

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**O Sistema EPOS como Suporte a Aplicações em
Redes de Sensores Sem Fios**

Lucas Francisco Wanner

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

O Sistema EPOS como Suporte a Aplicações em Redes de Sensores Sem Fios

Lucas Francisco Wanner

Prof. Dr. Antônio Augusto M. Fröhlich

Orientador

Banca Examinadora:

Prof. Dr. Rômulo Silva de Oliveira

Fauze Valerio Polpetta, B.Sc.

Palavras-chave: Redes de Sensores sem Fios, AVR, SO Orientado a Aplicação

Florianópolis, Fevereiro de 2004.

Aos meus avós,

José Tarcísio Hoff (in memoriam)

Reinildes von Frühauf

Arlindo Gregório Wanner

Maria Nair Schmidt

Agradecimentos

Gostaria de aproveitar esta oportunidade para agradecer a todas as pessoas que estiveram comigo durante todos estes anos aqui na UFSC, e em particular durante a execução desta pesquisa:

- Professor Antônio Augusto, por apoio, dedicação e comprometimento além das minhas mais altas expectativas (e, acredito, as de qualquer aluno de graduação). Mais do que um orientador, ele tornou-se um modelo para mim, e um exemplo a seguir.
- Fauze Valerio Polpeta, juntamente com meus colegas do grupo EPOS no LISHA, pelo seu apoio e valiosa orientação técnica .
- Professores Luis Fernando Friedrich e Leandro José Komosinski, pela sua orientação, amizade e apoio.
- Meus colegas e amigos no PET e no G8, pela sua amizade e muitas longas conversas sobre a vida, o universo e tudo mais...
- Minha mãe, Mena, e minha irmã, Joice, pelo seu amor e apoio durante todos estes anos. Eu não estaria aqui se não fosse por elas.
- Minha amada Gabriella, por me mostrar o significado do amor e da felicidade.

Sumário

Lista de Tabelas	p. vii
Lista de Figuras	p. viii
Resumo	p. ix
Abstract	p. x
1 Introdução	p. 1
1.1 Visão Geral da Apresentação	p. 2
2 Redes de Sensores Sem Fios	p. 3
2.1 Nodos de Sensor Sem Fios	p. 4
2.1.1 Mica Motes	p. 4
2.1.1.1 Organização do Hardware	p. 5
2.1.1.2 Placas de Sensores	p. 8
2.1.1.3 Programação	p. 8
2.1.1.4 Disponibilidade	p. 9
2.1.1.5 Desenvolvimento Futuro	p. 10
2.1.2 Trabalhos Relacionados	p. 10
2.2 Comunicação em Rede de Sensores sem Fios	p. 11
2.2.1 Camada de Enlace	p. 11
2.2.1.1 Mecanismo de Escuta	p. 12
2.2.1.2 Mecanismo Baseado em Contenção	p. 12

2.2.2	Roteamento	p. 13
2.2.2.1	Problemas de Projeto	p. 13
2.2.2.2	Protocolos Centrados em Dados	p. 14
2.2.2.3	Protocolos Hierárquicos	p. 15
2.3	Aplicações de Redes de Sensores	p. 15
2.3.1	Aplicações Militares	p. 16
2.3.2	Aplicações Ambientais	p. 16
2.3.2.1	Detecção de Incêndios Ambientais	p. 16
2.3.2.2	Monitoramento de Habitats	p. 17
2.3.3	Aplicações de Saúde	p. 18
2.3.4	Aplicações Comerciais	p. 18
3	A Arquitetura AVR	p. 19
3.1	Visão Geral da Arquitetura	p. 19
3.1.1	Registradores de Propósito Geral	p. 19
3.1.2	Registradores de I/O	p. 20
3.1.2.1	Registrador de Status	p. 21
3.1.2.2	Stack Pointer	p. 21
3.1.3	Memória de Dados	p. 21
3.1.4	Memória de Programa	p. 24
3.1.4.1	Auto-Programação	p. 24
3.1.5	Modos de Endereçamento	p. 25
3.1.5.1	Direto para Registradores – Único Registrador	p. 26
3.1.5.2	Direto para Registradores – Dois Registradores	p. 26
3.1.5.3	I/O Direto	p. 27
3.1.5.4	Direto para Dados	p. 27
3.1.5.5	Indireto para Dados com Deslocamento	p. 28

3.1.5.6	Indireto para Dados	p. 28
3.1.5.7	Acesso de Constantes Utilizando LPM	p. 29
3.1.5.8	Endereçamento Indireto de Programa	p. 29
3.1.5.9	Endereçamento Relativo de Programa	p. 30
3.2	Timers	p. 30
3.2.1	Eventos de Timer	p. 31
3.2.2	Watchdog Timer	p. 31
3.3	Interface Serial de Periféricos	p. 32
3.4	UART	p. 32
3.5	GPIO	p. 33
3.6	Modos de Economia de Energia	p. 33
4	Inicialização do EPOS no AVR	p. 35
4.1	Arquitetura do Sistema do EPOS	p. 35
4.1.1	Abstrações de Sistema	p. 35
4.2	Inicialização do EPOS	p. 36
4.2.1	O utilitário de Setup para o AVR	p. 36
4.2.2	O Utilitário de Inicialização	p. 37
4.2.3	Visão Geral da Inicialização do EPOS	p. 37
4.2.4	Considerações para a Arquitetura AVR	p. 37
5	Conclusões e Pesquisas Futuras	p. 39
	Referências	p. 40
	Anexo A – Código Objeto para a Imagem do EPOS Gerada	p. 43
	Anexo B – Artigo	p. 49

Lista de Tabelas

1	Componentes dos Mica Motes	p. 5
2	Registradores de I/O do AVR	p. 23

Lista de Figuras

1	O Nodo de Sensor Mica2	p. 5
2	Organização Geral da Arquitetura Mote	p. 6
3	Esquemático Geral do Mica2	p. 7
4	A Placa de Sensores Mica	p. 9
5	Kit de Motes da Crossbow	p. 9
6	O Mote Spec	p. 10
7	Mote Firebug Instalado	p. 17
8	Motes instalados na Great Duck Island	p. 18
9	Diagrama de Blocos da Arquitetura AVR [9]	p. 20
10	Mapas de Memória do AVR [9]	p. 22
11	Organização da Memória em AVRs Auto-Programáveis [10]	p. 25
12	Implementação SPI de Múltiplos Escravos [25]	p. 33
13	Famílias de Abstrações do EPOS [14]	p. 36
14	Visão Geral da Inicialização do EPOS [14]	p. 38

Resumo

O micro-sensoriamento 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 necessidades 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

Pervasive 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 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* 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 *Introdução*

Redes de Sensores sem Fios (RSSF) é uma tecnologia emergente que permite coleta de informações em diversos cenários, desde monitoramento ambiental até aplicações industriais e militares. Uma Rede de Sensores sem Fios consiste de grupos de nodos de sensores utilizando um link sem fio para realizar tarefas de sensoriamento distribuído [35]. Esses nodos tipicamente contêm um microprocessador embutido e uma quantidade muito pequena de memória. Este sistema embutido deve interfacear com sensores analógicos e digitais, prover uma pilha de comunicação e fazer uso eficiente da limitada capacidade de energia.

Enquanto o hardware de Rede de Sensores sem Fios está evoluindo para plataformas estáveis e comercialmente disponíveis, a fronteira Software/Hardware em Rede de Sensores sem Fios é um tópico de pesquisa aberto. Quando disponíveis, os Sistemas Operacionais para RSSF são, de acordo com seus próprios criadores, muito simplísticos e inadequados para programadores não-especialistas [28]

Este relatório apresenta o primeiro porte do sistema EPOS¹ para a família de microcontroladores AVR, uma arquitetura Harvard de 8 bits largamente utilizada em sistemas embutidos e Nodos de Redes 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.

¹EPOS: Embedded Parallel Operating System

1.1 Visão Geral da Apresentação

Capítulo 2 apresenta uma visão geral de tecnologias, hardware, modelos de comunicação e aplicações de Rede de Sensores sem Fios.

Capítulo 3 apresenta a arquitetura de microcontroladores AVR, largamente utilizada em designs de hardware para Rede de Sensores sem Fios.

Capítulo 4 apresenta o sistema EPOS e uma implementação do seu sistema de inicialização para a plataforma AVR.

Capítulo 5 apresenta as conclusões deste projeto, e sugere futuros trabalhos de pesquisa relacionados.

2 *Redes de Sensores Sem Fios*

Nos últimos anos os avanços na miniaturização e design de baixo custo e baixa potência levaram a extensivas pesquisas em redes em larga escala de microsensores pequenos, sem fios e de baixo consumo [24, 3]. Estes microsensores são equipados com um módulo de sensores (e.g. sensores acústicos, de luz, temperatura, magnéticos e de imagens), capazes de sentir alguma quantidade sobre o ambiente, um processador digital para processamento de sinais e para executar tarefas do sistema operacional e funções de rede e uma bateria para prover energia para operação [20]. Cada sensor obtém uma “visão” do ambiente, e envia os dados sentidos para uma estação base, através da qual um usuário final pode acessar as informações.

Rede de Sensores sem Fios permitem o monitoramento de uma variedade de ambientes possivelmente inóspitos que incluem segurança doméstica, diagnóstico de falha de máquinas, detecção química e biológica, e monitoramento médico e ambiental [20, 29]. Estas aplicações requerem monitoramento confiável, preciso, a prova de falhas e possivelmente em tempo real. Ao mesmo tempo, a baixa capacidade de processamento e energia requerem operação eficiente e com boa gerência de energia. Muitos pesquisadores visualizam o desenvolvimento de sensores de rede em escalas microscópicas, criando ambientes e dispositivos inteligentes, alimentados por energia ambiente [26] e utilizados em diversos espaços inteligentes. Ao mesmo tempo em que admite-se que as restrições em consumo de energia não permitirá uma grande capacidade de processamento nesta “po-eira inteligente”, uma interface sem fios com um grid de computação com computadores poderosos poderá facilmente prover as necessidades de conectividade, armazenamento e processamento dos nodos de rede.

Este capítulo descreve a arquitetura básica dos nodos de sensores, os princípios de comunicação em Rede de Sensores sem Fios e aplicações de RSSF.

2.1 Nodos de Sensor Sem Fios

Em uma Rede de Sensores sem Fios, um Nodo de Sensor é responsável pelo nível mais baixo da aplicação de sensoriamento. Vários nodos são colocados em áreas de interesse, e cada nodo de sensor coleta dados das suas imediações. Os dados coletados são então pré-processados no nodo, e enviados a uma estação-base distante através da rede formada com todos os nodos instalados.

Em um Nodo de Sensor, o módulo computacional é uma unidade programável que provê processamento, armazenamento e comunicação bidirecional com outros nodos no sistema. Este módulo interfaceia com os sensores analógicos e digitais no nodo, executa processamento básico de sinais e envia os dados de acordo com as necessidades da aplicação [29]. Os outros módulos em um Nodo de Sensor são formados por sensores e um rádio para comunicação.

Apesar de várias [3, 37, 33] plataformas terem sido propostas e implementadas para Nodos de Sensores sem Fios, as mais populares são as arquiteturas *Mote*¹ [22, 34], desenvolvidas na Universidade da Califórnia em Berkeley. Os Motes são dispositivos de *geração atual*, construídos com componentes disponíveis comercialmente, que possuem muitas das principais características da classe geral de Nodos de Sensores Sem Fios [22]. Eles provêm um microcontrolador com memória interna de programa, interfaces com placas de sensores, um rádio de baixa potência e um módulo de memória não-volátil.

