Reconfiguration Strategies for Environmentally Powered Devices: Theoretical Analysis and Experimental Validation*

Alex E. Şuşu[1], Michele Magno[2], Andrea Acquaviva[1,3], David Atienza[1,4], and Giovanni De Micheli[1,3]

[1] LSI/EPFL, Lausanne, Switzerland
{alex.susu,david.atienza,giovanni.demicheli}@epfl.ch
[2] DEIS/University of Bologna, Bologna, Italy
mmagno@deis.unibo.it
[3] STI/University of Urbino, Italy
acquaviva@urbino.it
[4] DACYA/Complutense University, Madrid, Spain
datienza@dacya.ucm.es

Abstract. Environmental energy is becoming a feasible alternative to traditional energy sources for ultra low-power devices such as sensor nodes. These devices can run reactive applications that adapt their control flow depending on the sensed data. In order to reduce the energy consumption of the platform and also to meet the timing constraints imposed by the application, we propose to dynamically reconfigure the system through the use of Field Programmable Gate Array (FPGA) fabric such that it executes more efficiently the tasks of the application.

In this paper we present a new approach that enables the designer to efficiently explore different reconfiguration strategies for environmentally powered systems. For this we define a stochastic model of a harvesting video sensor node that captures the behavior of the node and of its environment. We use this approach to investigate the impact of different reconfiguration strategies for a video surveillance node on metrics of interest, such as the expected lifetime or downtime of the system.

Then, we create a hardware implementation of an energy-aware reconfiguration manager on top of a custom multi-FPGA board.

Our results show that the systems improve their processing capabilities if suitable reconfiguration strategies are defined for their respective configuration environments.

Keywords: Wireless Sensor Nodes, FPGA, energy harvesting, probabilistic model checking, Markov chains.

1 Introduction

The existing and emerging energy harvesting technologies become feasible solutions to power up small electronic devices [21,24]. Using harvested energy has

* This article builds upon a paper prepared for the third conference on Computing Frontiers, 2006.

pluses and minuses: the energy from the environment is infinite, thus providing opportunity to increase the autonomy of the system, yet it is unpredictable. To cope with the unpredictability of the harvested energy we use power adaptation circuitry and energy storage elements such as rechargeable batteries and super-capacitors, which regulate the supplied power level corresponding to the demand. However, this scheme might not constantly offer the required power because it can happen that there is not enough energy coming from the environment and, also, not enough energy in the storage elements.

In order to meet timing constraints and, subsequently, in order to reduce the energy consumption of the platform, we propose to use low power FPGA logic that can execute more rapidly code that can be parallelized. More exactly, we use a sensor node which includes a small on-chip FPGA [1] that can implement signal processing routines together with a low-power microcontroller. Because of the limited capacity of the FPGA we can dynamically reconfigure it in order to load the most energy efficient task sets depending on the sensor context, for example. However, reconfiguration has a cost in terms of energy and time, so that a suitable strategy must be designed to determine if and when it is worth to perform system reconfiguration. Clearly, the reconfiguration process itself has an energy penalty, but our goal is to amortize this cost due to the future savings offered by the reconfiguration.

Power management policies can be considered another form of system reconfiguration w.r.t. the application context. The focus of generic power management policies for environmentally powered systems is different from the one of battery powered devices: while in the latter case we search to maximize the lifetime of the system, in the former case a good design objective is to increase the availability of the device for long periods of time.

Also, situations might arise where the harvester generates more energy than required by the device after the battery is already full. In this case, the additional energy, which we call *energy slack*, is wasted. This situation is of practical relevance with harvesters such as solar cells, where during long periods with intense external light conditions it becomes difficult to store all the available energy.

The work we present brings two contributions. First, we model realistic reconfigurable systems, using PRISM [17], a probabilistic model checking tool. The model is used to formally assess the impact of reconfiguration strategies on metrics of interest such as the lifetime and the availability of the system. We express quantitatively how various proposed policies improve these metrics.

Second, we describe the implementation of a reconfiguration strategy on a prototype board with FPGA, which uses hardware and software versions of the application tasks. The idea behind this policy is to use the otherwise wasted harvested energy (in case the battery is already full) to reconfigure the system, thus improving the energy efficiency in the future. In order to match the relatively high power requirements of the prototype board when compared to the power generated by practical energy harvesters and, in order to perform experiments in a controllable way, we emulate the harvester with a programmable power generator, which generates energy under the form of bursts of constant power and variable length.

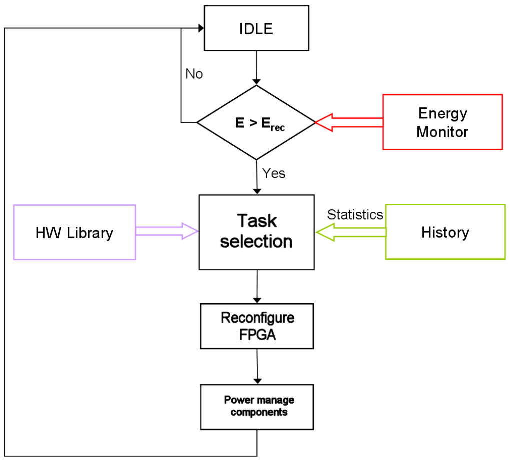

Fig. 1. Generic reconfiguration policy

2 Background Work

Advances in microelectronic technology have been recently exploited to build low-cost and low-power miniaturized sensor nodes that can collaborate together in sensing and relaying the information, building Wireless Sensor Networks (WSN). Such sensor nodes, integrated with harvesting devices, can exploit the environmental energy in order to increase their autonomy [21]. The energy level provided by the harvesting devices is determined by the technology in use. For instance, solar cells provide between $100mW/cm^2$ when directed toward bright sun and $100W/cm^2$ in an illuminated office, while vibrational microgenerators provide $4W/cm^3$ for human motion [21].

