An Opportunistic Reconfiguration Strategy for Environmentally Powered Devices

Igino Folcarelli, Alex Susu, Ties Kluter, Giovanni De Micheli
LSI - EPFL, Lausanne - Switzerland
{igino.folcarelli,alex.susu,ties.kluter, giovanni.demicheli}@epfl.ch

Andrea Acquaviva
STI - Urbino University, Urbino - Italy
acquaviva@sti.uniurb.it

ABSTRACT

Environmental energy is becoming a feasible alternative to traditional energy sources for ultra low-power devices such as sensor nodes and smart watches. Moreover, the increasing need for flexibility and reconfigurability of such devices makes its energy management even more challenging. As a result, to efficiently exploit the potentially unlimited environmental energy, new adaptation strategies are required. In this paper we present a novel system reconfiguration strategy that exploits the intrinsic unpredictability of environmental energy to opportunistically reconfigure the device. To assess the effectiveness of the proposed reconfiguration strategy we first perform a theoretical evaluation using statistical energy profile distribution and then we evaluate its energy efficiency on a prototype device in the presence of bursty energy profiles that we emulated using a programmable energy source.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms

experimentation

Keywords

reconfiguration, scavenging, sensor network

1. INTRODUCTION

The new frontier of ultra low-power computing is represented by highly reconfigurable devices providing adaptability to changing requirements to the physical and applicative context. Wireless sensor nodes, for instance, can be used in many heterogeneous domains such as environmental monitoring as well as Ambient Intelligence applications [1]. Personal devices like Personal Digital Assistants (PDAs) and palmtops are going to integrate even more functionalities to increase the range of ubiquitous personal services. In this context, the system must be able to reconfigure itself at various levels, ranging from the hardware level (through reprogrammable devices) to network and application level. Quality of Service (QoS) adaptation is an example of application level reconfiguration [7].

On the other hand, the developing technology of energy harvesters and the improved energy efficiency of electronic devices is reducing the gap between their requirements and the exploitable environmental energy [10, 11]. However, this gap is still hard to be filled because of two main reasons. First, the energy level provided by scavenging devices strongly depends on the technology used. For instance, solar cells provide between $100mW/cm^2$ (directed toward bright sun) and $100W/cm^2$ (illuminated office), while vibrational microgenerators provide $4W/cm^3$ for human motion) [10].

This energy is not always available or effective in the context in which these electronic devices have to operate. Even where the scavenger technology can be efficiently exploited there is still a mismatch between the energy profile of the device and the one provided by the environment. This is mainly due to the unpredictability of environmental energy, which is partially modulated by using rechargeable batteries and capacitors to smooth the energy profile. However, a complete adaptation is hard to achieve because of the limited capacity of energy storage elements, coupled with the uncertainty of the energy source. For instance, it could happen that an energy burst arrives right after the battery has been filled-up. In this case, the additional energy (that we call *energy slack*) is wasted if not used directly by the device itself. This situation is of practical relevance with photovoltaic (PV) systems like solar cells, where during long periods of "light" it becomes difficult to store all the available energy.

In this work we propose an adaptation strategy that exploits unused energy slacks to perform system reconfiguration. The main idea is to spend as much energy as possible when there is a burst to improve the energy efficiency of the system for later bursts. This approach is in contrast with traditional battery aware power management where the objective is to minimize the energy consumed by the system. In the proposed strategy, we promote an increase of the energy consumption in the presence of energy slack that cannot be absorbed by a storage element. The overall objective is to adapt the energy profile of the system with the energy available from the environment.

The proposed strategy has been first analyzed through a

CF'06, May 3–5, 2006, Ischia, Italy.

simple statistical model. Then, an energy-aware reconfiguration manager has been implemented in a reconfigurable prototype system. Energy bursts have been provided by a programmable power generator to perform experiments in a controllable way and match the requirements of the prototype board.

2. OPPORTUNISTIC ENERGY MANAGEMENT CONCEPT

In this section we explain the rationale behind the proposed strategy by first introducing the application and system context and then describing the issues related to environmentally powered devices.

