Chapter 7

Energy Supply

7.1 Energy Management and Distribution [Bosse, Behrmann*]

With increasing sensor node density and the shift from centralized towards decentralized data processing architectures we can observe that energy supply becomes a key technology and limiting factor in the design of sensor networks and their run-time stability. Centralized energy supply architectures have limited scaling ability and efficiency. Furthermore, they can introduce Single-point-of-Failures.

Decentralized energy supply architectures rely on local energy storage and energy harvesting, introducing hard energy constraints during run-time and the requirement for energy aware systems. Sensor nodes in common sensing applications are typically battery powered. There are different aims of energy aware systems:

1. Extension of the overall operational run-time (lifetime) of (A) The system that can be considered as the network level, and (B) The sensor node considered as the node level;
2. Enabling of more powerful operations (Algorithmic selection);
3. Safekeeping of the global operational system behaviour with or without the partial failure of system (sensor node) components. due to energy bottlenecks.

Energy aware systems address the minimization of the energy consumption and the energy management optimizing the usage of limited energy resources to safekeeping the system vividness, which present different strategies. The first major area focuses on the design of low-power systems reducing the power consumption of electronic devices, the

ISBN 978-3-527-33606-7

second area focuses on scheduling of tasks to be performed by sensor nodes and the network with the goal to minimize the consumed energy. But the problem of scheduling to minimize the energy consumption (especially on fine-grained device and component level) is a NP problem, i.e., the scheduling and computational complexity increases exponentially. And each data processing requires a specific amount of energy to be performed, and hence (software-based) energy management at run-time requires a specific amount of energy, too, which must be in balance with the benefit of energy reduction. Low-power design and energy management is already established in Distributed Sensor Networks. Details can be found in **[SIT05]**.

The activity of a sensor node can be partitioned in different major tasks:

1. Analog Signal Processing
2. Digital Data Processing
3. Communication
4. Sleeping and being idle (i.e., waiting for events)

The second and third produces dominant contributions to the overall energy consumption of a sensor node.

7.1.1 Design of Low-power Smart Sensor Systems

In electronic circuits an electrical current flowing through a resistor causes a thermal power dissipation. Static and dynamic current flows must be distinguished. There are basically two different transistor technologies: Bipolar (current-controlled current sources), and Field effect-based (voltage-controlled currents sources). Modern MOS field effect transistor technologies poses a huge ratio of a static current (no switching activity) to a dynamic current (switching activity), mainly a result of charging and discharging of parasitic electrical capacities.

Assuming major CMOS technology, the power consumption of a digital circuit on chip level can be expressed by a static (leakage) and dynamic (switching) part and by the total number N_{tot} and mean number of switching transistors N_{sw} per time unit T, respectively:

$$\begin{aligned} P_{\mathrm{static}} &\sim N_{tot} k_{\mathrm{tech}} V_{dd} \\ P_{\mathrm{dynamic}} &\sim f \bar{N}_{sw} C V_{dd} \end{aligned} \tag{7.1}$$

with f being the switching frequency (equal to $1/T$), C a technology parameter (i.e., a normalized electrical charge capacity), and V_{dd} the supply voltage of electronic circuit.

ISBN 978-3-527-33606-7

There are different optimization strategies that can be applied to a digital circuit:

1. Reduction of the switching frequency f of synchronous (clock-.driven) circuits, which leads to an increase of the data processing latency (the computation time), but that can be compensated by parallelization techniques;
2. Reduction of the number of switching transistors, which requires either application-specific designs (static) or the disabling of unused digital components (dynamic at run-time); Parallelization incresases transistor resources and the number of switching transistors;
3. Improving the technology (e.g., minimizing the parasitic capacity C);
4. Reducing the supply voltage V_{dd}(usually requiring a lowering of the switching frequency due to an increase of the switching latency).

Typical power consumption of different micro controllers, processors, wireless communication controllers, and analog-digital conversion circuits are shown in Tab. **7.1**.

Component	Description	Power (peak)
MSP430	Low-power 16 bit Micro controller, Texas Instruments[2]	Standby: 3µW 16MIPS: 1mW @ 16MHz
Atmel ATMEGA 8L	Low-power 8 bit Micro controller, Atmel[5]	Standby: 1mW 4MIPS: 12mW @ 4MHz
Intel® Core™ i5 4310U CPU	Generic and Graphics Processor for Mobile Computers, Intel[3]	40000MIPS: 15W @ 2GHz 10000MIPS:6W @ 750MHz
Wifi	Wireless communication, mid range[4]	20mW/Mbps, 2mW min.
Bluetooth®	Wireless communication, short range[4]	BR/DR: 33mW/MBps, 300µW min. Low En.: 100mW/MBps, 30µW min.
Zigbee®	Wireless communication, short range[4]	300mW/MBps, 50µW min.
wM-Bus, 868 MHz	Wireless communication, long range[1]	0.3W max./40ms
wM-Bus, 169 MHz	Wireless communication, long range[1]	1W max./100ms

Tab. 7.1 *Examples of typical power dissipation (and possible typical duration) of microcontrollers and peripheral components. MIPS: Million Instructions Per Second, MBps: Million Bit per Second, Msps: Million samples per second, TDP: Thermal Design Power, Sources: 1:**[FEC15]**, 2: **[ALB09]** 3: **[INT13]**, 4: **[TOC13]**, 5:**[ATM13]**, 6: **[MAX16]***

Component	Description	Power (peak)
GSM	Wireless communication, long range[1]	7W max./0.6ms
MAX11102	Analog-Digital Converter, 12 bit, SAR with S&H, Maxim[6]	5mW @ 3Msps

Tab. 7.1 *Examples of typical power dissipation (and possible typical duration) of microcontrollers and peripheral components. MIPS: Million Instructions Per Second, MBps: Million Bit per Second, Msps: Million samples per second, TDP: Thermal Design Power, Sources: 1:**[FEC15]**, 2: **[ALB09]** 3: **[INT13]**, 4: **[TOC13]**, 5:**[ATM13]**, 6: **[MAX16]***

In low-power systems wireless communication consumes commonly the dominant part of the electrical power and can reach a fraction of 99%, caused by the analog section of a communication system and the required radio power for wireless communication. Wired communication creates a much lower contribution below 20% **[SIT05]**. Low-power processors consume several order fewer energy than their desktop computer counterparts and provide more fine-grained power control at run-time.

The design of low-power systems, especially application-specific digital circuits, can profit from pre-design simulations on gate level, shown in Fig. **7.1**, evaluating the effect of algorithm complexity on power consumption. Additionally, this power analysis delivers an estimation of the power consumption is particular computations in a sensor node, which can be used an input for fine-grained dynamic power management.