A sensor node consists of the following hardware components [12,8,23]: a microprocessor, data storage, sensors, a data transceiver, and an energy source. The processing performance can be increased with a dedicated Digital Signal Processing (DSP) unit, an Application-Specific Integrated Circuit (ASIC) or reconfigurable hardware (e.g., FPGA) that implements computationally demanding or performance constrained tasks.

Currently, most examples of sensor node architectures are microprocessor-based. On the one hand, this provides flexibility for adaptation, since we can implement multi-modal programs on such nodes, or we can even dynamically upload new code on them in order to adapt to a new context.However, a processor executing software is far less efficient in performance, energy consumption and, even in manufacturing cost than an ASIC. On the other hand, ASICs do not have the flexibility for node-level adaptation. Thus, the use of field-programmable hardware, in particular FPGAs, in sensor nodes is a very recent area of research [18,12]. Reconfigurable logic can provide node flexibility with significantly greater energy efficiency than software-only solutions if low-power FPGA platforms are used in combination with suitable reconfiguration strategies. However, because of power and size constraints, suitable FPGAs are limited in the functionality they can implement. This limitation, coupled with the heterogeneous application context makes impractical the trivial solution of mapping simultaneously all the possible tasks on the FPGA. A solution to this problem is to dynamically load in the FPGA the required tasks exactly before being used.

Within the domain of WSN, video sensor nodes hold strong interest for military, security, robotics, and, recently, also, consumer applications [16,19]. However, due to the high computational requirements and energy costs of video coders for processors/microcontrollers, mixed hybrid architectures (i.e., microprocessor with DSP/ASIC/FPGA) with efficient partitioning algorithms have been suggested as a more suitable option to achieve sufficient performance with low energy consumption [22].

A similar hybrid approach is taken in the Low power Energy Aware Project (LEAP) [20]. The LEAP platform is a sensor node that can support intensive processing tasks. The LEAP architecture is composed of two modules: a general purpose computing module used for event-driven computationally intensive processing and a preprocessor module dedicated to low power sensing and energy accounting. The architecture integrates fine-grained energy dissipation monitoring and sophisticated power control scheduling for all subsystems including sensor subsystems. The LEAP architecture enables complex energy-aware algorithm design by providing a simple interface to control numerous platform and sensor power modes and report detailed energy usage information.

Lately, several processing power optimization techniques have been proposed for WSN. In the case of nodes with high duty cycle, one can tune the clock frequency and the supply voltage of the processing units depending on the workload [25]. Also, data aggregation strategies between multiple sensor nodes can reduce the redundant information transmitted and the power used in the network [6]. In addition, power-aware topology control algorithms have been proposed [7]. Another possibility (which is also present in LEAP) is to suspend the microcontroller, the coprocessors or the radio transceiver according to the communication [11] or computation features of the application. The first three methods are complementary to our reconfigurable approach of sensor nodes; thus, they could be used in combination with the reconfiguration strategies we suggest in this paper.

3 Analysis of Environmentally Powered Reconfigurable Systems

In this section we focus on the modeling of nodes, with emphasis on the energy generated by the harvesting device and on the reconfiguration policy. Our goal

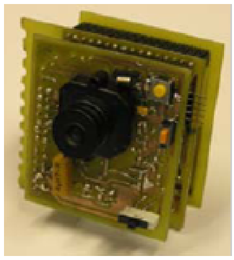

Fig. 2. The MicrelEye video surveillance node

is to analyze the model in order to express quantitatively the effectiveness of reconfiguration strategies in the context of environmentally powered devices.

To model the harvested energy, we use traces obtained from a real solar cell [3,9], which consist in the enumeration of the intensity values of the generated current over a period of time. The variation of the energy is periodic w.r.t. the day cycle and is caused by clouds, terrain obstacles from the sun and, in the longer term, by season changes. In Fig. 3 we present an arbitrary trace over 1 day. The voltage of the harvester is almost constant at a value of 5 V.

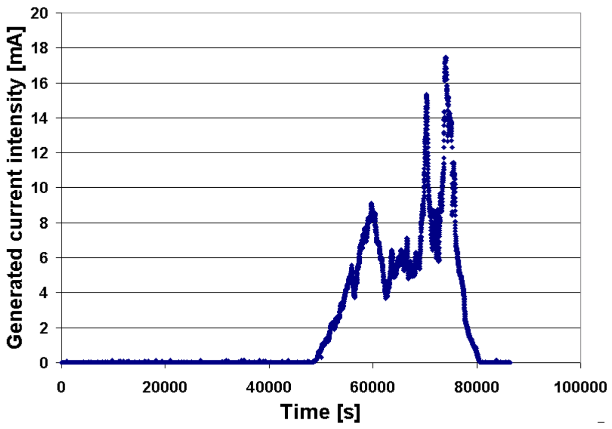

Fig. 3. Trace of the intensity of the generated current by a 5.5x15 cm^2 solar panel, on the roof of one of the buildings at EPFL starting from Nov 14, 2005 18:28, for 24 hours

The unpredictability of the environmental energy source can be modeled stochastically. Using, for example, the solar panel trace presented in Fig. 3, we can build a Discrete Time Markov Chain (DTMC) model for the harvested energy. To each state of the DTMC we associate an interval of energy levels generated in that specific state by the harvester (this is similar to the Power State Machine concept, used in [4] to model power manageable components). To assign probabilities on the transitions originating in a state of the DTMC, we go over the solar panel trace and count the occurrences of each transition from that state (by transition in the trace we understand the jump from one energy level at a moment, to the level corresponding for the next time instance). Then, we normalize these frequencies in order to have the sum of probabilities on the transitions from that state equal to 1.

