FEDERAL UNIVERSITY OF SANTA CATARINA

The EPOS System Supporting Wireless Sensor Networks Applications

Lucas Francisco Wanner

FEDERAL UNIVERSITY OF SANTA CATARINA

COMPUTER SCIENCES DEPARTMENT

B.SC PROGRAM IN COMPUTER SCIENCES

The EPOS System Supporting Wireless Sensor Networks Applications

Lucas Francisco Wanner

Prof. Dr. Antônio Augusto M. Fröhlich

Advisor

Keywords: Wireless Sensor Networks, Application Oriented Operating Systems

Florianópolis, February, 2004.

To my grandparents,

José Tarcísio Hoff (in memorian)
Reinilda von Frühauf

Arlindo Gregório Wanner
Maria Nair Schmidt

Acknowledgements

I would like to take this opportunity to acknowledge all the people that supported me through all these years here at UFSC, and particularly during the execution of this research:

- Professor Antônio Augusto, for support, dedication and commitment beyond my (and I believe, any undergraduate student's) wildest expectations. More than an advisor, he has become a role model to me, and an example to follow.
- Fauze Valerio Polpeta, along with all my colleagues from the EPOS group at LISHA, for their support and invaluable technical advice.
- Professors Luis Fernando Friedrich and Leandro José Komosinski, for their advice, friendship and support.
- My colleagues and friends at PET and G8, for their friendship and many long conversations about life, the universe and everything...
- My mother, Mena, and my sister, Joice for their love and support throughout all these years. I wouldn't be here if it wasn't for them.
- My beloved Gabriella, for showing me the meaning of love and happiness.

Contents

List of Tables

List of Figures

Resumo

O micro-sensoreamento pervasivo através de Redes de Sensores sem Fios está revolucionando a maneira como compreendemos e gerenciamos sistemas físicos complexos desde habitats de animais até plantas industriais. A possibilidade de monitoramento físico detalhado em virtualmente qualquer ambiente oferece oportunidades para quase todas as disciplinas científicas e é um campo de pesquisa aberto.

Compostas por milhares de pequenos dispositivos com recursos muito limitados, redes de sensores estão sujeitas a novos problemas e restrições de sistema. Enquanto o hardware de Redes de Sensores sem Fios está evoluindo para plataformas estáveis e comercialmente disponíveis, a fronteira Hardware/Software é um tópico de pesquisa aberto. Sistemas Operacionais para Redes de Sensores devem implementar abstrações que tratem de sensores analógicos e digitais, devem prover uma pilha de protocolos para comunicação e fazer uso eficiente da capacidade limitada de energia do sistema. Ao mesmo tempo, devem prover uma interface de sistema e sistema de configuração simples para o programador da aplicação, que provavelmente não será um especialista em Sistemas Operacionais nem terá grande conhecimento do design do sistema da Rede de Sensores.

O Sistema Operacional EPOS tem como objetivo dar a cada aplicação dedicada suporte de *runtime* adequado sem ter que desenvolver um novo sistema para cada aplicação e sem necessitar que o programador de aplicação passe por complexos processos de configuração, usando a técnica de engenharia de domínio *Design de Sistema Orientado à Aplicação* para produzir um sistema operacional baseado em componentes que pode ser automaticamente configurado de acordo com as necessidade de aplicações específicas.

Este relatório apresenta uma visão geral de tecnologias e arquitetura de sistema de Redes de Sensores e apresenta um porte do sistema EPOS para a arquitetura AVR, usada em várias plataformas de Redes de Sensores sem Fios.

Abstract

Pervasing micro-sensing through Wireless Sensor Networks is revolutionizing the way we understand and manage complex physical systems from animal habitats to industrial plants. The possibility of detailed physical monitoring in virtually every possible environment offers opportunities for almost every scientific discipline, and is a field of open research.

Composed by thousands of small devices with very limited resources, sensor networks are subject to novel system problems and constraints. While Wireless Sensor Networks (WSN) hardware designs are evolving into stable, commercially available platforms, the Hardware/Software boundary in WSN is a topic of open research. Operating Systems for WSN must implement abstractions to interface with digital and analog sensors, provide a communication stack, and make efficient use of the system's limited energy resources. Meanwhile, they must also provide a simple system interface and configuration system for the application programmer, who most likely will not be an operating systems expert nor have great knowledge of the WSN system design.

The EPOS operating system aims to give each dedicated application adequate run-time support without having to design a new system for each application and without requiring application programmers to undergo complicated configuration procedures, using the *Application Oriented System Design* domain engineering technique to produce a component-based operating system that can be automatically tailored according to the needs of particular applications.

This report presents an overview of Sensor Network technologies and system architecture and a port of the EPOS system for the AVR microcontroller architecture, used in many Wireless Sensor Networks research platforms.

1 Introduction

Wireless Sensor Networks is an emerging technology that enables information gathering in several different scenarios ranging from wildlife monitoring to industrial and military applications. A Wireless Sensor Network consists of groups of sensor nodes using wireless links to perform distributed sensing tasks [36]. These nodes are typically provided with an embedded microprocessor and a very small amount of memory.

This embedded system must interface with digital and analog sensors, provide a communication stack and make efficient use of it's usually limited energy resources.

While Wireless Sensor Networks (WSN) hardware designs are evolving into stable, commercially available platforms, the Hardware/Software boundary in WSN is a topic of open research. When available, the Operating Systems for WSN are, according to their own creators, too simplistic and unsuited for non-expert programmers [28].

This report presents the first port of the EPOS[1] system to the AVR family of microcontrollers, an 8-bit Harvard Architecture widely used in embedded systems and Wireless Sensor Nodes.

The EPOS system aims to give each dedicated application adequate runtime support without having to design a new system for each application and without requiring application programmers to undergo complicated configuration procedures, using the *Application Oriented System Design* [14] domain engineering technique to produce a component-based operating system that can be automatically tailored according to the needs of particular applications.

1.1 Presentation Overview

Chapter 2 presents an overview of Wireless Sensor Network technologies, hardware, communcation models and applications.

[1]EPOS: Embedded Parallel Operating System

Chapter 3 presents the AVR microcontroller architecture, widely used in Wireless Sensor Network hardware designs.

Chapter 4 presents the EPOS system and an implementation of it's initialization system to the AVR platform.

Chapter 5 presents the conclusions of this report, and suggests future related research projects.

2 Wireless Sensor Networks

In the past few years the advances in miniaturization and low-cost, low-power design have led to extensive research in large-scale networks of small, wireless, low-power, unattended microsensors [24, 3]. These microsensors are equipped with a sensor module (e.g. acoustic, light, temperature, magnetic, image sensor), capable of sensing some quantity about the environment, a digital processor for processing the signals from the sensors and performing operating system, application and network functions, a radio module for communication and a battery to provide energy for operation [20]. Each sensor obtains a "view" of the environment, and sends the view data to a distant base-station, through which an end-user can access the information.

Wireless sensor networks enable the monitoring of a variety of possibly inhospitable environments that include home security, machine-failure diagnosis, chemical/biological detection, medical and wild habitat monitoring [20, 29]. These applications require reliable, accurate, fault-proof and possibly real-time monitoring. Meanwhile, the low energy and processing capacities of the nodes require efficient and energy-aware operation. Many researchers envision driving the networked sensor down to microscopic scale, creating smart environments and devices, powered by ambient energy [26] and used in many smart space scenarios. While it is acknowledged that energy consumption restrictions will not likely allow great processing power in this "smart dust", a wireless grid interface with more powerful computers could easily fulfill connectivity, storage and processing needs in the network nodes.

This chapter describes the basic microsensor node architecture, the communication principles of Wireless Sensor Networks (WSN), and WSN applications.

2.1 Wireless Sensor Nodes

In a Wireless Sensor Network, a Sensor Node is responsible for the lowest level of the sensing application. Several nodes are placed in areas of interrest, and each sensor node

Mote Type	Renee	Mica	Mica2	Mica2Dot
Microcontroller				
Type	Atmega163	Atmega128	Atmega128	Atmega128
CPU Clock (Mhz)	4	4	8	4
Program Memory (KB)	16	128	128	128
RAM (KB)	1	4	4	4
Non-volatile Storage				
Size (KB)	32	512		
Radio Communication				
Radio	RFM TR1000		Chipcom CC1000	
Frequency	916		916 / 433	
Transmit Power Control	Programmable resistor potentiometer.		Programmable via CC1000 registers.	
Encoding	SecDed (Software)		Manchester (Hardware)	

Table 1: Mica Motes Components

collects data from it's immediate surroundings. The collected data is then pre-processed in the node, and forwarded to a base station through the network formed with all the deployed nodes.

In a Sensor Node, the computational module is a programmable unit that provides computation, storage and bidirectional communication with other nodes in the system. This module interfaces with the analog and digital sensors in the node, performs basic signal processing and dispatches the data according to the application's needs [29]. The other modules in a Wireless Sensor Node are comprised by sensors and a radio for communication.

While several [3, 38, 34] platforms have been proposed and implemented for Wireless Sensor Nodes, the most popular and representative are the U.C. Berkeley's Mote[1] architectures [22, 35]. These are "current generation" devices constructed from off-the-shell components that have many of the key characteristics of the general class of Wireless Sensor Nodes [22]. They provide a microcontroller with internal program memory, sensor board interfaces, a low power radio module and a non-volatile memory chip.

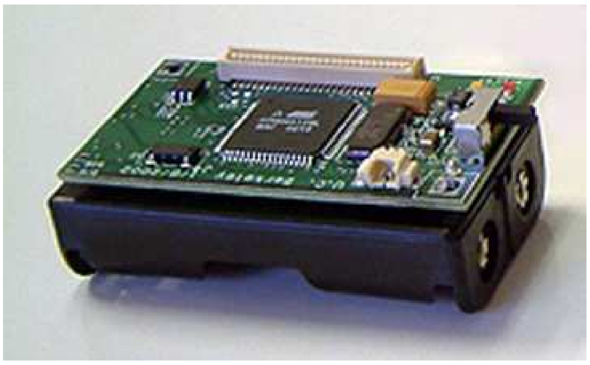


Figure 1: The Mica2 Sensor Node

2.1.1 Mica Motes

The UC Berkeley's Mote family (see Table 1) has evolved over the past few years into a stable platform for sensor networks research. It's current generation, the Mica2 (see Figure 1) uses a single channel radio from RF Monolithics (operating at 916Mhz in the USA and 433Mhz in Europe) to provide bidirectional communication at 40bps [31], an Atmel Atmega128 microcontroller (a member of Atmel's AVR architecture) running at 8Mhz, and a 512KB memory chip. A pair of conventional batteries is used as energy source. It's small size (aproximately 5 x 4 x 1.5 cm) allows deployment in remote locations with minimal interference with the existing habitat [31].

2.1.1.1 Hardware Organization

The processor within the Mica2 is an Atmel Atmega128 AVR. AVR is an 8-Bit Harvard architecture, with separete instruction and data memory. This architecture will be discussed at length in Chapter 3.

In the motes, the AVR interfaces with four hardware blocks (Radio, LEDS, Flash Memory and Sensor board / Programming interface). The general hardware organization is presented in Figure 2. A simple schematic of the Mica2 architecture is presented in Figure 3.

LEDs

Three Programmable LEDs are connected to the AVR in the Mica2 motes. These may be used for status and output of digital values.

[1]Mote, n. A small particle, as of floating dust; anything proverbially small; a speck: "The little motes in the sun do ever stir, though there be no wind" (Bacon).

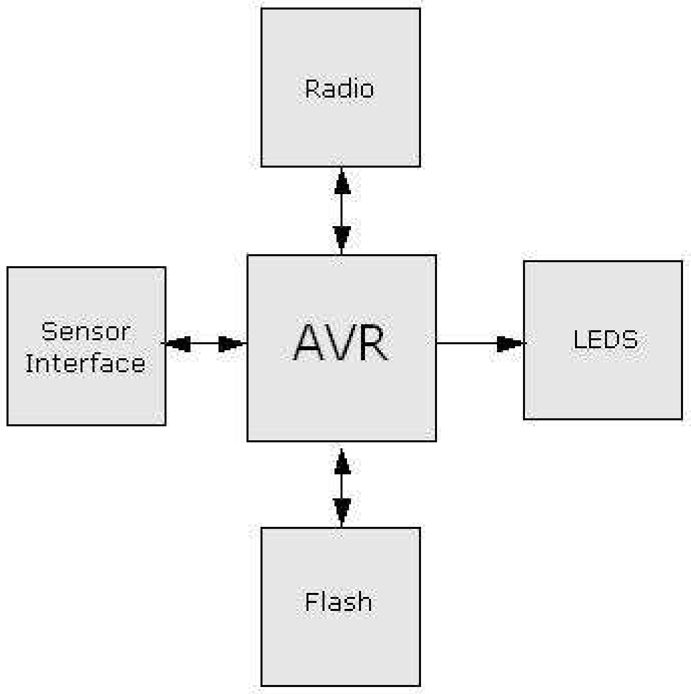


Figure 2: Mote Architecture General Organization

Flash Memory

In order to allow permanent storage and data logging in the motes, a 512KB Serial Flash memory chip is attached to one of the AVR's UART ports. If installed in conjuntion with a simple co-processor, this secondary memory could be also used for over-the-air reprogramming of the main microcontroller.

Radio

The Mica2 uses a low-power, single-chip UHF transciever from Chipcom as it's radio component. The CC1000 is designed for very low power and very low voltage wireless applications. The circuit is mainly intended for the ISM (Industrial, Scientific and Medical) and SRD (Short Range Device) frequency bands at 315, 433, 868 and 915 MHz, but can easily be programmed for operation at other frequencies in the 300-1000 MHz range. The main operating parameters of CC1000 can be programmed via an easy-tointerface serial bus, thus making CC1000 a very flexible and easy to use transceiver [6].

CC1000 is configured via a simple 3-wire interface. There are 36 8-bit configuration registers, each addressed by a 7-bit address. A Read/Write bit initiates a read or write operation. A full configuration of CC1000 requires sending 29 data frames of 16 bits each (7 address bits, R/W bit and 8 data bits). All registers are also

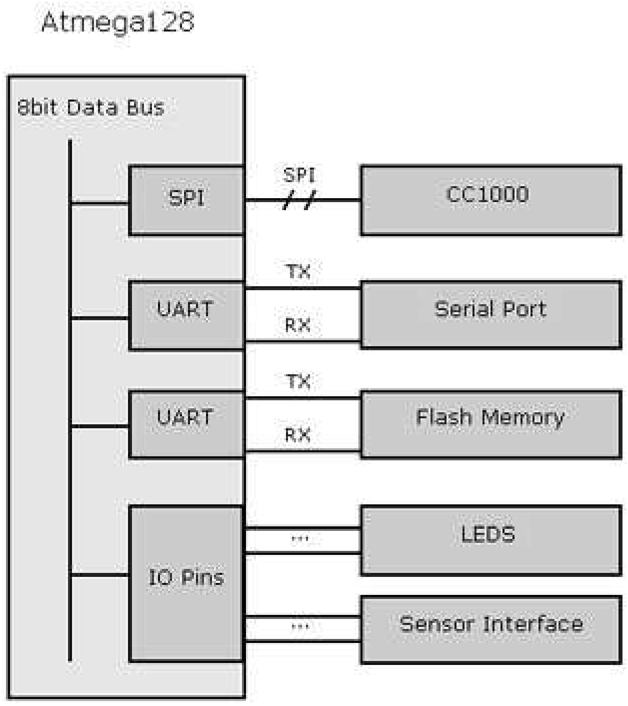


Figure 3: Mica2 General Schematic

readable.

Data is transferred to and from the AVR microcontroller via a dedicated SPI (Serial Peripheral Interface) Bus, and the Radio generates one interrupt every 8 bits when in receive mode.

Sensor and Programming Interface

The Mica2 interfaces with external devices through a 51-pin connector wired to CPU IO pins. This connector provides access to the AVR's GPIO (General Purpose Input or Output) Pins, UART and I^{2}C Bus, and is used for device programming and as a sensor board interface.