2.1 Reconfigurable System Context

In this section we describe the concept of opportunistic energy management as a suitable strategy for environmentally powered systems. The kind of system we are considering in this work are ultra low-power reconfigurable devices. Reconfigurability is one of the most promising research topics in the field of ultra-low power systems, since these devices must adapt to a more and more heterogeneous application context [8, 9, 6, 12]. These systems have to execute different tasks depending on user requests or external events in a context-aware system. For instance, a wireless sensor node could be programmed to sense different external events such as temperature or light changes. Depending on the external event, a different task is to be executed. System reconfiguration can be exploited to perform the current task to be executed in a more energy efficient way, for example through on-board field programmable logic (FPGA). Hardware reconfiguration is to be considered one of the key features for the development of future low-power and flexible multimedia applications. Low-power microcontroller used in sensor nodes can be equipped with small on-chip FPGA aimed at performing signal processing routines [2]. Moreover, reconfigurable processors for low-power applications have been recently proposed [13]. Another type of reconfiguration can be performed to achieve a better energy/QoS trade-off by executing a version with a different quality of the same task (Multi-Version Scheduling [7]).

Taking the previous example, a sensor node may be asked to take a picture of the environment when a particular event happens (e.g., a person enters the viewer of the sensor). Typically the image must be compressed before being sent to a gateway node to reach the data fusion system. The trade-off here could be between power consumption and quality of the reconstructed image. If the system is provided with reconfigurable hardware, the same compression routine could also be executed in hardware to save energy. In both cases (hardware or software reconfiguration), the reconfiguration requires additional energy.In the hardware case, it involves the physical reprogramming of the device, while in the software case the energy is taken by the decision and by the loading of the new task version in the system memory.

In this paper experimental validation has been carried out with a hardware reconfigurable platform, where the system is provided with a library of hardware routines corresponding to the software tasks. We assume that not all of the possible tasks to be executed can be programmed in the reconfigurable hardware at the same time, because of the limited size of FPGAs suitable for low-power systems [2].

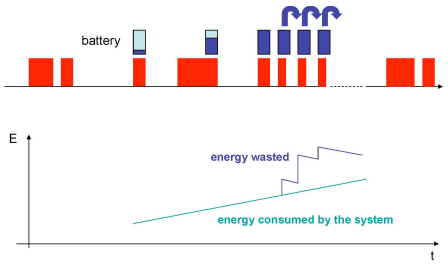


Figure 1: Energy wasted because of limited storage capacity.

2.2 Environmental Energy Context

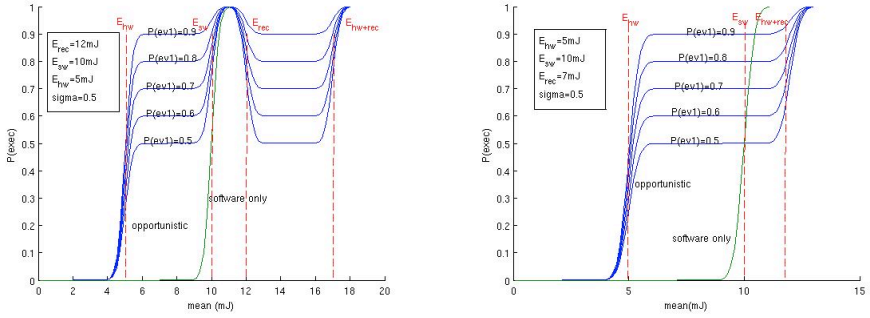
Typical scavengers provide energy in an unreliable way. As a general case, we consider a system that has an energy storage element with a limited capacity, either an ultracapacitor [14] or a rechargeable battery. When the scavenger generates is generating energy a certain amount of energy is used to refill the storage elements and the remaining energy is used to support ongoing computation. Due to the unreliability of the source, it is possible that the system receives two or more long light periods that fills-up the storage element [3]. In this case, the energy cannot be stored and must be consumed by the device. Without any adaptation, the amount of energy consumed by the device will probably not match the available energy that is then partially wasted. This situation is depicted in Figure 1. The opportunistic energy management approach is aimed at reducing the waste of energy by adapting the power consumption of the device. The adaptation is performed by reconfiguring the system such that the same task will be executed at a higher efficiency in the future. In this way, the energy that is otherwise wasted is consumed by the reconfiguration process. By doing so, we can also increase the probability of the successful completion of a task execution in the following bursts - another possible extension would be to achieve a better QoS for the same energy cost.