2.1.1 Mica Motes

Nos últimos anos a família de UC Berkeley Motes (ver Tabela 1) evoluiu para uma plataforma estável para pesquisa em Redes de Sensores. Sua geração atual, o Mica2 (ver Figura 1) usa um rádio de único canal da RF Monolithics (operando a 916 Mhz nos EUA e a 433 na Europa) para prover comunicação bidirecional a 40kbps [31], um microcontrolador Atmel Atmega128 rodando a 8Mhz, e um chip de memória de 512K. Um par de pilhas convencionais é usado como fonte de energia. Seu tamanho pequeno (aproximadamente 5 x 4 x 1.5 cm) permite instalação em locais remotos com mínima interferência com o habitat existente [31].

¹Mote, n. Uma pequena partícula, como de poeira; qualquer coisa proverbialmente pequena. “The little motes in the sun do ever stir, though there be no wind” (Bacon).

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)	

Tabela 1: Componentes dos Mica Motes

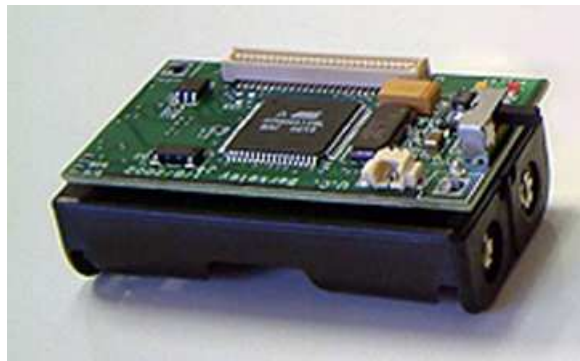


Figura 1: O Nodo de Sensor Mica2

2.1.1.1 Organização do Hardware

O processador dentro no Mica2 é um AVR Atmel Atmega128. AVR é uma arquitetura Harvard de 8-bits, com memórias separadas para código e dados. Esta arquitetura será discutida em profundidade no Capítulo 3.

Nos motes, o AVR interfaceia com quatro blocos de hardware (Rádio, LEDs, Memória Flash e Interface para Placas de Sensores / Programação). A organização geral do hardware é apresentada na Figura 2. Um esquemático simples da arquitetura Mica2 é apresentado na Figura 3.

LEDs

Três LEDs programáveis são conectados ao AVR nos motes Mica2. Estes podem

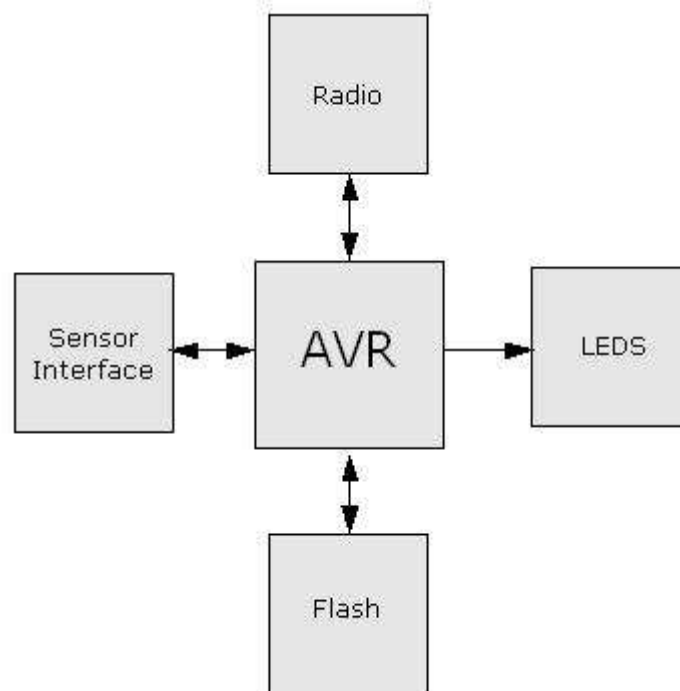


Figura 2: Organização Geral da Arquitetura Mote

ser utilizados para status e saída de valores digitais.

Memória Flash

De maneira a permitir armazenamento permanente e logging de dados nos notes, uma memória flash serial de 512K é ligada a uma das portas UART do AVR. Se instalada em conjunto com um co-processador simples, esta memória secundária pode ser utilizada para reprogramação do microcontrolador principal.

Radio

O Mica2 utiliza um transceiver UHF de único canal, implementado em um único chip e de baixa potência da Chipcom como seu componente de rádio. O CC1000 foi projetado para aplicações sem fio de baixíssima potência e voltagem. O circuito é utilizado principalmente para as bandas de frequência ISM (Industrial, Scientific and Medical; Industrial, Científica e Médica) e SRD (Short Range Device; Dispositivo de Baixo Alcance) nas faixas de 315, 433, 868 e 915 MHz, mas pode ser facilmente programado para operação em outras frequências nas faixas de 300-1000 Mhz. Os principais parâmetros operacionais do CC1000 podem ser programados através de um barramento serial facilmente interfaciável, fazendo assim o CC1000 um transceiver flexível e de uso simples [6].

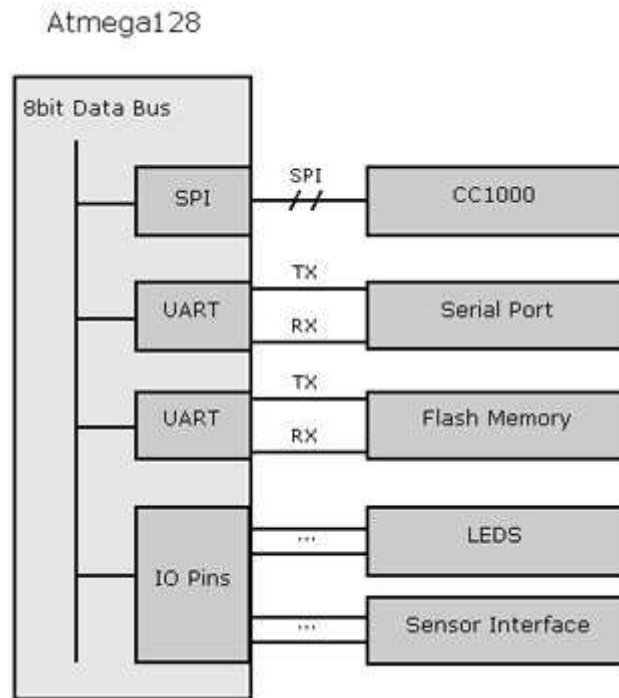


Figura 3: Esquemático Geral do Mica2

O CC1000 é configurado através de uma interface simples de 3 fios. Há 36 registradores de 8-bits, cada um endereçado por um endereço de 7-bits. Um bit de escrita ou leitura inicia uma operação de leitura ou escrita. Uma configuração completa do CC1000 requer o envio de 29 frames de dados de 16 bits cada (7 bits de endereço, bit de escrita ou leitura e 8 bits de dados). Todos os registradores são de escrita ou leitura.

Os dados são transferidos de e para o microcontrolador AVR via um barramento SPI (Serial Peripheral Interface) dedicado, e o rádio gera uma interrupção a cada 8 bits quando em modo de recepção.

Interface de Sensores e Programação

O Mica2 interfaceia com dispositivos externos através de um conector de 51 pinos ligados a pinos de I/O da CPU. Este conector provê acesso aos pinos de GPIO (General Purpose Input or Output), UART e barramento I²C do AVR, e é utilizado para programação do dispositivo e como interface para placas de sensor.

2.1.1.2 Placas de Sensores

Apesar do design modular dos motes permitir uma ampla gama de sensores analógicos e digitais ligados ao Nodo de Sensor, a placa de sensores de referência para a plataforma Mica é a Sensorboard” (Ver figura 4 [38]). Uma micasb completamente populada possui cinco módulos diferentes de sensores que permite uma ampla variedade de aplicações de redes de sensores. Estes sensores incluem: luz, temperatura, aceleração, campo magnético, e acústica, e cada um destes sensores está disponível comercialmente [23].

Um fotorresistor é utilizado como sensor de luz. Um acelerômetro ADXL202JE da Analog Devices é capaz de detectar aceleração em dois eixos. Para campo magnético, a placa é equipada com um magnetômetro de dois eixos HMC1002 da Honeywell. Um microfone omni-direcional Panasonic WM-62A é utilizado para capturar sinais acústicos, que são amplificados e passam por filtros passa-faixa da banda de voz antes de serem amostrados.

Além dos sensores acima, a placa é capaz de gerar saída acústica utilizando seu buzzer de 4kHz e único tom. Hardware opcional para detectar o tom gerado em nodos receptores é provido por um filtro passa-banda e um decodificador de tom LMC567 da National Semiconductor.

Todos os módulos na placa de sensor podem ser energizados independentemente, e são isolados energeticamente do processador do mote através de um switch analógico. Finalmente, a amplificação do magnetometro e do microfone é ajustável através de potenciômetros sobre o barramento I²C [18].

2.1.1.3 Programação

Os Mica Motes são programados por uma placa gateway que interfaceia com a interface de programação e sensores do mote e com uma porta serial de PC. Maiores discussões sobre a programação do processador AVR serão apresentadas no Capítulo 3.

Dado que muitas vezes o ambiente em que os motes são instalados é inóspito ou mesmo inalcançável, e que muitas vezes a aplicação de sensoriamento deve ser ajustada ou completamente alterada, reprogramação *over-the-air* é uma forte necessidade. Pesquisas atuais fazem uso de um co-processador simples e a memória flash nos motes para reprogramação completa [22], ou instalação de aplicações *over-the-air* fazendo uso de máquinas virtuais [28].

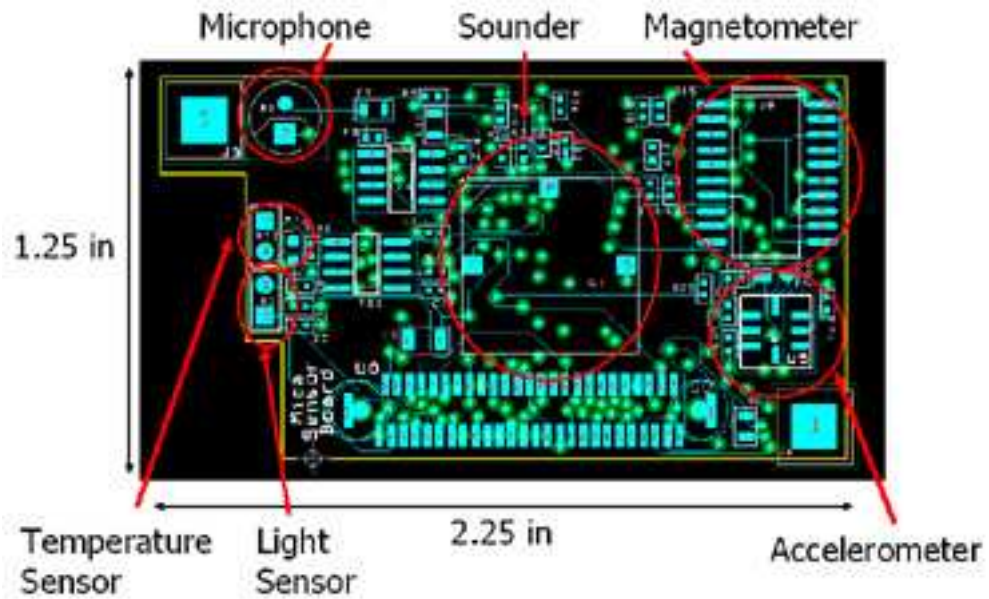


Figura 4: A Placa de Sensores Mica

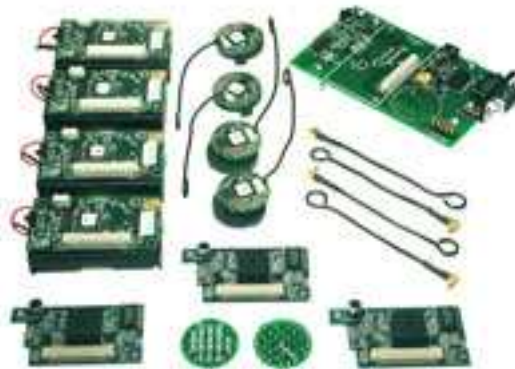


Figura 5: Kit de Motes da Crossbow

2.1.1.4 Disponibilidade

Kits UCB Mote são disponíveis comercialmente através da Crossbow Technology Inc., um fabricante de Sensores Integrados de Silicon Valley. Estes kits incluem de 2 a 20 motes Mica2 e Mica2Dot, placas de programação e sensores. Em Dezembro de 2003, o preço sugerido para um MOTE-KIT5040, incluindo 4 motes Mica2 e 4 motes Mica2Dot, 5 placas de sensores e uma placa programadora (Ver Figura 5) era de U\$ 1,995.00.