The choice of the number of states of the DTMC is a compromise between the accuracy of the DTMC abstraction w.r.t. the real harvester and the tractability of the analysis of the model, which we introduce in the following paragraphs. The statistical model can be considered representative for the days of the winter season in the geographic region of the experiments. Using the same technique, statistical models for different periods of the year or locations can be generated. In Fig. 4 we show the DTMC built from the trace in Fig. 3.

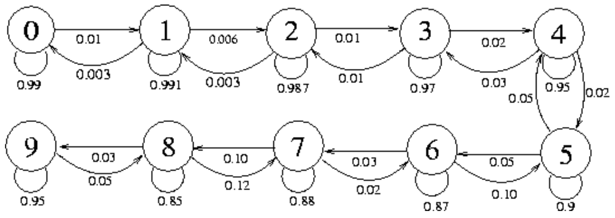

Fig. 4. Discrete Time Markov Chain built from the trace in Figure 3

In order to establish formally various properties like the expected system lifetime, the achievable activity duty cycle or the duration of the blackout periods, by automated analytical means, without performing expensive simulations, we use probabilistic model checking. Compared to simulation, which focuses on one admissible execution of a system, model checking explores all the possible behaviors of the modeled system [14]. One can argue that using a mean value approximation for the energy received in the time unit by the energy harvester and the energy consumed by the platform, we can easily compute analytically, for example, the average lifetime of the system. While this is true, the average approximation is no longer accurate, and therefore not suitable when we have, for example, a policy that changes the QoS of the application based on the energy level coming from the environment or remaining in the battery.

The probabilistic model is defined as the parallel composition of the modules of the system, such as the harvester, the radio channel and the battery. Each module has a certain number of states, among which transitions are defined. The transitions can be either probabilistic, building, for example, DTMCs, or deterministic.

The tool we use, the probabilistic model checker PRISM [17], is able to infer properties of the stochastic model through exhaustive exploration, many of them being non-trivial and amenable only by computer analysis. These properties are relevant for the hardware and software designers in order to adjust the sizes of the components, such as the harvester and the battery, as well as designing the software layers. For instance, if we employ backward recovery in order to cope with the blackouts [10] we can tune the checkpointing interval s.t. the average lost computation is minimized by using the average lifetime of the system determined with this method.

In order to instruct PRISM to perform the analysis of a desired property, a query must be built using the Probabilistic Computational Tree Logic (PCTL) temporal logic [5,13]. For example, we can ask the tool to compute the probability that the system runs out of power for a given battery size and initial level of environmental energy. In this case, the tool computes the probability that the battery module reaches the zero energy state. In the following section, we devise a model of a realistic reconfigurable system powered by a solar harvester and a rechargeable battery, the MicrelEye video sensor node used for security applications.

Concerning the reconfiguration policy of the system, we present the flowchart of a generic strategy in Fig. 1. The policy stores runtime statistics regarding the inputs of the application that decide its control flow (e.g., depending if the captured image contains a person or not, the system executes different sets of tasks). Based on these statistics, the policy decides which is the best task candidate for reconfiguration among the available ones and we load it in the FPGA, if it is not already there. The bitstreams of the task candidates are stored in a special module of the reconfiguration manager, namely, the hardware block library, which can be stored in the Flash memory of the available FPGA or in any other storage device directly accessible by the reconfigurable manager. The strategy is also responsible to power manage the components of the system.

The additional energy spent for reconfiguration could prevent the complete execution of a task that would be otherwise finalized if no reconfiguration is performed. But, our assumption is that the reconfiguration cost is amortized in the future by several executions of the more efficient reconfigured task. Moreover, the reconfiguration energy can be provided directly by the harvester at no cost when the battery is full and the power generated is bigger than the consumption.

To validate the proposed reconfiguration policy, we present in Sect. 4.2 a manager that implements such strategy in a proof-of-concept prototype system.

4 Case Studies and Experimental Measurements

In Sect. 4.1 we illustrate the use of PRISM to explore the possible reconfiguration opportunities of the model introduced in Sect. 3 for various hardware-software designs of the MicrelEye video sensor node. Then, in Sect. 4.2, we present the implementation of an energy-aware reconfiguration manager on top of a custom multi-FPGA board and experimental results performed with this platform.

4.1 Evaluation of Reconfiguration Policies Using PRISM

The MicrelEye Node. The MicrelEye (see Fig. 2) is equipped with an Omnivision 7640 video sensor, an ATMEL FPSLIC reconfigurable platform featuring an AVR microcontroller and a 40,000 gates FPGA. FPSLIC is one of the lowest power consuming reconfigurable boards on the market. Its latest version, FP-SLIC II, can put into low-power mode both the microcontroller and the FPGA.

For each captured image, the node performs a set of processing tasks, in order to determine if a human body is present in the viewer of the camera. In the case of a positive outcome, the original image is sent to a basestation for further processing (e.g., face recognition, using a database of features for comparisons). The FPGA is used to perform most of the image processing tasks.

The application running on the platform starts a normal execution cycle by capturing with the camera an image of 320*240 pixels; we call this the Camera Acquisition (CA) task. We apply on the resulting frame a Background Subtraction (BS) function, which removes the background of the image; this is accomplished by using an earlier-captured background frame as a reference. At this moment we can compare the original image (the output of task CA) and the result of the task BS. If the two images are very different, then it means that the original image is mostly background. In this case, we do not perform any more processing of the image, since the image is considered not to be interesting. If the image resulted from the task BS is not discarded, we continue with the Search Algorithm (SA) phase, which returns the position within the 320*240 frame of a 32*16 window that potentially contains a human body, positioned approximately 5 meters away from the camera. On this window of 32*16 pixels we apply the Feature Extraction (FE) function, which performs the average of the pixels for each row and column of the window and stores these values in a vector of size 32+16. This vector is handled by the Support Vector Machine (SVM) task that determines if the input vector corresponds to a human body or not. At the end of the SVM task, we know with a good degree of confidence if the captured image contains a human body.