2.1.1.2 Sensor Boards

While the modular design of the motes allows a wide range of analog and digital sensors to be attached to the Sensor Node, the reference sensor board for the Mica platform is the "Mica Sensorboard" (See Figure 4 [39]). A fully-populated micasb has five different sensor modules in order to support a wide variety of potential sensor networks applications. These sensors include: light, temperature, acceleration, magnetic field, and acoustic, and each of these sensors can be purchased off-the-shelf [23].

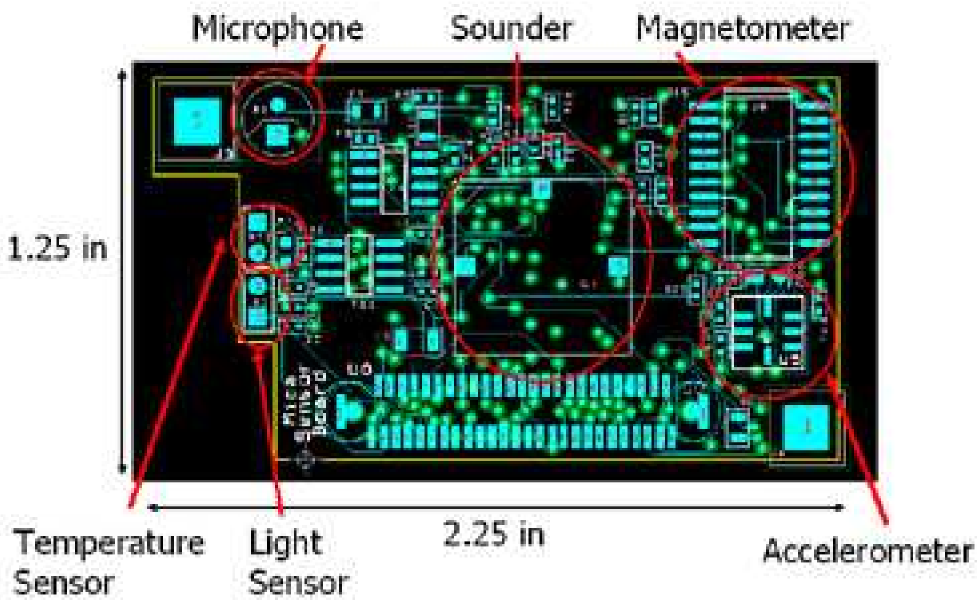


Figure 4: The Mica Sensorboard

A photo-resistor and thermistor are used to sense light and temperature. An Analog Devices ADXL202JE accelerometer is capable of delivering 2-axis acceleration sensing. For magnetic filed, the board is equipped with a Honeywell HMC1002 2-axis magnetometer. An omni-directional microphone, Panasonic WM-62A, is used to capture acoustic signal, which is amplified and bandpass-filtered to the voice band before being sampled.

In addition to the above sensors, the board is capable of generating acoustic output, using its 4kHz single tone buzzer. Optional hardware support to detect the generated tone on a receiving node is provided by an active bandpass filter and a LMC567 tone decoder from National Semiconductor, which has built in phase lock loop and adjustable threshold detection.

All modules in the sensor board can be power cycled independently, and are power isolated from the Mica's processor through an analog switch. Finally, gain of the magnetometer and the microphone amplification is adjustable by tuning the two digital potentiometers over the I^2C bus [18].

2.1.1.3 Programming

The Mica Motes are programmed through a gateway board that interfaces with the mote's sensor and programming interface and a PC's Serial Port. Further discussion on the programming of the AVR processors will be presented in Chapter 3.

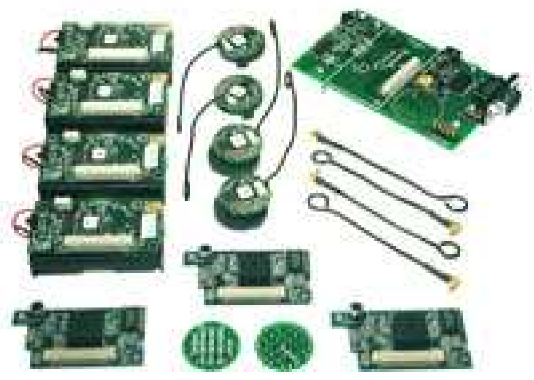


Figure 5: Mote Kit from Crossbow

Given that many times the environment in witch the motes are deployed is inhospitable or even unreachable, and that many times the sensing application must be adjusted or completely changed, over-the-air reprogramming of the motes is a strong need. Current researches make use of a simple coprocessor and the flash memory in the mote for full reprogramming [22], or over-the-air deployment of applications making use of virtual machines [28].

2.1.1.4 Availability

UCB Mote "Kits" are commercially available through Crossbow Technology Inc., an Integrated Sensors Manufacturer from Silicon Valley. These kits include from 2 to 20 Mica2 and Mica2Dot motes, programming and sensor Boards. As of December 2003, the suggested price for a MOTE-KIT5040, including 4 Mica2 and 4 Mica2Dot motes, 5 sensor boards and a programming board (See Figure 5) was U$ 1,995.00.

2.1.1.5 Future Development

The next step for the Mote family is, undoubtedly, single chip design. In the first semester of 2003, the first successful tests with Spec, the first single chip mote, were realized. Spec (See Figure 6) measures approximately 2 x 2.5 mm, has an AVR-like RISC core on it, 3K of memory, 8 bit On-chip ADC, radio transmitter, paged memory system, register windows, SPI programming interface, RS232 compatible UART, 4-bit input port and 4-bit output port. The expected production cost for each Spec Mote is U$ 0.30, batteries included.

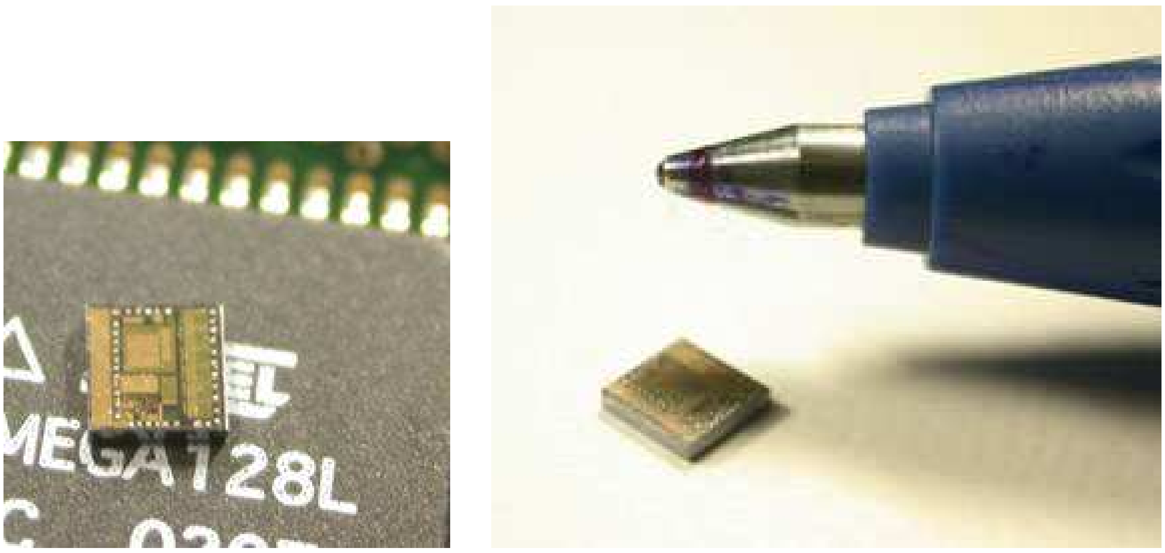


Figure 6: The Spec Mote

2.1.2 Related Work

Although most of the current research in Wireless Sensor Nodes comes from, or is related to the UCB Motes [2], there are several other related hardware designs developed or in development.

PicoRadio, also developed at Berkeley, aims to develop meso-scale low cost (less than 50 cents) transceivers for ubiquitous wireless data acquisition that minimizes power and energy dissipation.

The Manatee project, developed at the University of Copenhagen, aims is to study Bluetooth-based data dissemination and monitoring applications.

WINS (Wireless Integrated Network Sensors), from UCLA, is another single-chip Wireless Sensor Node design iniciative.

Smart Dust, closely related to the Mote Projects, aims to pack an entire system of sensing and communication into a cubic millimeter at relatively low-cost.

2.2 Communication in Wireless Sensor Networks

The Network component of Wireless Sensor Networks presents a series of new design challenges and is a topic of open research.

[2]The Motes/TinyOS group at Berkeley is coordinated by Prof. David Culler, and supported by Intel.

Sensor Networks must be power-aware. Most current network protocols are conservative only in their use of bandwidth. In a sensor node, all communication – including passive listening – will have a significant effect on the node's limited energy reserves.

Sensor Networks are highly dynamic. Over time, sensors may fail or new sensors may be added. Sensors are likely to experience changes in their position and reachability. These changes make static configuration unacceptable.

Sensor Networks must be self-configuring. A single human may be responsible for thousands of nodes in a dense sensor network, and a design where each sensor node requires individual attention would be impractical.

All of these characteristics, presented in [24] and discussed at length in [12, 40, 19], may affect many aspects of the system's design, including routing and addressing mechanisms, naming services, security mechanisms and so forth. This section, while not addressing any of these challenges at length, presents the basic concepts for WSN communication, addressing Data Link Layer issues and discussing routing mechanisms.

2.2.1 Data Link Layer

The data link layer is responsible for the multiplexing of data streams, data frame detection, medium access and error control [27], and ensures reliable point-to-point and point-to-multipoint connections in a communication network.

While Media Access Control (MAC) and transmission control protocols are well-studied for traditional computer networks, the different wireless technologies, applications characteristics and usage scenarios create a complex mix of issues related to the design of the MAC for Wireless Sensor Networks.

The capabilities of sensor devices are also very different from traditional nodes in a computer network. Typically on these platforms, a low power radio delivers bandwidth in a single channel. There is little or no dedicated support for carrier sensing, collision detection, and no specific framing or encoding enforced by the hardware. Furthermore, there is no specific protocol stacks in place to dictate the MAC protocol design [40].

This section presents an overview of the main MAC design issues in Wireless Sensor Networks, and presents possible solutions based on the work of [40].

2.2.1.1 Listening Mechanism

Listening Mechanisms like the CSMA/CD (Carrier Sense Multiple Access With Collision Detection) are very effective when all nodes can hear each other. In spite of being simple, listening comes with an energy cost, since the radio must be on in order to listen. To conserve energy, the carrier sensing time must be shortened. In many protocols, such as IEEE 802.11, the channel must be sensed even during backoff. The CSMA for Sensor Networks should take this opportunity to turn off the radio.

Given the fact that Sensor Networks use a simple Radio without hardware mechanism for collision detection, nodes that send data at the same time will corrupt each other. The solution for this is to introduce random delay for transmission to unsyncronize the nodes.

2.2.1.2 Contention Based Mechanism

Explicit contention control schemes used in many MAC protocols, such as IEEE 802.11, require the use of control packets, such as Request to Send (RTS), Clear to Send (CTS) and Acknowledgements (ACK). For networks where packets are large, this small control packets impose little overhead. However, this overhead can be very substantial in Sensor Networks, where the packets are expected to be a few bytes long.

Therefore, a contention scheme for WSN should use a minimum number of control packets. Woo and Culler [40] suggest that a node wishing to transmit first sends a RTS packet to it's parent and waits for a CTS reply. If no CTS is received for a timeout period, the node goes into backoff with a binary exponential increasing backoff window. Similarly, if it receives a CTS not destined to it, it will also go into backoff. If no CTS has been received after five retries, the transmission should be dropped. Furthermore, if a node hears a CTS before any of it's own transmissions, it will defer transmission for one packet time to avoid corrupting the traffic.

2.2.2 Routing

Sensor Networks present several characteristics that distinguish them from contemporary communication and wireless ad-hoc networks [1]:

- It is not feasible to build a global addressing scheme for the deployment of sheer number of sensor nodes.

- Contrary to typical communication networks, most applications of sensor networks require the flow of data from multiple regions (sources) to a particular sink.
- Generated traffic has significant redundancy in it since multiple sensors may generate same data within the vicinity of the phenomenon.
- Sensor nodes are very constrained in terms of transmission power, energy resources, processing cabacity and storage.

These characteristics have brought about many new algorithms for the problem of routing in sensor networks. This section summarizes the system architecture design issues for sensor networks, as presented in [1] and presents some relevant routing protocols for Sensor Networks.

2.2.2.1 Design Issues

As the performance of a routing protocol is closely related to the architectural model, this section captures the principal architectural and design issues to be considered by a routing protocol for Sensor Networks.

Network Dynamics: While most WSN setups use stationary sensor nodes, individual sensors may fail or run out of power at any time, thus creating the need for dynamic route changing.

Node Deployment: The topology of the nodes may be self-organizing, in situations where the nodes are scattered at random, thus creating an ad-hoc infrastructure.

Energy Considerations: As the energy required to transmit and receive data in a sensor node is much higher than that required to sense a phenomenon or process data in the node, routing protocols must be energy-aware.

Data Delivery Models: The data delivery model in a Sensor Network can be, depending on the application, continuous, event-driven or query-driven. The routing protocol is highly influenced by the data delivery model, especially with regart to the minimization of energy and route stability [1].

Data Aggregation: Since sensor nodes might generate significant redundant data, similar packets from multiple nodes can be aggregated so the number of transmissions can be reduced. Data aggregation is the combination of data from different sources using fuctions such as *suppression* (eliminating duplicates), *min*, *max* and *average* [1].

2.2.2.2 Data-centric Protocols

In data-centric routing, the sink sends queries to certain regions and waits for data from the sensors located in the selected regions. Since data is being requested through queries, and there is no global identifier in the node, attributed-based naming is necessary to specify the properties of data. This section presents the most relevant data-centric protocols in development today.

Directed Diffusion: In this solution, each sensor names data that it generates using one or more attributes. A sink may query for data by dissiminating interrests, and intermediate nodes propagate these interrests. For example [36], a seismic sensor may generate a data: (type = seismic, id = 66, location = SE, timestamp = 04.01.13), and sink may send an interest of the form: (type = seismic, location, SE). The intermediate nodes propagate the interrest to the selected region, and the sensors that match the request send back the sensed data.

SPIN: The SPIN (Sensor Protocols for Information via Negotiation) solutions are designed to disseminate individual sensor information to all the sensor nodes, assuming all of them are potential sinks [36].

Rumor Routing: This solution is a variant of the Directed Diffusion Technique intended for contexts in which geographic routing criteria are not applicable.

2.2.2.3 Hierarchical Protocols

The main aim of hierarchical routing is to efficiently maintain the energy consumption of sensor nodes by involving them in multihop communication within a particular cluster and by performing data aggregation and fusion in order to decrease the number of transmitted messages to the sink [1]. Cluster formation is typically based on energy reserve of sensors and sensor's proximity to the cluster head. LEACH (Low-Energy Adaptative Clustering Hierarchy) [21] is the most popular and representative hierarchical routing protocol for Sensor Networks.

2.3 Sensor Network Applications

Sensor networks may consist of many different types of sensors such as seismic, low sampling rate magnetic, thermal, visual, infrared, acoustic and radar, which are able to monitor a wide variety of ambient conditions that include the following [2]:

- temperature,
- humidity,
- vehicular movement,
- lightning condition,
- pressure,
- soil makeup,
- noise levels,
- the presence or absence of certain kinds of objects,
- mechanical stress levels on attached objects, and
- the current characteristics such as speed, direction, and size of an object.

Sensor nodes can be used for continuous sensing, event detection, location sensing, and local control of actuators [2]. The concept of pervasing micro-sensing through Wireless Sensor Networks promise many new application areas. This section presents and categorizes the applications of WSN into military, environment, health, and commercial areas.