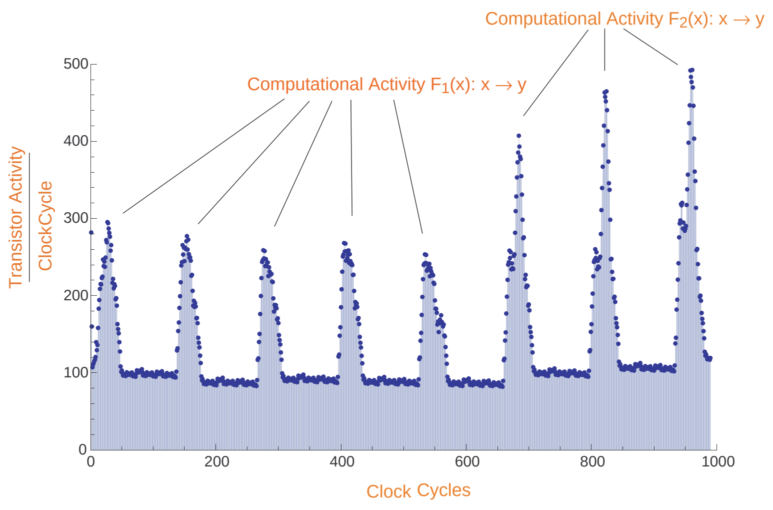
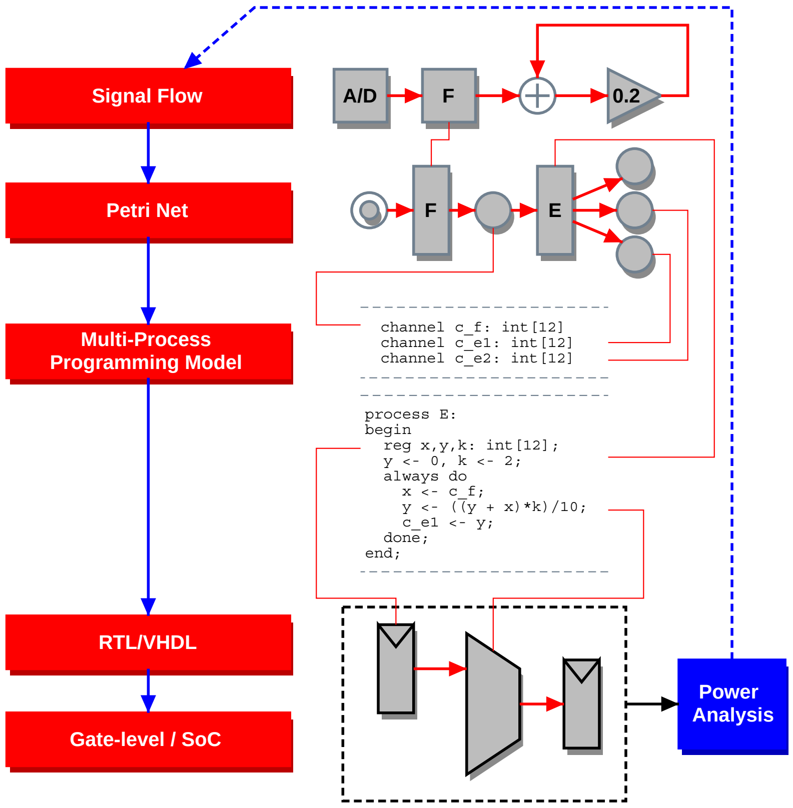


Fig. 7.1 *Gate-level simulation of a digital circuit analyzing the switching activity of gates in relation to algorithmic activity **[BOS11]**. The peaks in the transistor switching activity accumulated of all logic gates in the Device under Test (DUT) result from computational activity, and the height of the peaks are related to the computational complexity of the used algorithm,*

If there is no explicit computational activity, there is still transistor switching activity, mainly resulting from clock-driven register and control state machine logic. The power analysis workflow is shown in Fig. **7.2**. On top-level, an algorithm consisting of a set of functions $F=\{fn_1,fn_2,..\}$ is modelled with a signal flow graph, which is transformed in State-Transition (SN) Petri Net. Functional blocks fn_i are mapped to transitions, and states represent data that is exchanged between those functional blocks. The partitioning of functional blocks to transitions of the net can be performed at different composition and complexity levels **[BOS11]**. Sensor data is acquired periodically and passed to the data processing system. A token of the net is equal to a data set of one computation processed by the functional blocks. Different functional blocks can be placed in concurrent paths of the net.



```
  channel c_f: int[12]
  channel c_e1: int[12]
  channel c_e2: int[12]

process E:
begin
  reg x,y,k: int[12];
  y <- 0, k <- 2;
  always do
    x <- c_f;
    y <- ((y + x)*k)/10;
    c_e1 <- y;
  done;
end;
```

Fig. 7.2 *Power Analysis Workflow deriving the relation between algorithm activity and complexity with the power consumption* ***[BOS11]****.*

The Petri Net is then used 1. To derive the communication architecture; and 2. To determine an initial configuration for the communication network, which is related to a multi-process programming model, finally used to derive a hardware model for a SoC design on

Register-Transfer level. Functional blocks with a feedback path require the injection of initial tokens in the appropriate states.

The aim of the power analysis is to retrieve an explicit assignment of the power (or the integral energy) required to execute a specific function block fn_i under specific conditions like the data processing rate, r_i=data sets/time unit, or the data set size w_i, i.e., the set F is mapped on a parameterized power set $P = \{p_1(fn_1,r_1,w_1), p_2(fn2,r_1,w_1), .. , p_i(fn_i,r_j,w_k), ..\}$. The power-function set can be used (1) At design-time for the optimization of a hardware design; (2) At run-time for dynamic power management, e.g., by selecting from a set of functions that perform basically the same computation but with different accuracy or quality and energy consumption.

Possible power savings on device level at run-time address **[SIT05]**:

- Disabling of idle (not-used) components (e.g., leakage current control techniques for memories);
- Switching between multiple low-power modes reducing the performance (e.g., data transmission bandwidth), i.e., a range between full operation (active), standby, power-down, and disabled;
- Supply voltage, clock frequency, and threshold voltage scaling (but affecting the performance of the device);
- Dynamic variation of ADC sampling rates;
- Algorithmic selection from a set of computational units performing basically the same computation but differing in power consumption and accuracy or quality of service;
- Reconfiguration of device components.

7.1.2 A Toolbox for Energy Analysis and Simulation [Bosse, Behrmann]

In the last years the number of available high performance but low-power embedded systems grown exponentially, enabling application scenarios for tailored sensor systems. For many application cases battery powered or self-powered systems are needed, e.g. in the context of wireless sensor networks. A designer has to assure that the system is provided with the appropriate amount of energy and assuring enough power to fulfil its given tasks. Often this can only be done when the analysis is carried out in the context of the real application. Consequently a simulation has to consider the environmental condition of this context as well.

Therefore a simulation toolbox for energy and power analysis of independent sensor nodes is required, a tool for designing modular sensor systems, proposed in **[BEH10]** using the Matlab/Simulink framework. The focus of this tool will be on the economic and efficient use of power and energy performed on the embedded system level.

The base of this toolbox is a set of simulation blocks modelling the power behaviour of embedded modules like energy sources, converters, storage and load. A set of tools for observing losses, energy throughput and power lags, assists the system designer to set up an economic solution. A strong emphasis lies on the modelling of modern energy harvesting principles and the embedding physical situation.

One main goal of this toolbox is to overcome the over-sizing of the power supply of electric systems for the "worst case". Instead a situation-dependant adaptive energy management will control different operation modes of embedded systems to cope with different power supply and energy situations. Therefore these systems can be specified more accurate and economic.

To save energy, the different operation modes will lead to a tailored sensor data processing. Beside of using a software-controlled micro-processor system, dedicated and application specific configurable hardware blocks can be added. The simulation and the energy management rely on load models that consider different implementations on work task level.