The CA, BS and SA tasks are very computing intensive. To meet deadlines, since a microcontroller does not provide short execution times for the aforementioned tasks, we have to execute these tasks on the FPGA. Therefore, tasks CA, BS, SA and FE are only executed on the FPGA. We execute the SVM task on the AVR microcontroller, since it can perform fast multiplications, and the SVM task is multiplication intensive.

The characteristics of the tasks of the detection application are given in Table 1. For the energy consumption values, we assume that we suspend the microcontroller or the FPGA whenever they are not used.

We model in PRISM the MicrelEye node as the parallel composition of the following modules: i) the harvesting device (*SolarHarvester*); ii) a rechargeable battery (*Battery*); iii) the consumer part of the video node (*MicrelEyeNode*) that-models the energy intake of the camera, microcontroller and FPGA; iv) the "view" of the camera (*CameraView*) that determines if the camera captures a frame only with background information or not. To coordinate the simultaneous execution of the modules we define the *Clock* module. All these modules are synchronized on the *tick* action (the term action comes from the terminology used by PRISM) generated by Clock. The structure of the system can be seen in Fig. 5.

Table 1. Characterization of the Tasks

Task	Execution time [ms]	Energy consumed [mJ]
CA	40	2.64
BS	54.5	12.58
BS2	27.25	6.29
SA	19.2	4.43
FE	0.27	0.0623
SVM	89	33.1881
FPGA Reconfiguration	22	3.63

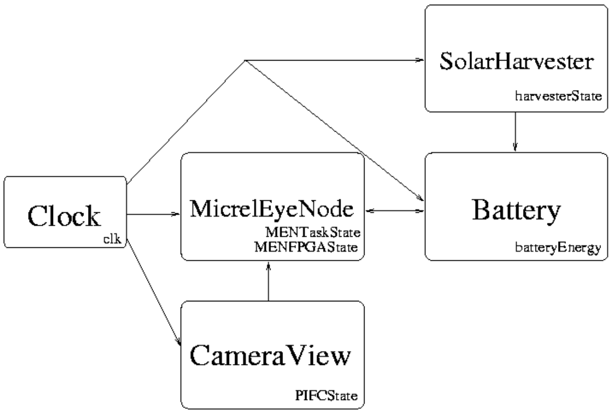

Fig. 5. Block diagram of the system

Battery. The rechargeable battery is modeled as a set of states that represent the battery energy levels with deterministic transitions defined by the energy consumption and generation levels in the current state.

CameraView. We model the view of the camera using two states (the boolean variable PIFCState specifies what is the current state) and probabilistic transitions between them. The transitions represent basically the probability of having a person (or something that is recognized as a person) in the frame captured by the camera.

Clock. The clock module does not correspond to a physical component of the system. The module has two states that generate the actions tick and tock. As previously explained, Clock is used to trigger the activity of the other components through the tick action.

MicrelEyeNode. The node is modeled using the MENFPGAState boolean variable for the two configurations of the FPGA and the MENTaskState variable with 6 states that keeps track which task is the node currently executing. The

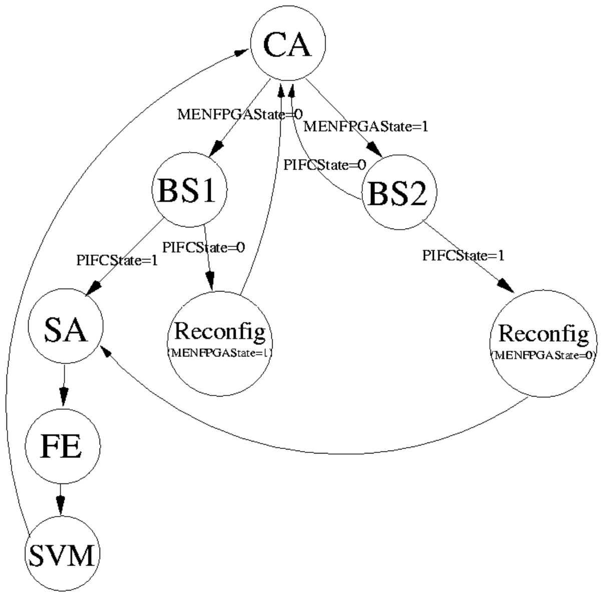

Fig. 6. The task graph of the software application implemented on the MicrelEye node for Policy3

transitions between the states are deterministic and take into account the state of the CameraView and the level of energy in the battery.

The actual specification of the system is done in the Reactive Modules language [2], but we do not present it due to space limitation. We define each module in this language by specifying: i) the variables of the module and their initial values and ii) the behavior of each module by a set of commands, where each describes a guard (a predicate that needs to be true to execute the statement) and one deterministic variable update (transition) or more probabilistic updates with specified probabilities.

The behavior of the model is the following. Initially the battery is full. In every time step we add to the battery energy level (variable *batteryEnergy*) the contribution from the harvester and subtract the energy consumed by the node, which is computed based on the state of the node (MENTaskState and MENFPGAState). When the battery is full, the system uses directly the energy from the harvester. If this energy is not enough for the given time step, it consumes energy from the battery as well. If the energy from the harvester is bigger than what the platform requires, then the surplus is wasted. If the system runs out of energy (i.e., the system does not have the required energy for executing the current application cycle) it constantly checks the battery level and restarts only when there is enough energy in the battery to be able to execute the initial application cycle.