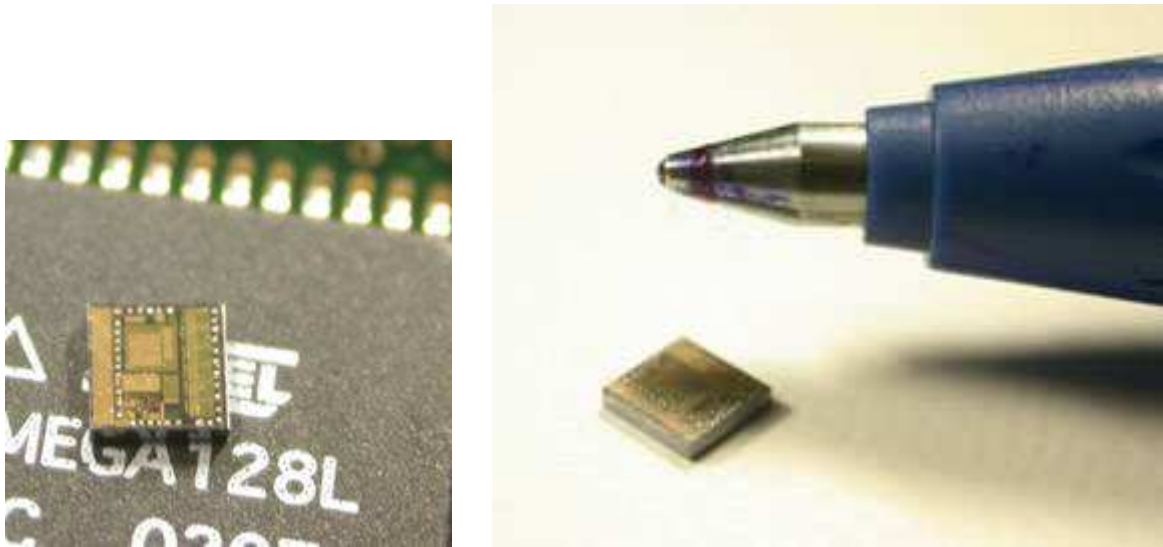


Figura 6: O Mote Spec

2.1.1.5 Desenvolvimento Futuro

O próximo passo para a família Mote é, sem dúvidas, design em chip único. No primeiro semestre de 2003, os primeiros testes bem sucedidos com o Spec, o primeiro mote em um único chip, foram realizados. O Spec (Ver Figura 6) mede aproximadamente 2 x 2.5 mm, possui um core AVR-like, 3K de memória, ADC de 8-bits on-chip, interface de programação SPI, janelas de registradores, sistema de memória paginado, UART compatível com RS232, porta de entrada de 4-bits e porta de saída de 4-bits. O custo de produção previsto para o SPEC é de U\$ 0.30, com as baterias inclusas.

2.1.2 Trabalhos Relacionados

Apesar da maior parte da pesquisa atual em Nodos de Sensores sem Fios vir de, ou estar relacionada ao grupo dos Motes na UCB ², há vários outros projetos de hardware relacionados em desenvolvimento.

PicoRadio, também desenvolvido em Berkeley, tem como objetivo desenvolver transceivers em meso-escala de baixo custo (menos de 50 centavos de dólar) para sensoria-mento pervasivo sem fios com minimização de consumo de dissipação de energia.

The Manatee project, desenvolvido na Universidade de Copenhagen, tem como objetivo estudar a disseminação de dados baseada em Bluetooth em aplicações de

²O Grupo Motes/TinyOS em Berkeley é coordenado pelo Prof. David Culler, e apoiado pela Intel.

monitoramento.

WINS (Wireless Integrated Network Sensors), da UCLA, é outra iniciativa de design de nodos de sensores sem fios em um único chip.

Smart Dust, diretamente relacionado aos projetos dos Motes, tem como objetivo encapsular um sistema completo de sensoriamento e comunicação em um milímetro cúbico por um custo relativamente baixo.

2.2 Comunicação em Rede de Sensores sem Fios

O componente de rede de Rede de Sensores sem Fios apresenta uma série de novos desafios de projetos é um tópico aberto de pesquisa.

Redes de Sensores devem ser cientes de energia. A maioria dos protocolos de rede atuais são conservativos somente em seu uso de largura de banda. Em um nodo de sensor, toda comunicação, incluindo escuta passiva, terá efeito significativo nas reservas limitadas de energia do nodo.

Redes de Sensores são altamente dinâmicas. Com o tempo, sensores podem falhar, ou novos sensores podem ser adicionados. É provável que os sensores mudem de posição e alcançabilidade. Estas mudanças tornam inaceitável uma configuração estática.

Redes de Sensores devem ser autoconfiguráveis. Um único humano pode ser responsável por milhares de nodos em uma rede de sensores densa, e um projeto onde cada nodo requer atenção individual seria impraticável.

Todas estas características, apresentadas em [24] e discutidas profundamente em [12, 39, 19], podem afetar muitos aspectos do projeto do sistema, incluindo mecanismos de roteamento e endereçamento, serviços de nomes, mecanismos de segurança e assim por diante. Esta seção, ainda que não discutindo a fundo nenhum destes desafios, apresenta os conceitos básicos para comunicação em RSSF, incluindo projeto de Camada de Enlace e mecanismos de Roteamento.

2.2.1 Camada de Enlace

A camada de enlace é responsável pela multiplexação de streams de dados, detecção de frame de dados, acesso ao meio e controle de erro [27], e garante conexões ponto-a-ponto e ponto-a-multiponto em uma rede de comunicações.

Apesar do controle de transmissão e acesso ao meio (MAC) serem bem estudados para redes tradicionais de computadores, as diferenças tecnologias sem fio, características de aplicações e cenários de uso criam um complexo mix de assuntos relacionados ao design do MAC para Rede de Sensores sem Fios

As capacidades dos dispositivos sensores também são muito diferentes das de nodos tradicionais em uma rede de computadores. Tipicamente nestas plataformas um rádio de baixa potência entrega banda em um único canal. Há pouco ou nenhum suporte dedicado para *carrier sensing*, detecção de colisões, e não há nenhum enquadramento ou codificação forçados pelo hardware. Além disso, não há pilhas de protocolo específicas instaladas para ditar o projeto do protocolo MAC [39].

Esta seção apresenta uma visão geral dos principais aspectos de projeto do MAC em Rede de Sensores sem Fios, e apresenta possíveis soluções baseadas no trabalho de [39].

2.2.1.1 Mecanismo de Escuta

Mecanismos de escuta como o CSMA/CD (Carrier Sense Multiple Access With Collision Detection) são bastante efetivos quando todos os nodos podem ouvir um ao outro. Apesar de ser simples, escutar tem um custo de energia, já que o rádio deve estar ligado para poder ouvir o meio. Para conservar energia, o tempo de *carrier sensing* deve ser diminuído. Em muitos protocolos, como o IEEE 802.11, o canal deve ser escutado mesmo durante o tempo de *backoff*. O CSMA para redes de sensores deve tomar esta oportunidade para desligar o rádio.

Dado o fato que Redes de Sensores utilizam um rádio simples sem mecanismos de detecção de colisão por hardware, nodos que enviam dados ao mesmo tempo irão corromper uns aos outros. A solução para isto é introduzir delay aleatório na transmissão para desincronizar os nodos.

2.2.1.2 Mecanismo Baseado em Contenção

Mecanismos de controle explícito de contenção utilizados em muitos protocolos MAC como o IEEE 802.11, requerem o uso de pacotes de controle, como Request to Send (RTS), Clear to Send (CTS) and Acknowledgements (ACK). Para redes onde os pacotes são grandes, estes pequenos pacotes de controle impõe pouco overhead. Entretanto, este overhead pode ser muito substancial em Redes de Sensores, onde espera-se que os pacotes tenham apenas alguns bytes.

Sendo assim, um mecanismo de contenção para RSSF deve utilizar um número mínimo de pacotes de controle. Woo e Culler [39] sugerem que um nodo que deseja transmitir deve primeiro enviar um pacote RTS a seu nodo pai e esperar uma resposta CTS. Se nenhuma resposta é recebida por um período de timeout, o nodo entra em backoff com uma janela binária de crescimento exponencial. Similarmente, se recebe um CTS que não é destinado a ele, o nodo também entra em backoff. Se nenhum CTS é recebido após cinco tentativas, a transmissão é cancelada. Além disso se um nodo ouve um CTS antes de qualquer uma de suas transmissões, ele atrasa a transmissão pelo tempo de um pacote de modo a evitar corromper o tráfego.

2.2.2 Roteamento

Redes de Sensores apresentam várias características que as distinguem de redes de comunicação sem fios ad-hoc contemporâneas [1]:

- Não é possível ou prático construir um esquema de endereçamento global para a instalação de um amplo número de nodos de sensor.
- Ao contrário das redes de comunicação típicas, a maioria das aplicações em redes de sensores tem fluxo de dados partindo de múltiplas regiões (fontes) para um único coletor (sink).
- O tráfego gerado tem uma redundância significativa, já que múltiplos sensores em uma mesma vizinhança podem gerar os mesmos dados a respeito de um fenômeno.
- Os nodos de sensores tem grandes limitações em termos de poder de transmissão, recursos de energia, capacidade de processamento e armazenamento.

Estas características fizeram surgir muitos novos algoritmos para o problema de roteamento em redes de sensores. Esta seção sumariza os problemas de projeto de arquitetura de sistema para redes de sensores, conforme apresentados em [1] e apresenta alguns protocolos de roteamento relevantes para Rede de Sensores sem Fios.

2.2.2.1 Problemas de Projeto

Como a performance de um protocolo está diretamente relacionada com o modelo de arquitetura, esta seção apresenta os principais problemas de projeto a serem considerados por um protocolo de roteamento para Rede de Sensores sem Fios

Dinâmica da rede: Ainda que a maior parte das instalações de RSSF utilizem sensores estacionários, sensores individuais podem falhar ou esgotar suas reservas de energia, criando necessidade de alteração dinâmica de rota.

Instalação dos Nós: A topologia dos nós de sensor pode ser auto-organizável em situações em que os nós são lançados aleatoriamente, criando assim uma infraestrutura ad-hoc.

Conservação de Energia: Como a energia requerida para transmitir e receber dados é muito maior do que a requerida para sentir um fenômeno ou processar dados no nó, os protocolos de roteamento devem ser cientes de energia.

Modelos de Entrega de Dados: O modelo de entrega de dados em uma Rede de Sensores pode ser, dependendo da aplicação, contínuo, disparado por eventos ou disparado por queries. O protocolo de roteamento é altamente influenciado pelo modelo de entrega de dados, especialmente com respeito à minimização de gastos de energia e estabilidade de rotas [1].

Agregação de Dados: Já que nós de sensor podem gerar dados significativamente redundantes, pacotes similares podem ser agregados de maneira a diminuir o número de transmissões. Agregação de dados é a combinação de dados de diferentes fontes usando funções como *supressão* (eliminar duplicatas), *min*, *max* e *média* [1].