Every measurement systems designer dealing with battery- or self-powered systems have to consider the energy behavior. Most classical measuring devices were designed to work on a constant and sufficient power supply. Measurement experts often are not experienced in low-power design. A lot of optimizations can be applied using technology from mobile computing, control theory and artificial intelligence. Using sensors not only for measuring, but also for energy harvesting, creates new perspectives in the sensor node design and deployment. Future methods for sensorial materials are consisting of low power design, adaptation in the power consumption and energy management up to self-organization, self-localization, fault tolerance, cognition, and grid intelligence.

Sensor nodes could be implemented according the architectural model shown in Fig. **7.3**. The model of a sensor node can be divided into the parts data (acquisition, signal and digital data processing, and communication) and energy (supply, storage, consumption). Though the design of low-power systems focus on the first level on the optimization of data processing and communication in respect to the energy consumption, this toolbox will concentrate on modelling, simulation and analysis of the energy branch of the entire system. For most self-powered applications the energy provided by the energy harvesting device has to get converted, shown in Fig. **7.4**.

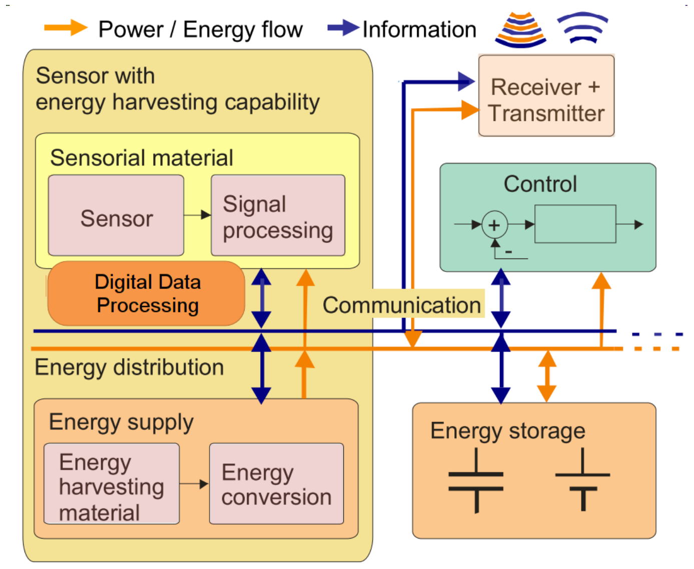


Fig. 7.3 *Architecture of a Sensorial Material embedding a Sensor Node with Energy Management and Energy Harvesting* **[BEH10]**.

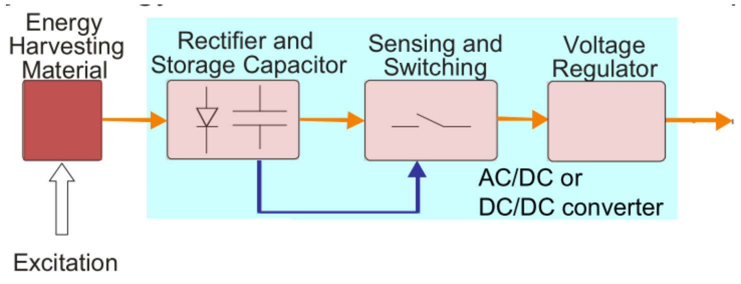


Fig. 7.4 *Principle Energy Harvesting Architecture transforming some physical property into an electrical signal that can stored.*

The structure of the toolbox is close to the node architecture scheme according Fig. **7.3** and is ordered from the energy's point of view (details can be found in **[BEH13]**):

Energy Sources

- Ideal Sources
- Batteries
- Inductive/Magnetic
- Mechanical

- Piezo Electric
- Photo Electric
- Thermal Electric
- Transformers

Energy Converters

- AC/DC
- DC/DC
- Voltage Regulators + PWM
- Amplifier

Energy Storage

- Accumulators
- Capacitors

Energy Consumers

- Micro controllers (including Memory)
- Sensors and Signal Processing
- Actuators
- Communication Modules (including Radio)
- Illumination

For each element there are generic blocks in the structure to cover the main functionality. Special parts can be derived and added to the Library. According to the naming conventions in Matlab/Simulink an in-put signal is connected to an "in-port" and the output of a block is fed through the "out-port". A parameter mask in Simulink can be provided as a GUI for setting parameter values comfortably, which are then connected to constant values inside the block model implemented in Simulink **[SIM10]**. All blocks of the energy toolbox are masked for convenience e.g. to parameterize the block according to the data sheet.

An example power model of a sensor node is shown in Fig. **7.5**, consisting of different power components (energy sources, converters, and consumers) and a power balancer performing some kind of power analysis and management (controlling the components). The power model is a parametric diagram with a constraint block (the power balancer implementing a power optimization function) and a set of constraint parameters (the sensor node components).

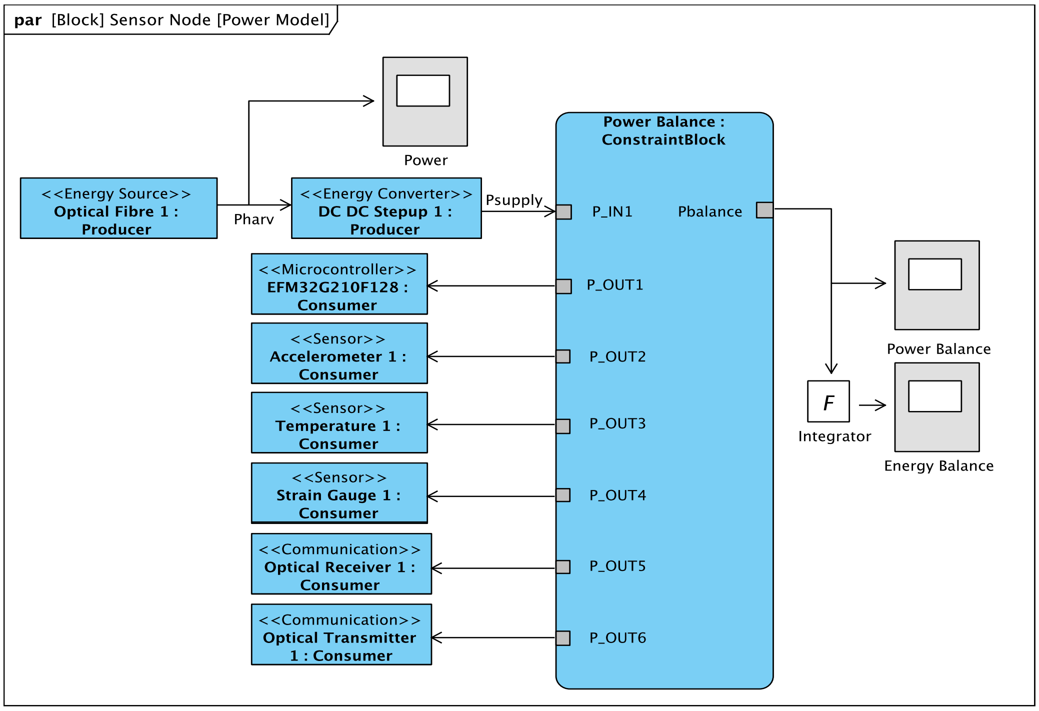

par [Block] Sensor Node [Power Model]
Power
Power Balance : ConstraintBlock
<<Energy Source>> Optical Fibre 1 : Producer
Pharv
<<Energy Converter>> DC DC Stepup 1 : Producer
Psupply
P_IN1
Pbalance
<<Microcontroller>> EFM32G210F128 : Consumer
P_OUT1
<<Sensor>> Accelerometer 1 : Consumer
P_OUT2
<<Sensor>> Temperature 1 : Consumer
P_OUT3
<<Sensor>> Strain Gauge 1 : Consumer
P_OUT4
<<Communication>> Optical Receiver 1 : Consumer
P_OUT5
<<Communication>> Optical Transmitter 1 : Consumer
P_OUT6
Power Balance
F
Integrator
Energy Balance