The Strategies. We consider several operation policies, which we compare quantitatively afterwards. The first two of them disallow runtime reconfiguration, while the following two are variations of dynamic reconfiguration policies.

- *Policy1 (static, FPGA always active)*, which assumes that the system is not able to dynamically reconfigure the FPGA, so it has tasks CA, BS, SA and FE statically mapped on the FPGA. Also, we assume that the FPGA cannot be put into low power mode. On the other hand, the AVR microcontroller is put into sleep mode when it is not used.
- *Policy2 (static, low power FPGA)*: Once again we assume that the configuration of the FPGA is assigned at the beginning and cannot be changed during runtime. We model the use of the FPSLIC II board, and, therefore, the FPGA can be suspended as well, besides the microcontroller.
- *Policy3 (dynamic, low power FPGA)*: This policy takes advantage of the possibility to dynamically reconfigure the FPGA and to suspend the FPGA and the AVR microcontroller, when no longer used. It means that the microcontroller is turned on only during the execution of the SVM task, while the FPGA is suspended only for the tasks CA and SVM.

  The dynamic reconfiguration allows the execution of different versions of a task. For example, since the task BS is the most computing intensive, we can parallelize it by creating two instances of sub-tasks BS that work on half of the image each. Thus, this parallelized version of the task BS (which we call BS2) has almost half the execution time of the original task BS, but this comes at the expense of occupying almost double space on the FPGA. This makes task BS2 to have almost half the energy consumption w.r.t. the original task BS, if we consider that the power of the FPGA is the same for the two different mapping scenarios (tasks CA, BS, SA and FE, versus task BS2). However, the parallelized version of the task BS occupies a big part of the FPGA, and therefore it does not leave space for the SA or the FE tasks.

  Having two different FPGA mapping scenarios, we can make use of one or the other at the right moment by employing dynamic reconfiguration. In the case we detect that the image is not interesting (immediately after running the task BS) we assume that the following images will not be interesting either with a high probability, and, therefore, we execute task BS2 from now on, as long as possible. For doing this, we need to dynamically reconfigure the FPGA with the task BS2, if this task is not already mapped on the FPGA, such that for the future frames we benefit of the lower energy consumption. In case we have mapped task BS2 on the FPGA, which occupies the entire reconfigurable logic estate, and we receive a frame that is declared by BS2 as being valuable, then we are forced to reconfigure the FPGA in order to load the tasks BS, SA and FE on it. This reconfiguration policy is depicted in Fig. 6, where some of the transitions are annotated with predicates which decide if the transition is taken or not.
- *Policy4 (dynamic, low power FPGA, harvester used as sensor)*: This policy enhances Policy3 by sensing the light conditions with the solar panel. If we detect through the solar panel that there is no light in the environment

(i.e., the power from the solar panel is almost zero), then we power down the node. The node restarts when the solar panel captures light and, of course, we have enough energy in the battery to sustain the computations.

We consider Policy2 the baseline non-reconfigurable strategy for our experiments.

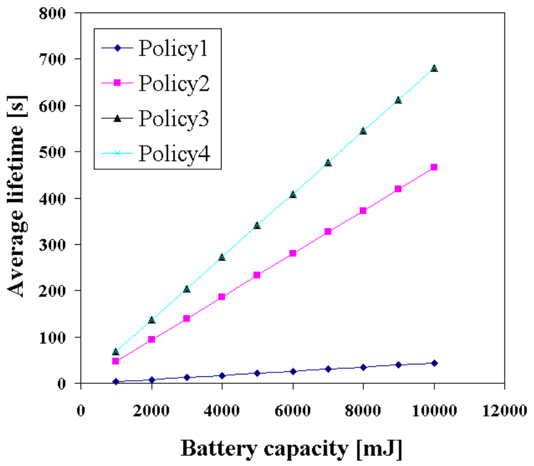

Fig. 7. Variation of the average lifetime for Policy1, Policy2, Policy3 and Policy4 w.r.t. the capacity of the battery. The node is turned on invariably at 11AM.

Exploration Results of the MicrelEye Node. Before presenting the results, we define precisely the terminology we use. By average lifetime we understand the expected period of time from the moment we start the system until the moment it runs out of power (note that the system can sleep during its lifetime). By the downtime of the system over a given period we understand the expected sum of periods of time over the given time frame in which the node cannot run because it does not have enough energy to proceed with the execution. The downtime of the system is the complementary of the uptime for the same period, which can be easily converted in availability. It is important to mention that the average downtime and lifetime are not perfectly complementary: the average downtime is the sum of periods of blackout for the given period of time, while the lifetime is just until the first blackout.

For the experiments we run, we assume a probability of 1% of having a person in front of the camera. In Fig. 7 we present the expected lifetime of the system running each of the defined policies, as a function of the initial (and maximum) capacity of the battery. Clearly, the lifetime depends on the capacity of the battery in a linear way, since we are outside the energy neutral operation mode (i.e., the power consumption, which is dependent on the duty cycle of the application, is higher than the generated power, on average). The initial battery capacity only affects the initial behavior of the system. We can also see that here Policy3 and Policy4 are equally efficient. This is so because the node is supposed to start running at 11AM and, since the power consumption is bigger than the power generated by the harvester, the system runs out of power in the order of minutes. Thus, the node is not able to get into a period of darkness and, because of this, the two policies behave in a similar way.

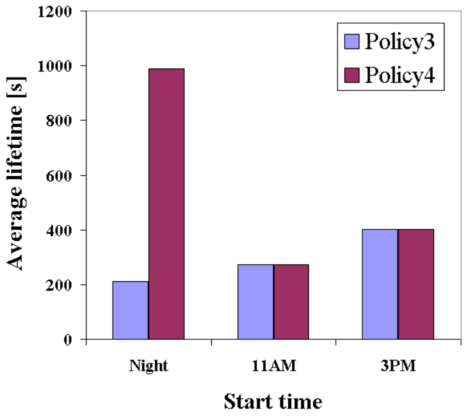

Fig. 8. Variation of the average lifetime for Policy4 w.r.t. the moment of the day when we turn on the system. The capacity of the battery is 4J.