2.3.1 Military applications

Wireless sensor networks can be an integral part of military command, control, communications, computing, intelligence, surveillance, reconnaissance and targeting (C4ISRT) systems. The rapid deployment, self-organization and fault tolerance characteristics of sensor networks make them a very promising sensing technique for military C4ISRT [2].

Since sensor networks are based on the dense deployment of disposable and low-cost sensor nodes, destruction of some nodes by hostile actions does not affect a military operation as much as the destruction of a traditional sensor, which makes sensor networks

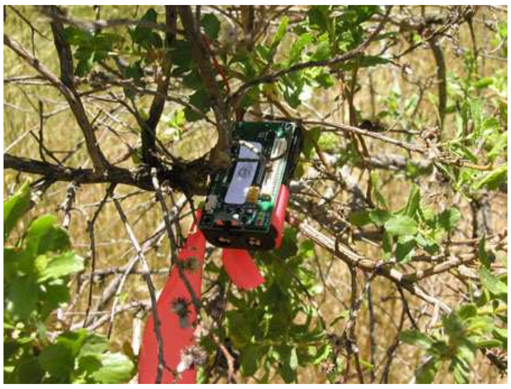


Figure 7: Deployed Firebug Mote

concept a better approach for battlefields. Some of the military applications of sensor networks are monitoring friendly forces, equipment and ammunition; battlefield surveillance; reconnaissance of opposing forces and terrain; targeting; battle damage assessment; and nuclear, biological and chemical attack detection and reconnaissance [2].

2.3.2 Environmental applications

Environmental and habitat monitoring is a driving field for wireless sensor networks [4]. It's applications include tracking the movements of small animals; monitoring environmental conditions; chemical/biological detection; precision agriculture; forest fire detection; meteorological or geophysical research; flood detection; bio-complexity mapping of the environment; and pollution study.

2.3.2.1 Forest fire detection

Since sensor nodes may be strategically, randomly, and densely deployed in a forest, sensor nodes can relay the exact origin of the fire to the end users before the fire is spread uncontrollable [2].

The FireBug project [5] uses of a network of GPS-enabled, wireless thermal sensors, a control layer for processing sensor data, and a command center for interactively communicating with the sensor network. Each mote has sufficient power, radio communication and processing capabilities to support location and thermal sensors and data handling. Along with geographic position, motes (See Figure 7) measure humidity, temperature and light intensity.

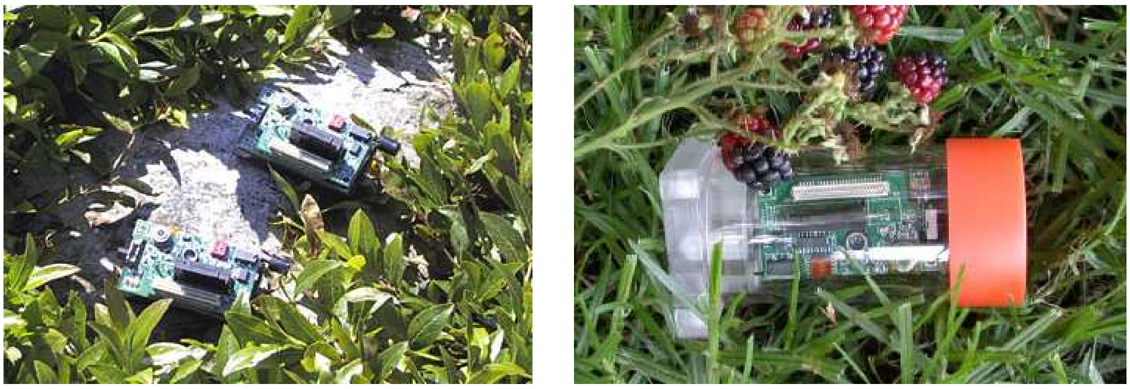


Figure 8: Motes deployed on Great Duck Island

2.3.2.2 Habitat Monitoring

Researchers in the Life Sciences are becoming increasingly concerned about the potential impacts of human presence in monitoring plants and animals in field conditions [29]. In this context, sensor networks represent a significant advance over traditional invasive methods of monitoring. Sensors can be deployed prior to the onset of the breeding season or other sensitive period (in the case of animals) or while plants are dormant or the ground is frozen (in the case of botanical studies). Sensors can be deployed on small islets where it would be unsafe or unwise to repeatedly attempt field studies [29].

The Great Duck Island Habitat Monitoring project [29] is a pilot application for the monitoring of migratory seabirds in the coast of Maine. In the spring of 2002, 32 wireless sensor nodes were deployed on Great Duck Island, Maine (See figure 8). These nodes monitor the microclimates in and around nesting burrows. At the end of the field season in November 2002, well over 1 million readings had been logged from the 32 motes deployed on the island and made available on the internet through the GDI Project Website [33].

2.3.3 Health applications

Some of the health applications for sensor networks are providing interfaces for the disabled; integrated patient monitoring; drug administration in hospitals; monitoring the movements and internal processes of insects or other small animals; telemonitoring of human physiological data; and tracking and monitoring doctors and patients inside a hospital [2].

2.3.4 Commercial applications

Commercial applications for WSN include monitoring material fatigue; managing inventory; monitoring product quality; constructing smart office spaces; robot control and guidance in automatic manufacturing environments; interactive toys; factory process control and automation; smart structures with sensor nodes embedded inside; machine diagnosis; transportation; factory instrumentation; local control of actuators; and vehicle tracking and detection [2].

3 The AVR Architecture

AVR is a widely used family of 8-bit RISC microcontrollers from Atmel. Usually deployed in the form of MCUs (Microprocessor Control Units[1]), the AVR provides good performance at low cost and low power consumption in a simple Harvard Architecture[2], making it the natural choice for Wireless Sensor Nodes processing and control.

This chapter describes the AVR microcontroller core architecture and the AT90S8515 AVR MCU, used in the first implementation of the EPOS initialization system for AVR architecture (See Chapter 4).

3.1 Architectural Overview

The AVR CPU resembles most RISC processors but has smaller registers. The core features 32 identical 8-bit registers that can hold addresses or data. Since 8-bit address pointers are fairly worthless even in an 8-bit device, the last six registers can be used in pairs, as address pointers. Dubbed X, Y, and Z, these three meta-registers can be used for any load or store operation [37]. All operations are register-toregister; the chip follows a strict load/store model.

Figure 9 presents the Block Diagram of the AVR Architecture.

3.1.1 General Purpuse Registers

The AVR's fast-access Register file contains 32 x 8-bit general purpose registers with a single clock cycle access time, allowing sincle-cycle Arithmetic Logic Unit (ALU) operation. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing.

The register file is mapped into the data address space. The first 32 bytes of data

[1] In an MCU, the processor, memory and I/O all reside in the same physical IC (integrated circuit).
[2]A Harvard Architecture provides separate momories and buses for program and data.

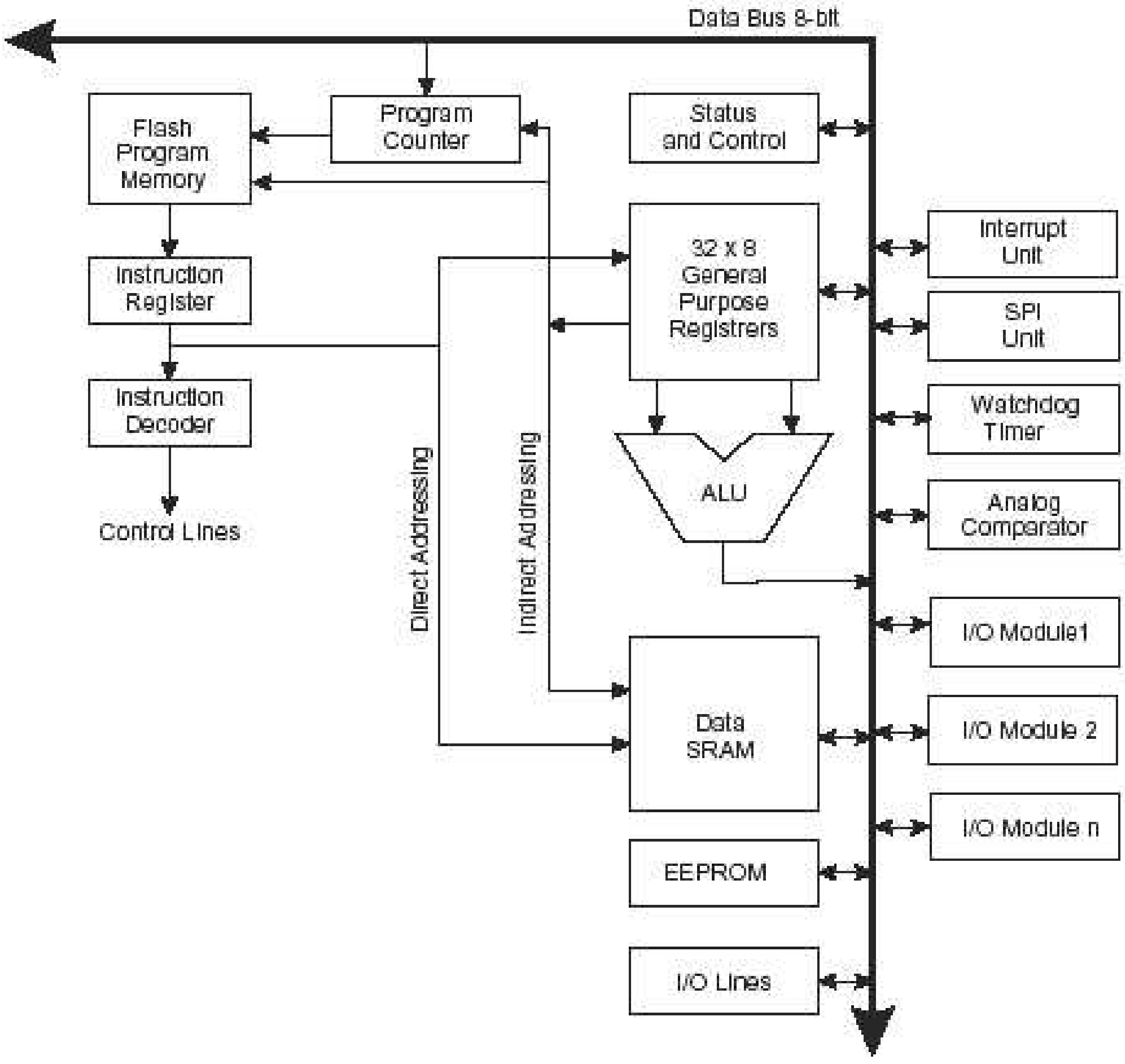


Figure 9: Block Diagram of the AVR Architecture [9]

memory, \$0x0000 – \$0x001F, correspond to registers R0-R31.

3.1.2 I/O Registers

The AVR's 64 I/O Registers are memory-mapped into addresses \$0x0020 – \$005F. These registers include status, interrupt and timer control, stack pointer, GPIO (General-Pourpose Input and Output) and SPI (Serial Programming Interface) and UART registers. Table 2 presents the I/O registers for the AT90S8515 MCU. Unused and reserved locations are not shown in the table.

3.1.2.1 Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. The AVR Status Register is defined as:

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C

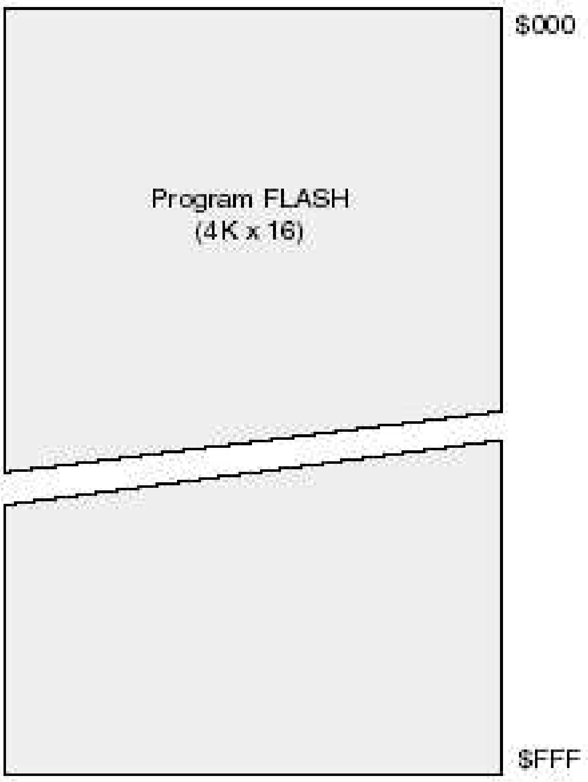
- Bit 7 - I: Global Interrupt Enable
- Bit 6 - T: Bit Copy Storage
- Bit 5 - H: Half Carry Flag
- Bit 4 - S: Sign Bit, $S = N \oplus V$
- Bit 3 - V: Two's Complement Overflow Flag
- Bit 2 - N: Negative Flag
- Bit 1 - Z: Zero Flag
- Bit 0 - C: Carry Flag

3.1.2.2 Stack Pointer

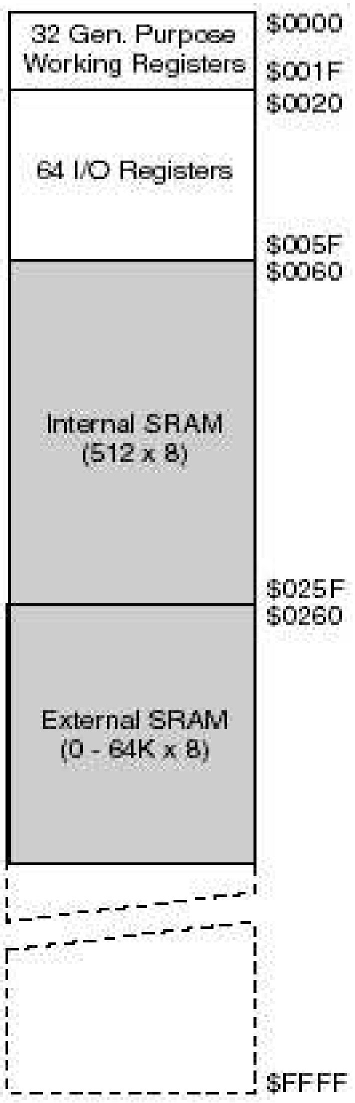
The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the stack. The stack is implemented as growing from higher memory to lower memory locations, thus a Stack PUSH command decreases the Stack Pointer. The AVR Stack Pointer is implemented as two 8-bit registers in the I/O Space, SPH and SPL.

3.1.3 Data Memory

In the AT90S8515 MCU, the first 96 memory locations address the Register file and I/O memory. The next 512 locations address the internal data SRAM. An optional external data SRAM can be placed in the same SRAM memory space, filling the AVR's 64K address space. Figure 10(b) presents the AT90S8515 data memory map.