Fig. 7.5 *Parametric diagram of the power model of a Sensor Node using the toolbox components (Power sources, consumers, and analyzers, adapted from* ***[BEH13]****)*

The application of the analysis tool delivers information about the layout of the sensor node's power system by showing the dynamic energy flows of the different components. These results and models can be incorporated into a subsequent low-power design and DPM. Most of the modelled components are dominated by a sleep schedule executing a cyclic wake-up reducing energy demand by an estimated duty cycle of 1:100 **[BEH13]**.

Energy Source Block

The energy sources class covers the different methods for providing energy. In traditional design approaches, there is an assumption of a constant source always providing enough power. This tool enables the user to tailor the ratings for average and maximum use. For the possibility of environment-dependent energy sources, the blocks have the ability to be controlled by environmental parameters, for example, by solar radiation in the case of a photo voltaic cell. A solar cell generator converts light pulses into electrical power P_{MPP}, calculated as follows: $P_{\mathrm{MPP}} = \eta\, A\, E = FF\, I_{\mathrm{sc}}\, U_{\mathrm{oc}} = I_{\mathrm{m}} U_{\mathrm{m}}$, where η is the efficiency, E is the irradiance, A is the area of the cell, FF is the fill factor, I_{sc} is the short-circuit current, U_{oc} is the open-circuit voltage, I_{m} is the optimum operating current, and U_{m} is the optimum operating voltage.

Energy Storage Block

The generic capacitor block contains a simple model of a capacitor. There is a series resistance R representing effects of dielectric losses and conductor resistance. The capacitor inductance is represented by the series inductance L. A parallel resistance R_p models leakage current flow and the self-discharge. These values have to be entered into the blocks parameter mask. If the value of the series resistance is not known, the dissipation factor tan(d) and the according frequency can be entered instead. Note that the time needed for self-discharge is approximately $5 \times R \times C$. An initial voltage across the capacitor can be entered into the mask. The in-port signal represents the power to be stored on the capacitor.

Energy Converter Blocks

Components for converting and regulating energy flows are technically mandatory. Focusing on the energy and efficiency, the most important fact is that these devices have electric losses and so the loss of power has to be considered. Most modern devices like direct current DC-to-DC converters use switched-power circuits and smoothing capacitors. A generic model for calculating losses without implementing the real operation is sufficient at the level of system definition and could be recalculated when the hardware layout is fixed.

Energy Consumer Block

The consumer block provides generic constant power consumption. It is a good way to estimate the component's energy demand using datasheet's values, i.e, useful for modeling components that have not been modeled in detail yet. A cyclic wake-up consumer block models the power consumption with an electrical switch having a resistance when in "on" position and switching losses. The power consumed is calculated as $P_{\text{loss}} = t_{\text{switch}}\, f_{\text{pwm}}\; UI + \underline{I}^2\; R_{\text{on}}$, where t_{switch} is the time needed to switch between on and off, f_{pwm} is the switching frequency, U is the voltage of the power source, I is the current at maximum voltage, $\underline{I}$ is the average current, and R_{on} is the ohmic resis-tance of the switch when in "on" position (Graovac and Pürschel, 2009). The output port provides information on power consumption.

Typically, energy management is performed by a central controller with limited fault tolerance and the requirement of a well-known environment world model for energy sources, sinks, and storage. Energy management in a network can additionally involve the transfer of energy between network nodes.

As already introduced, System-on-Chip hardware design uses advanced high-level synthesis approaches on higher algorithmic level that can improve energy management and power efficiency based on results from toolbox analysis. Models and model parameters provided by the toolbox can be used for algorithm design, configuration, and hardware synthesis gathered by the power analysis introduced in Sec. **7.1.1**.

7.1.3 Dynamic Power Management

The application of low-power hardware design techniques is static and cannot exploit energy reduction at run-time. Beside sensor nodes implemented with application-specific digital circuits (System-on-Chip designs), sensor nodes are often operating with micro controllers and light-weighted Operating Systems.

An Operating System (OS) is the central instance of a computer system. They control resources (memory, devices, processors), schedule device access, implement a hardware abstraction layer, and provide unified programming interfaces for services accessing devices and data structures (e.g., file systems). Since the past decade, the OS additionally performs Dynamic Power Management (DPM) of the entire system, originally required for mobile devices (e.g. , notebooks, later smart phones), finally deployed in server computers, too.

DPM basically affects two main resources of a computer system: (1) The processor; (2) Input/Ouput Devices. Run-time power control of system components is closely related to optimized scheduling of access operations.

Assuming central processing units (CPU) and peripherals supporting different power states there are two different DPM techniques that can be combined:

- CPU-centric DPM: Based on Eq. **7.1**, dynamic core voltage and frequency scaling (DVS/DFS) of processors based on processor workload is one major power reduction technique at run-time.
- IO-centric DPM: Peripheral IO devices supporting different power states can be shutdown or switched to stand-by mode if there is no activity. In contrast to DVS/ DFS techniques introduces power state switching of peripheral devices a time delay affecting the access operations that compromises IO performance.

The main action performed by DPM is the scheduling of a set of tasks, $T=\{t_1,t_2,..\}$, which has to be executed. Each task is parameterized by the tuple $t_i=(a_i,d_i,l_i)$, with a_i as the arrival time, d_i the latest deadline time, and l_i the estimated duration time of the task in instruction cycles. Assuming the CPU can operate on different core voltages $V=\{v_1,v_2,..\}$, associated with a set of speed levels $F=\{f_1,f_2,..\}$. An algorithm is required to perform all tasks in T with minimal energy consumption, i.e., assigning an appropriate speed level $f_j(v_j)$ to each task t_i to meet all (or at least as many) deadlines of the tasks:

$$E \sim \sum_{i=1}^{n} v_j^2 f_j l_i \tag{7.2}$$

CPU-centric DPM

EDF Algorithm
The well-known Earliest Deadline First (EDF) algorithm is a dynamic scheduling algorithm primarily used in real-time operating systems. It places the task set T in a priority queue **[JEF91]**, i.e., an ordered ready list.

LEDF Algorithm
The Low-energy EDF (LEDF) algorithm **[SWA01]**, shown in Alg. **7.1**, is an extension of the EDF algorithm. It considers different CPU speeds and adapts scheduling based on tasks missing their deadlines. The scheduling policy tries to minimize the total energy consumed by the task set and guarantees the deadline for periodic tasks. For all tasks an absolute deadline is computed and that is recalculated at each release based on the absolute time of release and the relative deadline **[SIT05]**. The task with the earliest (first) deadline is selected for execution. Energy optimization is performed by checking the task deadline and if it can be met by executing it at a lower voltage (speed). Each speed at which the processor can run is considered for deadline computation in order from the lowest to the highest. For a given speed, the approximation of the worst-case execution time of the task is calculated based on the estimated instruction count of the task. If this execution time is too high to meet the current absolute deadline for the task, then the next higher speed is considered.