To test Policy3 and Policy4 in conditions that would differentiate them, we consider we turn on the node at different hours, in the same day. We show in Fig. 8 the variation in expected lifetime of the system for the two policies, when turning on the node at various moments of the day. We can see that only in the case we turn on the node during the night, the policies are very different: the lifetime of the node with Policy4 increases 4.7 times when compared to Policy3. Otherwise, if we turn the node on during the day time, it does not survive until the night comes, because it runs out of power relatively fast.

In Fig. 9 we present the expected downtime of the system for a period of one hour, for each of the defined policies. We notice an increase in availability of 37% for Policy3 w.r.t. Policy2, the baseline non-reconfigurable strategy.

We notice that the dynamic reconfiguration policies reduce the expected downtime, leading to larger periods of activity of the system with a given environmental energy. This leads to an increased throughput, i.e. number of frames processed per second.

We also compute the average wasted energy for one hour (we waste energy when the battery is full and the generated power is bigger than the one consumed) for Policy1: we obtain a value of zero, which is easy to understand since Policy1 is very power hungry even w.r.t. the maximum power generated by the harvester.

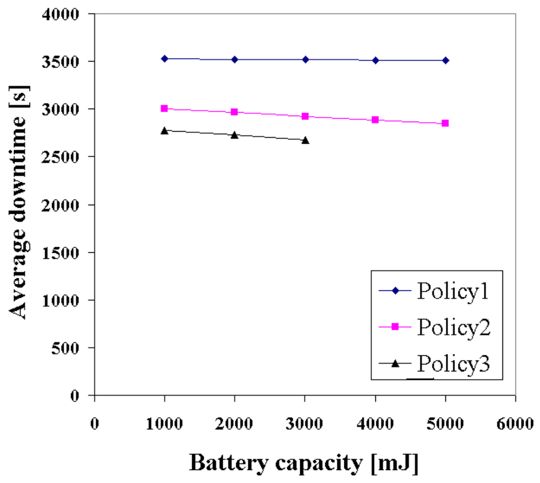

Fig. 9. Variation of the average downtime for a period of one hour for Policy1, Policy2, Policy3, Policy4 w.r.t. the capacity of the battery

We conclude that the reconfiguration strategies, Policy3 and Policy4, can improve the lifetime of the video node at least with 40% when compared to the baseline for non-reconfigurable strategies, Policy2, and the availability up to 37%.

4.2 Proof-of-Concept of the Reconfiguration Manager

For validation purposes we build a hardware implementation of an energy-aware reconfiguration manager on top of a custom multi-FPGA board. The considered platform is equipped with two FGPAs, where one contains a manager that drives the reconfiguration of the other FPGA, which contains the processing logic.

In this set of experiments, we consider a reactive application, which consists of two different tasks. Both tasks implement, either in software or in the FPGA, slightly different versions of a fourth order Finite Impulse Response filter (FIR), and are named accordingly FIR1 and FIR2. The software version is used when a task needs to be executed and its hardware counterpart is not loaded yet in the FPGA, in the idea of completing the task as soon as possible. The choice of the FIR routine is 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 for a low-power device; (iii) it is a workload independent routine, which allows controllable experiments to be performed. In our application, we select for execution one of the tasks FIR1 or FIR2 based on the particular value of a sensor reading.

Clearly, this application is simpler than the one presented in Sect. 4.1. Also, an important difference is the fact that we consider now the environmental energy to come in bursts of different lengths, but of constant power, assumption which holds better for vibrational or indoors photovoltaic harvesters, for example.

In the rest of this section we describe the reconfiguration policy running on the prototype board, then, the implementation of the board, and, in the end, we discuss the results of the measurements that assess the effectiveness of our strategy.

Reconfiguration Policy. The strategy is implemented as code running on the processor of the prototype board. This simple policy, depicted in Fig. 10, allows us to study inherent properties of the reconfiguration strategy in a repeatable and controllable way.

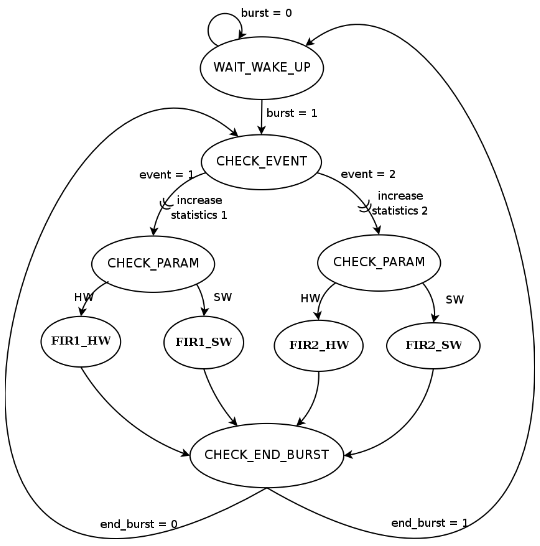

Fig. 10. The event-driven application model

The execution of a task is triggered by the arrival of an energy burst ($burst = 1$). We create fictive sensor readings by using a pseudo-random generator. Depending on the value of the reading, one of two possible events is generated ($event = 1,\ or\ event = 2$). Each event triggers the execution of one of the two FIR tasks. Along with this, the execution statistics are incremented for the corresponding task. The policy has to select between the hardware and software version of the selected task. This is done by checking a shared memory location ($CHECK\_PARAM$) written by the reconfiguration manager, which stores which routine is actually loaded in the FPGA. The system continues running as long as there is energy available.

