Fábio Lúcio Corrêa Júnior
Orientador: Claudionor José Nunes Coelho Júnior

# Desenvolvimento de Dispositivo Nó Sensor com Arquitetura Reconfigurável para Redes de Sensores Sem Fio

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte - MG
30 de Setembro de 2004

**A Deus, aos meus pais Neuza e Fábio,**
**à minha irmã Viviane e à Cibele**

# Resumo

Redes de Sensores Sem Fio (RSSFs) surgem como uma alternativa de baixo custo para a realização do monitoramento de sistemas em diversas áreas de aplicação (militar, industrial, biomédica, aviação, ambiental, etc). RSSF é um tipo de rede cujos nós integram um ou mais sensores, conhecidos como nós sensores. Estes nós possuem como principais características, limitada capacidade computacional, tamanho reduzido, baixo custo, forte restrição de recurso de energia e a possibilidade para comunicar, utilizando-se formas de comunicação sem fio.

Esta dissertação descreve o desenvolvimento e a concepção de um protótipo de um Dispositivo Nó Sensor com Arquitetura Reconfigurável, denominado por RANS-300, para aplicações em RSSFs. A proposta desta arquitetura é de oferecer não apenas uma plataforma eficiente e flexível, mas também que possua recursos configuráveis de hardware, objetivando potencializar o emprego do RANS-300 no monitoramento de sistemas complexos que requeiram maior recurso computacional. Como resultado, o RANS-300 pode operar nos modos de baixo consumo de potência ou de alta performance, respondendo dinamicamente aos requisitos ambientais.

# Abstract

Wireless Sensor Networks (WSNs) appear as a low cost alternative for monitoring systems in a wide variety of fields (WSN can be applied in the military, industrial, biomedical, aviation or environmental areas). WSN is a kind of network whose nodes contain one or more sensors, also known as sensor nodes. The nodes main features are a limited computational capacity, a reduced size, a low cost, a strong restriction of energy resource and the possibility of communicating using wireless devices.

In this thesis we describe the development and the prototype of a Node Sensor Device with a Reconfigured Architecture, named RANS-300, applied in WSNs. The purpose of this architecture is to offer not only a platform which could be efficient and flexible, but also one that can have a hardware that could be configured in order to strengthen the RANS-300 in monitoring complex systems, which require a huge computational resource. As result, RANS-300 can operate in very low power or high-performance mode, responding dynamically to environment requirements.

# Agradecimentos

Aos professores Claudionor José Nunes Coelho Júnior, Antônio Otávio Fernandes e Antônio Alfredo Loureiro pela confiança, disposição, amizade, incentivo, apoio e orientação.

Aos funcionários do Departamento de Ciência da Computação da UFMG pelo apoio, paciência e atenção sempre demonstrada.

Aos professores do Departamento de Ciência da Computação da UFMG.

A todos participantes dos projetos SENSORNET[1] e ao Programa Nacional de Microeletrônica[2].

A Fundação para Inovações Tecnológicas (Fitec) pelo incentivo e colaboração.

A todos que contribuíram de alguma forma para a realização deste trabalho.

[1]Processo CNPq 55.2111/2002-3
[2]Processo CNPq 57.0097/2003-7

# Sumário

# Índice de Figuras

# Índice de Tabelas

# CAPÍTULO 1

# Introdução

Sensores são dispositivos empregados no monitoramento e estudo de sistemas e aplicados em diversas áreas de interesse como, por exemplo, militar, industrial, biomédica, aviação, ambiental, etc. A utilização de poucos, ou talvez apenas um sensor, pode ser suficiente para determinar o comportamento ou conhecer as características de sistemas simplificados. Por outro lado, sistemas complexos exigem um grande número de sensores para coletar eventos e informações, e é neste contexto que as Redes de Sensores Sem Fio (RSSF ou *WSN*) [2, 3, 15, 12] vem adquirindo considerável importância como uma alternativa de oferecer soluções com eficiência, precisão, segurança e baixo custo.

Figura 1.1: Aplicabilidade das Redes de Sensores.