The LEDF algorithm has a computational complexity of $O(N \log N)$ for a processor with two speed levels, where N is the total number of tasks to be executed. Each scheduling requires computation and hence reduces the overall energy optimization efficiency, though the LEDF algorithm introduces only a low overhead compared with other parts of an Operating System. The complexity of the LEDF algorithm becomes $O(N \log N+kN)$, with k as the number of speed settings that are supported by the CPU. In **[SIT05]** the computational overhead of the scheduling is reported to be below 1‰ and can be considered as negligible compared to the execution time of the tasks. The power saving of the LEDF algorithm compared to the conventional EDF is shown in Fig. **7.6**, and reaches up to 40% with low CPU utilization.

One disadvantage of this approach is the requirement for task deadlines. Event-driven tasks with a short run-time can be assigned to computable deadlines, but pure computational tasks with varying instruction lengths (depending on the state of the program and input data) cannot be captured very well by this approach.

Material-integrated sensor networks will be equipped with low-resource and low-power micro controllers supporting at least one active and one sleep state, demanding for power state transition algorithms, similar to the approaches discussed in IO-centric DPM. Micro controllers consist of a microprocessor, memory, and a large number of peripheral devices, all integrated on a System-on-Chip design, hence profiting from IO-centric DPM.

Alg. 7.1 *The simplified LEDF algorithm adapted from* ***[SWA01]***

```
Procedure LEDF()
Begin
  Do
    If (tasks waiting to be scheduled in ready list) Then
      Sort(deadlines) in ascending order
      Schedule(task) with earliest deadline
      Check if deadline can be met at lower speed/voltage
      If (deadline can be met) Then
       Schedule(task) to execute at lower speed/voltage
      ElsIf (deadline cannot be met) Then
        Check if deadline can be met at higher speed/voltage:
        If (deadline can be met) Then
          Schedule(task) to execute at higher speed/voltage
        ElsIf (deadline cannot be met) Then
          Task cannot be scheduled. Call exception handler!
    Else
      Do nothing.
  Done
End
```

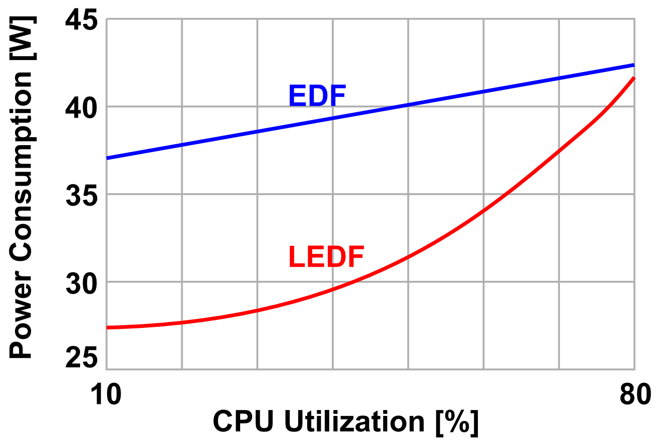


Fig. 7.6 *Illustration of the benefit for power saving of the LEDF compared to the EDF algorithm (based on data from* **[SIT05]**).

IO-centric DPM

Scheduling of IO devices is a technique that was originally introduced in non-real-time OS to minimize the access time for IO requests. For example, a hard disk can only read consecutive blocks on the disk with a high throughput. Random access causes significant read/write head positioning delays. One major algorithm to schedule different IO hard disk requests is a classical elevator algorithm solving multiple elevator requests on different floors and different directions. Such methods perform well in reactive systems, but a minimization of access times in real-time systems do not guarantee deadlines. With respect to energy-aware systems the elevator algorithm minimizes the energy consumption of IO devices, but cannot profit from different power states of such devices. Again an algorithm is required for managing and scheduling of multi-power-state IO devices, with the restriction but common case of devices supporting two power states (active and sleep/stand-by). It is assumed that a system consists of a set of p devices $IO=\{io_1, io_2, .., io_p\}$. Different power states and power state transitions must be considered with respect to their power consumption and duration for each device io_j:

- Wake-up transition: Sleep→Working with $P_{wu,i}\bullet t_{wu,j}$
- Shutdown transition: Working→Sleep with $P_{sd,j}\bullet t_{sd,j}$
- Working state: $P_{w,j}$
- Sleep state: $P_{s,j}$

A wake-up or shutdown state transition is expensive, and hence there is a break-even time t_{be} for a sleep state between two working states consuming equal or more power than the case without the temporary shutdown of the device. This is a constraint not existing in CPU speed/voltage switching and complicates the scheduling algorithm significantly! For idle time periods shorter than the break-even time power is saved by keeping the working state.

According to **[SIT05]**, there is again a set of n tasks accessing the IO devices $T=\{t_1,t_2,..,t_n\}$, which has to be scheduled. Each task is parameterized by the tuple $t_i=(a_i, c_i, p_i, d_i, L_i)$, with a_i as the arrival time, c_i a worst-case execution time, p_i a period, d_i a deadline time, and L_i a device usage list of a task. Commonly, the deadline of the task is equal or less than the period time $d_i \le p_i$. The energy consumed by a specific device io_i is given by:

$$E_i = P_{w,i}t_{w,i} + P_{s,i}t_{s,i} + MP_{st,i}t_{st,i} \tag{7.3}$$

with M as the number of power state transitions requiring the power $P_{st}=P_{wu}=P_{sd}$ for a time t_{st}. Associated with a task set T there is a set of l jobs $J=\{j_1,j_2,..,j_l\}$ containing all tasks to be executed. Except for the job period, a job inherits all parameters of the task. A job is an instance of a task. If there is a job set J that uses a set of IO devices IO, a scheduling algorithm have to identify a set of start times $S=\{s_1,s_2,..,s_l\}$ for the jobs such that the total energy consumed $\Sigma_{i=1}^{p} E_i$ is minimal and all jobs meet their deadline.

EDS Algorithm

From the tasks to be executed a job tree is generated. A common approach to find a scheduling solution is called pruning technique by iterartively pruning branches in the tree with respect to energy and time. A vertex *v* of this scheduling tree is a tuple (*i*,*t*,*e*), where *i* is the index of the job j_i, *t* a start time for the job, and *e* the energy consumed by the device until time *t*. A path from the root vertex to an intermediate vertex *v* is termed a partial schedule with some of the jobs. A path from the root vertex to a leaf vertex is a complete schedule. A complete schedule in that no job misses its deadline is suitable to solve the scheduling problem because temporal pruning eliminates all infeasible partial schedules.

The EDS algorithm, shown in Alg. **7.2**, uses the job list as an input and computes all possible non-preemptive minimum energy schedules for the given job set. Details can be found in **[SIT05]**. The algorithm shows the complexity to find an optimal energy-aware solution for IO devices. Though this is traditionally part of an Operating System only available on more powerful computer systems, low-resource embedded systems deployed in material-integrated sensing systems cannot compute the EDS algorithm due to required memory and computational resources.