With respect to traditional battery-aware power management strategies [4], the proposed approach is not simply aimed at reducing the energy consumed by the system. It also increases the energy consumption inside an energy burst in order to match the energy profile provided by the environment, thus improving the energy efficiency in the following bursts.

3. STATISTICAL ANALYSIS

In this section we provide a statistical analysis of the effectiveness of opportunistic reconfiguration strategy. We suppose to have a software model with two alternative tasks to be executed depending on associated events that trigger them. Moreover, we suppose to have a hardware version of both routines and because of the limited capacity of the FPGA (this is a reasonable assumption in for a FPGA in a low-power device), only one task can be loaded into the FPGA at any moment. Let us consider that we have at most one execution of a certain task to perform for each energy burst. In this case, it is of interest to compute the probability to perform a single execution.

We distinguish two cases: (i) the reconfiguration has a considerable cost (E_{REC}), higher than the energy required



a) $E_{rec} > E_{sw}$. b) $E_{rec} < E_{sw}$.

Figure 2: Probability of one task execution as a function of the mean energy burst value.

for software execution (E_{SW}) and hardware execution (E_{HW}); (ii) the reconfiguration takes lower energy compared to at least E_{SW}. In both cases we suppose that $E_{HW} + E_{REC}$ is greater than E_{SW}. If this is not the case, it is always convenient to perform reconfiguration and hardware execution. In the first case, the probability to perform a task execution is given by:

$$\begin{aligned} &P(HW) \cdot P(E \geq E_{HW}) + \\ &(1 - P(HW)) \cdot (P(E_{REC} > E \geq E_{SW}) \\ &+ P(E \geq E_{HW} + E_{REC}) \end{aligned} \quad (1)$$

$P(HW)$ is the probability that the task to be executed is programmed into the reconfigurable hardware, which depends on the status of the reconfigurable device. This means that the probability of executing the task once in an energy burst is given by the sum of the probability of executing the task in hardware, if the task is loaded in the FPGA. Otherwise, if the task is not loaded in the FPGA $(1 - P(HW))$, the probability of execution is given by the probability of executing the task in software ($P(E_{REC} > E \geq E_{SW})$) or the probability of loading the task in FPGA and executing it in the hardware ($P(E \geq E_{HW} + E_{REC})$).

It must be noted that if the task to be executed is not present in the reconfigurable hardware, then the task will be executed in software if the energy burst is larger than software energy but lower than energy required for reconfiguration. In fact, we consider an aggressive policy that performs reconfiguration each time the task to be executed is not in the reconfigurable hardware and we have energy to do it. If burst energy is larger than reconfiguration energy, we use it for reconfiguration and we loose the opportunity of execution, unless burst energy is larger than $P(E_{REC} + E_{HW}$. In this case, we perform reconfiguration and hardware execution.

The second case we consider is when the reconfiguration energy is smaller than energy for software execution. In this case, the first part of the equation is not modified, while in the second part the probability of execution is larger than zero only if the energy of the burst is larger that $P(E_{REC} + E_{HW}$. In fact, if the energy is lower than E_{REC} the task cannot be executed in hardware nor in software. The probability of execution is given by:

$$\begin{aligned} &P(HW) \cdot P(E \geq E_{HW}) \\ &+ (1 - P(HW)) \cdot P(E \geq E_{HW} + E_{REC}) \end{aligned} \quad (2)$$

Since the target of the probabilistic model is to set the bounds of the effectiveness of the opportunistic reconfiguration, we suppose for the moment that the policy is implemented by an oracle which always perform the best choice in terms of reconfigurations. Between two possible reconfigurations, the oracle always decide to reconfigure the most frequent event. Which event is the most frequent depends on the probabilistic distributions of execution between events. If we express the probability of an event i as $P(ev_i)$, we can say that the probability of having in the reconfigurable hardware the task selected for execution corresponds to: $P(HW) = max(P(ev_1), P(ev_2))$.

Most of the environmental energy sources can be modeled through a Gaussian density distribution [3]. If we use this distribution for energy bursts $P(E)$ and we use a uniform distribution of events $P(ev_i)$, we can compute the probability of one execution as a function of energy and event distribution.