2.2.2.2 Protocolos Centrados em Dados

No roteamento centrado em dados, o coletor envia queries para certas regiões e esperada pelos dados dos sensores nas regiões selecionadas. Como os dados estão sendo requisitados por queries, e não há um identificador global no nó, é necessário um sistema de identificação baseada em atributos para especificar as propriedades dos dados. Esta seção apresenta alguns protocolos centrados em dados relevantes em desenvolvimento hoje.

Directed Diffusion: Nesta solução, cada sensor nomeia os dados que gera usando um ou mais atributos. Um coletor pode propagar dados disseminando interesses, e nós intermediários propagam estes interesses. Por exemplo [35], um sensor sísmico pode gerar dados: (type = seismic, id = 66, location = SE, (type = seismic, location, SE)). Os nós intermediários propagam o interesse para a região de interesse, e os sensores que casam com este interesse mandam os dados de volta para o coletor.

SPIN: As soluções SPIN (Sensor Protocols for Information via Negotiation) foram desenvolvidas para disseminar informações de sensores individuais para todos os outros nodos, assumindo que todos são coletores em potencial [35].

Rumor Routing: Esta solução é uma variante da técnica Directed Diffusion proposta para contextos e que criterios de roteamento geográficos não se aplicam

2.2.2.3 Protocolos Hierárquicos

O principal objetivo do roteamento hierárquico é manter eficientemente o consumo de energia dos nodos de sensor envolvendo-os em uma comunicação multihop dentro de um cluster restrito e utilizando agregação e fusão de dados de maneira a diminuir o numero de mensagens transmitidas para o coletor [1]. A formação do cluster é tipicamente baseada nas reservas de energia e na proximidade dos nodos do *cluster head* LEACH (Low-Energy Adaptative Clustering Hierarchy) [21] é a família de protocolos hierárquicos mais popular e representativa para Redes de Sensores.

2.3 Aplicações de Redes de Sensores

Redes de Sensores podem ser constituídas de diferentes tipos de sensores como sísmicos, magnéticos, termais, visuais, infravermelhos, acústicos e radares, que são capazes de monitorar uma variedade de condições ambientais incluindo [2]:

- temperatura,
- umidade,
- movimento veicular,
- condições de iluminação,
- pressão,
- composição do solo,
- níveis de ruído,
- presença ou ausência de certos tipos de objetos,
- níveis de desgaste mecânico em objetos, e

- características atuais de objeto, como velocidade, direção e tamanho.

Nodos de sensor podem ser utilizados para sensoriamento contínuo, detecção de eventos, e controle local de atuadores [2]. O conceito de micro-sensoriamento pervasivo através de Rede de Sensores sem Fios promete muitas novas áreas de aplicação. Esta seção apresenta e categoriza as aplicações de RSSF em militares, ambientais, de saúde e comerciais.

2.3.1 Aplicações Militares

Rede de Sensores sem Fios podem ser uma parte integral de sistemas militares de comando, controle, comunicação, computação, inteligência, reconhecimento e alvo (C4ISRT). A instalação rápida e auto-organizável e a tolerância a falhas de redes de sensores são características que tornam-as muito promissoras para C4SIRT militares [2].

Como redes de sensores são baseadas na instalação densa de nodos de sensores descartáveis e de baixo custo, a destruição de alguns nodos não afeta tanto a operação militar quanto a destruição de um sensor tradicional, o que faz das RSSF uma boa alternativa para campos de batalha. Algumas das aplicações militares de Rede de Sensores sem Fios incluem monitoramento de forças amigas, equipamento e munição; vigilância de campo de batalha; reconhecimento de terreno e forças inimigas; alvo; avaliação de danos de batalha; e detecção química, biológica e nuclear.

2.3.2 Aplicações Ambientais

Aplicações ambientais e de monitoramento de habitat são um campo motivador para redes de sensores [4]. Suas aplicações incluem rastreamento dos movimentos de pequenos animais; monitoramento de condições ambientais; detecção química e biológica; agricultura de precisão; detecção de incêndios florestais; pesquisa meteorológica e geofísica; detecção de enchentes; mapeamento da bio-complexidade de ambientes e estudos de poluição.

2.3.2.1 Detecção de Incêndios Ambientais

Como nodos de sensores podem ser estrategicamente, aleatoriamente e densamente instalados em uma floresta, podem enviar a origem exata de incêndios para os usuários finais antes que o fogo se alastre incontrolavelmente [2].



Figura 7: Mote Firebug Instalado

O Projeto Firebug [5] utiliza uma rede de sensores termais sem fios com GPS, uma camada de controle para processamento de dados dos sensores e um centro de comando para comunicar-se interativamente com a rede de sensores. Cada nodo de sensor tem capacidades de energia, comunicação de dados e processamento para suportar, localização, sensoriamento termal e tratamento de dados. Juntamente com a posição geográfica, os motes Firebug (Ver Figura 7) medem temperatura, umidade e intensidade da luz.

2.3.2.2 Monitoramento de Habitats

Pesquisadores de Ciências Naturais tem uma crescente preocupação com os impactos potenciais da presença humana no monitoramento de plantas e animais selvagens em habitat natural [29]. Neste contexto, redes de sensores representam um avanço significativo sobre métodos tradicionais e invasivos de monitoramento. Os sensores podem ser instalados antes da temporada de migração ou outro período relevante (no caso de animais), ou enquanto as plantas estão dormentes ou enquanto o chão está congelado (no caso de estudos botânicos). Sensores podem ser instalados em pequenas ilhas onde seria inseguro ou desaconselhável que houvessem estudos de campo repetitivos [29].

O Projeto de Monitoramento Great Duck Island [29] é uma aplicação piloto para o estudo de aves marinhas migratórias na costa do Maine. Na primavera de 2002, 32 nodos de sensor sem fios foram instalados na Great Duck Island, Maine (Ver figura 8). Estes nodos monitoram os micro-climas dentro e ao redor dos ninhos. No final da temporada migratória, em Novembro de 2002, mais de 1 milhão de leituras haviam sido coletadas dos 32 motes instalados na ilha e disponibilizados pela Internet através do website do Projeto GDI.



Figura 8: Motes instalados na Great Duck Island

2.3.3 Aplicações de Saúde

Algumas das aplicações de saúde para Rede de Sensores sem Fios incluem prover interfaces para deficientes; monitoramento integrado de pacientes; administração de medicamentos em hospitais; monitoramento de movimentos e processos internos de insetos ou outros animais pequenos; tele-monitoramento de dados fisiológicos humanos; e rastreamento e monitoramento de médicos e pacientes em hospitais [2].

2.3.4 Aplicações Comerciais

Aplicações comerciais para RSSF incluem monitoramento de materiais; gerência de estoques; monitoramento de qualidade de produtos; construção de espaços de escritório inteligentes; controle de robôs; brinquedos interativos; controle e automação de processos de fábrica; estruturas inteligentes com nodos de sensores embutidos; diagnóstico de máquinas; instrumentação de fábricas; controle local de atuadores; e detecção de veículos [2].

3 *A Arquitetura AVR*

AVR é uma família amplamente utilizada de microcontroladores RISC e 8-bits da Atmel. Normalmente implementada na forma de MCUs (Microprocessor Control Units¹), o AVR provê boa performance por custo baixo e consumo baixo de energia em uma arquitetura de Harvard² simples, fazendo dele a escolha natural para processamento e controle de Nodos de Sensor Sem Fios.

Este capítulo descreve a arquitetura AVR e a MCU AVR AT90S8515, utilizada na primeira implementação do sistema de inicialização do EPOS para a arquitetura AVR (Ver Capítulo 4).

3.1 Visão Geral da Arquitetura

A CPU AVR lembra a maior parte dos processadores RISC, mas com registradores menores. O core tem 32 registradores idênticos de 8-bits que podem armazenar dados ou endereços. Como ponteiros de 8-bits são insuficientes mesmo em um dispositivo de 8-bits, os últimos seis registradores podem ser utilizados em pares, como ponteiros de endereço. Chamados de X, Y, e Z, estes três meta-registradores podem ser utilizados por qualquer operação de load ou store [36]. Todas as operações são registrador-registrador; o chip segue um modelo load/store estrito.

A Figura 9 apresenta um Diagrama de Blocos para a arquitetura AVR.

3.1.1 Registradores de Propósito Geral

O banco de registradores de rápido acesso do AVR contém 32 x 8-bits registradores de propósito geral com acesso em ciclo único, permitindo operações na Unidade Lógica e Aritmética (ULA) em ciclo único. Seis dos 32 registradores podem ser utilizados como

¹ Em uma MCU, o processador, memória e I/O residem em um único CI (Circuito Integrado).

² Uma Arquitetura de Harvard provê barramentos separados para programa e dados.

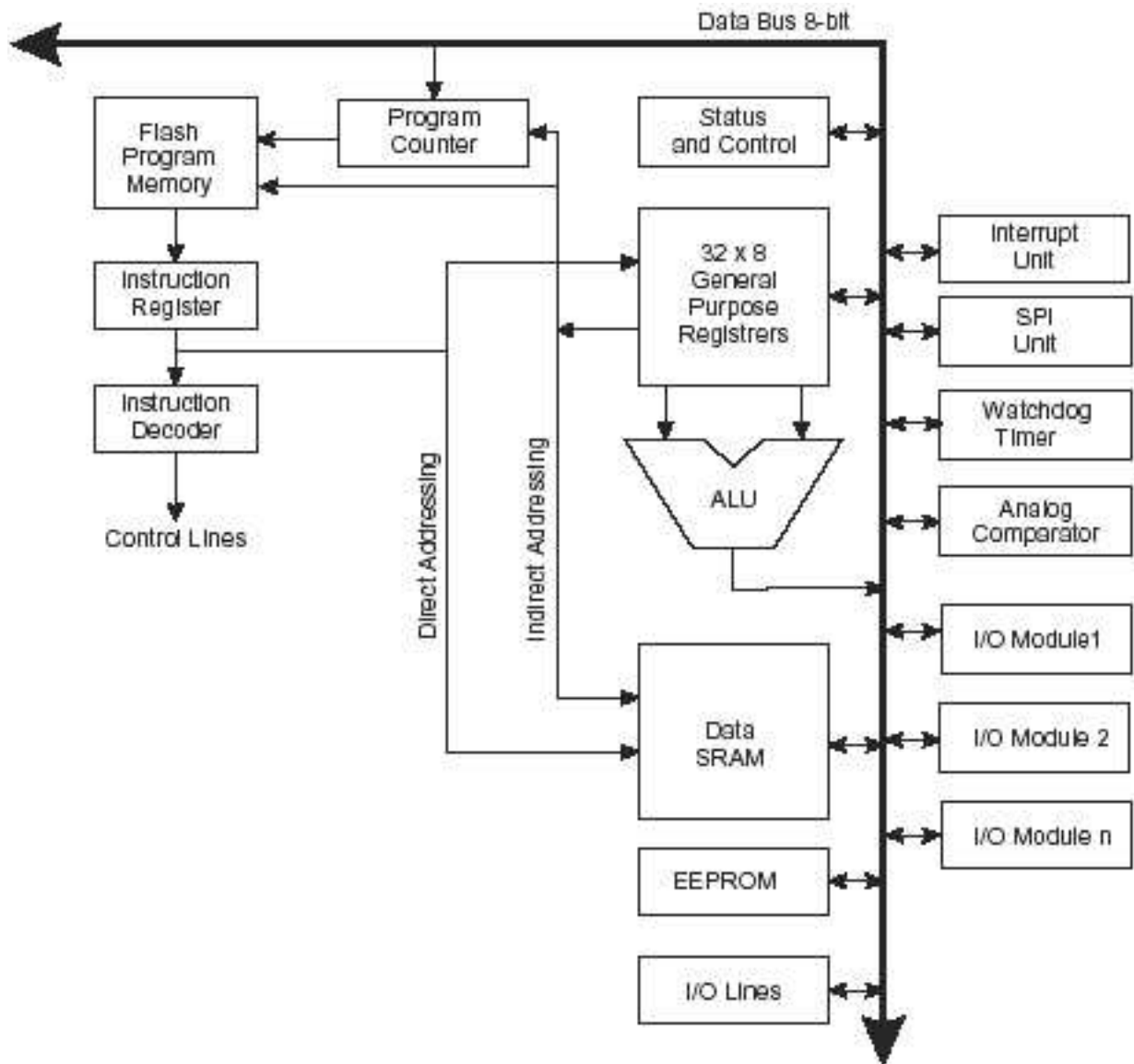


Figura 9: Diagrama de Blocos da Arquitetura AVR [9]

três ponteiros indiretos de 16-bits para endereços de Dados.