(a) Program Memory



(b) Data Memory

Figure 10: AVR Memory Maps [9]

Memory Location	Register Name	Description
\$0x5F	SREG	Status Register
\$0x5E	SPH	Stack Pointer High
\$0x5D	SPL	Stack Pointer Low
\$0x5B	GIMSK	General Interrupt Mask register
\$0x5A	GIFR	General Interrupt Flag Register
\$0x59	TIMSK	Timer/Counter Interrupt Mask register
\$0x58	TIFR	Timer/Counter Interrupt Flag register
\$0x55	MCUCR	MCU general Control Register
\$0x53	TCCR0	Timer/Counter0 Control Register
\$0x52	TCNT0	Timer/Counter0 (8-bit)
\$0x4F	TCCR1A	Timer/Counter1 Control Register A
\$0x4E	TCCR1B	Timer/Counter1 Control Register B
\$0x4D	TCNT1H	Timer/Counter1 High Byte
\$0x4C	TCNT1L	Timer/Counter1 Low Byte
\$0x4B	OCR1AH	Timer/Counter1 Output Compare Register A High Byte
\$0x4A	OCR1AL	Timer/Counter1 Output Compare Register A Low Byte
\$0x49	OCR1BH	Timer/Counter1 Output Compare Register B High Byte
\$0x48	OCR1BL	Timer/Counter1 Output Compare Register B Low Byte
\$0x45	ICR1H	T/C 1 Input Capture Register High Byte
\$0x44	ICR1L	T/C 1 Input Capture Register Low Byte
\$0x41	WDTCR	Watchdog Timer Control Register
\$0x3E	EEARH	EEPROM Address Register High Byte (AT90S8515)
\$0x3E	EEARL	EEPROM Address Register Low Byte
\$0x3D	EEDR	EEPROM Data Register
\$0x3C	EECR	EEPROM Control Register
\$0x3B	PORTA	Data Register, Port A
\$0x3A	DDRA	Data Direction Register, Port A
\$0x39	PINA	Input Pins, Port A
\$0x38	PORTB	Data Register, Port B
\$0x37	DDRB	Data Direction Register, Port B
\$0x36	PINB	Input Pins, Port B
\$0x35	PORTC	Data Register, Port C
\$0x34	DDRC	Data Direction Register, Port C
\$0x33	PINC	Input Pins, Port C
\$0x32	PORTD	Data Register, Port D
\$0x31	DDRD	Data Direction Register, Port D
\$0x30	PIND	Input Pins, Port D
\$0x2F	SPDR	SPI I/O Data Register
\$0x2E	SPSR	SPI Status Register
\$0x2D	SPCR	SPI Control Register
\$0x2C	UDR	UART I/O Data Register
\$0x2B	USR	UART Status Register
\$0x2A	UCR	UART Control Register
\$0x29	UBRR	UART Baud Rate Register
\$0x28	ACSR	Analog Comparator Control and Status Register

Table 2: AVR I/O Registers

3.1.4 Program Memory

In the AT90S8515 MCU contains 8K bytes of Programmable Flash Memory for program storage. Since all instructions are 16- or 32-bit words, the Flash is organized as 4Kx16. The AT90S8515 Program Counter is 12 bits wide, thus addressing the 4096 program memory addresses.

In early AVR models, such as the AT90S8515, the program memory can only be updated by writing a full binary image to the flash. Once the program data is downloaded, no further updates of the flash are possible, as there is no instruction capable of writing to the program memory. These devices can be programmed serially, via ISP (In-System Programming) or parallelly, via High-Voltage Programming.

In-System Programming uses the AVR internal SPI (Serial Peripheral Interface) to download code into the flash and EEPROM memory of the AVR. ISP programming requires only VCC , GND, RESET and 3 signal lines for programming. All AVR devices except AT90C8534, Attiny11 and ATtiny28 can be ISP programmed. The AVR can be programmed at the normal operating voltage, normally 2.7V-6.0V. No high voltage signals are required. The ISP programmer can program both the internal flash and EEPROM [11].

For High-Voltage programming a 12V programming voltage is applied to the RESET pin of the AVR device. All AVR devices can be programmed with High-Voltage programming, and the target device can be programmed while it is mounted in its socket.

3.1.4.1 Self-Programming

Recent AVR MCUs, such as the Atmega128, used in the Mica Motes, provide a Store Program Memory SPM) instruction capable of erasing and writing a page in the program memory.

In the MCUs where the SPM instruction is available, the Flash memory is divided into two sections, one Application section and one Boot Loader section. The SPM instruction can only be executed from the Boot Loader section [10]. The Flash memory is divided into pages containing 32, 64, or 128 words each. Memory organization is shown in Figure 11.

All Self-programming operations are performed using the SPM instruction. Different operations (page erasing, buffer filling and page writing) are selected using the SPMCR

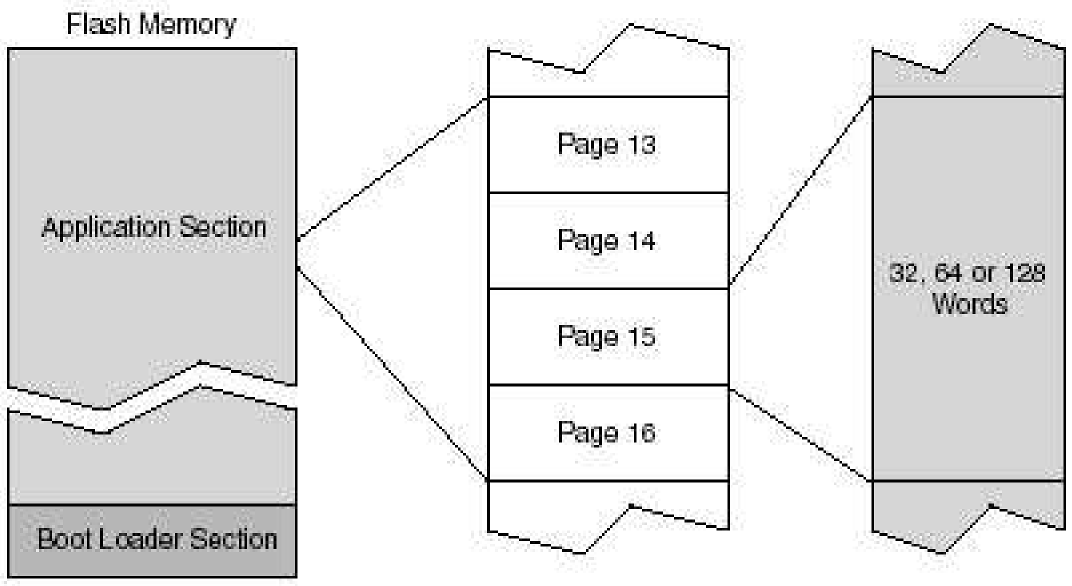


Figure 11: Self-Programmable AVR Program Memory Organization [10]

Register. Flash memory updates are done page by page.

Before writing new data to a page, the page must be erased. The Z-register (R31:R30) is used to select the page to be erased.

To write new data to a page, the Page Buffer must be filled first. The Page Buffer is a separate (not SRAM) write-only buffer holding one temporary page, and must be filled word by word, using the R1:R0 registers.

When the Page Buffer is loaded with new data, it must be written to Flash memory. For this, the Z-register is used to select the page to be written, and the page writing operation is selected in the SPMCR Register.

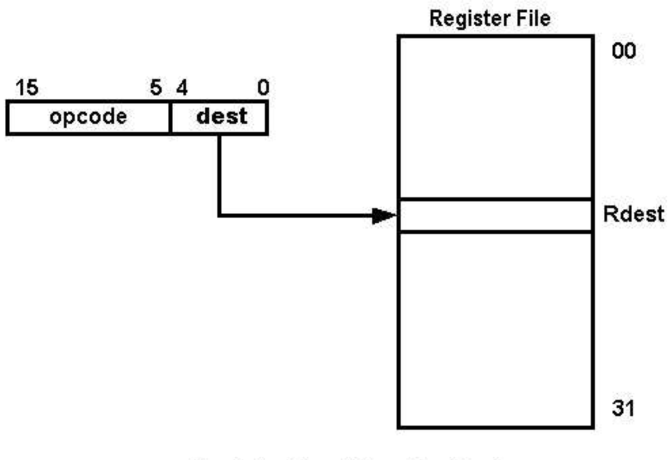
3.1.5 Addressing Modes

This section describes the several addressing modes provided by the AVR architecture for access to the program memory (Flash) and data memory (SRAM, Regiter File and I/O Memory).

3.1.5.1 Register Direct – Single Register

One of the 32 general purpose registers (dest) contains the instruction operand.

Example: `CLR  R0           ; R0 is cleared`

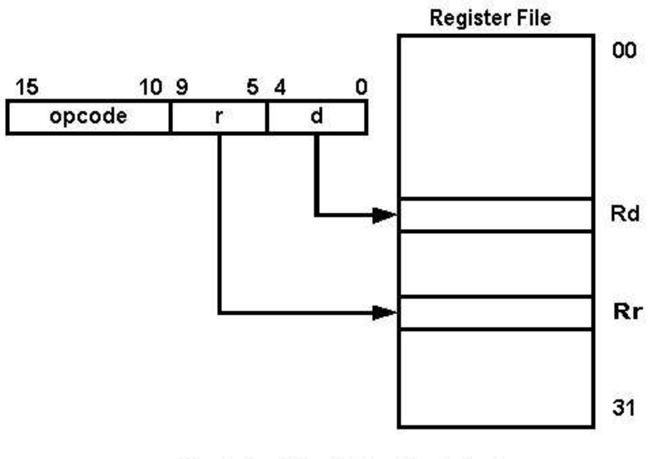


Register Direct (One Register)

3.1.5.2 Register Direct – Two Registers

Two of the 32 general purpose registers contains the instruction operands; one is the Source Register and the other is the Destiny Register.

Example: `ADD  R0,R1          ; R0 = R0 + R1`

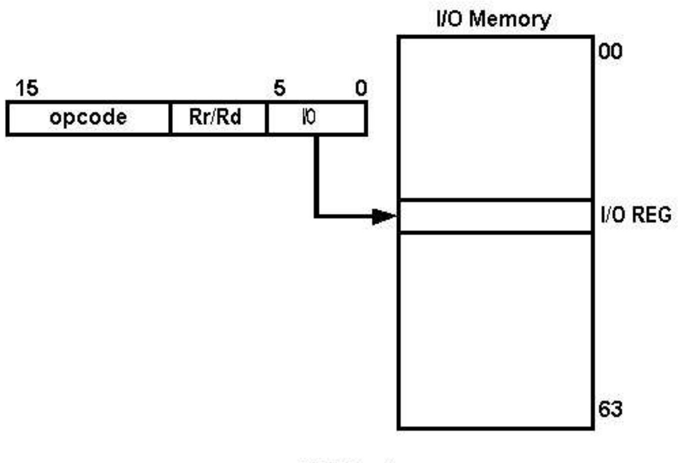


Register Direct (Two Registers)

3.1.5.3 I/O Direct

An address from the 64 Special Function Registers is contained in the 6 bit I/O portion of the instruction. The address of the Source or Destiny Register is contained in the remaining 6 bits of the instruction operands.

Example: `IN   R16,MCUCR     ; R16 = MCUCR (I/O Address 0x35)`

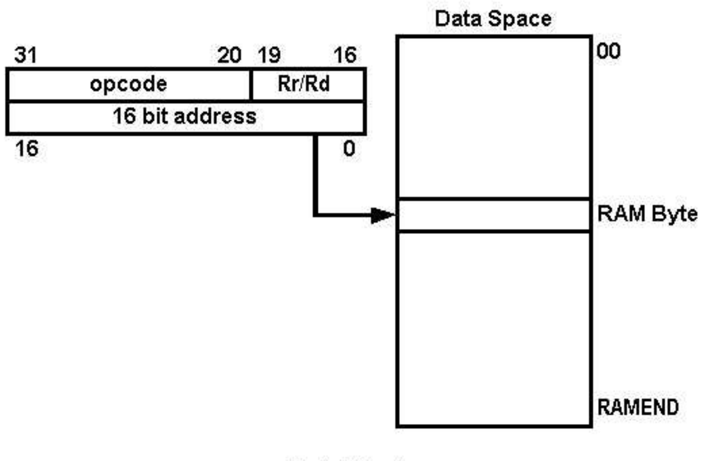


I/O Direct

3.1.5.4 Data Direct

A 16 bit Data Address is specified as an operand (to access up to 64K of RAM memory). The address of the Source or Destiny Register is contained in the remaining 6 bits of the instruction operands.

Example: `LDS  R0,$1234       ; R0 = &0x1234`

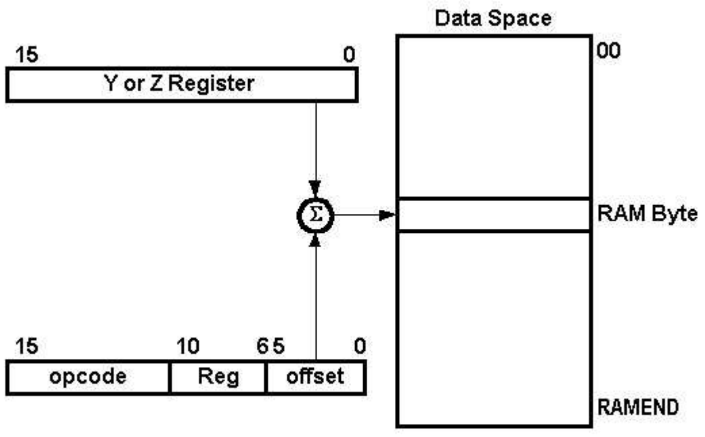


Data Direct

3.1.5.5 Data Indirect with Displacement

Data operand address is the result of the addition of the Y or Z register and the 6 bit Offset found on the instruction. Reg is the Source or Destiny Register

Example: `LDD  R0,Y + $3F    ; R0 = &($3F + [Y])`

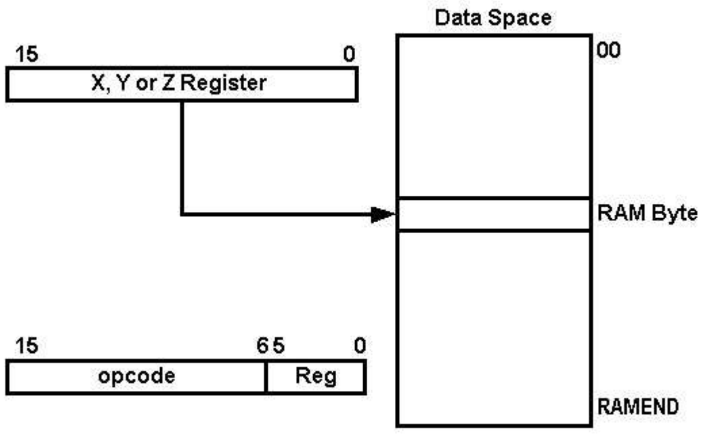


Data Indirect with Displacement

3.1.5.6 Data Indirect

Data operand address is the contents of the X, Y or Z register. Reg is the Source or Destiny Register.

Example: `LD    R0,X            ; R0 = &[X]`



Data Indirect (No Displacement)

This mode can also be used with Pre-Increment and Post-Increment of the Address Register.

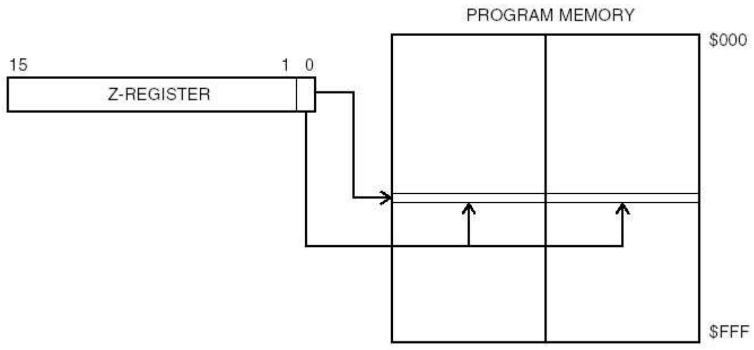
Pre-Decrement Example: `LD    R0,Z-      ; R0 = &[--Z]`

Post-Increment Example: `LD    R0,Z+      ; R0 = &[Z++]`

3.1.5.7 Constant Addressing Using LPM

Program memory word address operand is specified by the 15 MSB bits of the Z register (Z15:1). The LSB (Z0) selects which byte is to be fetched and stored in R0.

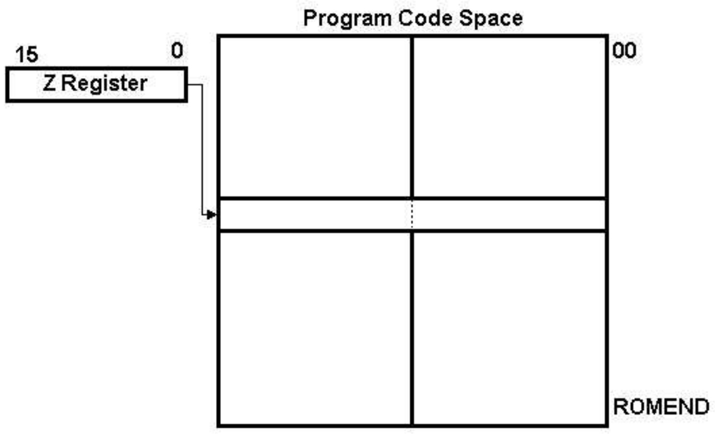
Example: `LPM   ; R0 = &(Program Memory Address [Z] >> 1)` (Z0 selects high or low byte.)