Alg. 7.2 *EDS Algorithm* **[SIT05]**

```
Procedure EDS(J,l)
 J is the Job Set with elements (i,time,energy)
 l is the number of jobs
 openList is a list of unexpanded vertices
 currentList is a list of vertices at the current depth
 t is a time counter

 t:=0; d:=0; currentList:={}; openList:={(0,0,0)};
 Add(vertex(0,0,0) to openList)
 ForEach vertex v=($j_i$,time,_) ∈ openList
   t := time + completionTime(ji);
   Find all jobs J' released up to time t:
   J' := {j elem J | time($j_i$)≤t}
   ForEach job j ∈ J' Do
    If (j has not been previously scheduled) Then
      V := {Find all possible scheduling instants for j};
      Computer energy for each generated vertex in V
      currentList := {currentList ∪ V};
    EndIf
   Done
   ForEach pair of vertices ($v_1$,$v_2$) ∈ currentList Do
     If $v_1$=$v_2$ and partialSchedule($v_1$)=partialSchedule($v_2$) Then
       If energy($v_1$) > energy($v_2$) Then
         Prune($v_1$)
       Else
         Prune($v_2$);
     EndIf
   Done
   openList := { openList ∪ {Unpruned vertices of currentList}};
   currentList := {};
   Incr(d);
   If d=l Then Terminate;
 Done
```

The Low-Energy Device Schedule (LEDES) algorithm **[SWA03]**, a near-optimal, deterministic device-scheduling algorithm for two-state I/O devices that can be used for online device scheduling. It is assumed that the start time for each job is fixed and known a priori. Under this assumption, the energy-aware device scheduling problem P_{io} can be simplified. In contrast to EDS, the LEDES algorithm schedules devices and not jobs. The only critical check is that the idle time of a device is greater than the break-even time.

The P_{io} scheduling problem: Given the start times $S=\{s_1,s_2,..,s_n\}$ and completion times $C=\{c_1,c_2,..,c_n\}$ of the n tasks in a real-time task set T that uses a set $IO=\{io_1, io_2, .., io_p\}$ of I/O devices, determine a sequence of sleep/working states for each I/O device io_i such that total energy consumed $\Sigma_{i=1}^{p} E_i$ by IO is minimal and all tasks are executed within the deadline interval. This algorithm - though not giving the best possible solution compared with EDS - is more suitable for material-integrated low-resource system due to lower memory and computational requirements. The LEDES algorithm is shown in Alg. **7.3**.

Alg. 7.3 *LEDES Algorithm* **[SWA03]** *for on-line scheduling of j-th device io_j ($t_{0,j}$ is the device power state transition time of device j, L the device usage list, and curr the current scheduling step)*

```
Procedure LEDES($io_j$,$\tau_i$,$\tau_{i+1}$,curr)
  If (Scheduling step curr is the start of $\tau_i$) Then
    If (state($io_j$) = powered-up) Then
      If ($io_j \notin \{L_i \cup L_{i+1}\}$) Then
        shutdown($io_j$)
      EndIf
      If ($io_j \in L_{i+1}$}) Then
        If ($s_{i+1}$ - ($s_i$+$c_i$) >= $t_{0,j}$) Then
          shutdown($io_j$)
        EndIf
      EndIf
    Else
      If ($io_j \in L_{i+1}$ and $s_{i+1}$ - (si-ci) < $t_{0,j}$) Then
        wakeup($io_j$)
      EndIf
    EndIf
  ElsIf (Scheduling step curr is completion of task $\tau_i$) Then
    If (state($io_j$) = powered-up) Then
      If ($io_j$ nelem $L_{i+1}$ and $s_{i+1}$-curr >= $t_{0,j}$) Then
        shutdown($io_j$)
      EndIf
    Else
      wakeup($io_j$)
    EndIf
  EndIf
End
```

A significant advantage of online I/O device scheduling compared with off-line approaches is that the DPM decision-making can exploit underlying hardware features such as buffered read and write operations. In contrast, an off-line device schedule constructed as a table in system memory prevents the use of such features. Thus the flexibility of online scheduling enhances the effectiveness of device scheduling **[SIT05]**.

7.1.4 Energy-aware Communication in Sensor Networks

The total energy consumed in a sensor network can be classified in these contributions with a rough weighting:

- Analog Data Processing (ADC) [5%]
- Digital Data Processing (Computation) [30%]
    - Microprocessor [25%]
    - Memory [5%]
- Data Communication [60%]
    - On-chip [<5%]
    - Inter-chip (same node) [5%]
    - Inter-node [50%]
- Idle / Stand-by (including auxiliary electronics and electrical losses) [5%]

In large scale sensor network the inter-node communication consumes the dominant part of the energy. Energy consumed by transmitting data using wireless communication over a 10 to 100m range consume as much energy as thousands of data processing operations **[SIT05]**. Large scale sensor networks commonly uses message-based communication and routing (delivery of a message along a path) creates a significant overhead. Smart communication is required to reduce the energy consumptions. Avoiding communication is the most successful energy saving in sensor networks! Following there are some techniques shown saving energy by reducing communication and overhead.

The big volume of sensed information in large-scale networks and the requirement to aggregate data from the different sensed nodes for detecting events of interest leads to a large number of communications across the different nodes. The energy consumption of communication is proportional to the bit-size of a message, and in the case of wireless transmission proportional to the square of distance. Routing increased the energy for one message multiple times and depends on the number of intermediate nodes forwarding the message along its path.

Clustering

Assuming mesh-like network structures, clustering of sensor nodes in Regions of Interest (ROI) and limiting communication mostly to these regions can reduce communication significantly. These clusters perform local processing of the sensed data and extract some useful information that is communicated to the nodes outside the cluster. See, for example, **[LOH09]**, for a deeper discussion of clustering and dedicated cluster communication protocols. The clustering concept is close to the Multi-agent model and self-organizing systems with entities interaction with each other in a limited region.

Caching-Based Communication

In contrast to generic purpose communication networks, sensor nodes exchange mostly raw sensor and aggregated sensor data. This enables classification of communication messages. In stream-based sensor processing the sensor data is sent multiple times without a change in the sensor value. This offers the opportunity to cache sensor data that is encapsulated and transferred in messages. Each node in a defined cluster caches the n previous data values sensed in this cluster. Assuming all the sensor nodes in the cluster have a coherent cache, then if the new sensed value that needs to be transmitted matches with one of the cached values, then only the index number of the matching cache entry is transmitted instead of the actual data value. This can reduce the size and hence the costs of communication significantly if the size of the generated sensor data is large compared to the data size of a cache index. Details can be found in **[SIT05]**, chapter 35. If a dedicated and optimized communication protocol is used, the minimal packet size is $|p| = 1 + w + \log_2 n$, with w as the sensor data bit-size and n the number of cache entries. The first bit indicates the kind of a message (cache or data). This very simple caching approach is suitable for low-resource material-integrated sensor nodes. One disadvantage of this simple approach is the proper handling of cache inconsistency due to communication or node failures.