For the given implementation, the number of loops executed during an energy burst of constant length becomes a metric for the energy efficiency of the reconfiguration strategy. Obviously, the lower the amount of energy consumed by each loop, the higher the number of loops that get executed within a single energy burst.

Reconfiguration Manager Implementation and Experimental Results. The custom design board we use to run the reconfiguration policy is equipped with the URLAP processor [15] (a low power ARM-based processor with 256 KB of internal SRAM), 8MB of external DRAM, 512KB of Flash memory and two FPGAs. One of the two on-board FPGAs is used for the execution of the FIR tasks, while the other one is used as a reconfiguration manager. The overall system architecture is shown in Fig. 11.

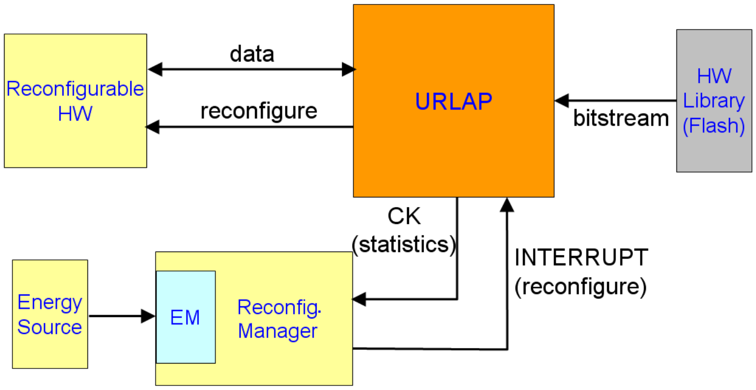

Fig. 11. Reconfigurable system architecture

In this design, the FPGAs can be configured to be either memory-mapped or to have a coprocessor interface to the URLAP processor. We use the first alternative in our experiments to build the interface to the reconfiguration manager. The FPGAs and the URLAP communicate through the shared memory view and interrupt lines. In this case, the reconfiguration of the FPGA can only be done at run-time by the URLAP, by using two additional CPLDs.

Since the board is not optimized for being powered by a real harvester, we use a burst emulation system based on the 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 paper. The hardware reconfiguration manager is directly connected to the energy source in order to detect the power of the harvester and the status of the battery, thus being able to detect the energy slack that might be normally wasted. This component that measures the energy and power levels is indicated in Fig. 11 as the Energy Monitor (EM).

Since we do not know in advance the size of the energy burst, we start the reconfiguration process as soon as the power level of the burst is larger than the reconfiguration power. Moreover, we restrict our analysis to the case where we always have enough energy to perform the reconfiguration process of the FPGA.

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 statistics collection process. We collect statistics by inserting code checkpoints, which write information about the last task execution in a dedicated shared memory location. The reconfiguration manager reads then this information and uses it to update the execution counters of the tasks and other related variables. In order to decide which task to load in the FPGA, we implement a simple moving average filter, which selects the most frequent task from the 15 previous task executions.

We perform now a set of experiments in which we search to obtain efficiency bounds for the proposed reconfiguration policy. In the following paragraphs we use the following terminology: i) the *burst size* indicates the duration of an energy burst; ii) the *event distribution* is the ratio between the number of consecutive events of type 1 and the number of consecutive events of type 2. For instance, an event distribution of 4:6 means that we have four consecutive events of type 1, followed by six consecutive events of type 2.

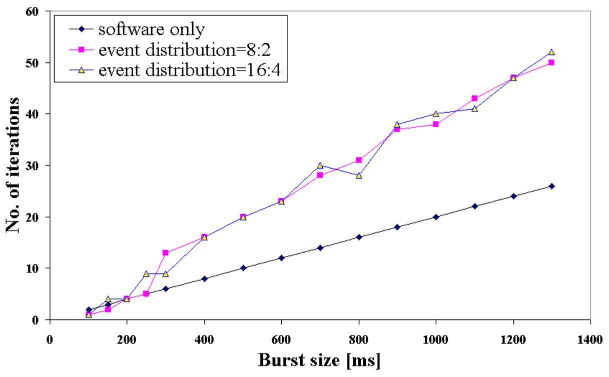

Fig. 12. Number of task executions for various unbalanced event distributions

We evaluate the effectiveness of the dynamic reconfiguration policy for different energy burst sizes. We also measure the energy consumed during the process of reconfiguration process. We find a peak power of 132mW at a frequency of 30 MHz of the reprogrammation of the FPGA, with a reconfiguration time of about 70ms.

In Fig. 12 we report the number of iterations performed per energy burst, for various unbalanced event distributions. The *software only* line represents the number of iterations obtained when we do not employ any reconfiguration policy, for arbitrary event distributions, since both tasks have identical characteristics in software. We can see that the proposed policy is more effective for larger energy bursts, because of the good adaptability of the prediction policy for the actual input event sequence.

In Fig. 13 we present the energy per iteration consumed as a function of the burst size. This energy takes into account the additional energy spent for the reconfiguration process. The plot is performed using a balanced event distribution, for which a history-based prediction algorithm is more effective. In this plot we compare our approach with a *no reconfiguration* case, in which the FPGA is statically programmed with one of the two routines and is never reconfigured.

We conclude that the system using the proposed reconfiguration strategy consumes less energy per iteration than the system using only the software implementation of the tasks, in most of the cases. Since the reconfiguration manager uses a simple prediction policy, with the help of more complex prediction algorithms or of policies that exploit more predictable external events, we should improve the effectiveness of the proposed strategies.