O banco de registradores é mapeado no espaço de endereçamento de dados. Os primeiros 32 bytes de memória de dados, \$0x0000 – \$0x001F, correspondem aos registradores R0-R31.

3.1.2 Registradores de I/O

Os 64 Registradores de I/O do AVR são mapeados em memória nos endereços \$0x0020 – \$005F. Estes registradores incluem status, controle de interrupções e timer, stack pointer, registradores de GPIO (General-Purpose Input and Output), SPI (Serial Program-

ming Interface) e de UART. A Tabela 2 apresenta os registradores de I/O para a MCU AT90S8515. Endereços reservados e não utilizados não são apresentados na tabela.

3.1.2.1 Registrador de Status

O registrador de Status contém informações sobre o resultado a última operação aritmética executada. Esta informação pode ser utilizada para alterar o fluxo de execução do programa através de operações condicionais. O Registrador de Status do AVR é definido como:

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

A pilha é utilizada principalmente para armazenar dados temporários, variável locais e endereços de retorno depois de chamadas de sub-rotinas e interrupções. O registrador Stack Pointer sempre aponta para o topo da pilha. A pilha é implementada crescendo das posições mais altas para posições mais baixas, assim um comando de PUSH na pilha diminui o Stack Pointer. O Stack Pointer do AVR é implementado como dois registradores no espaço de I/O, SPL e SPH.

3.1.3 Memória de Dados

Na MCU AT90S8515, os primeiros 96 endereços referem-se ao banco de registradores e memória de I/O. As próximas 512 posições endereçam a memória SRAM de dados interna.

Uma memória SRAM externa opcional pode ser colocada no mesmo espaço de endereços, preenchendo os 64K de espaço de endereçamento do AVR. A Figura 10(b) apresenta o mapa de memória de dados do AT90S8515.

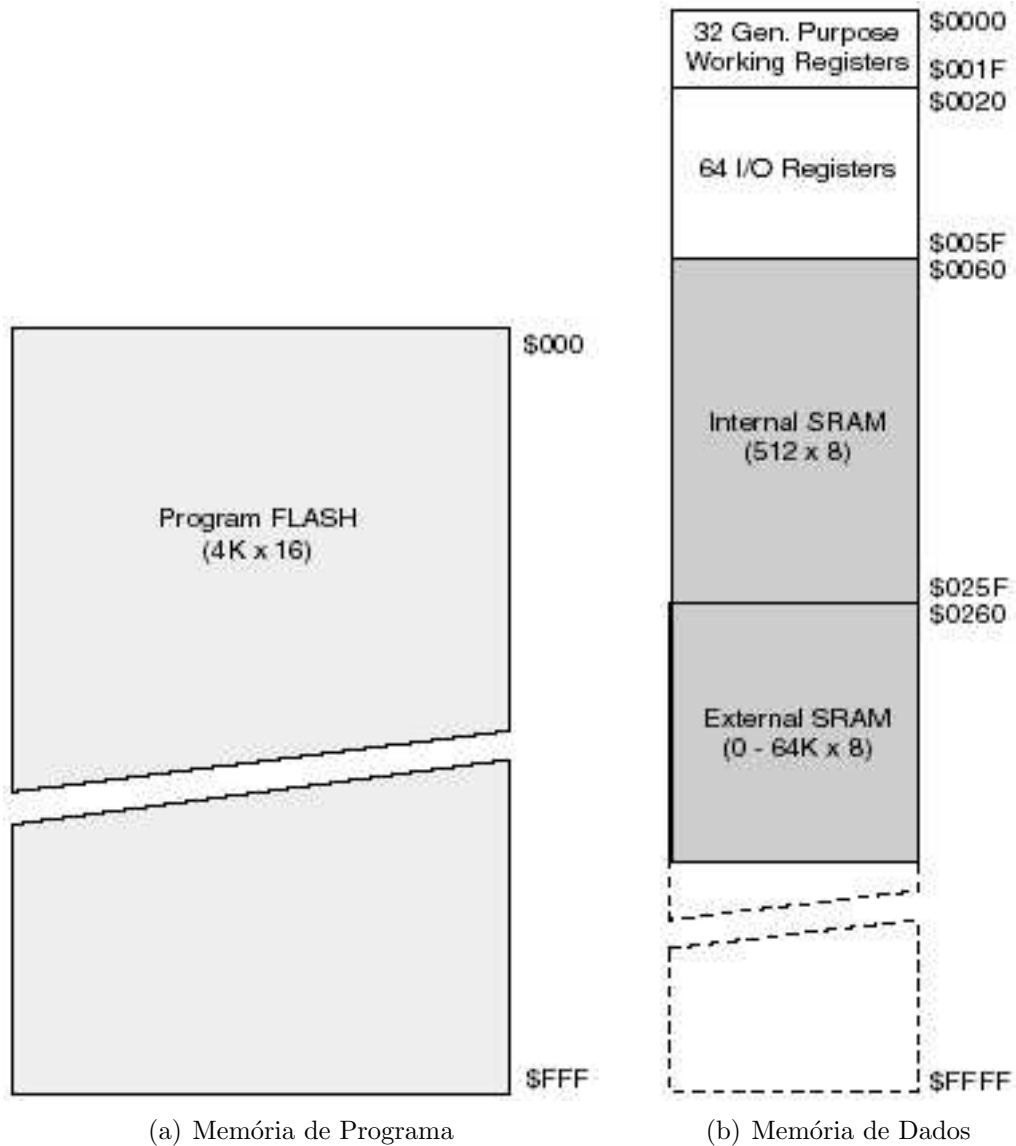


Figura 10: Mapas de Memória do AVR [9]

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

Tabela 2: Registradores de I/O do AVR

3.1.4 Memória de Programa

A MCU AT90S8515 contém 8K bytes de Memória Flash Programável para armazenamento de programas. Como todas as instruções são palavras de 16-bits ou 32-bits, a Flash é organizada como 4Kx16. O Program Counter do AT90S8515 tem 12 bits de largura, endereçando os 4096 endereços de memória de programa.

Nos primeiros modelos de AVR, como o AT90S8515, a memória de programa somente pode ser alterada escrevendo uma imagem binária completa para a flash. Uma vez que os dados de programa são baixados, não é mais possível alterar a flash, e não há instrução capaz de escrever na memória de programa. Estes dispositivos podem ser programados serialmente, via ISP (In-System Programming) ou paralelamente, via Programação em Alta Voltagem.

ISP (In-System Programming) utiliza a interface SPI (Serial Peripheral Interface) interna do AVR para fazer download do código para a flash e memória EEPROM do AVR. ISP requer apenas os pinos VCC, GND, RESET e 3 linhas de sinal para programação. Todos os dispositivos AVR, exceto os AT90C8534, Attiny11 e ATtiny28 podem ser programados via ISP. O AVR pode ser programado na voltagem normal de operação, normalmente 2.7V-6.0V. Nenhum sinal de alta voltagem é requerido. O programador ISP pode escrever tanto na flash interna quanto na memória EEPROM [11].

Para programação em alta voltagem, um sinal de 12V é aplicado ao pino de RESET do dispositivo AVR. Todos os AVRs podem ser programados via Programação em Alta Voltagem, e o dispositivo alvo pode ser programado enquanto está montado em seu socket.

3.1.4.1 Auto-Programação

MCUs AVR recentes, como o Atmega128, utilizado nos Mica Motes, provém uma instrução SPM (Store Program Memory) capaz de escrever e apagar uma página na memória de programa.

Nos MCUs onde a instrução SPM está disponível, a memória Flash é dividida em duas seções, uma seção de Aplicação e uma de Boot Loader. A instrução SPM somente pode ser executada a partir da Seção Boot Loader [10]. A memória Flash é dividida em página contendo 32, 64, ou 128 palavras cada. A organização da memória é apresentada na Figura 11.

Todas as operações de Auto-Programação são feitas utilizando a instrução SPM. Di-

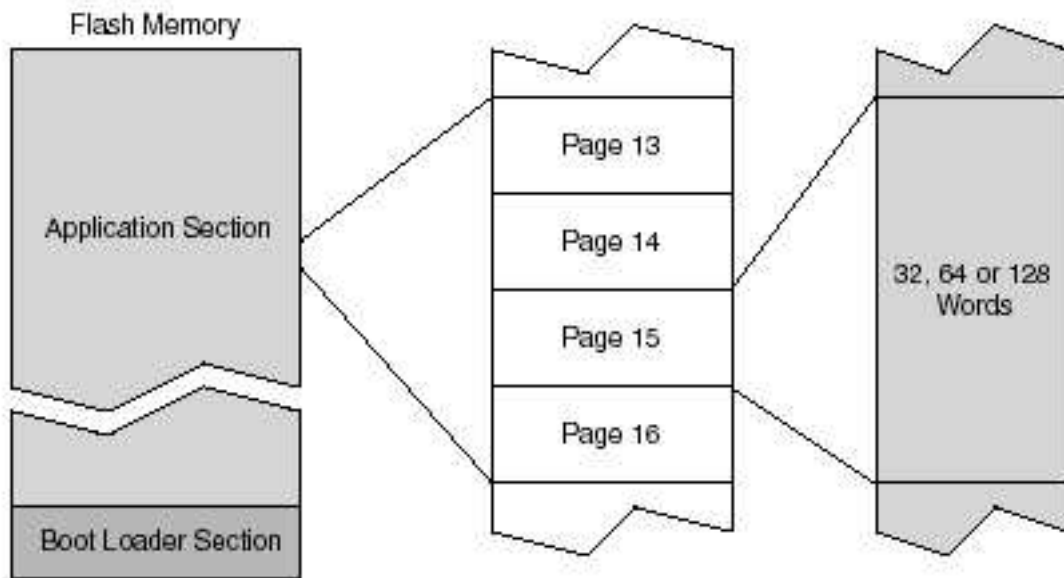


Figura 11: Organização da Memória em AVR's Auto-Programáveis [10]

ferentes operações (apagar páginas, preencher buffers e escrever páginas) são selecionadas utilizando o Registrador SPMCR. As atualizações da memória Flash são feitas página por página.

Antes de escrever novos dados para uma página, esta deve ser apagada. O Registrador Z (R31:R30) é utilizado para selecionar a página a ser apagada.

Para escrever novos dados a uma página, o Buffer de Página deve ser previamente preenchido. O Buffer de Página é buffer somente de escrita separado (fora da SRAM) que armazena uma página temporária, e que deve ser preenchido palavra por palavra, utilizando os registradores R1:R0.

Quando o Buffer de Página é carregado com novos dados, deve ser escrito para a Memória Flash. Para isto, o registrador Z é utilizado para selecionar a página a ser escrita, e a operação de escrita de página é selecionada no Registrador SPMCR.