In Figure 2.a we draw the probability of one execution in an energy burst as a function of the mean of the gaussian distribution of energy bursts for different values of probabilities of events ev1 and ev2 and in case of E_{rec} larger than E_{sw}. In Figure 2.b the probability is computed for E_{rec} smaller than E_{sw}.

The probability is shown in the case of "software only execution" ($P(E > E_{SW})$) and in the case of the opportunistic reconfiguration strategy. When E_{REC} is large, the probability of execution is greater than "'software only"' case for values of energy burst between E_{HW} and E_{SW}. This is because of the contribution of the probability of having the task in hardware. The execution probability goes to one for both curves when there is energy enough to perform the task in software. However, in case of opportunistic strategy, as soon as we have enough energy to perform reconfiguration ($E > E_{REC}$) the aggressive policy will perform it and, compared to the software only case, we loose opportunity for execution if the energy is not enough for reconfiguration and hardware execution.

When E_{REC} is small (Figure2.b), the probability of execution is greater than "'software only"' for energy bursts lower than E_{SW}.

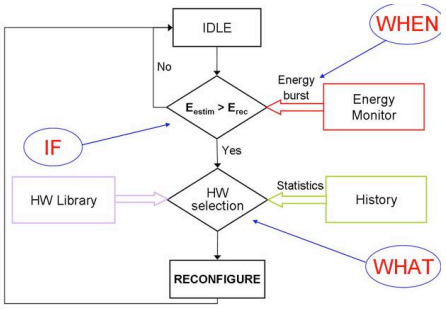


Figure 3: Reconfiguration policy.

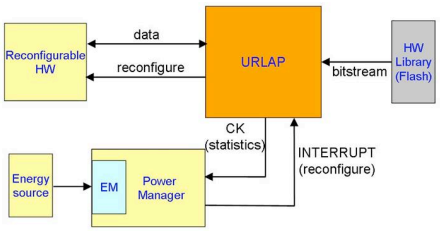


Figure 4: System architecture.

The most important observation is that the probability is that we have a clear improvement of the probability of executing a task in a burst when we are using opportunistic reconfiguration w.r.t. the normal "software only" execution. There are also cases in which the probability is lower, due to the cost of reconfiguration and the aggressive policy. This behavior depends also on the relationship between reconfiguration energy and hardware/software energy. For this reason, the presented analysis can be used to design a suitable reconfiguration policy.

Finally, because of the oracle, the probability of execution in case of increased probability of event 1 increases as well as outlined by the different curves shown in Figure 2.

4. RECONFIGURATION MANAGER DESIGN

In the previous section, in order to achieve upper bounds in the probability of the execution of a task within an energy burst, we use an oracle that decides what is the most suitable task to be loaded in the FPGA. In this section we consider a real system that does not know in advance what the best reconfiguration choice is. A viable solution is for the power manager to collect statistics regarding the number of times tasks are executed, and decide which task to load in the FPGA based on these statistics.

The reconfiguration policy implemented by the energy manager is represented in Figure 3. When an energy burst arrives from the environment, the energy manager checks if the energy slack is big enough to support the reconfiguration. If this is the case, it looks at the execution statistics - we will describe later how they can be collected - to decide which is the best candidate for reconfiguration among the available ones, which are stored in a hardware block library. In the proposed strategy we always perform reconfiguration whenever there is a slack. This is because the amount of energy slack is not known when the burst arrives. Our opportunistic approach always tries to profit from the presence of a slack. Clearly, the additional energy spent for reconfiguration could prevent the execution of a task that would be otherwise executed if the whole slack was exploited. However, as pointed out in the statistical analysis, this can be compensated by the improved energy efficiency depending on the distribution of burst intervals. Reconfiguration is not performed if the task that is predicted to be executed is already loaded in the FPGA or there is no hardware version available. Bitstreams of task candidates (hardware library) are stored in the FLASH memory.

The policy described has been implemented in our proof-of-concept prototype system. It is represented by a custom design board equipped by the URLAP processor [5] (a low power ARM-based processor), two FPGAs, an external DRAM and a FLASH. FPGAs can be configured to be either memory mapped or to have a coprocessor interface to the URLAP processor. We used the first configuration in our experiments. FPGAs and URLAP communicate through the shared memory view and interrupt lines. FPGA reconfiguration can only be done by the URLAP. Since the board is not optimized for being powered by a real scavenger, we used a burst emulation system based on Labview software and a Data AcQuisition Board (DAQ). A detailed description of the board and the burst emulation system is beyond the scope of this work.