3.1.5.8 Indirect Program Addressing

Program Execution continues from the memory location specified on the Z register.

Example: `IJMP    ; PC = Z`

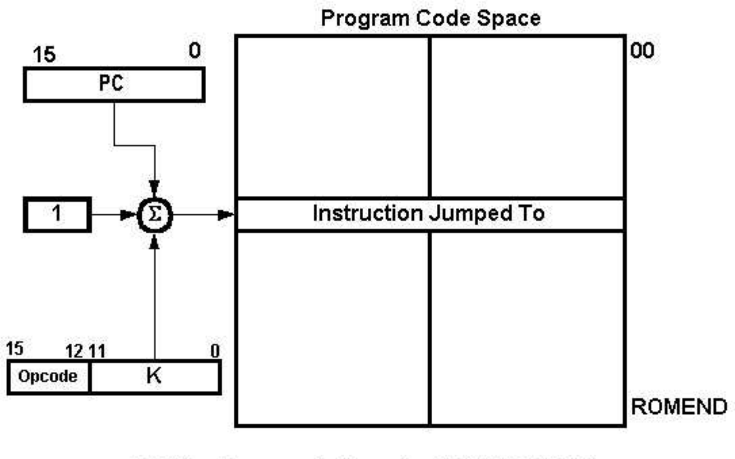


Indirect Program Addressing (IJMP,ICALL)

3.1.5.9 Relative Program Addressing

Program execution continues from the memory location specified by adding the PC, the relative offset K and one. The relative offset goes from -2048 to 2047.

Example: `RJMP  $020        ; PC = PC + 20 + 1`



Relative Program Addressing (RJMP, RCALL)

3.2 Timers

The AT90S8515 AVR MCU provides two timers (one 8-bit and one 16-bit). In principle, a timer is a simple counter. Its advantage is that the input clock and operation of the timer is independent of the program execution. The deterministic clock makes it possible to measure time by counting the elapsed cycles and take the input frequency of the timer into account [7].

3.2.1 Timer Events

The timers of the AVR can be specified to monitor several events:

Timer Overflow: The counter has counted up to its maximum value and will be reset to zero in the next timer clock cycle.

Compare Match: The Output Compare Register can be loaded with a value which the timer will be checked against every timer cycle. When the timer reaches the compare value, an event is signaled, and the Timer can be configured to clear the count register to "0".

Input Capture: The AVR has an input pin to trigger the input capture event. A signal change at this pin causes the timer value to be read and saved in the Input Capture Register. This is useful to measure the width of external pulses.

The timer operates independently of the program execution, and for each timer event there is a corresponding status flag in the Timer Interrupt Flag Register. The occurrence of timer events can be monitored by constantly polling of status flags or by breaking of program flow and execution of Interrupt Service Routines.

3.2.2 Watchdog Timer

A watchdog timer is a piece of hardware that can cause a processor reset when it judges that the system has hung, or is no longer executing the correct sequence of code [30].

The hardware component of a watchdog timer is a counter that is set to a certain value and then counts down towards zero. It is the responsibility of the software to set the count to its original value often enough to ensure that it never reaches zero. If it does reach zero, it is assumed that the software has failed in some manner and the CPU is reset, or an interrupt is generated.

In the AT90S8115, the Watchdog Timer (WDT) is independent from the rest of the system. It has its own internal oscillator, which runs as long as one of the WDT operating modes is enabled (Reset or Interrupt). This ensures safe operation even if the main CPU oscillator fails [8].

3.3 Serial Peripheral Interface

Serial Peripheral Interface (SPI) is a serial bus standard established by Motorola and supported in silicon products from various manufacturers. It is a synchronous serial data link that operates in full duplex (signals carrying data in both directions simultaneously) [25].

Devices communicate using a master/slave relationship, in which the master initiates the data frame. When the master generates a clock and selects a slave device, data may be transferred in both directions simultaneously.

SPI specifies four signals: clock ($SCLK$); master data output, slave data input ($MOSI$); master data input, slave data output ($MISO$); and slave select ($\overline{SS}$). SCLK is generated by the master and input to all slaves. $MOSI$ carries data from master to slave. $MISO$ carries data from slave back to master. A slave device is selected when the master asserts its $\overline{SS}$ signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave. Figure 12 presents a single master, multiple slave SPI implementation.

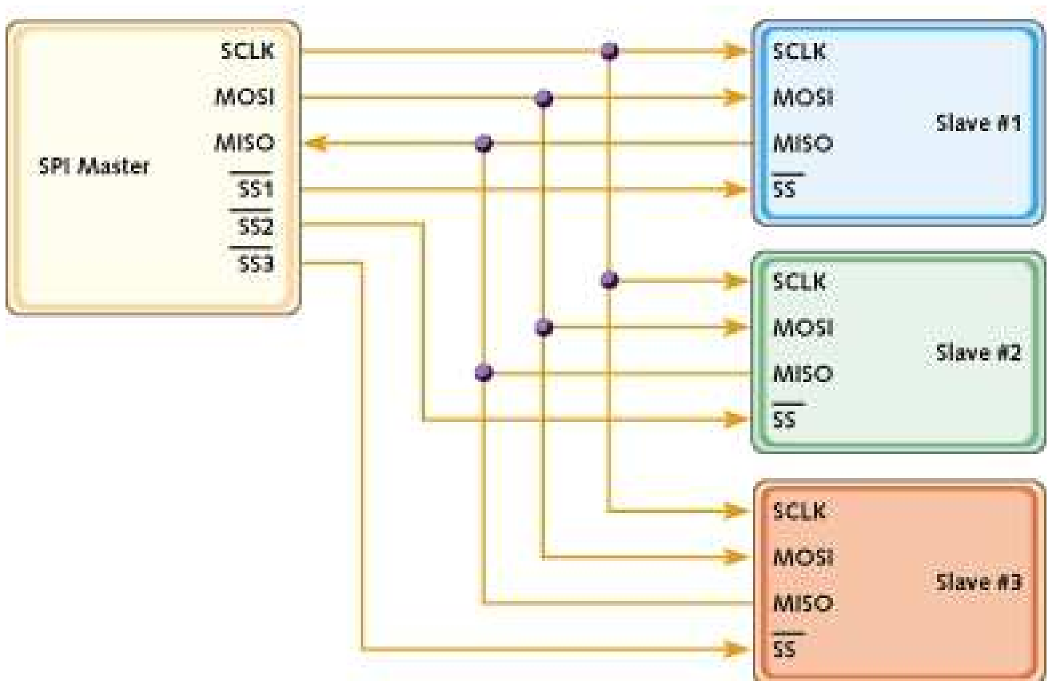


Figure 12: Multiple slave SPI implementation [25]

The AT90S8515 provides a fully functional SPI implementation, capable of working in either master or slave mode and controlled by I/O memory mapped registers.

3.4 UART

The AT90S8515 MCU provides a full-duplex Universal Asynchronous Receiver/Transmitter (UART), featuring:

- Baud Rate generator
- Noise Filtering
- Overrun detection
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete.

Data transmission and reception, as well as UART setup is controlled by I/O memory mapped registers.

3.5 GPIO

The AT90S8515 architecture provides four bi-directional I/O ports. Three I/O memory address locations are allocated for each port, one each for the Data Register, PORTx, Data Direction Register, DDRx, and the Port x Input Pins, PINx. The last enables access to the physical value on each Port x pin. The Port x Input Pins address is read only, while the Data Register and the Data Direction Register are read/write. All Port x Pins can be used for General Purpose Input or Output (GPIO).

3.6 Sleep Modes

AVR microcontrollers provide several sleep modes. The purpose of these modes is to provide a way of suspending program execution when necessary, thereby reducing power consumption [13]. The Sleep Modes available in the AT90S8515 MCU are (in order from maximal to minimal power consumption):

- Idle mode

  The idle mode stops the CPU but leaves peripherals (UART, Analog Comparator etc.) running. The MCU will continue program execution immediately after waking up from Idle mode.

- Powersave mode

  This mode is identical to the Powerdown mode, with one exception: The Timer Crystal Oscillator will continue to operate and the Timer can continue to count. The device can wake up from either a Timer Overflow or Output Compare event.

- Powerdown mode

  In this mode, all Oscillators are stopped while the External Level interrupts and the Watchdog continue operating. Only an External Reset, a Watchdog Reset or an External Level interrupt can wake up the MCU.

The device is sent into sleep mode by selecting the desired sleep mode in the MCU Control Register, enabling interrupts that should be able to wake the MCU up from sleep and executing a SLEEP intruction.

4 EPOS initialization in the AVR

The EPOS system was born in 1997 as a project to experiment with the concepts and mechanisms of application-oriented system design [14]. EPOS is thus an intrinsically application-oriented operating system, and today is evolving into a fully functional, multi-platform, very high performance OS. Current results include implementations for high-performance Clusters of Commodity Workstations based on Myrinet Networks [16, 17, 15] and ports to the PowerPC (32-bit) and H8 (8-bit) architectures [32]. EPOS aims to deliver functionality (giving the application it's necessary runtime support), customizability (being tailored to specific applications) and efficiency (making resources available to the application with the lowest possible overhead) [14].

This chapter presents an overview of the EPOS system, focusing on the EPOS initialization process and it's implementation for the AVR architecture.

4.1 EPOS System Architecture

EPOS relies on System Abstractions, Hardware Mediators and Aspects to ensure component reusability. Abstractions describe scenario-independent functionalities, are widely reusable and represent most of the components in the system. A hardware mediator is a system-dependent abstraction of elements of the hardware platform that are used by system abstractions and scenario aspects [14]. Aspects provide configurable functionalities for applications, such as sharing, protection and atomicity.

4.1.1 System Abstractions

EPOS families of System Abstractions are the result of the decomposition of the dedicated computing domain . System abstractions are modeled independently of execution scenario aspects and and specific system architectures [14]. Figure 13 presents a top level representation of EPOS families of abstractions.

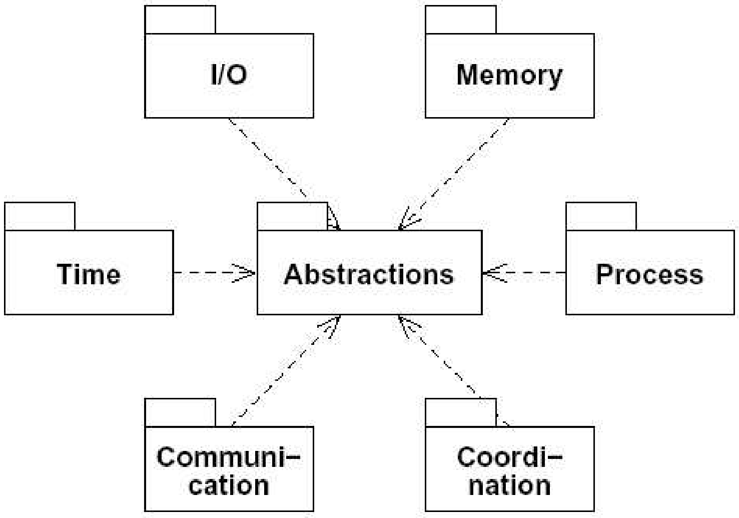


Figure 13: EPOS Families of Abstractions [14]

4.2 EPOS Initialization

The first phase of the EPOS initialization process in comprised by the *bootstrap* and a setup utility. The setup utility runs previous to the operating system and builds an elementary execution context for EPOS, initializing hardware components.

The second phase, the init utility, concerns the initialization of system data structures and the creation of the first (and possibly unique) application process.

4.2.1 The Setup Utility for the AVR

EPOS Setup Utility is responsible for building an elementary execution context for the OS. It runs after the bootstrap and previous to the Init Utility.

In the AVR, the bootstrap simply disables interrupts and calls setup, passing the SysInfo Structure as a parameter. The SysInfo structure describes the relevant characteristics of the forthcoming EPOS configuration.

As the Setup utility initiates, it proceeds with hardware setup, updating and completing SysInfo, including information about the physical resources configured, a memory map describing how the operating system has been loaded, the node's logical id, etc [14].

In the AVR, Setup is mainly responsible for setting up the interrupt controller, checking system integrity, setting up the Init entry point and setting up system data structures.

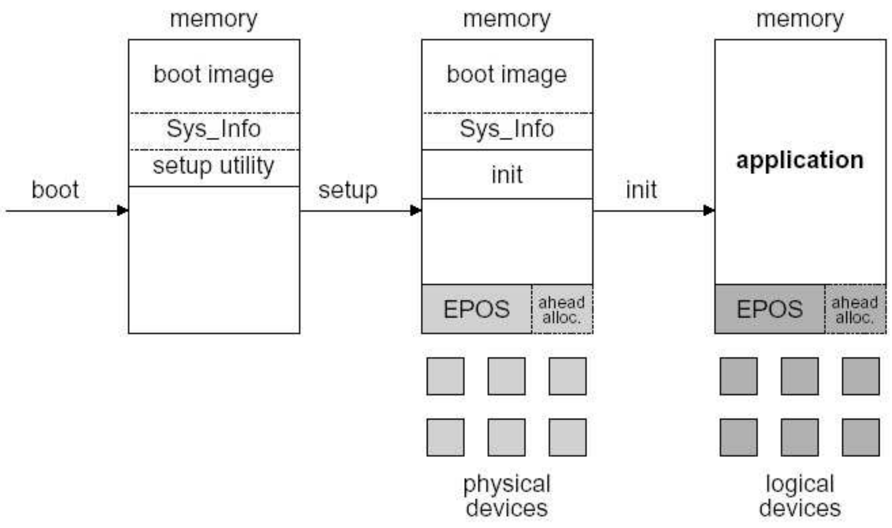


Figure 14: Overview of EPOS initialization [14]

4.2.2 The Init Utility

EPOS init is a routine that has plain access to the address space of the operating system, thus being able to invoke system operations. The initialization procedure carried out by the init utility consists in checking the traits of each abstraction to determine whether it has been included in the current system configuration and invoking the init class method for present abstractions [14].

After calling the init class method for all present abstractions, the init utility invokes EPOS operations, which by now are fully operational, to create the first process. If the dedicated application running on EPOS is executed by a single process, then the process created by the init utility is the application's unique process. Otherwise, this process is a loader that subsequently creates application processes in a multitasking environment [14].

4.2.3 Overview of EPOS initialization

An overview of EPOS initialization is presented in Figure 14. After loading the boot image, which includes a preliminary system description (SysInfo), the bootstrap invokes the setup utility to configure the hardware platform. Setup then utility builds an elementary memory model, configures required devices, loads EPOS, loads and activates the init utility. The init utility invokes the init class method of every abstraction included in the system to initialize its logical structure. It finishes loading the executable provided in the boot image to create the first process [14].

4.2.4 Considerations for the AVR Architecture

Having being designed bearing in mind a Von Neuman, self programming architecture, the EPOS initialization process has to undergo some changes when ported to a device such as the AT90S8515 AVR MCU, a Harvard Architecture unable of changing program memory at execution time.

In a regular system setup, the EPOS initialization system is eliminated after execution, and the resources it occupied are returned to the system's pool of free resources. This is not possible in the AT90S8515, since program memory cannot be altered, and therefore freed, at execution time. Since processes cannot be dynamically loaded at execution time, application pointers must be pre-adjusted in the binary image uploaded to the MCU.

The original structure of the EPOS image must also be altered bearing in mind two different address spaces and buses. Data structures, such as the SysInfo must now be placed together with the code and copied to RAM memory at initialization time.