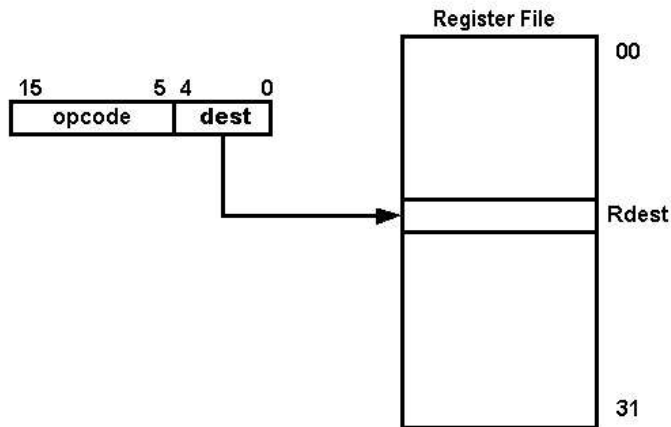
3.1.5 Modos de Endereçamento

Esta seção descreve os vários modos de endereçamento fornecidos pela arquitetura AVR para acesso à memória de programa (Flash) e dados (SRAM, Arquivo de Registradores e Memória de I/O).

3.1.5.1 Direto para Registradores – Único Registrador

Um dos 32 registradores de propósito geral (dest) contém o operando da instrução.

Exemplo: CLR R0 ; R0 is cleared

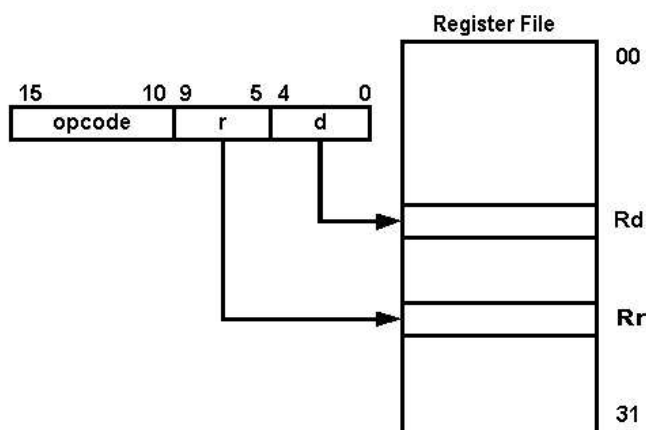


Register Direct (One Register)

3.1.5.2 Direto para Registradores – Dois Registradores

Dois dos 32 registradores de propósito geral contém os operandos da instrução; um é o Registrador Fonte e outro é o Registrador de Destino.

Exemplo: ADD R0,R1 ; R0 = R0 + R1

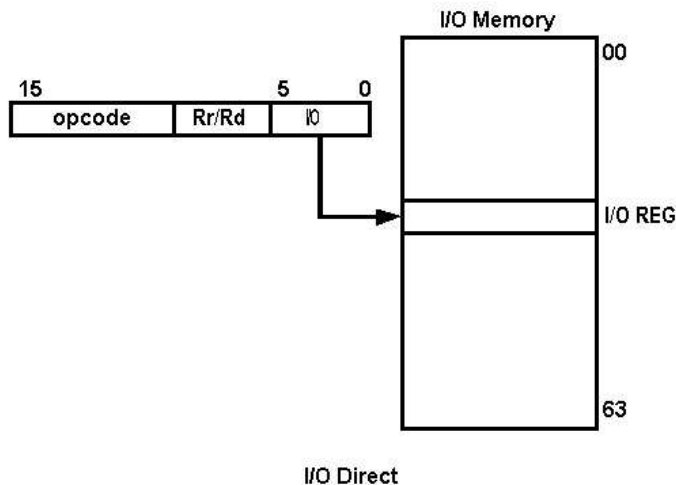


Register Direct (Two Registers)

3.1.5.3 I/O Direto

Um endereço dos 64 Registradores de I/O fica nos 6 bits da porção I/O da instrução. O endereço do Registrador Fonte ou Destino fica nos 6 bits restantes dos operandos da instrução.

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

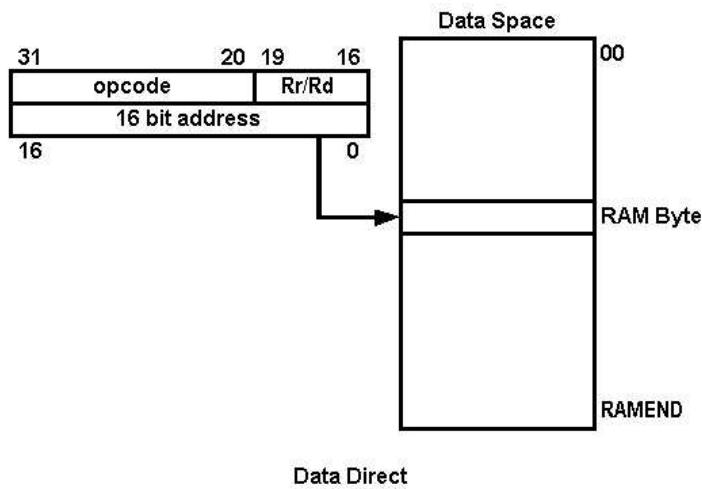


3.1.5.4 Direto para Dados

Um endereço de Dados de 16-bits é especificado com um operando (para acessar até 64K de Memória RAM). O endereço do Registrador Fonte ou Destino fica nos 6 bits restantes dos operandos da instrução.

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.

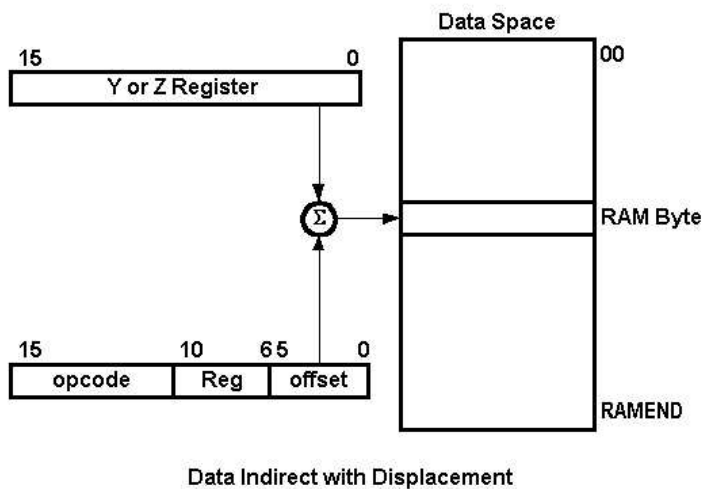
Exemplo: `LDS R0,$1234 ; R0 = &0x1234`



3.1.5.5 Indireto para Dados com Deslocamento

O endereço a ser acessado é o resultado da soma dos registradores Y ou Z e o offset de seis bits encontrado na instrução. Reg é o Registrador Fonte ou Destino.

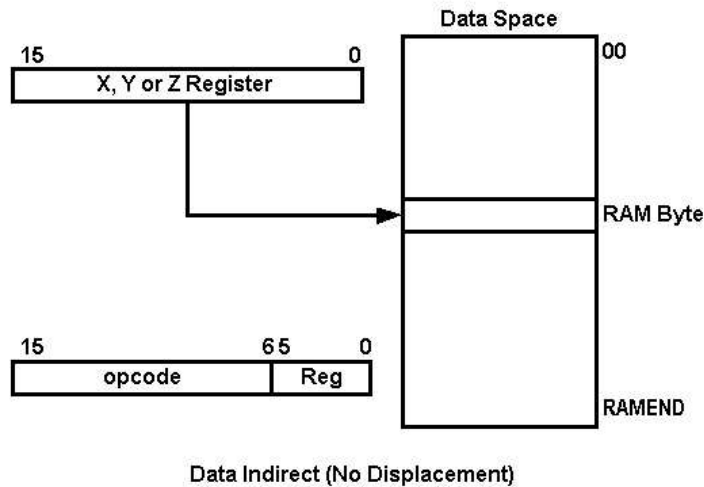
Exemplo: LDD R0,Y + \$3F ; R0 = &(\$3F + [Y])



3.1.5.6 Indireto para Dados

O endereço a ser acessado é o conteúdo dos registradores X, Y ou Z. Reg é o Registrador Fonte ou Destino.

Exemplo: LD R0,X ; R0 = &[X]



Este modo também pode ser utilizado com Pré-Incremento e Pós-Incremento do Registrador de Endereço.

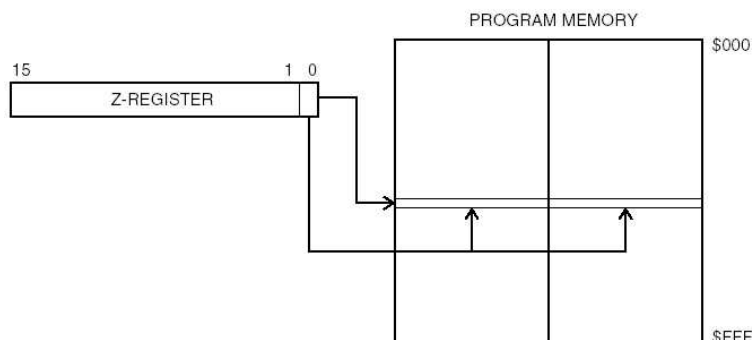
Pre-Decrement Example: LD R0,Z- ; R0 = &[--Z]

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

3.1.5.7 Acesso de Constantes Utilizando LPM

Uma palavra da memória de código é especificada pelo 15 bits mais significativos do registrador z (Z15:1). O bit menos significativo seleciona o bit a ser buscado e armazenado no registrador R0.

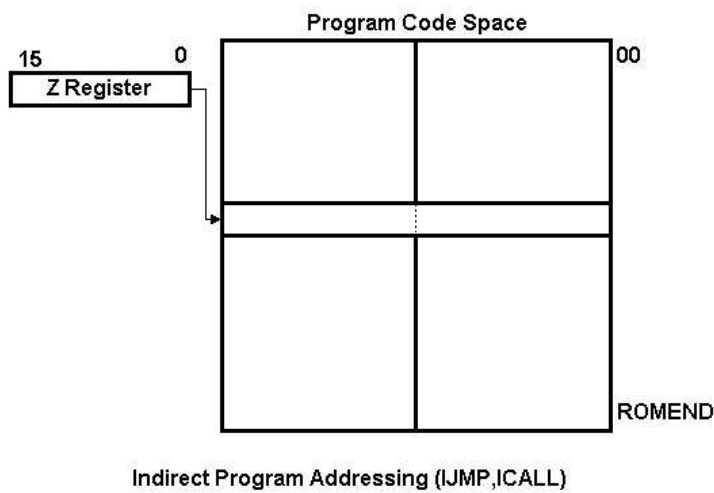
Exemplo: LPM ; R0 = &(Program Memory Address [Z] >> 1) (Z0 selects high or low byte.)



3.1.5.8 Endereçamento Indireto de Programa

A execução do Programa continua a partir do local especificado pelo registrador Z.

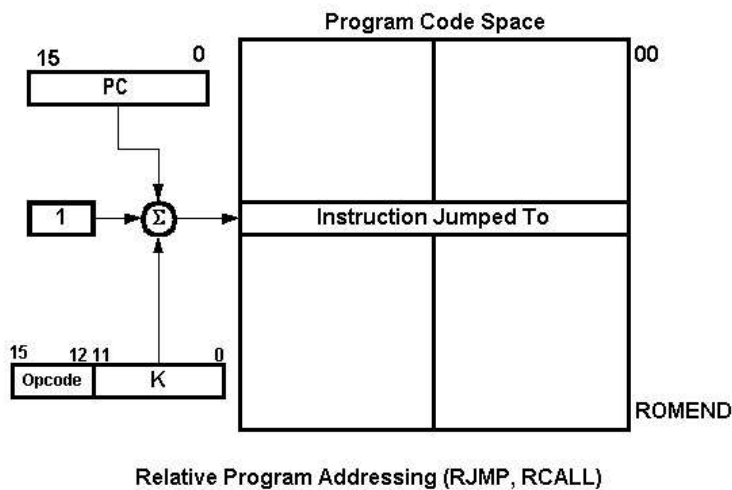
Exemplo: IJMP ; PC = Z



3.1.5.9 Endereçamento Relativo de Programa

A execução do programa continua a partir do local especificado pela soma do PC, o offset relativo K e um. O offset relativo vai de -2048 a 2047.