Data-centric and Event-based Communication

Instead of transmitting sensor data periodically (request-based) in a data stream, the sensor nodes (as a data source) can decide if there was a significant change of sensor data that can be of interest for other nodes (as a data sink and collector). Only if there is an event was detected sensor data is send to sink nodes. This event-based communication (discussed in Sec. **6.2.7**) can reduce the overall communication activity significantly and require no further synchronization, though nodes in a cluster can collectively decide if there was an event. This is discussed in Sec. **6.4.7** and applies clustering and event-based communication in distributed machine learning.

There are many popular data-centric routing algorithms for minimizing energy consumption, e.g., diffusion routing, which use local spatial gradients to select paths for sending information from data source to sink nodes. Energy-aware metric and the geographical position of each node can be used to determine a route, too. Hybrid approaches can deal with constrained energy and quality-of-service considerations to improve routing. The lowest energy path is not always be the optimal path considering long-term network connectivity. Global energy minimization can result in uneven energy consumption patterns across particular sensor nodes. Consequently, some nodes could deplete their energy resources sooner than other nodes, reducing the system vividness and information quality of the entire sensor network.

7.1.5 Energy Distribution in Sensor Networks

With increasing miniaturization and sensor density, decentralized self-supplied energy concepts and energy distribution architectures are preferred over centralized.

Self-powered sensor nodes collect energy from local sources, for example, by using thermo-electrical energy harvester or solar cells. But additionally they can be supplied by external energy sources. Nodes in a sensor network can use communication links to transfer energy, for example, optical links are capable of transferring energy using Laser or LE diodes in conjunction with photo diodes on the destination side, with a data signal modulated on an energy supply signal. An example can be found in **[BUD11]**. The sensor node uses an optical link based on a fibre connection (wired communication). Two different optical wavelengths are used to transmit data and energy over the fibre link. The principle node architecture is shown in Fig. **7.7**, simplifying the transmission of data and communication by using one optical wavelength, which finally reduces the node complexity. In this approach, the communication signal is modulated on the (strong) energy signal on the sender side, and filtered out on the receiver side **[BOS12]**.

The main building blocks of a sensor node, the proposed technical implementation of the optical serial interconnect modules, and the local energy management module collecting energy from a local source, for example a thermo-electric generator, and energy retrieved from the optical communication receiver modules. The data processing system can use the communication unit to transfer data (*D*) and superposed energy (*E*) pulses using a light-emitting or laser diode. The diode current, driven by a differential-output sum amplifier, and the pulse duration time determine the amount of energy to be transferred. The data pulses have a fixed intensity several orders lower than the adjustable energy pulses. On the receiver side, the incoming light is converted into an electrical current by using a photo diode. The data part is separated by a high-pass filter, the electrical energy is stored by the harvester module.

Sharing of one interconnect medium for both data communication and energy transfer significantly reduces node and network resources and complexity, a prerequisite for a high degree of miniaturization required in high-density sensor networks embedded in sensorial materials. Point-to-point connections and mesh-network topologies are preferred in high-density networks because they allow good scalability (and maximal path length) in the order of O(log *N*), with *N* as the number of nodes.

Assuming a mesh-like network topology with peer-to-peer connectivity of neighbour nodes, nodes can exchange energy based on demand and negotiation, offering energy distribution and balancing capabilities. Nodes with energy surplus can transfer energy from their local energy storage to neighbour nodes with low energy deposit. This technical feature enables distributed and global energy management, discussed in the next section. Energy distribution capabilities of sensor nodes can eliminate a centralized energy supply infrastructure.

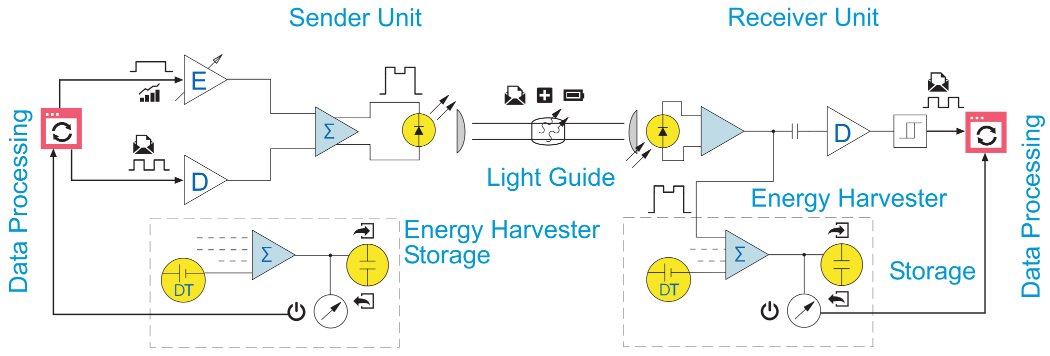


Fig. 7.7 *Principle sensor node architecture with energy harvesting using an optical link to transmit data and energy.*

7.1.6 Distributed Energy Management in Sensor Networks using Agents

Having the technical abilities explained in the previous section, it is possible to use active messaging to transfer energy from good nodes having enough energy towards bad nodes, requiring energy. An agent can be sent by a bad node to explore and exploit the near neighbourhood. The agent examines sensor nodes during path travel or passing a region of interest (perception) and decides to send agents holding additional energy back to the original requesting node (action). Additionally, a sensor node is represented by a node agent, too. The node and the energy management agents must negotiate the energy request.

The agents have a data state consisting of data variables and the control state, and a reasoning engine, implementing behaviour and actions. Table **7.2** poses the different agent behaviour required for the smart energy management.

A node having an energy level below a threshold can send a help agent with a delta-distance vector specifying the region of interest in a randomly chosen direction. The help agent hops from one sensor node to the next until the actual delta-vector is zero. If there is a good node found, with local energy above a specified threshold, the help agent persists on this node and tries to send periodically deliver agents transferring additional energy to the originator node. An additional behaviour, help-on-way, changes the deliver agent into an exploration agent, too. Such a modified agent examines the energy level of nodes along the path to the destination. If a bad node was found, the energy carried with the agent is delivered to this node, instead to the original destination node. This approach is independent of the agent processing platform and the mobile agent model.

Agent Type	Behaviour
`Request`	*Point-to-point agent*: this agent requests energy from a specific destination node, returned with a `Reply` agent.
`Reply`	*Point-to-point agent*: Reply agent created by a `Request` agent, which has reached its destination node. This agent carries energy from one node to another.
`Help`	*ROI agent*: this agent explores a path starting with an initial direction and searches a good node having enough energy to satisfy the energy request from a bad node. This agent resides on the final good node for a couple of times and creates multiple deliver agents periodically in dependence of the energy state of the current node.
`Deliver`	*Path agent*: this agent carries energy from a good node to a bad node (response to `Help` agent). Depending on selected sub-behaviour (`HELPONWAY`), this agent can supply bad nodes first, found on the back path to the original requesting node.
`Distribute`	*ROI agent*: this agent carries energy from and is instantiated on a good node and explores a path starting with an initial direction and searches a bad node supplying it with the energy.

Tab. 7.2 *Agents with different behaviour used to manage and distribute energy (ROI: Region of Interest).*

Simulation carried out in **[BOS12]** with the SeSam agent simulator environment was used to show the suitability of this approach. Each sensor node is modelled with an agent, too. Energy management agents and sensor node agents negotiate energy demands and communicate with each other by using globally shared variables. If the energy deposit of a node is below a threshold $E_i<E_{low}$ (called bad node), help agents are sent out, if $E_i>E_{high}>E_{good}$ then distribute agents are used to distribute energy to surrounding bad nodes. If $E_i>E_{good}>E_{low}$ then the node is fully functional (called good node).