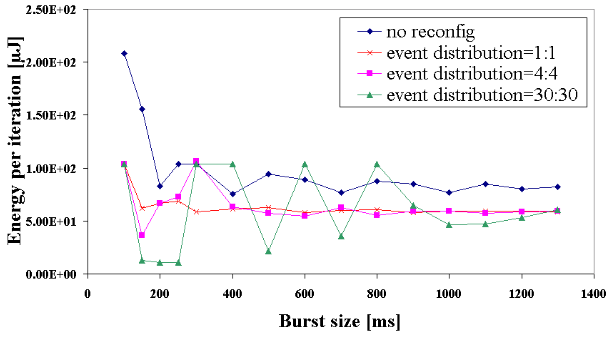

Fig. 13. Energy per iteration with balanced event distribution

5 Conclusions

In this paper we have presented modeling and implementation approaches that use the reconfigurable hardware existing in the latest and forthcoming sensor node architectures. Our goal is to model the energy coming from the environment and the possibilities of reconfiguration that target to maximize the energy efficiency. We have first performed explorations of stochastic models for environmental energy and sensor node architectures and we have shown improvements in the lifetime and the availability of the system of 40% and 37%, respectively, by employing several proposed strategies. Then, we have shown an implementation of the proposed reconfiguration manager architecture using a prototype board with microcontroller and reconfigurable hardware. Our energy efficiency measurements have demonstrated the effectiveness of the employed reconfiguration policies.

As future work we plan to extend our modeling methodology to be able to infer the expected downtime or the energy wasted for more interesting periods of time (e.g., months, years). Also, we plan to analyze additional real-life working environments (e.g., main doors of buildings, bridges, etc.) and types of harvesting devices.

Acknowledgments

We would like to thank Matthias Fruth, Dave Parker, Rajesh Gupta and Andrey Rybalchenko for the insightful discussions regarding the model checking of our prototype system. Also, we would like to thank Henri Dubois-Ferrière and Guillermo Barrenetxea for helping us obtain the traces of the solar panel. This work is partially supported by the Swiss NFS Research Grant 20021-109450/1 and the Spanish Government Research Grant TIN2005-5619.

References

1. FPSLIC (AVR with FPGA) from Atmel, ATMEL Corporation - www.atmel.com/products/FPSLIC/.
2. Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Form. Methods Syst. Des.*, 15(1):7–48, 1999.
3. Guillermo Barrenetxea, Henri Dubois-Ferriere, Roger Meier, and John Selker. A weather station for SensorScope. In *Demo Session, In Information Processing in Sensor Networks (IPSN 2006)*, 2006.
4. Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, June 2000.
5. Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 15, 1995.
6. Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wirel. Netw.*, 8(5):481–494, 2002.
7. Xiuzhen Cheng, Bhagirath Narahari, Rahul Simha, Maggie Xiaoyan Cheng, and Dan Liu. Strong minimum energy topology in wireless sensor networks: Np-completeness and heuristics. *IEEE Transactions on Mobile Computing*, 02(3): 248–256, 2003.
8. David Culler, Deborah Estrin, and Mani Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, 37(8):41–49, 2004.
9. Henri Dubois-Ferriere. Sensorscope presentation at NCCR-MICS WG2, 2005.
10. E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.
11. Christian C. Enz, Amre El-Hoiydi, Jean-Dominique Decotignie, and Vincent Peiris. Wisenet: An ultralow-power wireless sensor network solution. *Computer*, 37(8): 62–70, 2004.
12. Jessica Feng, Farinaz Koushanfar, and Miodrag Potkonjak. System-architectures for sensor networks issues, alternatives, and directions. *ICCD*, 00:226, 2002.
13. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and probability. *Formal Apsects of Computing*, 6, 1994.
14. Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model checking.* MIT Press, Cambridge, MA, USA, 1999.
15. Ties Kluter. URLAP Processor, EPFL LAP Technical Report, 2004.

16. Greg Kogut, Mike Blackburn, and H.R. Everett. Using video sensor networks to command and control unmanned ground vehicles. In *AUVSI Unmanned Systems in International Security (USIS)*, 2003.
17. Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 2.0: A tool for probabilistic model checking. *QEST*, 00:322–323, 2004.
18. John Lach, David Evans, Jon McCune, and Jason Brandon. Power-efficient adaptable wireless sensor networks. In *International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, 2003.
19. Enrico Magli, Massimo Mancin, and Luca Merello. Low-complexity video compression for wireless sensor networks. *Proceedings of the International Conference on Multimedia and Expo, ICME 2003*, 3:585–588, 2003.
20. Dustin McIntire, Kei Ho, Bernie Yip, Amarjeet Singh, Winston Wu, and William J. Kaiser. The low power energy aware processing (leap)embedded networked sensor system. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 449–457, New York, NY, USA, 2006. ACM Press.
21. Joseph A. Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. *Pervasive Computing, IEEE*, 4(1):18–27, 2005.
22. Jan M. Rabaey, M. Josie Ammer, Julio L. da Silva, Danny Patel, and Shad Roundy. Picoradio supports ad hoc ultra-low power wireless networking. *Computer*, 33(7):42–48, 2000.
23. Kishore Raja, Ioannis Daskalopoulos, Hamadoun Diall, Stephen Hailes, Tom Torfs, Chris Van Hoof, and George Roussos. Sensor Cubes: A modular, ultra-compact, power-aware platform for sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN SPOTS)*, April 2006.
24. Shad Roundy, Eli S. Leland, Jessy Baker, Eric Carleton, Elizabeth Reilly, Elaine Lai, Brian Otis, Jan M. Rabaey, Paul K. Wright, and V. Sundararajan. Improving power output for vibration-based energy scavengers. *Pervasive Computing, IEEE*, 4(1):28–36, 2005.
25. Amit Sinha and Anantha Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Des. Test*, 18(2):62–74, 2001.