Exemplo: RJMP \$020 ; PC = PC + 20 + 1



3.2 Timers

A MCU AVR AT90s8515 provê dois timers (um de 8-bits e outro de 16-bits). Em princípio, um timer é um simples contador. Sua vantagem é que o clock de entrada e a operação do timer é independente da execução do programa. O clock determinístico possibilita medir o tempo contando ciclos passados e levando a frequência de entrada do

timer em consideração [7].

3.2.1 Eventos de Timer

Os timers do AVR podem ser configurados para monitorar diversos eventos:

Timer Overflow: O contador contou até seu valor máximo e será resetado para zero no próximo ciclo de relógio do timer.

Compare Match: O Registrador de I/O “Output Compare” pode ser carregado com um valor contra o qual o timer será checado a cada ciclo de clock do timer. Quando o timer chega ao valor de comparação, um evento é sinalizado e o Timer pode ser configurado para limpar o valor do contador para “0”.

Input Capture: O AVR tem um pino de entrada para disparar o evento captura de entrada. Uma mudança neste sinal faz com que o valor do timer seja lido e salvo no registrador “Input Capture”. Isto é útil para medir a largura de pulsos externos.

O timer opera independentemente da execução do programa, e para cada evento de timer é um flag de status diferente no registrador de Interrupção de Timer. A ocorrência de um evento de timer pode ser monitorada por polling constante de flags de status ou quebrando o fluxo de execução através da execução de rotinas tratadoras de interrupção.

3.2.2 Watchdog Timer

Um Watchdog Timer é um hardware que pode resetar um processador quando julga que o sistema pendurou, ou não está mais executando a seqüência correta de código [30].

O Componente de Hardware de um Watchdog Timer é um contador que é setado para um certo valor e então conta para baixo até zero. É responsabilidade do software setar o contador para seu valor original frequentemente o suficiente para garantir que nunca alcance zero. Se o contador alcançar zero, assume-se que o software falhou de alguma maneira e a CPU é resetada ou uma interrupção é gerada.

No AT90S8515, o Watchdog Timer (WDT) é independente do restante do sistema. Tem seu próprio oscilador interno, que executa enquanto um dos modos de operação (Reset ou Interrupção) está habilitado . Isto garante operação segura mesmo quando o oscilador da CPU principal falhar [8].

3.3 Interface Serial de Periféricos

Serial Peripheral Interface (SPI) é um padrão de barramento serial estabelecido pela Motorola e suportado em produtos de silício de diversos fabricantes. É um link de dados seriais síncrono que opera em modo full duplex (Sinais carregando dados nas duas direções simultaneamente) [25].

Os dispositivos comunicam-se usando um relacionamento mestre/escravo, no qual o mestre inicia um frame de dados. Quando o mestre gera um clock e seleciona um dispositivo escravo, dados podem ser transferidos em ambas direções simultaneamente.

O barramento SPI especifica quatro sinais: clock (*SCLK*); master data output, slave data input (*MOSI*); master data input, slave data output (*MISO*); and slave select (\overline{SS}). *SCLK* é gerado pelo mestre e é entrada de todos escravos. *MOSI* carrega dados do mestre para o escravo. *MISO* carrega dados do escravo para o mestre. Um dispositivo escravo é selecionado quando o mestre indica o seu sinal \overline{SS} . Se existem múltiplos dispositivos escravos, o mestre deve gerar um sinal separado de seleção para cada dispositivo. A Figura 12 apresenta uma implementação SPI de único mestre e múltiplos escravos.

O AT90S8515 provê uma implementação completamente funcional de SPI, capaz de trabalhar em modo mestre ou escravo e controlada por registradores de I/O mapeados em memória.

3.4 UART

A MCU provê uma UART (Universal Asynchronous Receiver/Transmitter) full-duplex, com:

- Gerador de Baud Rate
- Filtro de Ruídos
- Detecção de Overrun
- Três interrupções separadas para TX Complete, TX Data Register Empty e RX Complete.

A transmissão e recepção de dados, bem como configuração da UART é controlada por registradores de I/O mapeados em memória.

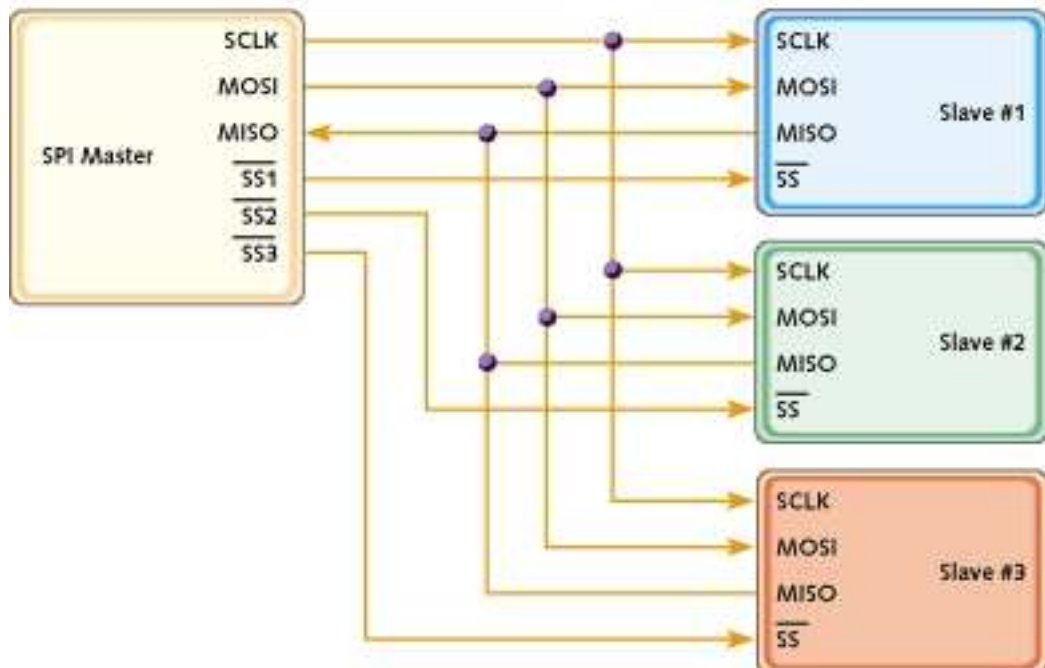


Figura 12: Implementação SPI de Múltiplos Escravos [25]

3.5 GPIO

A MCU AVR AT90S8515 provê quatro portas de I/O bi-direcionais. Três registradores de I/O são alocados para cada porta, um cada para o registro de dados, PORTx, Registro de Direção de Dados, DDRx, e os pinos de entrada da Porta x, PINx. Este último permite acesso ao valor físico em cada pino da Porta x. Os pinos de entrada de cada Porta são valores apenas de leitura, já o Registro de Dados e de Direção de Dados são de leitura e escrita. Todas as portas podem ser utilizadas para GPIO (General Purpose Input or Output).

3.6 Modos de Economia de Energia

Os microcontroladores AVR provém diversos modos de economia de energia. O propósito destes modos é permitir uma maneira de suspender a execução do programa quando necessário, diminuindo assim o consumo de energia [13]. Os modos de economia do AT90S8515 são (em ordem de consumo de energia máximo para mínimo):

- Modo Idle

O módulo idle pára a CPU mas deixa os periféricos (UART, Comparador Analógico,

etc.) executando. A MCU continua a execução imediatamente após acordar do modo idle.

- Modo Powersave

Este modo é idêntico ao modo Power-down, com uma exceção: O oscilador do cristal do timer continuará a operar e o timer pode continuar a contar. O dispositivo pode acordar de um evento de Timer Overflow ou Output Compare.

- Modo Powerdown

Neste modo, todos os osciladores são parados, e somente as interrupções de nível externo e o Watchdog Timer continuam operando. Somente um Reset externo, um Reset do Watchdog ou uma interrupção externa podem acordar a 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 instruction.

4 *Inicialização do EPOS no AVR*

O sistema EPOS nasceu em 1997 como um projeto para experimentar com os conceitos e mecanismos do design de sistema orientado a aplicação [14]. O EPOS é assim um sistema operacional intrinsecamente orientado a aplicação, e hoje está se transformando em um SO totalmente funcional, multi-plataforma e de altíssima performance. Resultados atuais incluem implementações em Clusters de Workstations baseados em Redes Myrinet [16, 17, 15] e portes para as arquiteturas PowerPC (32-bits) e H8 (8-bits) [32]. O EPOS tem como objetivo entregar funcionalidade (dando à aplicação seu suporte de execução necessário), adaptatividade (sendo montado para aplicações específicas) e eficiência (tornando os recursos disponíveis com o menor overhead possível) [14].

Este capítulo apresenta uma visão geral do sistema EPOS, concentrando-se no processo de inicialização e sua implementação para a arquitetura AVR.

4.1 **Arquitetura do Sistema do EPOS**

O EPOS utiliza Abstrações de Sistema, Mediadores de Hardware e Aspectos para garantir reusabilidade de componentes. Abstrações descrevem funcionalidades independentes de cenário, são amplamente reutilizáveis e representam a maior parte dos componentes no sistema. Um mediador de hardware é uma abstração de elementos da plataforma de hardware dependente de sistema que são utilizado por abstrações de sistema e aspectos de cenário [14]. Aspectos provêm funcionalidades configuráveis para aplicações, como compartilhamento, proteção e atomicidade.

4.1.1 **Abstrações de Sistema**

As famílias de Abstrações de Sistema do EPOS são resultado da decomposição do domínio da computação dedicada. Abstrações de sistema são modeladas independentemente de aspectos de cenários de execução e arquiteturas específicas de sistema [14]. A

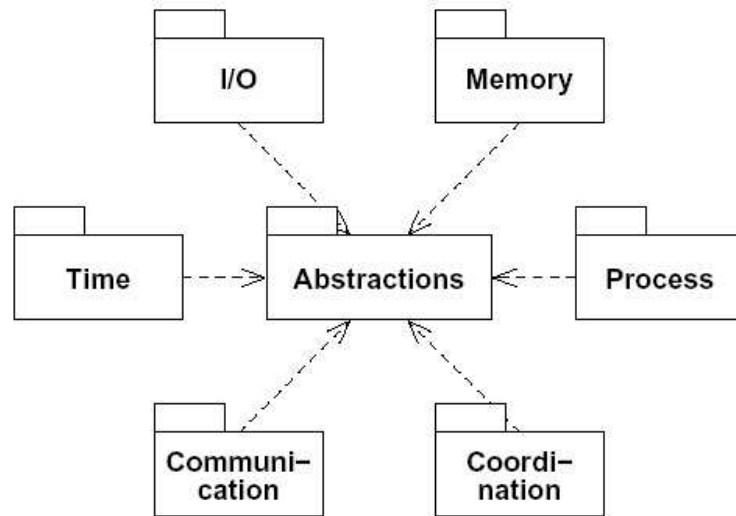


Figura 13: Famílias de Abstrações do EPOS [14]

Figura 13 apresenta uma representação do nível mais alto das famílias de abstrações do EPOS.

4.2 Inicialização do EPOS

A primeira fase do processo de inicialização do EPOS é composta pelo *bootstrap* e um utilitário de setup. O utilitário de setup executa antes do sistema operacional e constroi um contexto de execução básico para o EPOS, inicializando componentes de hardware.

A segunda fase, o utilitário de inicialização, trata da inicialização das estruturas de dados do sistema e criação do primeiro (e possivelmente único) processo de aplicação.

4.2.1 O utilitário de Setup para o AVR

O utilitário de setup do EPOS é responsável por construir um contexto elementar de execução para o SO. Ele executa depois do bootstrap e antes do utilitário de inicialização.