Alguns trabalhos apresentados [4, 5, 11] mostram que é possível integrar dispositivos nós sensores para auxiliar na aquisição de eventos (sísmicos, acústicos, posição) e de grandezas físicas (temperatura, aceleração). A idéia básica é tirar proveito da implementação de

dispositivos pequenos e baratos que possam ser utilizados em larga escala [4]. Por exemplo, em reserva ecológica, distribuindo-se diversos nós sensores e recebendo suas informações, poderíamos conhecer o comportamento deste sistema, efetuando o levantamento dos níveis de poluentes (gases, substâncias químicas) e suas características ambientais (temperatura, umidade, nível pluvial) [6, 7, 27].

Desenvolver dispositivos nós sensores eficientes e otimizados, que conciliem performance computacional e autonomia de operação estendida ou auto-suficiente, é um dos grandes desafios tecnológicos a ser enfrentado na implementação e disseminação das RSSFs. Os requisitos funcionais e construtivos requeridos por esta nova classe de dispositivos, têm exigido soluções de *hardware* e *software* mais aprimoradas e integradas, e também o desenvolvimento de algoritmos específicos e de sistemas operacionais compactos [16, 42] melhores adaptados ao novo perfil de operação requerido por este novo tipo de rede.

O surgimento de novas tecnologias de componentes com baixíssimo consumo de potência, tensão de alimentação reduzida, integrando periféricos analógicos e digitais, provendo comunicação sem fio (infravermelho, rádio transceptor), memória, sensores, com tamanho reduzido e baixo custo, é um dos fatores relevantes que têm contribuindo para disseminação e desenvolvimento de dispositivos nós sensores.

## 1.1. Motivação e Objetivos

A implantação de uma de RSSF baseia-se na distribuição ou "pulverização" de inúmeros nós sensores de tamanho físico reduzido e com o propósito de realizar um determinado monitoramento. De uma maneira geral, os nós sensores adquirem as informações provenientes dos seus sensores e as transmitem no sentido de propagá-las através da rede. Esta atividade consome uma significativa quantidade de energia, principalmente, no caso da transmissão via enlace de rádio. Assim, é fundamental que estes dispositivos sejam projetados de forma a consumirem pouca energia, o que demanda a utilização de componentes eletrônicos específicos e de excelente performance de consumo de potência. Entretanto, muitos destes componentes atendem a este requisito com uma certa limitação computacional (memória reduzida, baixa velocidade de processamento, poucas interfaces, etc) e, neste caso,

a capacidade de processamento local que um determinado nó sensor é capaz de realizar fica comprometida.

Este trabalho tem como objetivo desenvolver um dispositivo nó sensor para RSSF, batizado como RANS-300 (*Reconfigurable Architecture for Network Sensor* – 300 *kgates*), com a finalidade de incorporar recursos reconfiguráveis de *hardware* a fim de aprimorar e expandir o conjunto de funcionalidades e de monitoramento atualmente desempenhados pelos nós sensores convencionais, possibilitar o processamento de eventos complexos que exijam maior eficiência computacional e precisão, e de minimizar o consumo de energia quando estes recursos não são requeridos pela aplicação. A arquitetura de hardware do RANS-300 foi desenvolvida a fim de:

- permitir soluções baseadas em síntese lógica de circuitos digitais, que podem ser configuradas localmente e/ou remotamente;
- possibilitar a execução de determinados algoritmos de tempo real em *hardware*;
- possibilitar a realização de cálculos matemáticos complexos;
- efetuar processamento diferenciado em situações de emergência (incêndio, inundação, etc), onde as informações provenientes de sensores (tratamento de imagem, filtragem de ruído, comunicação com sensores) necessitem ser tratadas localmente;
- controlar diversas unidades de monitoramento e múltiplos módulos de sensores (função *gateway* ou líder de *cluster*);
- oferecer alternativas de interfaces de comunicação com maiores taxas de transferência de dados (IEEE 802.11, Bluetooth);
- oferecer grande capacidade de armazenamento temporário e permanente de dados, uma vez que, neste sentido, os nós convencionais dispõem de poucos kbytes.