Recent AVRs, such as the Atmega128, enable the possibility of dynamically loading code at boot time. This is done by writing code memory pages based on data from RAM memory. This process makes use of the SPM (Store Program Memory) instruction, which can only be executed from the Boot Loader section of Program Memory. Such AVR MCUs would enable the EPOS initialization system (bootstrap, setup and init) to be executed from the Boot Loader section, and freeing this section for application code after execution.

5 Conclusions and Further Research

Wireless Sensor Networks research and application is one of the most promising fields in Computer Sciences today, and presents a series of new challenges, among which adequate runtime support for applications is a key issue.

This research represented the first effort in the implementation functional release of the EPOS system for the UCB Wireless Sensor Network Plattform (Mica Motes), the "state-of-the-art" hardware platform for WSN. While the Motes group at Berkeley provides it's own Operating System for WSN (TinyOS), it does not provide the advanced functionality nor the Application Oriented design EPOS does. As WSN applications evolve, TinyOS will provide increasingly inadequate support, while EPOS can be easily configured and expanded in order to support the application programmers needs. The highly portable nature of EPOS also ensures reusability, both in system and application levels, as new hardware platforms emerge.

The port of a fully functional EPOS system for the Mica Platforms is a work in progress. Current results present a functional EPOS image of 1.3 KB for the AT90S8515 MCU (The object code for this image is presented in Annex A). Recent fund grants from FUNGRAD/UFSC will alow LISHA to acquire commercial Motes Kits, thus allowing further development focused on hardware mediators for Sensor Boards and Radio Transceivers and on the implementation of Communication Systems.

Energy control is one of the fundamental problems of WSN, and the research and implementation of WSN energy control mechanisms, including energy-aware communication protocols is also on the "LISHA WSN" agenda.

References

[1] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks, 2003.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless integrated network sensors. *Computer Networks*, 38(4):393–422, 2002.

[3] G. Asada, T. Dong, F. Lin, G. Pottie, W. Kaiser, and H. Marcy. Wireless integrated network sensors: Low power systems on a chip, 1998.

[4] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology, 2001.

[5] M. M. Chen, C. Majidi, D. M. Doolin, S. Glaser, and N. Sitar. Design and construction of a wildfire instrumentation system using networked sensors (poster), 2003.

[6] Chipcon. Smartrf cc1000 datasheet, 2002.

[7] Atmel Corporation. *AVR130: Setup and Use the AVR Timers*. San Jose, California, 2002.

[8] Atmel Corporation. *AVR132: Using the Enhanced Watchdog Timer*. San Jose, California, 2002.

[9] Atmel Corporation. *AVR 8515 Microcontroller Datasheet*. San Jose, California, 2003.

[10] Atmel Corporation. *AVR Application Note 109: Self Programming*. San Jose, California, 2003.

[11] Atmel Corporation. *STK500 User Guide*. San Jose, California, 2003.

[12] David E. Culler, Jason Hill, Philip Buonadonna, Robert Szewczyk, and Alec Woo. A network-centric approach to embedded software for tiny devices. *Lecture Notes in Computer Science*, 2211, 2001.

[13] AVR Freaks. *Design Note 003: AVR Sleep Modes*, 2002.

[14] Antônio Augusto Fröhlich. *Application-Oriented Operating Systems*. Number 17 in GMD Research Series. GMD - Forschungszentrum Informationstechnik, Sankt Augustin, August 2001.

[15] Antônio Augusto Fröhlich, Philippe Olivier Alexander Navaux, Sérgio Takeo Kofuji, and Wolfgang Schröder-Preikschat. Snow: a parallel programming environment for clusters of workstations. In *Proceedings of the 7th German-Brazilian Workshop on Information Technology*, Maria Farinha, Brazil, September 2000.

[16] Antônio Augusto Fröhlich and Wolfgang Schröder-Preikschat. On component-based communication systems for clusters of workstations. *ACM Applied Computing Review*, 1(1):1–1, November 2001.

[17] Antônio Augusto Fröhlich, Gilles Pokam Tientcheu, and Wolfgang Schröder-Preikschat. EPOS and Myrinet: Effective Communication Support for Parallel Applications Running on Clusters of Commodity Workstations. In *Proceedings of 8th International Conference on High Performance Computing and Networking*, pages 417–426, Amsterdam, The Netherlands, May 2000.

[18] TinyOS Group. *TinyOS Mica Developers Guide*. University Of California, Berkeley, 2002.

[19] John S. Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, pages 146–159, 2001.

[20] W. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, 2000.

[21] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, 2000.

[22] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[23] Crossbow Technology Inc. Mts sensor, data acquisition boards overview, 2002.

[24] Deborah Estrin Jeremy Elson. An address-free architecture for dynamic sensor networks.

[25] David Kalinsky and Roee Kalinsky. Introduction to serial peripheral interface, 2003.

[26] John Kymissis, Clyde Kendall, Joseph A. Paradiso, and Neil Gershenfeld. Parasitic power harvesting in shoes. In *ISWC*, pages 132–139, 1998.

[27] lan F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, August 2002.

[28] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002. To appear.

[29] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.

[30] Niall Murphy. Watchdog timers, 2003.

[31] Joseph Robert Polastre. Design and implementation ofwireless sensor networks for habitat monitoring. Master's thesis, University of California, Berkeley, 2003.

[32] Fauze Valério Polpeta and Antônio Augusto Fröhlich. Portability in component-based systems. LISHA, 2004.

[33] Great Duck Island Project. Habitat monitoring on great duck island – http://www.greatduckisland.net/.

[34] Pico Radio Project. Pico radio – http://bwrc.eecs.berkeley.edu/research/pico_radio/.

[35] TinyOS Project. Tinyos hardware designs.

[36] Praveen Rentala, Ravi Musunuri, Shashidhar Gandham, and Udit Saxena. Survey on sensor networks.

[37] Jim Turley. Atmel avr brings risc to 8-bit world. *Microprocessor Report*, 11(9), 1997.

[38] Brett Warneke and Sunil Bhave. Smart dust mote core architecture.

[39] Alec Woo. The mica sensing platform, 2002.

[40] Alec Woo and David E. Culler. A transmission control scheme for media access in sensor networks. In *Mobile Computing and Networking*, pages 221–235, 2001.

ANNEX A – Object Code for the Generated EPOS Image

```
at90s8515_loader:     file format elf32-avr

Disassembly of section .text:

00000000 <__vectors>:
   0: 10 c0       rjmp .+32      ; 0x22
   2: 36 c0       rjmp .+108     ; 0x70
   4: 35 c0       rjmp .+106     ; 0x70
   6: 34 c0       rjmp .+104     ; 0x70
   8: 33 c0       rjmp .+102     ; 0x70
   a: 32 c0       rjmp .+100     ; 0x70
   c: 31 c0       rjmp .+98      ; 0x70
   e: 30 c0       rjmp .+96      ; 0x70
  10: 2f c0       rjmp .+94      ; 0x70
  12: 2e c0       rjmp .+92      ; 0x70
  14: 2d c0       rjmp .+90      ; 0x70
  16: 2c c0       rjmp .+88      ; 0x70
  18: 2b c0       rjmp .+86      ; 0x70

0000001a <__ctors_start>:
  1a: f2 00       .word 0x00f2
  1c: 28 01       movw r4, r16

0000001e <__ctors_end>:
  1e: f8 00       .word 0x00f8
  20: 2e 01       movw r4, r28

00000022 <__dtors_end>:
  22: 11 24       eor r1, r1
  24: 1f be       out 0x3f, r1 ; 63
  26: cf e5       ldi r28, 0x5F ; 95
  28: d2 e0       ldi r29, 0x02 ; 2
  2a: de bf       out 0x3e, r29 ; 62
  2c: cd bf       out 0x3d, r28 ; 61

0000002e <__do_copy_data>:
  2e: 10 e0       ldi r17, 0x00 ; 0
  30: a0 e6       ldi r26, 0x60 ; 96
  32: b0 e0       ldi r27, 0x00 ; 0
  34: ee e8       ldi r30, 0x8E ; 142
  36: f2 e0       ldi r31, 0x02 ; 2
  38: 03 c0       rjmp .+6       ; 0x40

0000003a <.do_copy_data_loop>:
  3a: c8 95       lpm
  3c: 31 96       adiw r30, 0x01 ; 1
  3e: 0d 92       st X+, r0

00000040 <.do_copy_data_start>:
  40: a8 3e       cpi r26, 0xE8 ; 232
  42: b1 07       cpc r27, r17
  44: d1 f7       brne .-12      ; 0x3a
```

```
00000046 <__do_clear_bss>:
  46: 11 e0       ldi r17, 0x01 ; 1
  48: a8 ee       ldi r26, 0xE8 ; 232
  4a: b0 e0       ldi r27, 0x00 ; 0
  4c: 01 c0       rjmp .+2      ; 0x50

0000004e <.do_clear_bss_loop>:
  4e: 1d 92       st X+, r1

00000050 <.do_clear_bss_start>:
  50: a5 30       cpi r26, 0x05 ; 5
  52: b1 07       cpc r27, r17
  54: e1 f7       brne .-8      ; 0x4e
  56: 0d d0       rcall .+26     ; 0x72

00000058 <__do_global_ctors>:
  58: 10 e0       ldi r17, 0x00 ; 0
  5a: ce e1       ldi r28, 0x1E ; 30
  5c: d0 e0       ldi r29, 0x00 ; 0
  5e: 04 c0       rjmp .+8      ; 0x68

00000060 <.do_global_ctors_loop>:
  60: 22 97       sbiw r28, 0x02 ; 2
  62: fd 2f       mov r31, r29
  64: ec 2f       mov r30, r28
  66: 02 d1       rcall .+516    ; 0x26c

00000068 <.do_global_ctors_start>:
  68: ca 31       cpi r28, 0x1A ; 26
  6a: d1 07       cpc r29, r17
  6c: c9 f7       brne .-14     ; 0x60
  6e: c6 c0       rjmp .+396    ; 0x1fc

00000070 <__bad_interrupt>:
  70: c7 cf       rjmp .-114    ; 0x0

00000072 <_i_start>:
  72: 01 d0       rcall .+2      ; 0x76
  74: 08 95       ret

00000076 <_Z9epos_initPc>:
  76: 0f 93       push r16
  78: 1f 93       push r17
  7a: cf 93       push r28
  7c: df 93       push r29
  7e: 80 e0       ldi r24, 0x00 ; 0
  80: 90 e8       ldi r25, 0x80 ; 128
  82: 90 93 e9 00 sts 0x00E9, r25
  86: 80 93 e8 00 sts 0x00E8, r24
  8a: 80 e0       ldi r24, 0x00 ; 0
  8c: 90 e0       ldi r25, 0x00 ; 0
  8e: 20 91 04 80 lds r18, 0x8004
  92: 30 91 05 80 lds r19, 0x8005
  96: 82 17       cp r24, r18
  98: 93 07       cpc r25, r19
  9a: 20 f4       brcc .+8      ; 0xa4
  9c: 01 96       adiw r24, 0x01 ; 1
  9e: 82 17       cp r24, r18
  a0: 93 07       cpc r25, r19
  a2: e0 f3       brcs .-8      ; 0x9c
  a4: c0 e6       ldi r28, 0x60 ; 96
  a6: d0 e0       ldi r29, 0x00 ; 0
  a8: 0c 2f       mov r16, r28
  aa: 1d 2f       mov r17, r29
  ac: 0c 57       subi r16, 0x7C ; 124
  ae: 1f 4f       sbci r17, 0xFF ; 255
  b0: e9 91       ld r30, Y+
  b2: f9 91       ld r31, Y+
  b4: 30 97       sbiw r30, 0x00 ; 0
  b6: 21 f4       brne .+8      ; 0xc0
  b8: 0c 17       cp r16, r28
  ba: 1d 07       cpc r17, r29
  bc: c8 f7       brcc .-14     ; 0xb0
  be: 06 c0       rjmp .+12      ; 0xcc
  c0: 80 91 e8 00  lds r24, 0x00E8
  c4: 90 91 e9 00  lds r25, 0x00E9
  c8: 09 95       icall
  ca: f6 cf       rjmp .-20      ; 0xb8
  cc: 80 e0       ldi r24, 0x00 ; 0
  ce: 90 e0       ldi r25, 0x00 ; 0
  d0: df 91       pop r29
  d2: cf 91       pop r28
  d4: 1f 91       pop r17
  d6: 0f 91       pop r16
  d8: 08 95       ret

000000da <_ZN6System4AVR84initEPNS_11System_InfoE>:
  da: 80 e0       ldi r24, 0x00 ; 0
  dc: 90 e0       ldi r25, 0x00 ; 0
  de: 08 95       ret

000000e0 <_ZN6System17AT90S8515_Display4initEPNS_11System_InfoE>:
  e0: 80 e0       ldi r24, 0x00 ; 0
  e2: 90 e0       ldi r25, 0x00 ; 0
  e4: 08 95       ret

000000e6 <_ZN6System12AT90S8515_IC4initEPNS_11System_InfoE>:
  e6: 80 e0       ldi r24, 0x00 ; 0
  e8: 90 e0       ldi r25, 0x00 ; 0
  ea: 08 95       ret

000000ec <_ZN6System9AT90S85154initEPNS_11System_InfoE>:
  ec: 80 e0       ldi r24, 0x00 ; 0
  ee: 90 e0       ldi r25, 0x00 ; 0
  f0: 08 95       ret

000000f2 <_ZN6System8AVR8_MMU4initEPNS_11System_InfoE>:
  f2: 80 e0       ldi r24, 0x00 ; 0
  f4: 90 e0       ldi r25, 0x00 ; 0
  f6: 08 95       ret

000000f8 <_ZN6System15AT90S8515_Timer4initEPNS_11System_InfoE>:
  f8: 80 e0       ldi r24, 0x00 ; 0
  fa: 90 e0       ldi r25, 0x00 ; 0
  fc: 08 95       ret

000000fe <_ZN6System8AVR8_TSC4initEPNS_11System_InfoE>:
  fe: 80 e0       ldi r24, 0x00 ; 0
 100: 90 e0       ldi r25, 0x00 ; 0
 102: 08 95       ret

00000104 <_ZN6System3Imp16Exclusive_Thread4initEPNS_11System_InfoE>:
 104: cf 93       push r28
 106: df 93       push r29
 108: cd b7       in r28, 0x3d ; 61
 10a: de b7       in r29, 0x3e ; 62
 10c: 27 97       sbiw r28, 0x07 ; 7
 10e: 0f b6       in r0, 0x3f ; 63
 110: f8 94       cli
 112: de bf       out 0x3e, r29 ; 62
 114: 0f be       out 0x3f, r0 ; 63
 116: cd bf       out 0x3d, r28 ; 61
 118: f9 2f       mov r31, r25
 11a: e8 2f       mov r30, r24
 11c: 42 ef       ldi r20, 0xF2 ; 242
 11e: 50 e0       ldi r21, 0x00 ; 0
 120: 2c 2f       mov r18, r28
 122: 3d 2f       mov r19, r29
 124: 2f 5f       subi r18, 0xFF ; 255
 126: 3f 4f       sbci r19, 0xFF ; 255
 128: e2 5b       subi r30, 0xB2 ; 178
 12a: ff 4f       sbci r31, 0xFF ; 255
 12c: 60 81       ld r22, Z
 12e: 71 81       ldd r23, Z+1 ; 0x01
 130: b3 2f       mov r27, r19
 132: a2 2f       mov r26, r18
 134: 11 96        adiw r26, 0x01 ; 1
 136: 80 91 fb 00  lds r24, 0x00FB
 13a: 88 23        and r24, r24
 13c: 19 f4        brne .+6       ; 0x144
 13e: 81 e0        ldi r24, 0x01 ; 1
 140: 80 93 fb 00  sts 0x00FB, r24
 144: 80 91 03 01  lds r24, 0x0103
 148: 90 91 04 01  lds r25, 0x0104
 14c: 00 97        sbiw r24, 0x00 ; 0
 14e: 71 f4        brne .+28      ; 0x16c
 150: e0 91 e8 00  lds r30, 0x00E8
 154: f0 91 e9 00  lds r31, 0x00E9
 158: e0 5b        subi r30, 0xB0 ; 176
 15a: ff 4f        sbci r31, 0xFF ; 255
 15c: 80 81        ld r24, Z
 15e: 91 81        ldd r25, Z+1 ; 0x01
 160: 8e 83        std Y+6, r24 ; 0x06
 162: 9f 83        std Y+7, r25 ; 0x07
 164: 90 93 04 01  sts 0x0104, r25
 168: 80 93 03 01  sts 0x0103, r24
 16c: 12 96        adiw r26, 0x02 ; 2
 16e: 8d 93        st X+, r24
 170: 9c 93        st X, r25
 172: 13 97        sbiw r26, 0x03 ; 3
 174: 80 50        subi r24, 0x00 ; 0
 176: 91 40        sbci r25, 0x01 ; 1
 178: 90 93 04 01  sts 0x0104, r25
 17c: 80 93 03 01  sts 0x0103, r24
 180: 81 50        subi r24, 0x01 ; 1
 182: 9f 4f        sbci r25, 0xFF ; 255
 184: f3 2f        mov r31, r19
 186: e2 2f        mov r30, r18
 188: 81 83        std Z+1, r24 ; 0x01
 18a: 92 83        std Z+2, r25 ; 0x02
 18c: 81 81        ldd r24, Z+1 ; 0x01
 18e: 92 81        ldd r25, Z+2 ; 0x02
 190: 6e 83        std Y+6, r22 ; 0x06
 192: 7f 83        std Y+7, r23 ; 0x07
 194: 85 e0        ldi r24, 0x05 ; 5
 196: b5 2f        mov r27, r21
 198: a4 2f        mov r26, r20
 19a: 01 90        ld r0, Z+
 19c: 0d 92        st X+, r0
 19e: 8a 95        dec r24
 1a0: e1 f7        brne .-8       ; 0x19a
 1a2: 41 15        cp r20, r1
 1a4: 51 05        cpc r21, r1
 1a6: 21 f0        breq .+8       ; 0x1b0
 1a8: 95 2f        mov r25, r21
 1aa: 84 2f        mov r24, r20
 1ac: 01 96        adiw r24, 0x01 ; 1
 1ae: 02 c0        rjmp .+4       ; 0x1b4
 1b0: 95 2f        mov r25, r21
 1b2: 84 2f        mov r24, r20
 1b4: 90 93 f1 00  sts 0x00F1, r25
 1b8: 80 93 f0 00  sts 0x00F0, r24
 1bc: 80 91 f3 00  lds r24, 0x00F3
 1c0: 90 91 f4 00  lds r25, 0x00F4
 1c4: 80 91 f3 00  lds r24, 0x00F3
 1c8: 90 91 f4 00  lds r25, 0x00F4
 1cc: 80 e0        ldi r24, 0x00 ; 0
 1ce: 90 e0        ldi r25, 0x00 ; 0
 1d0: 27 96        adiw r28, 0x07 ; 7
 1d2: 0f b6        in r0, 0x3f ; 63
 1d4: f8 94        cli
 1d6: de bf        out 0x3e, r29 ; 62
 1d8: 0f be        out 0x3f, r0 ; 63
 1da: cd bf        out 0x3d, r28 ; 61
 1dc: df 91        pop r29
 1de: cf 91        pop r28
 1e0: 08 95        ret

000001e2 <_Z41__static_initialization_and_destruction_0ii>:
 1e2: 08 95        ret

000001e4 <_GLOBAL__I__ZN6System2siE>:
 1e4: 6f ef        ldi r22, 0xFF ; 255
 1e6: 7f ef        ldi r23, 0xFF ; 255
 1e8: 81 e0        ldi r24, 0x01 ; 1
 1ea: 90 e0        ldi r25, 0x00 ; 0
 1ec: fa df        rcall .-12      ; 0x1e2
 1ee: 08 95        ret

000001f0 <_GLOBAL__D__ZN6System2siE>:
 1f0: 6f ef        ldi r22, 0xFF ; 255
 1f2: 7f ef        ldi r23, 0xFF ; 255
 1f4: 80 e0        ldi r24, 0x00 ; 0
 1f6: 90 e0        ldi r25, 0x00 ; 0
 1f8: f4 df        rcall .-24      ; 0x1e2
 1fa: 08 95        ret

000001fc <main>:
 1fc: cf e5        ldi r28, 0x5F ; 95
 1fe: d2 e0        ldi r29, 0x02 ; 2
 200: de bf        out 0x3e, r29 ; 62
 202: cd bf        out 0x3d, r28 ; 61
 204: 8f ef        ldi r24, 0xFF ; 255
 206: 87 bb        out 0x17, r24 ; 23
 208: 81 e0        ldi r24, 0x01 ; 1
 20a: 88 bb        out 0x18, r24 ; 24
 20c: fd cf        rjmp .-6       ; 0x208

0000020e <_Z41__static_initialization_and_destruction_0ii>:
 20e: cf 93        push r28
 210: df 93        push r29
 212: cd b7        in r28, 0x3d ; 61
 214: de b7        in r29, 0x3e ; 62
 216: 22 97        sbiw r28, 0x02 ; 2
 218: 0f b6        in r0, 0x3f ; 63
 21a: f8 94        cli
 21c: de bf        out 0x3e, r29 ; 62
 21e: 0f be        out 0x3f, r0 ; 63
 220: cd bf        out 0x3d, r28 ; 61
 222: 28 2f        mov r18, r24
 224: 39 2f        mov r19, r25
 226: 6f 5f        subi r22, 0xFF ; 255
 228: 7f 4f        sbci r23, 0xFF ; 255
 22a: 49 f4        brne .+18      ; 0x23e
 22c: 21 30        cpi r18, 0x01 ; 1
 22e: 31 05        cpc r19, r1
 230: 31 f4        brne .+12      ; 0x23e
 232: 80 91 f7 00  lds r24, 0x00F7
 236: 90 91 f8 00  lds r25, 0x00F8
 23a: 89 83        std Y+1, r24 ; 0x01
 23c: 9a 83        std Y+2, r25 ; 0x02
 23e: 22 96        adiw r28, 0x02 ; 2
 240: 0f b6        in r0, 0x3f ; 63
 242: f8 94        cli
 244: de bf        out 0x3e, r29 ; 62
 246: 0f be        out 0x3f, r0 ; 63
 248: cd bf        out 0x3d, r28 ; 61
 24a: df 91        pop r29
 24c: cf 91        pop r28
 24e: 08 95        ret

00000250 <_GLOBAL__I__ZN6System3Imp20Address_Space_Common12_sys_segmentE>:
 250: 6f ef        ldi r22, 0xFF ; 255
 252: 7f ef        ldi r23, 0xFF ; 255
 254: 81 e0        ldi r24, 0x01 ; 1
 256: 90 e0        ldi r25, 0x00 ; 0
 258: da df        rcall .-76      ; 0x20e
 25a: 08 95        ret

0000025c <_GLOBAL__D__ZN6System3Imp20Address_Space_Common12_sys_segmentE>:
 25c: 6f ef        ldi r22, 0xFF ; 255
 25e: 7f ef        ldi r23, 0xFF ; 255
 260: 80 e0        ldi r24, 0x00 ; 0
 262: 90 e0        ldi r25, 0x00 ; 0
 264: d4 df        rcall .-88      ; 0x20e
 266: 08 95        ret

00000268 <__tablejump2__>:
 268: ee 0f        add r30, r30
 26a: ff 1f        adc r31, r31

0000026c <__tablejump__>:
 26c: c8 95        lpm
 26e: 31 96        adiw r30, 0x01 ; 1
 270: 0f 92        push r0
 272: c8 95        lpm
 274: 0f 92        push r0
 276: 08 95        ret

00000278 <__do_global_dtors>:
 278: 10 e0        ldi r17, 0x00 ; 0
 27a: ce e1        ldi r28, 0x1E ; 30
 27c: d0 e0        ldi r29, 0x00 ; 0
 27e: 04 c0        rjmp .+8       ; 0x288

00000280 <.do_global_dtors_loop>:
 280: fd 2f        mov r31, r29
 282: ec 2f        mov r30, r28
 284: f3 df        rcall .-26      ; 0x26c
 286: 22 96        adiw r28, 0x02 ; 2

00000288 <.do_global_dtors_start>:
 288: c2 32        cpi r28, 0x22 ; 34
 28a: d1 07        cpc r29, r17
 28c: c9 f7        brne .-14      ; 0x280
Disassembly of section .data:

00800060 <_ZN6System10init_tableE>:
...
  800068: 00 00        nop
  80006a: 6d 00        .word 0x006d
...
  800074: 00 00        nop
  800076: 7f 00        .word 0x007f
...
  800080: 00 00        nop
  800082: 79 00        .word 0x0079
...
  80008c: 00 00        nop
  80008e: 76 00        .word 0x0076
...
  80009c: 73 00        .word 0x0073
...
  8000aa: 7c 00        .word 0x007c
...
  8000c0: 00 00        nop
  8000c2: 70 00        .word 0x0070
  8000c4: 00 00        nop
  8000c6: 82 00        .word 0x0082
...

008000e6 <_ZN6System7machineE>:
  8000e6: ec 00        .word 0x00ec
```