No AVR, o bootstrap simplesmente desabilita interrupções e chama o setup, passando como parâmetro a estrutura SysInfo. Esta estrutura descreve as características relevantes para a futura configuração do EPOS.

Quando inicia, o utilitário de Setup continua com a configuração do hardware atualizando e completando a estrutura SysInfo, incluindo informações sobre os recursos físicos configurados, um mapa de memória descrevendo como o sistema operacional foi carregado,

a identificação lógica do nodo, etc [14].

No AVR, o Setup é responsável principalmente por configurar o controlador de interrupções, verificar a integridade do sistema, configurar o ponto de entrada do utilitário de Inicialização e configurar estruturas de dados do sistema.

4.2.2 O Utilitário de Inicialização

O utilitário de inicialização do EPOS é uma rotina que tem acesso a todo o espaço de endereçamento do sistema operacional, sendo assim capaz de invocar operações de sistema. O procedimento de inicialização executado pelo Init consiste em verificar as configurações de cada abstração de modo a determinar se estão incluídas na configuração atual do sistema e invocar o método de classe init para cada abstração presente [14].

Depois de invocar o método init para cada abstração presente, o utilitário de inicialização invoca operações do EPOS, que neste ponto estão completamente operacionais, para criar o primeiro processo. Se a aplicação dedicada executando sobre o EPOS roda em um processo único, o processo criado pelo utilitário de inicialização é o próprio processo único da aplicação. De outra maneira, este processo é um carregador que subseqüentemente cria processos de aplicação em um ambiente multi-tarefa [14].

4.2.3 Visão Geral da Inicialização do EPOS

Uma visão geral da inicialização do EPOS é apresentada na Figura 14. Depois de carregar a imagem de boot, que inclui uma descrição preliminar do sistema (SysInfo), o bootstrap invoca o utilitário de setup para configurar a plataforma de hardware. Depois o setup constrói um modelo elementar de memória, configura dispositivos, carrega o EPOS, carrega e ativa o utilitário de inicialização. O init invoca o método de classe init para cada abstração incluída no sistema para inicializar sua estrutura lógica. Termina carregando o executável incluído na imagem de boot para criar o primeiro processo [14].

4.2.4 Considerações para a Arquitetura AVR

Tendo sido projetado tendo em mente uma arquitetura Von Neuman, auto-programável, o processo de inicialização do EPOS teve que sofrer algumas mudanças quando portado para um dispositivo como o AT90S8515, uma Arquitetura de Harvard incapaz de alterar memória de programa em tempo de execução.

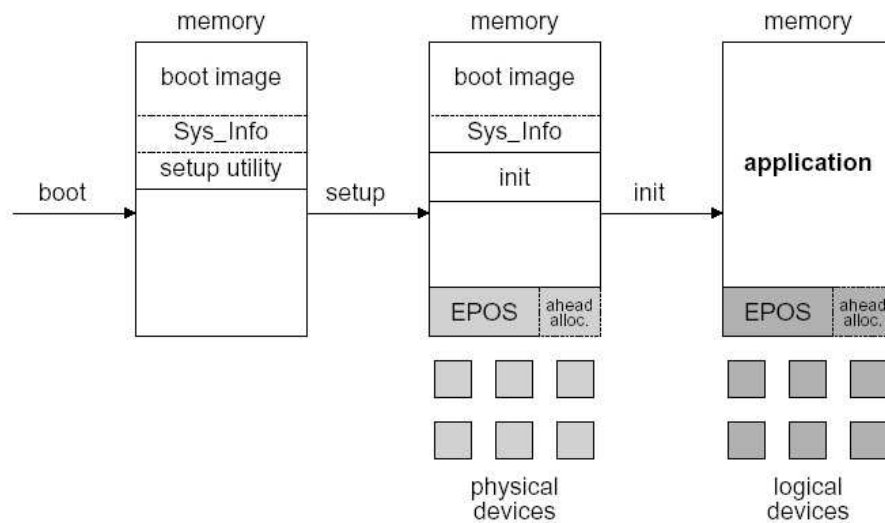


Figura 14: Visão Geral da Inicialização do EPOS [14]

Numa configuração padrão de sistema, o sistema de inicialização do EPOS é eliminado após a execução, e os recursos por ele utilizados retornam ao pool de recursos livres do sistema. Isto não é possível no AT90S8515, já que a memória de programa não pode ser alterada, e portanto liberada, em tempo de execução. Como os processos não pode ser carregados dinamicamente em tempo de execução, os ponteiros da aplicação devem ser pré-ajustados na imagem binária carregada para a MCU.

A estrutura original da imagem do EPOS também deve ser alterada tendo em mente dois espaços de endereçamento e barramentos separados. Estruturas de dados, como a SysInfo agora devem ser carregadas juntamente com o código e copiadas para memória RAM em tempo de execução.

AVRs mais recentes, como o Atmega128, permitem a possibilidade de carregar código dinamicamente em tempo de boot. Isto é feito escrevendo páginas de memória baseado em dados da memória RAM. Este processo faz uso da instrução SPM (Store Program Memory), que só pode ser executada a partir da seção Boot Loader da memória de programa. Estas MCUs AVR permitiriam que o sistema de inicialização do EPOS (bootstrap, setup e init) fosse executado da seção Boot Loader, liberando esta seção para código de aplicação depois da execução.

5 *Conclusões e Pesquisas Futuras*

A pesquisa em Rede de Sensores sem Fios é um dos campos mais promissores nas Ciências da Computação hoje, e apresenta uma série de novos desafios, entre os quais o suporte adequado de execução para aplicações é um assunto chave.

Esta pesquisa representou um primeiro esforço na implementação de um release totalmente funcional do sistema EPOS para a plataforma de RSSF da UCB (Mica Motes), a plataforma de hardware “Estado da Arte” para Redes de Sensores. Ainda que o grupo de pesquisa dos Motes em Berkeley provenha seu próprio sistema operacional para RSSF (TinyOS), ele não provê a funcionalidade avançada nem o design orientado a Aplicação que o EPOS provê. Conforme as aplicações de Redes de Sensores vão evoluindo, o TinyOS terá suporte cada vez mais inadequado, ao mesmo tempo em que o EPOS pode ser facilmente configurado e expandido para suportar as necessidades dos programadores de aplicação. A natureza altamente portátil do EPOS também garante reusabilidade, tanto no nível de sistema quanto de aplicação, conforme novas plataformas de hardware forem surgindo.

O porte de um sistema EPOS totalmente funcional para as plataformas Mica é um trabalho em constante desenvolvimento. Resultados atuais apresentam uma imagem funcional de 1.3 KB do EPOS para a MCU AT90S8515 (O código objeto desta imagem é apresentado no anexo A). Fundos do FUNGRAD/UFSC permitirão ao LISHA adquirir kits comerciais de Motes, permitindo desenvolvimento futuro concentrado em mediadores de hardware para Placas de Sensores e Tranceivers de Rádio e na implementação de Sistemas de Comunicação.

Controle de energia é um dos problemas fundamentais de RSSF, e a pesquisa e implementação de mecanismos de controle de energia para RSSF, incluindo protocolos cientes de energia também está na “Agenda de Rede de Sensores sem Fios” do LISHA

Referências

- [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 Roe 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] Ian 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 of wireless 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] Pico Radio Project. Pico radio – http://bwrc.eecs.berkeley.edu/research/pico_radio/.
- [34] TinyOS Project. Tynyos hardware designs.
- [35] Praveen Rentala, Ravi Musunuri, Shashidhar Gandham, and Udit Saxena. Survey on sensor networks.
- [36] Jim Turley. Atmel avr brings risc to 8-bit world. *Microprocessor Report*, 11(9), 1997.
- [37] Brett Warneke and Sunil Bhave. Smart dust mote core architecture.
- [38] Alec Woo. The mica sensing platform, 2002.
- [39] 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.

ANEXO A – Código Objeto para a Imagem do EPOS Gerada

```
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

00000da <_ZN6System4AVR84initEPNS_11System_InfoE>:
da: 80 e0      ldi r24, 0x00 ; 0
dc: 90 e0      ldi r25, 0x00 ; 0
de: 08 95      ret

00000e0 <_ZN6System17AT90S8515_Display4initEPNS_11System_InfoE>:
e0: 80 e0      ldi r24, 0x00 ; 0
e2: 90 e0      ldi r25, 0x00 ; 0
e4: 08 95      ret

00000e6 <_ZN6System12AT90S8515_IC4initEPNS_11System_InfoE>:
e6: 80 e0      ldi r24, 0x00 ; 0
e8: 90 e0      ldi r25, 0x00 ; 0
ea: 08 95      ret

00000ec <_ZN6System9AT90S85154initEPNS_11System_InfoE>:
ec: 80 e0      ldi r24, 0x00 ; 0
ee: 90 e0      ldi r25, 0x00 ; 0
f0: 08 95      ret

00000f2 <_ZN6System8AVR8_MMU4initEPNS_11System_InfoE>:
f2: 80 e0      ldi r24, 0x00 ; 0
f4: 90 e0      ldi r25, 0x00 ; 0
f6: 08 95      ret

00000f8 <_ZN6System15AT90S8515_Timer4initEPNS_11System_InfoE>:
f8: 80 e0      ldi r24, 0x00 ; 0
fa: 90 e0      ldi r25, 0x00 ; 0
fc: 08 95      ret

00000fe <_ZN6System8AVR8_TSC4initEPNS_11System_InfoE>:
fe: 80 e0      ldi r24, 0x00 ; 0
100: 90 e0     ldi r25, 0x00 ; 0
102: 08 95     ret

0000104 <_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

```

```

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

```

ANEXO B – Artigo

The EPOS System Supporting Wireless Sensor Networks Applications

Lucas Wanner

¹Laboratory for Software and Hardware Integration
Federal University of Santa Catarina

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 necessidades 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.

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 [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 EPOS¹ 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.

1.1. Presentation Overview

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

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

Finally the last part presents the conclusions of this paper, 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 [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, 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 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 [16] and

¹EPOS: Embedded Parallel Operating System

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 section 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 interest, and each sensor node collects data from its 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 Mote² 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 separate instruction and data memory. In the motes, the AVR interfaces with four hardware blocks (Radio, LEDs, Flash Memory and Sensor board / Programming interface).

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.

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 [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.

²Mote, 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).

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. Sensor nodes can be used for continuous sensing, event detection, location sensing, and local control of actuators [1]. The concept of pervading 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 [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 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 [1].

2.3.2. Environmental applications

Environmental and habitat monitoring is a driving field for wireless sensor networks [3]. Its 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.

2.3.3. 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].

2.3.4. 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 [1].

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 Units³), the AVR provides good performance at low cost and low power consumption in a simple Harvard Architecture⁴, making it the natural choice for Wireless Sensor Nodes processing and control.

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 [23]. All operations are register-to-register; the chip follows a strict load/store model.

3.1.1. General Purpose Registers

The AVR's fast-access Register file contains 32 x 8-bit general purpose registers with a single clock cycle access time, allowing single-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.

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-Purpose Input and Output) and SPI (Serial Programming Interface) and UART registers.

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.

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

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

⁴A Harvard Architecture provides separate memories and buses for program and data.

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.

3.2. 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.

3.3. 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.

3.4. 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).

3.5. 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 instruction.

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 [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 its necessary runtime support), customizability (being tailored to specific applications) and efficiency (making resources available to the application with the lowest possible overhead) [7].

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 [7]. Aspects provide configurable functionalities for applications, such as sharing, protection and atomicity.

4.1.1. 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.

4.1.2. 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].

4.2. 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].

4.3. 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 Platform (Mica Motes), the “state-of-the-art” hardware platform for WSN. While the Motes group at Berkeley provides its 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 allow 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.

Referências

- [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 efficient 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 Roe 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. Tynyos 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 Bhawe. 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.