O *hardware* proposto para o RANS-300 considerou as premissas funcionais, físicas, elétricas e estratégicas (baixo consumo de potência, custo, tamanho reduzido) exigidas por esta classe de dispositivos.

## 1.2. Arquitetura do RANS-300

Podemos identificar como pontos fortes da arquitetura deste novo dispositivo o emprego de arquiteturas reconfiguráveis de *hardware,* a grande densidade de memória, a capacidade de adicionar interfaces e periféricos e ao seu controle modular de potência. Todas estas características tornam este dispositivo ideal para a implementação de soluções complexas, configuráveis e dinâmicas.

De uma maneira geral, os dispositivos nós sensores apresentam certas similaridades construtivas de *hardware* e também funcionais. Para efeito de análise e comparação, podemos considerar que estas partes comuns constituem uma espécie de núcleo básico responsável pela implementação de grande parte dos requisitos demandados atualmente pelas RSSFs. Uma característica observada neste tipo de núcleo e, conseqüentemente, na arquitetura destes nós sensores, é a pouca flexibilidade apresentada pelo seu *hardware*, uma vez que, são utilizados na sua implementação componentes com funcionalidades dedicadas. A Figura 1.2 mostra detalhes da arquitetura de *hardware* deste núcleo básico.

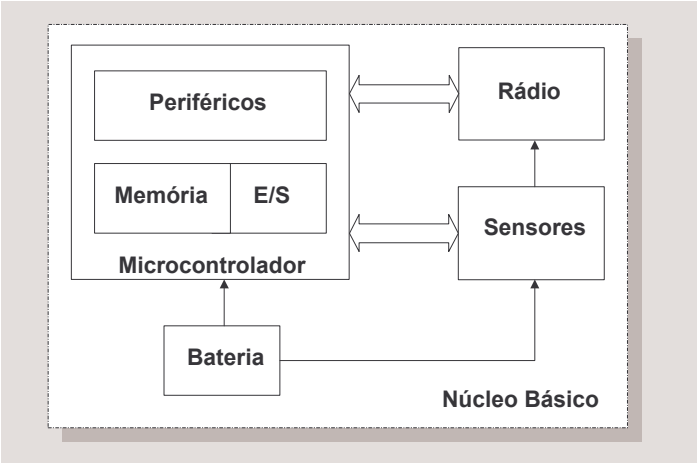



Figura 1.2: Arquitetura do Núcleo Básico.

Para oferecer maior flexibilidade, o dispositivo proposto incorpora à estrutura do núcleo básico um novo bloco de *hardware,* formando um conjunto modular com características reconfiguráveis (Figura 1.3).

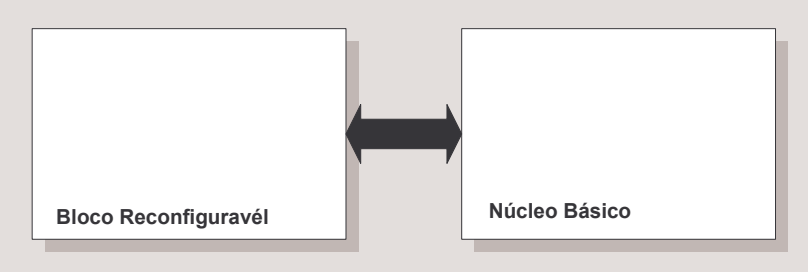



Figura 1.3: Estrutura da Arquitetura do RANS-300.

A característica modular está associada à capacidade do núcleo básico do RANS-300 de permitir a conexão e o controle de módulos auxiliares (externos e bloco reconfigurável) que podem conter sensores ou outros tipos de periféricos, e também relacionada ao fato dele permitir a alocação funcional e elétrica dos recursos computacionais, com a finalidade de proporcionar o gerenciamento dinâmico do consumo de energia. Em virtude desta capacidade de alocação é possível estabelecer que partes do circuito eletrônico deste dispositivo permaneçam no estado desenergizado sem comprometer o funcionamento das demais. Este requisito é essencial para a redução do consumo de potência e é aplicado sempre que o bloco reconfigurável do dispositivo não esteja sendo utilizado pela aplicação, proporcionando assim, um aumento da autonomia de operação.