In this paper, we present a hardware implementation of the energy-aware reconfiguration manager on top of one of the two on-board FPGAs. However, a software implementation is also possible. One of the two on-board FPGAs has been used for reconfigurable computation, while the other one as a reconfiguration manager. The overall system architecture is shown in Figure 4.

The hardware reconfiguration manager is directly connected to the energy source to detect supply power levels. It has also knowledge of the status of the battery, so that it is able to detect slacks. Since we do not know in advance the shape of the energy burst, we start the reconfiguration process as soon as the power level of the slack is larger than the reconfiguration power. The block performing energy detection is indicated as Energy Monitor (EM) in Figure 4.

The interface of the reconfiguration manager to the main processor is represented by an interrupt signal (INTERRUPT) and a checkpoint signal (CK). The first one gives to the reconfiguration manager the capability of issuing a *reconfigure* command to the main processor in the presence of an energy burst. The second one is used in the statistic collection process. This interface is logically represented by a dedicated shared memory location where information about the last execution is written by the software and read by the reconfiguration manager. After being read, this information is used to update execution counters. The reason why the statistic signal is called checkpoint can be found in the software implementation. Statistics are collected by the insertion of code checkpoints that indicate how many times a certain task is executed. Code checkpointing is a well-known solution for designing application-aware power management policies. In this work we exploit the same concept for the implementation of the proposed environmental energy-aware reconfiguration strategy. Execution statistics drive reconfiguration decision. The most probable task to be executed is estimated depending on past executions. In this work we do not explore task prediction policies since we focus on the opportunistic reconfiguration concept.

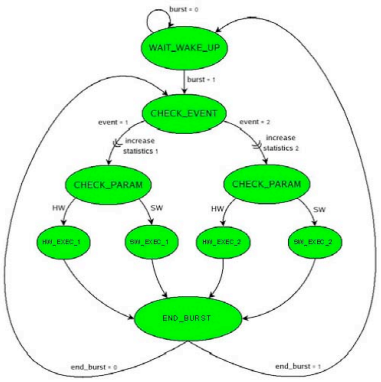


Figure 5: Event-driven application model.

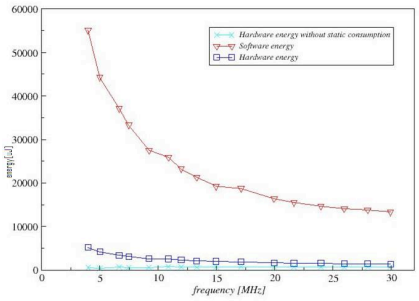


Figure 6: Energy for hardware and software task versions.

5. EVENT-DRIVEN APPLICATION MODEL

In this section we describe the software model we consider for the study of the feasibility of the proposed strategy. Our model is an event-driven application which is composed of two alternative tasks that are executed depending on some external event (i.e. a sensor outcome). This model is implemented as software running on the main processor of the prototype board. The external event has been emulated thanks to a randomly generated variable, depending on which of the two possible tasks is executed. This simple software model, represented in Figure 5 allows us to study inherent properties of the reconfiguration strategy in a repeatable and controlled way.

Software execution is triggered by the arrival of an energy burst ($burst = 1$). The external event is then responsible for selecting one of the two tasks. If an event of type 1 (2) is detected ($event = 1(2)$) execution statistics are updated correspondingly through code checking points (*increase statistics 1 (2)*). Then, the application has to select between hardware and software execution. This is obtained by checking a shared memory location ($CHECK\_PARAM$) written by the reconfiguration manager depending on which routine has been actually reconfigured in the FPGA. If there is still energy, another computational loop is performed.

Thanks to this implementation, the number of execution loops becomes a metric for the energy efficiency of the reconfiguration strategy. In fact, the lower the energy taken by each loop, the higher the number of loops that get executed within a single energy burst. Thus, instead of measuring the energy consumed, we can just annotate the number of loop iterations per burst.

