at 1.01 ms of sleep time.

7 Acknowledgements

This work was partly supported by the German Research Foundation (DFG) under grant no. FOR 1508 and grant no. KA 3171/3-2. Special thanks go to Chia-Yu Hsieh, who simulated our control logic for the energy estimation in our evaluation.

8 References

[1] S. Amelon, D. Dalton, J. Millspaugh, and S. Wolf. Radiotelemetry techniques and analysis. In *Ecological and behavioral methods for the study of bats*, pages 57–77, 2009.

[2] C. Brandolese, W. Fornaciari, L. Rucco, and F. Terraneo. Introducing smart drivers a way to conceive smart data sensing in wireless sensor networks. In *Information Communication and Embedded Systems (ICICES)*, pages 1–6, 2013.

[3] P. Felber, C. Fetzer, P. Marlier, and T. Riegel. Time-based software transactional memory. *IEEE Transactions on Parallel and Distributed Systems*, pages 1793–1807, 2010.

[4] Guangzhou Markyn Battery Co., Ltd. *Specifications for GMB-300910*. Rev. A/0.

[5] T. Hönig, H. Janker, C. Eibel, O. Mihelic, and R. Kapitza. Proactive energy-aware programming with peek. In *Conference on Timely Results in Operating Systems (TRIOS)*, 2014.

[6] K. Ishida, T. Yasufuku, S. Miyamoto, H. Nakai, M. Takamiya, T. Sakurai, and K. Takeuchi. A 1.8 v 30nj adaptive program-voltage (20v) generator for 3d-integrated nand flash ssd. In *Solid-State Circuits Conference - Digest of Technical Papers, (ISSCC)*, pages 25–33, 2009.

[7] H. Jayakumar, A. Raha, and V. Raghunathan. Hypnos: An ultra-low power sleep mode with sram data retention for embedded microcontrollers. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–10, 2014.

[8] V. Mohan, S. Gurumurthi, and M. Stan. Flashpower: A detailed power model for nand flash memory. In *Design, Automation Test in Europe Conference Exhibition*, pages 502–507, 2010.

[9] Nano Power Solutions, Inc. *NanoPower -CMOS Comparator*. Visited on October 2015, `http://www.npsi.jp/j/product/nps1101.pdf`.

[10] Philips Semiconductors. *Protecting Microcontrollers against Power Supply Imperfections*. AN468, published in May 2001.

[11] SAPNA and B. P. SINGH. Low power schmitt trigger in sub-threshold region. In *Proceeding Of International Conference On Recent Trends In Computing and Communication Engineering*, RTCCE, pages 88–91, 2013.

[12] Texas Instruments. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide*, 2012. Rev. G, Revised October 2015.

[13] Texas Instruments. *MSP430FR59xx Mixed-Signal Microcontrollers*, 2012. Rev. G, Revised March 2015.

[14] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat. Worst-Case Energy Consumption Analysis for Energy-Constrained Embedded Systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–114, 2015.

[15] L. Yerva, B. Campbell, A. Bansal, T. Schmid, and P. Dutta. Grafting energy-harvesting leaves onto the sensornet tree. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*, IPSN, pages 197–208. ACM, 2012.

[16] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided Design*, ICCAD, pages 35–40. IEEE Computer Society, 2004.