A característica reconfigurável está associada à capacidade atribuída pelo seu *hardware* de permitir ser reprogramado localmente ou remotamente para atender a certos requisitos. Também demonstra, a capacidade do dispositivo em adequar as novas situações de operação, de possibilitar implementações complexas e soluções de problemas que requeiram maior recurso computacional, de possuir flexibilidade e maior reconfigurabilidade do *hardware*.

Na definição do modelo da arquitetura de *hardware* do RANS-300 foi avaliado o grau de flexibilidade pretendido considerando os aspectos de custo, tamanho e consumo. A solução de implementação focaliza na utilização de microcontrolador e componentes de memória de baixo consumo, rádio transceptor, conversores de energia e de dispositivos de lógica programável, *FPGA* [10], com capacidade para sintetizar vários módulos funcionais e computacionais utilizando-se das linguagens de descrição de *hardware* do tipo *Verilog* [36] e *VHDL* [53]. Através da síntese lógica, podemos implementar na FPGA do nó sensor processadores do tipo *RISC* (*ARM, MIPS, SPARC*) [45-47], CISC [48], microcontroladores,

diferentes periféricos e outros tipos de circuitos dedicados [34]. Os detalhes construtivos desta arquitetura serão apresentados e melhor detalhados no capítulo 4.

## 1.3. Metodologia de Desenvolvimento e Processo de Prototipação

O desenvolvido do *hardware* do RANS-300 foi orientado para o emprego preferencial de componentes eletrônicos de tecnologia SMD a fim de obter tamanho reduzido. Determinados componentes eletrônicos são ofertados pelos seus fabricantes em diversos tipos de encapsulamentos mecânicos, muitos deles com ganho expressivo de área e alta integração, mas exigem que sua montagem seja realizada através de máquinas automáticas. Toda esta tecnologia de montagem é, atualmente, restrita às empresas de fabricação de produtos eletrônicos, o que torna dispendioso o custo envolvido para a prototipação de poucas unidades. Estes aspectos orientaram a escolha dos componentes eletrônicos, do desenvolvimento da placa de circuito impresso (PCB) e definiu os processos de elaboração dos protótipos.

As atividades globais relacionadas ao desenvolvimento do RANS-300 são apresentadas na Figura 1.4.

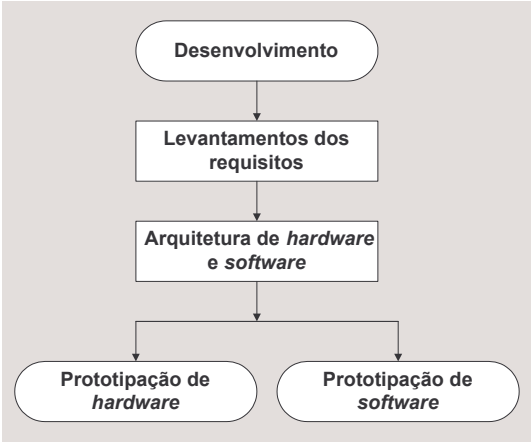



Figura 1.4: Fluxograma das atividades de desenvolvimento do RANS-300.

Na atividade de levantamento dos requisitos foram considerados os aspectos técnicos e funcionais desejados para a arquitetura do RANS-300, bem como as características disponíveis em catálogos de alguns modelos comerciais de nós sensores [15, 19].

A atividade de desenvolvimento da arquitetura de *hardware* pode ser caracterizada pela sistematização dos requisitos estabelecidos. Nesta fase, obteve-se a sedimentação dos conceitos e das propostas que orientaram o desenvolvimento do RANS-300. Assim, os aspectos previamente levantados na fase de requisitos foram organizados e representados na forma de diagramas de blocos, desenhos funcionais e diagramas de funções e eventos. O detalhamento deste estudo foi realizado nas fases de prototipação de *hardware* e *software*, as quais serão apresentados no capítulo 4.