The computation performed by the tasks is a 4th order Finite Impulse Response filter (FIR). The choice of this software routine has been motivated by the following reasons: i) it is a signal processing algorithm suitable for typical sensor networking application; ii) it results in a hardware implementation easy to fit in a small-sized FPGA suitable to a low-power device; iii) it is a workload independent routine that allows controllable experiments to be performed. In future work we plan to perform similar experiments on a wireless sensor node running a real-life application. In this work we are focusing on achieving efficiency bounds for the reconfiguration policy. The proposed benchmark turned out to be suitable for this purpose.

6. EXPERIMENTAL RESULTS

All the experiments have been carried out on a prototype board based on the URLAP processor, which is a RISC ARM-compliant processor with 256 KB of internal SRAM. The board has 512KB of FLASH, 8MB of DRAM, 2 FPGAs and 2 CPLDs, that are exploited by the URLAP to perform run-time reconfiguration of the FPGA.

First of all, we characterized hardware and software energy consumption of the FIR filter at different board clock frequencies (Figure 6). It can be noted that the hardware implementation is more efficient than the software one as expected. We also plotted the hardware energy consumption without static power contribution, which is dependent on the FPGA technology. This plot outlines power consumption due to the execution of the FIR filter in hardware. Power of the whole board has been measured through the same DAQ system used for emulating energy burst.

In Figure 6 the energy is expressed in μJ and the clock frequency in MHz. We also measured the energy consumed during the process of reconfiguration process. We found a peak power of 132mW at frequency of 30 MHz of the reprogrammation of the FPGA, with a reconfiguration time of about 70ms.

In the second part of the experiment we exploit the metric described in Section 5 to evaluate energy efficiency of the proposed approach. In the following figures we will use the following naming conventions: i) *burst size* indicates the duration of the energy burst; ii) *burstiness* is the number of consecutive loop iterations triggered by the same type of event. For instance, burstiness 4:6 means that the software performs 4 consecutive executions related to event 1 and then 6 executions related to event 2.

In these experiments, we implement a simple moving average filter with history of 15 executions. Moreover, we restrict our analysis to the case in which we always have enough energy to perform the reconfiguration process of the FPGA.

In Figure 7 the number of iterations performed in a burst is shown as a function of the energy burst size for an event distribution of 50%. The burst size has been selected to reflect typical range of a solar cell exposed to variable indoor light conditions. This distribution is a worst case for our execution prediction policy, and it is obtained by deterministically programming a sequence (burstiness) of events of the same type.

The "software only" line represents the number of iterations obtained with no reconfiguration policy. It can be seen that the proposed approach provides better results for

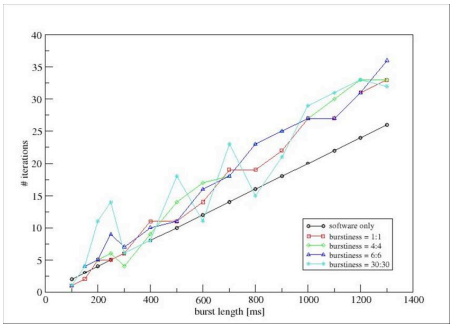


Figure 7: Task executions with balanced event distribution.

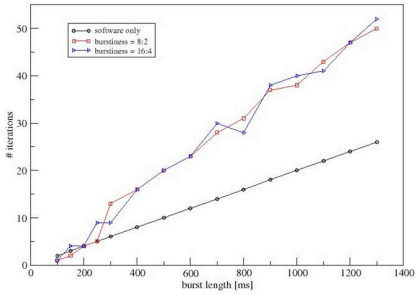


Figure 8: Task executions with unbalanced event distribution.

all the burst size. A considerably larger number of iterations is achieved that compensates for the additional energy and time spent for reconfiguration. For larger burst size, this is not-dependent on the burstiness. For smaller burst sizes the reconfiguration policy is very close to the software one in some points. This is because of the adaptation mismatches between the prediction policy and the event sequence.

If we consider a more unbalanced situation the history-based prediction algorithm is more effective. In Figure 8 we reported results about iterations obtained with an event distribution of 80% for event 1 and 20% for event 2. It can be noted how the proposed policy is more effective also for small energy bursts, because of the reduced impact of adaptation mismatches.