ANNEX B – Article

The EPOS System Supporting Wireless Sensor Networks Applications

Lucas Wanner
lucas@inf.ufsc.br

Abstract— **Pervasing micro-sensing through Wireless Sensor Networks is revolutionizing the way we understand and manage complex physical systems from animal habitats to industrial plants. Composed by thousands of small devices with very limited resources, sensor networks are subject to novel system problems and constraints.**

Operating Systems for WSN must implement abstractions to interface with digital and analog sensors, provide a communication stack, and make efficient use of the system's limited energy resources. The EPOS operating system aims to give each dedicated application adequate runtime support, using the *Application Oriented System Design* domain engineering technique to produce a component-based operating system that can be automatically tailored according to the needs of particular applications.

This report presents an overview of Sensor Network technologies and system architecture and a port of the EPOS system for the AVR microcontroller architecture, used in many Wireless Sensor Networks research platforms.

Resumo— **O micro-sensoreamento pervasivo através de Redes de Sensores sem Fios está revolucionando a maneira como compreendemos e gerenciamos sistemas físicos complexos desde habitats de animais até plantas industriais. Compostas por milhares de pequenos dispositivos com recursos muito limitados, redes de sensores estão sujeitas a novos problemas e restrições de sistema.**

Sistemas Operacionais para Redes de Sensores devem implementar abstrações que tratem de sensores analógicos e digitais, devem prover uma pilha de protocolos para comunicação e fazer uso eficiente da capacidade limitada de energia do sistema. O Sistema Operacional EPOS tem como objetivo dar a cada aplicação dedicada suporte de *runtime* adequado, usando a técnica de engenharia de domínio *Design de Sistema Orientado à Aplicação* para produzir um sistema operacional baseado em componentes que pode ser automaticamente configurado de acordo com as necessidade de aplicações específicas.

Este artigo apresenta uma visão geral de tecnologias e arquitetura de sistema de Redes de Sensores e apresenta um porte do sistema EPOS para a arquitetura AVR, usada em várias plataformas de Redes de Sensores sem Fios.

Keywords— **Wireless Sensor Networks, Application Oriented Operating Systems**

I. INTRODUCTION

Wireless Sensor Networks is an emerging technology that enables information gathering in several different scenarios ranging from wildlife monitoring to industrial and military applications. A Wireless Sensor Network consists of groups of sensor nodes using wireless links to perform distributed sensing tasks [22]. These nodes are typically provided with an embedded microprocessor and a very small amount of memory.

While Wireless Sensor Networks (WSN) hardware designs are evolving into stable, commercially available platforms, the Hardware/Software boundary in WSN is a topic of open research. When available, the Operating Systems for WSN are, according to their own creators, too simplistic and unsuited for non-expert programmers [17].

This paper presents the first port of the EPOS[1] system to the AVR family of microcontrollers, an 8-bit Harvard Architecture widely used in embedded systems and Wireless Sensor Nodes.

The EPOS system aims to give each dedicated application adequate runtime support without having to design a new system for each application and without requiring application programmers to undergo complicated configuration procedures, using the *Application Oriented System Design* [7] domain engineering technique to produce a component-based operating system that can be automatically tailored according to the needs of particular applications.

A. Presentation Overview

The first part of this paper presents an overview of Wireless Sensor Network technologies, hardware, communcation models and applications; as well as the the AVR microcontroller architecture, widely used in Wireless Sensor Network hardware designs.

The second part presents the EPOS system and an implementation of it's initialization system to the AVR platform.

Finally the last part presents the conclusions of this paper, and suggests future related research projects.

II. WIRELESS SENSOR NETWORKS

In the past few years the advances in miniaturization and low-cost, low-power design have led to extensive research in large-scale networks of small, wireless, low-power, unattended microsensors [2], [14]. These microsensors are equipped with a sensor module (e.g. acoustic, light, temperature, magnetic, image sensor), capable of sensing some quantity about the environment, a digital processor for processing the signals from the sensors and performing operating system, application and network functions, a radio module for communication and a battery to provide energy for operation [12]. Each sensor obtains a "view" of the environment, and sends the view data to a distant base-station, through which an end-user can access the information.

Wireless sensor networks enable the monitoring of a variety of possibly inhospitable environments that include home security, machine-failure diagnosis, chemical/biological detection,

[1]EPOS: Embedded Parallel Operating System

medical and wild habitat monitoring [12], [18]. These applications require reliable, accurate, fault-proof and possibly real-time monitoring. Meanwhile, the low energy and processing capacities of the nodes require effi cient and energy-aware operation. Many researchers envision driving the networked sensor down to microscopic scale, creating smart environments and devices, powered by ambient energy [16] and used in many smart space scenarios. While it is acknowledged that energy consumption restrictions will not likely allow great processing power in this "smart dust", a wireless grid interface with more powerful computers could easily fulfi ll connectivity, storage and processing needs in the network nodes.

This section describes the basic microsensor node architecture, the communication principles of Wireless Sensor Networks (WSN), and WSN applications.

A. Wireless Sensor Nodes

In a Wireless Sensor Network, a Sensor Node is responsible for the lowest level of the sensing application. Several nodes are placed in areas of interrest, and each sensor node collects data from it's immediate surroundings. The collected data is then pre-processed in the node, and forwarded to a base station through the network formed with all the deployed nodes.

In a Sensor Node, the computational module is a programmable unit that provides computation, storage and bidirectional communication with other nodes in the system. This module interfaces with the analog and digital sensors in the node, performs basic signal processing and dispatches the data according to the application's needs [18]. The other modules in a Wireless Sensor Node are comprised by sensors and a radio for communication.

While several [2], [20], [24] platforms have been proposed and implemented for Wireless Sensor Nodes, the most popular and representative are the U.C. Berkeley's Mote[2] architectures [13], [21]. These are "current generation" devices constructed from off-the-shell components that have many of the key characteristics of the general class of Wireless Sensor Nodes [13]. They provide a microcontroller with internal program memory, sensor board interfaces, a low power radio module and a non-volatile memory chip.

The processor within the Mica2 is an Atmel Atmega128 AVR. AVR is an 8-Bit Harvard architecture, with separete instruction and data memory. In the motes, the AVR interfaces with four hardware blocks (Radio, LEDS, Flash Memory and Sensor board / Programming interface).

B. Communication in Wireless Sensor Networks

The Network component of Wireless Sensor Networks presents a series of new design challenges and is a topic of open research.

Sensor Networks must be power-aware. Most current network protocols are conservative only in their use of bandwidth. In a sensor node, all communication – including passive listening – will have a signifi cant effect on the node's limited energy reserves.

Sensor Networks are highly dynamic. Over time, sensors may fail or new sensors may be added. Sensors are likely to experience changes in their position and reachability. These changes make static confi guration unacceptable.

Sensor Networks must be self-confi guring. A single human may be responsible for thousands of nodes in a dense sensor network, and a design where each sensor node requires individual attention would be impractical.