As Figuras 1.5, 1.6 e 1.7 apresentam um roteiro prático que foi adotado durante o desenvolvimento da fase de prototipação do RANS-300, ilustrando as principais atividades de projeto e parte da logística (compra de componentes, fabricação, contratação de serviços, etc.) envolvida.

A prototipação é uma fase presente no desenvolvimento de sistemas que integram *hardware* e *software* e tem como finalidade a implementação e a verificação do modelo físico definido na fase de elaboração da arquitetura. Esta atividade requer a realização de testes básicos de desempenho, levantamento e correção de problemas inesperados, comprovação técnica dos requisitos, verificação da estabilidade funcional, estabelecimento dos limites de operação e a execução de medidas elétricas.

A simulação otimiza o desenvolvimento de sistemas de *hardware* e *software*, abreviando e atenuando o esforço de realizar algumas etapas de implementação e contribui para a redução de erros e riscos inerentes ao processo de desenvolvimento. A realização de simulações funcionais e paramétricas foi um recurso, sempre que possível, adotado durante o desenvolvimento do RANS-300.

A prototipação de *hardware* envolveu um conjunto específico de atividades que demandaram por recursos financeiros e laboratoriais, como a aquisição de componentes eletrônicos e partes, a contratação de serviços especializados de terceiros para fabricação de placa de circuito impresso, um conjunto de ferramentas para montagem e manutenção de componentes em SMD, equipamentos de medição e depuração lógica (osciloscópio, analisador lógico, multímetro), microcomputadores e outros.