In Figure 9 we also reported the energy per iteration consumed as a function of burst size. This energy takes into account the additional energy spent for the reconfiguration process. The plot has been performed for the worst case of balanced event distribution. In this plot we compared our approach with a "no reconfiguration" case, in which the hardware is programmed with one of the two routines and is never reconfigured. It can be noted that the opportunistic reconfiguration strategy provides lower energy per iteration thus improving energy efficiency in most of the cases. Adaptation mismatches happen only in case of very unrealistic burstiness conditions, with a sequence of 30 events of the same type.

Finally, it must be noted that these results have been obtained with a simple execution prediction policy. We expext an improvement using more complex prediction algorithms exploiting a more predictable event.

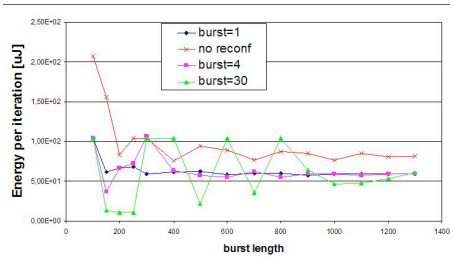


Figure 9: Energy per iteration with balanced event distribution.

7. CONCLUSION

In this paper we present an opportunistic reconfiguration strategy, aimed at exploiting the unused energy due to the unpredictability of the environmental energy. It exploits energy slacks to dynamically reconfigure ultra-low power devices such as sensor nodes to increase their energy efficiency. We first performed a statistical analysis, then we tested the reconfiguration strategy through a reconfigurable prototype board.

8. REFERENCES

[1] L. Benini, M. Poncino, "Ambient intelligence: a computational platform perspective," *in Ambient intelligence: impact on embedded system design*, Kluwer Academic Publishers, pp. 31–50, 2003,

[2] ATMEL Corporation, "FPSLIC (AVR with FPGA) from Atmel ,". www.atmel.com/products/FPSLIC/.

[3] A. Kansal, D. Potter and M.B. Srivastava, "Performance Aware Tasking for Environmentally Powered Sensor Networks," *ACM Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2004.

[4] J. Khan, R. Vemuri, "Battery-Efficient Task Execution on Reconfigurable Computing Platforms with Multiple Processing Units," *IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[5] Theo Kluter, "URLAP Processor," *EPFL LAP Technical Report*, 2004.

[6] P. J. M. Havinga, L.T. Smit, G. J. M. Smit, M. Bos, P.M. Heysters, "Dynamic Reconfiguration in Mobile Systems,". *Heterogeneous Computing Workshop*, 2001.

[7] C. Rusu, R. Melhem, D. Mosse, "Multi-version Scheduling in Rechargeable Energy-aware Real-Time Systems,". *Journal of Embedded Computing*, pages 2–11, June 2004.

[8] N. Ngoc, G. Lafruit, J.Y. Mignolet, S. Vernalde, G.Deconink, R. Lauwereins, "A Framework for Mapping Scalable Networked Multimedia Applications on Run-Time Reconfigurable Platforms,". *Proceedings of ICME*, pages 469–472, June 2003.

[9] D. Panigrahi, C. N. Taylor, S. Dey, "A Hardware/Software Reconfigurable Architecture for Adaptive Wireless Image Communication,". *Proceedings of ASP-DAC/VLSI Design*, pages 553–559, June 2002.

[10] J.A. Paradiso and T. Starner, "Energy Scavenging for Mobile and Wireless Electronics," IEEE Pervasive Computing, vol. 4, no. 1, pp. 18–27, 2005.

[11] S. Roundy, E. Leland, J. Baker, E. Carleton, E. Reilly, E. Lai, B. Otis, J. Rabaey, P. Wright, V. Sundararajan, "Improving power Output for Vibration-Based Energy Scavengers," IEEE Pervasive Computing, 2005.

[12] Gerard J. M. Smit, P. J. M. Havinga, L.T. Smit, P.M. Heysters, M. A. J. Rosien, "Dynamic Reconfiguration in Mobile Systems,". *Lecture Notes In Computer Science*, Vol. 2438, pages 171–181, 2002.

[13] Paul M. Heysters, Gerard J. M. Smit, Egbert Molenkamp, "Energy-Efficiency of the MONTIUM Reconfigurable Tile Processor,". *Proceedings of ERSA*, pages 29–32, 2004.

[14] Maxwell technologies, "Ultracapacitors," www.maxwell.com/ultracapacitors/, 2005.