All of these characteristics, presented in [14] and discussed at length in [5], [11], [25], may affect many aspects of the system's design, including routing and addressing mechanisms, naming services, security mechanisms and so forth.

C. Sensor Network Applications

Sensor networks may consist of many different types of sensors such as seismic, low sampling rate magnetic, thermal, visual, infrared, acoustic and radar, which are able to monitor a wide variety of ambient conditions. Sensor nodes can be used for continuous sensing, event detection, location sensing, and local control of actuators [1]. The concept of pervasing micro-sensing through Wireless Sensor Networks promise many new application areas. This section presents and categorizes the applications of WSN into military, environment, health, and commercial areas.

1) Military applications: Wireless sensor networks can be an integral part of military command, control, communications, computing, intelligence, surveillance, reconnaissance and targeting (C4ISRT) systems. The rapid deployment, self-organization and fault tolerance characteristics of sensor networks make them a very promising sensing technique for military C4ISRT [1].

Since sensor networks are based on the dense deployment of disposable and low-cost sensor nodes, destruction of some nodes by hostile actions does not affect a military operation as much as the destruction of a traditional sensor, which makes sensor networks concept a better approach for battlefi elds. Some of the military applications of sensor networks are monitoring friendly forces, equipment and ammunition; battlefi eld surveillance; reconnaissance of opposing forces and terrain; targeting; battle damage assessment; and nuclear, biological and chemical attack detection and reconnaissance [1].

2) Environmental applications: Environmental and habitat monitoring is a driving fi eld for wireless sensor networks [3]. It's applications include tracking the movements of small animals; monitoring environmental conditions; chemical/biological detection; precision agriculture; forest fi re detection; meteorological or geophysical research; flood detection; bio-complexity mapping of the environment; and pollution study.

3) Health applications: Some of the health applications for sensor networks are providing interfaces for the disabled; integrated patient monitoring; drug administration in hospitals; monitoring the movements and internal processes of insects or other small animals; telemonitoring of human physiological data; and tracking and monitoring doctors and patients inside a hospital [1].

[2]Mote, n. A small particle, as of floating dust; anything proverbially small; a speck: "The little motes in the sun do ever stir, though there be no wind" (Bacon).

4) Commercial applications: Commercial applications for WSN include monitoring material fatigue; managing inventory; monitoring product quality; constructing smart office spaces; robot control and guidance in automatic manufacturing environments; interactive toys; factory process control and automation; smart structures with sensor nodes embedded inside; machine diagnosis; transportation; factory instrumentation; local control of actuators; and vehicle tracking and detection [1].

III. THE AVR ARCHITECTURE

AVR is a widely used family of 8-bit RISC microcontrollers from Atmel. Usually deployed in the form of MCUs (Microprocessor Control Units[3]), the AVR provides good performance at low cost and low power consumption in a simple Harvard Architecture[4], making it the natural choice for Wireless Sensor Nodes processing and control.

A. Architectural Overview

The AVR CPU resembles most RISC processors but has smaller registers. The core features 32 identical 8-bit registers that can hold addresses or data. Since 8-bit address pointers are fairly worthless even in an 8-bit device, the last six registers can be used in pairs, as address pointers. Dubbed X, Y, and Z, these three meta-registers can be used for any load or store operation [23]. All operations are register-toregister; the chip follows a strict load/store model.

1) General Purpuse Registers: The AVR's fast-access Register file contains 32 x 8-bit general purpose registers with a single clock cycle access time, allowing sincle-cycle Arithmetic Logic Unit (ALU) operation. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing.

The register file is mapped into the data address space. The first 32 bytes of data memory, \$0x0000 – \$0x001F, correspond to registers R0-R31.

2) I/O Registers: The AVR's 64 I/O Registers are memory-mapped into addresses \$0x0020 – \$005F. These registers include status, interrupt and timer control, stack pointer, GPIO (General-Pourpose Input and Output) and SPI (Serial Programming Interface) and UART registers.

3) Data Memory: In the AT90S8515 MCU, the first 96 memory locations address the Register file and I/O memory. The next 512 locations address the internal data SRAM. An optional external data SRAM can be placed in the same SRAM memory space, filling the AVR's 64K address space.

4) Program Memory: In the AT90S8515 MCU contains 8K bytes of Programmable Flash Memory for program storage. Since all instructions are 16- or 32-bit words, the Flash is organized as 4Kx16. The AT90S8515 Program Counter is 12 bits wide, thus addressing the 4096 program memory addresses.

[3]In an MCU, the processor, memory and I/O all reside in the same physical IC (integrated circuit).

[4]A Harvard Architecture provides separate momories and buses for program and data.

In early AVR models, such as the AT90S8515, the program memory can only be updated by writing a full binary image to the flash. Once the program data is downloaded, no further updates of the flash are possible, as there is no instruction capable of writing to the program memory. These devices can be programmed serially, via ISP (In-System Programming) or parallelly, via High-Voltage Programming.

Recent AVR MCUs, such as the Atmega128, used in the Mica Motes, provide a Store Program Memory SPM) instruction capable of erasing and writing a page in the program memory.

In the MCUs where the SPM instruction is available, the Flash memory is divided into two sections, one Application section and one Boot Loader section. The SPM instruction can only be executed from the Boot Loader section [4]. The Flash memory is divided into pages containing 32, 64, or 128 words each.

B. Serial Peripheral Interface

Serial Peripheral Interface (SPI) is a serial bus standard established by Motorola and supported in silicon products from various manufacturers. It is a synchronous serial data link that operates in full duplex (signals carrying data in both directions simultaneously) [15].

Devices communicate using a master/slave relationship, in which the master initiates the data frame. When the master generates a clock and selects a slave device, data may be transferred in both directions simultaneously.

SPI specifies four signals: clock ($SCLK$); master data output, slave data input ($MOSI$); master data input, slave data output ($MISO$); and slave select ($\overline{SS}$). SCLK is generated by the master and input to all slaves. $MOSI$ carries data from master to slave. $MISO$ carries data from slave back to master. A slave device is selected when the master asserts its $\overline{SS}$ signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave.

The AT90S8515 provides a fully functional SPI implementation, capable of working in either master or slave mode and controlled by I/O memory mapped registers.

C. UART

The AT90S8515 MCU provides a full-duplex Universal Asynchronous Receiver/Transmitter (UART), featuring:

- Baud Rate generator
- Noise Filtering
- Overrun detection
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete.

Data transmission and reception, as well as UART setup is controlled by I/O memory mapped registers.

IV. GPIO

The AT90S8515 architecture provides four bi-directional I/O ports. Three I/O memory address locations are allocated for each port, one each for the Data Register, PORTx, Data Direction Register, DDRx, and the Port x Input Pins, PINx.

The last enables access to the physical value on each Port x pin. The Port x Input Pins address is read only, while the Data Register and the Data Direction Register are read/write. All Port x Pins can be used for General Purpose Input or Output (GPIO).

A. *Sleep Modes*

AVR microcontrollers provide several sleep modes. The purpose of these modes is to provide a way of suspending program execution when necessary, thereby reducing power consumption [6]. The Sleep Modes available in the AT90S8515 MCU are (in order from maximal to minimal power consumption):

- Idle mode
  The idle mode stops the CPU but leaves peripherals (UART, Analog Comparator etc.) running. The MCU will continue program execution immediately after waking up from Idle mode.
- Powersave mode
  This mode is identical to the Powerdown mode, with one exception: The Timer Crystal Oscillator will continue to operate and the Timer can continue to count. The device can wake up from either a Timer Overflow or Output Compare event.
- Powerdown mode
  In this mode, all Oscillators are stopped while the External Level interrupts and the Watchdog continue operating. Only an External Reset, a Watchdog Reset or an External Level interrupt can wake up the MCU.

The device is sent into sleep mode by selecting the desired sleep mode in the MCU Control Register, enabling interrupts that should be able to wake the MCU up from sleep and executing a SLEEP intruction.

V. EPOS INITIALIZATION IN THE AVR

The EPOS system was born in 1997 as a project to experiment with the concepts and mechanisms of application-oriented system design [7]. EPOS is thus an intrinsically application-oriented operating system, and today is evolving into a fully functional, multi-platform, very high performance OS. Current results include implementations for high-performance Clusters of Commodity Workstations based on Myrinet Networks [8]–[10] and ports to the PowerPC (32-bit) and H8 (8-bit) architectures [19]. EPOS aims to deliver functionality (giving the application it's necessary runtime support), customizability (being tailored to specific applications) and efficiency (making resources available to the application with the lowest possible overhead) [7].

A. *EPOS System Architecture*

EPOS relies on System Abstractions, Hardware Mediators and Aspects to ensure component reusability. Abstractions describe scenario-independent functionalities, are widely reusable and represent most of the components in the system. A hardware mediator is a system-dependent abstraction of elements of the hardware platform that are used by system abstractions and scenario aspects [7]. Aspects provide configurable functionalities for applications, such as sharing, protection and atomicity.

1) The Setup Utility for the AVR: EPOS Setup Utility is responsible for building an elementary execution context for the OS. It runs after the bootstrap and previous to the Init Utility.

In the AVR, the bootstrap simply disables interrupts and calls setup, passing the SysInfo Structure as a parameter. The SysInfo structure describes the relevant characteristics of the forthcoming EPOS configuration.

As the Setup utility initiates, it proceeds with hardware setup, updating and completing SysInfo, including information about the physical resources configured, a memory map describing how the operating system has been loaded, the node's logical id, etc [7].

In the AVR, Setup is mainly responsible for setting up the interrupt controller, checking system integrity, setting up the Init entry point and setting up system data structures.

2) The Init Utility: EPOS init is a routine that has plain access to the address space of the operating system, thus being able to invoke system operations. The initialization procedure carried out by the init utility consists in checking the traits of each abstraction to determine whether it has been included in the current system configuration and invoking the init class method for present abstractions [7].

After calling the init class method for all present abstractions, the init utility invokes EPOS operations, which by now are fully operational, to create the first process. If the dedicated application running on EPOS is executed by a single process, then the process created by the init utility is the application's unique process. Otherwise, this process is a loader that subsequently creates application processes in a multitasking environment [7].

3) Overview of EPOS initialization: After loading the boot image, which includes a preliminary system description (SysInfo), the bootstrap invokes the setup utility to configure the hardware platform. Setup then utility builds an elementary memory model, configures required devices, loads EPOS, loads and activates the init utility. The init utility invokes the init class method of every abstraction included in the system to initialize its logical structure. It finishes loading the executable provided in the boot image to create the first process [7].

4) Considerations for the AVR Architecture: Having being designed bearing in mind a Von Neuman, self programming architecture, the EPOS initialization process has to undergo some changes when ported to a device such as the AT90S8515 AVR MCU, a Harvard Architecture unable of changing program memory at execution time.

In a regular system setup, the EPOS initialization system is eliminated after execution, and the resources it occupied are returned to the system's pool of free resources. This is not possible in the AT90S8515, since program memory cannot be altered, and therefore freed, at execution time. Since processes cannot be dynamically loaded at execution time, application pointers must be pre-adjusted in the binary image uploaded to the MCU.

The original structure of the EPOS image must also be altered bearing in mind two different address spaces and buses. Data structures, such as the SysInfo must now be placed together with the code and copied to RAM memory at initialization time.

Recent AVRs, such as the Atmega128, enable the possibility of dynamically loading code at boot time. This is done by writing code memory pages based on data from RAM memory. This process makes use of the SPM (Store Program Memory) instruction, which can only be executed from the Boot Loader section of Program Memory. Such AVR MCUs would enable the EPOS initialization system (bootstrap, setup and init) to be executed from the Boot Loader section, and freeing this section for application code after execution.

VI. Conclusions and Further Research

Wireless Sensor Networks research and application is one of the most promising fi elds in Computer Sciences today, and presents a series of new challenges, among which adequate runtime support for applications is a key issue.

This research represented the fi rst effort in the implementation functional release of the EPOS system for the UCB Wireless Sensor Network Plattform (Mica Motes), the "state-of-the-art" hardware platform for WSN. While the Motes group at Berkeley provides it's own Operating System for WSN (TinyOS), it does not provide the advanced functionality nor the Application Oriented design EPOS does. As WSN applications evolve, TinyOS will provide increasingly inadequate support, while EPOS can be easily confi gured and expanded in order to support the application programmers needs. The highly portable nature of EPOS also ensures reusability, both in system and application levels, as new hardware platforms emerge.

The port of a fully functional EPOS system for the Mica Platforms is a work in progress. Current results present a functional EPOS image of 1.3 KB for the AT90S8515 MCU (The object code for this image is presented in Annex A). Recent fund grants from FUNGRAD/UFSC will alow LISHA to acquire commercial Motes Kits, thus allowing further development focused on hardware mediators for Sensor Boards and Radio Transceivers and on the implementation of Communication Systems.

Energy control is one of the fundamental problems of WSN, and the research and implementation of WSN energy control mechanisms, including energy-aware communication protocols is also on the "LISHA WSN" agenda.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless integrated network sensors. *Computer Networks*, 38(4):393422, 2002.

[2] G. Asada, T. Dong, F. Lin, G. Pottie, W. Kaiser, and H. Marcy. Wireless integrated network sensors: Low power systems on a chip, 1998.

[3] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology, 2001.

[4] Atmel Corporation. *AVR Application Note 109: Self Programming*. San Jose, California, 2003.

[5] David E. Culler, Jason Hill, Philip Buonadonna, Robert Szewczyk, and Alec Woo. A network-centric approach to embedded software for tiny devices. *Lecture Notes in Computer Science*, 2211, 2001.

[6] AVR Freaks. *Design Note 003: AVR Sleep Modes*, 2002.

[7] Antônio Augusto Fröhlich. *Application-Oriented Operating Systems*. Number 17 in GMD Research Series. GMD - Forschungszentrum Informationstechnik, Sankt Augustin, August 2001.

[8] Antônio Augusto Fröhlich, Philippe Olivier Alexander Navaux, Sérgio Takeo Kofuji, and Wolfgang Schröder-Preikschat. Snow: a parallel programming environment for clusters of workstations. In *Proceedings of the 7th German-Brazilian Workshop on Information Technology*, Maria Farinha, Brazil, September 2000.

[9] Antônio Augusto Fröhlich and Wolfgang Schröder-Preikschat. On component-based communication systems for clusters of workstations. *ACM Applied Computing Review*, 1(1):1–1, November 2001.

[10] Antônio Augusto Fröhlich, Gilles Pokam Tientcheu, and Wolfgang Schröder-Preikschat. EPOS and Myrinet: Effective Communication Support for Parallel Applications Running on Clusters of Commodity Workstations. In *Proceedings of 8th International Conference on High Performance Computing and Networking*, pages 417–426, Amsterdam, The Netherlands, May 2000.

[11] John S. Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building effi cient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, pages 146–159, 2001.

[12] W. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, 2000.

[13] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[14] Deborah Estrin Jeremy Elson. An address-free architecture for dynamic sensor networks.

[15] David Kalinsky and Roee Kalinsky. Introduction to serial peripheral interface, 2003.

[16] John Kymissis, Clyde Kendall, Joseph A. Paradiso, and Neil Gershenfeld. Parasitic power harvesting in shoes. In *ISWC*, pages 132–139, 1998.

[17] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002. To appear.

[18] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.

[19] Fauze Valério Polpeta and Antônio Augusto Fröhlich. Portability in component-based systems. LISHA, 2004.

[20] Pico Radio Project. Pico radio – http://bwrc.eecs.berkeley.edu/research/pico_radio/.

[21] TinyOS Project. Tinyos hardware designs.

[22] Praveen Rentala, Ravi Musunuri, Shashidhar Gandham, and Udit Saxena. Survey on sensor networks.

[23] Jim Turley. Atmel avr brings risc to 8-bit world. *Microprocessor Report*, 11(9), 1997.

[24] Brett Warneke and Sunil Bhave. Smart dust mote core architecture.

[25] Alec Woo and David E. Culler. A transmission control scheme for media access in sensor networks. In *Mobile Computing and Networking*, pages 221–235, 2001.