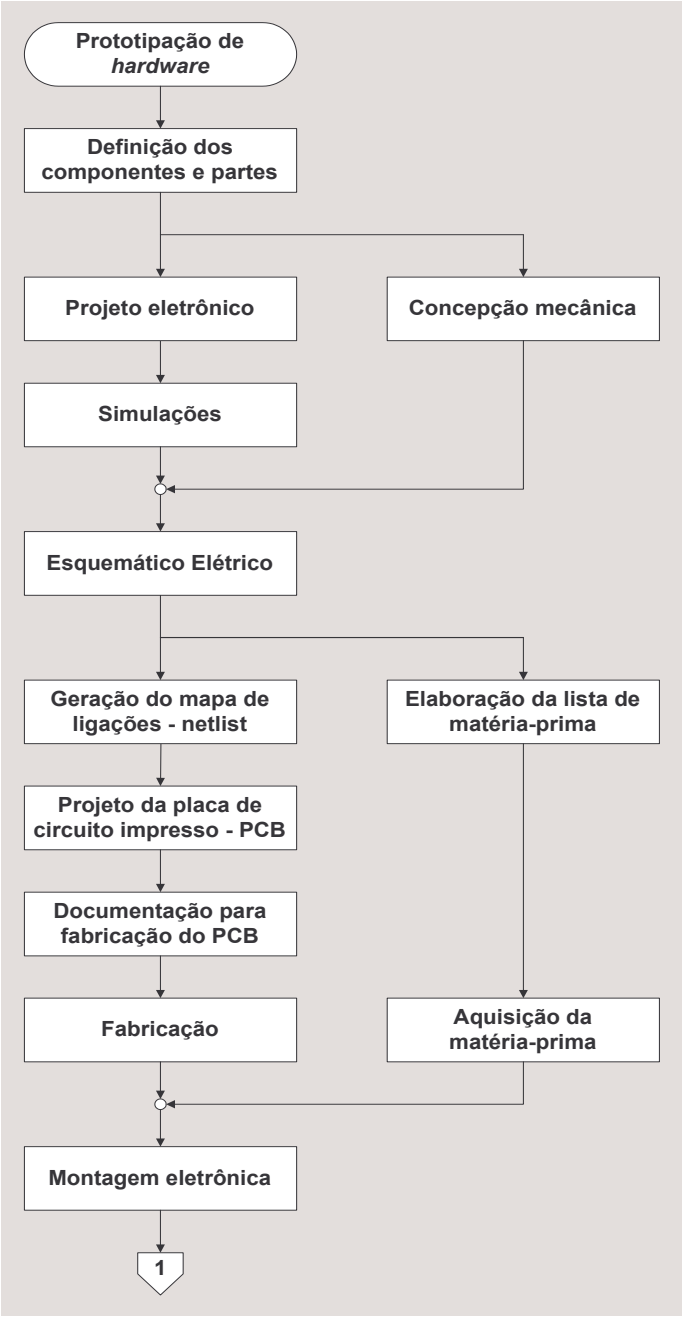



Figura 1.5: Fluxograma da fase de confecção do protótipo de *hardware*.

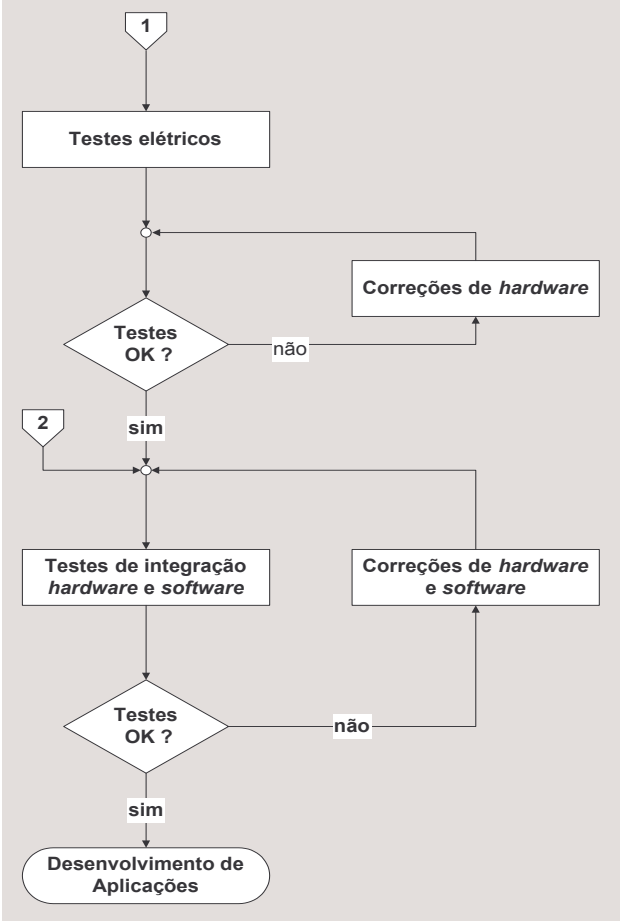



Figura 1.6: Fluxograma da fase de teste e integração *hardware/software*.

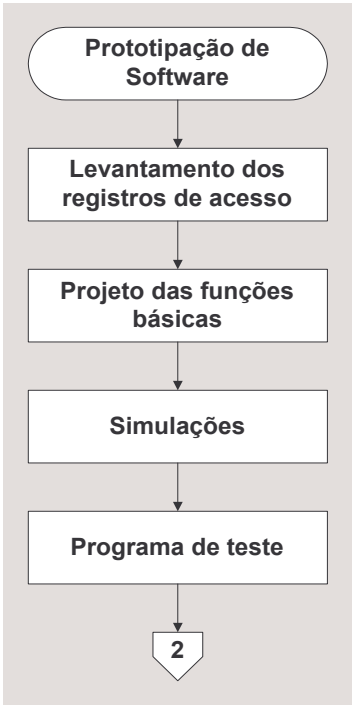



Figura 1.7: Fluxograma da fase de *software* básico de testes.

A prototipação de *software* teve como finalidade implementar um conjunto básico de funções de acesso ao *hardware* para auxiliar na elaboração de um programa de teste e para ser utilizado no desenvolvimento de aplicações futuras.

Na prática, a fase de prototipação encerra-se quando se obtém o refinamento técnico e funcional do dispositivo desenvolvido, podendo exigir em muitos casos, novas confecções de protótipos para as devidas adequações e correções.