The simulation showed that help agents with simple exploration and exploitation behaviour are suitable to meet the goal of a regular energy distribution and a significant reduction of bad nodes unable to process sensor information, but additional distribute agents create no significant benefit. The multi-agent implementation offers a distributed management service rather than a centralized approach commonly used. The simple agent behaviours can be easily implemented in digital logic hardware (application-specific platform approach).

The temporal development of bad and good nodes with and without smart energy management is show in Fig. **7.8**.

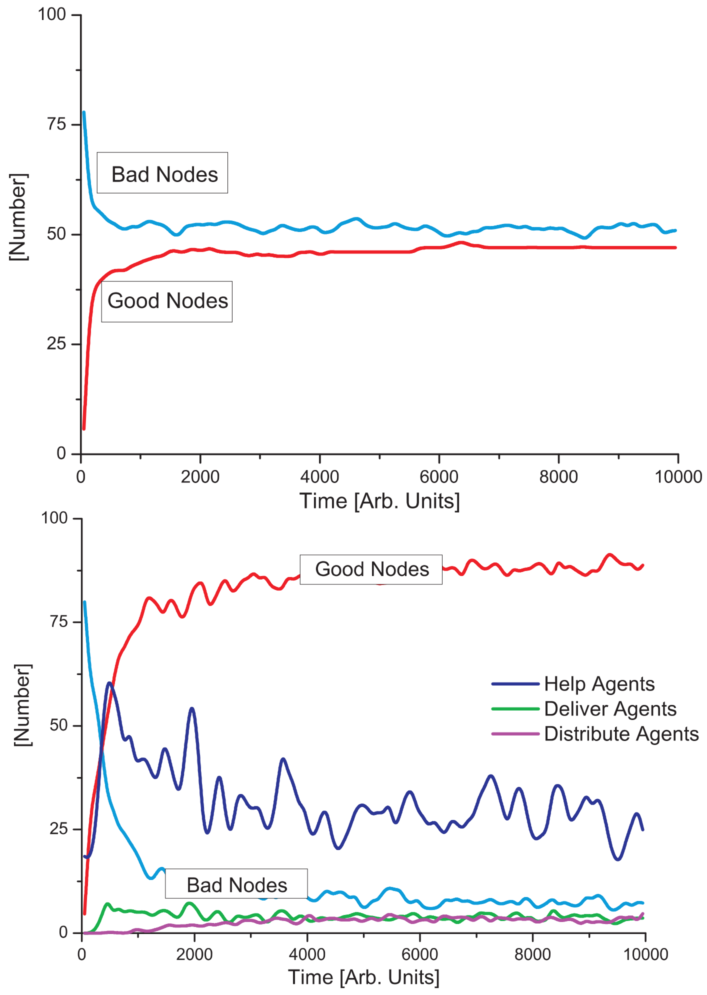


Fig. 7.8 *Simulation results from **[BOS12]** for a distributed sensor network of self-supplying nodes without (top) and with (bottom) energy distribution using agents.*

7.2 Bibliography

[ALB09] Z. Albus, A. Valenzuela, M. Buccini, *Ultra-Low Power Comparison: MSP430 vs. Microchip XLP* , Tech Brief, Texas Instruments (2009)

[ATM13] *8-bit Atmel with 8KBytes In-System Programmable Flash*, ATmega8/ATmega8L, Rev.2486AA–AVR–02/2013, Atmel

[BEH10] T. Behrmann, C. Zschippig, M. Lemmel, S. Bosse, *Toolbox for Energy Analysis and Simulation of self-powered Sensor Nodes*, in Proceedings of 55. IWK Internationales Wissenschaftliches Kolloquium, Workshop A3 Analysis and Synthesis to Energy Efficiency Optimisation, (IWK-10), 13.9.-17.9.2010, Ilmenau

[BEH13] T. Behrmann, C. Budelmann, M. Lemmel, S. Bosse, D. Lehmhus, *Tool chain for harvesting, simulation and management of energy in Sensorial Materials*, Journal of Intelligent Material Systems and Structures 24(18) 2245–2254, DOI: 10.1177 1045389X13488248

[BOS11] S. Bosse, T. Behrmann, *Smart Energy Management and Low-Power Design of Sensor and Actuator Nodes on Algorithmic Level for Self-Powered Sensorial Materials and Robotics*, Proceedings of the SPIE Microtechnologies 2011 Conference, 18.4.-20.4.2011, Prague, Session EMT 101 Smart Sensors, Actuators and MEMS, 2011, DOI:10.1117/12.888124

[BOS12] S. Bosse, F. Kirchner, *Smart Energy Management and Energy Distribution in Decentralized Self-Powered Sensor Networks Using Artificial Intelligence Concepts* , Proceedings of the Smart Systems Integration Conference 2012, Session 4, Zürich, Schweiz, 21 – 22 Mar. 2012, 2012, ISBN: 978-3-8007-3423-8.

[BUD11] Budelmann, C. and Krieg-Brückner, B. 2011. *Sensor Network Based on Fibre Optics for Intelligent Sensorial Materials*, EUROMAT 2011 - European Conference and Exhibition on Advanced Materials and Processes, 12.- 15.9.2011, Montpellier, France.

[FEC15] F. Feckl, *IoT long-life wireless sensors require ultra-low power architectures*, Electronics Component News, 21.5.2015 - 3:43pm, https://www.ecnmag.com/article/2015/05/iot-long-life-wireless-sensors-require-ultra-low-power-architectures

[INT13] *Intel® Core™ i5-4300U Processor (3M Cache, up to 2.90 GHz)*, WEB Specification, https://ark.intel.com/products/76308/Intel-Core-i5-4300U-Processor-3M-Cache-up-to-2_90-GHz

[JEF91] Jeffay, K. et al., *On non-preemptive scheduling of periodic and sporadic tasks with varying execution priority*, in Proceedings of the Real-Time Systems Symposium, December 1991, 129.

[LOH09] P. Loh and Y. Pan, *An Energy-Aware Clustering Approach for Wireless Sensor Networks*, International Journal of Communications, Network and System Sciences, Vol. 2 No. 2, 2009, pp. 131-141. doi: 10.4236/ijcns.2009.22015.

[MAX16] *Low-Power ADC ICs*, Maxim, https://www.maximintegrated.com/en/products/analog/data-converters/analog-to-digital-converters/precision-adcs/low-power-adc-ics.html

[SIM10] *Mathworks: Simulink Reference*, Mathworks, 2010

[SIT05] S. S. Iyengar and R. R. Brooks, (Ed.), *Distributed Sensor Networks*. CRC Press, 2005.

[SWA01] V. Swaminathan, K. Chakrabarty, *Real-time task scheduling for energy-aware embedded systems*, Journal of the Franklin Institute (2001)

Volume: 338, Issue: 6, Pages: 729-750

[SWA03] V. Swaminathan, K Chakrabarty, *Energy-conscious, deterministic I/O device scheduling in hard real-time systems*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 22(7), 847, 2003.

[TOC13] M. Tochigi, *Achieving low power wireless connectivity for battery powered healthcare sensors*, Murata Europe, (2013), MURA053-D3 Technical Note