## 1.4. Trabalhos Relacionados

Trabalhos relacionados com a implementação de nós sensores [43] e diversos modelos disponibilizados comercialmente [15,19,26-32], adotam uma arquitetura baseada em microcontroladores de baixo consumo, com tensão de operação reduzida (2,7 a 3,6V) e memória interna de dados e programa. Os estágios de rádios utilizados permitem efetuar comunicação bidirecional de dados (transceivers) e podem ser encontrados utilizando as tecnologias Bluetooth [17-18,20,23], IEEE 802.11[21,22] e recentemente o padrão IEEE 802.15.4 [49]. Estes dispositivos utilizam como suporte para processamento, controle e comunicação de dados alguns sistemas operacionais específicos e aplicados à RSSF, como por exemplo, os sistemas TinyOs [16] e YATOS [42]. Os aspectos construtivos e funcionais de alguns destes dispositivos serão abordados no capítulo 2, o qual descreve o atual cenário tecnológico e relaciona o dispositivo desenvolvido neste trabalho.

# CAPÍTULO 2

# Rede de Sensores

As RSSFs são um tipo de rede cujos nós integram um ou mais sensores, processador, memória, elementos que efetuam comunicação sem fio (rádio, laser, infravermelho, etc), bateria e em alguns casos atuadores.

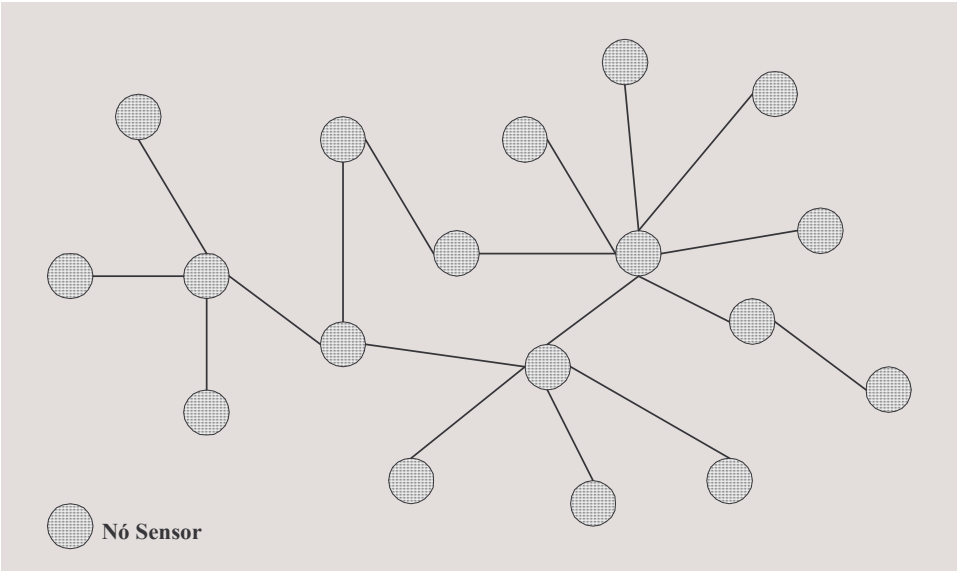



Figura 2.1: Topologia das Redes de Sensores Sem Fio.

De uma forma geral, as RSSFs possuem forte restrição de consumo de potência [2, 8, 9, 13]. Uma certa quantidade de energia é necessária para que os nós sensores desempenhem determinadas funções como, por exemplo, o tratamento e aquisição das informações dos sensores, processamento, armazenamento e comunicação. A ausência de "fios" para fornecer esta energia constante através da rede restringe a capacidade e a maneira de como os nós sensores são alimentados, levando a maioria destes dispositivos a adotarem elementos do tipo bateria como fonte principal de energia. Para o melhor aproveitamento deste recurso limitado e distribuído de energia, as RSSFs demandam por metodologias de controle e gerenciamento operacional mais eficientes que os encontrados nas redes convencionais, objetivando a melhor performance com o menor custo energético.

Numa infra-estrutura baseada em RSSF não se consideram procedimentos relacionados à manutenção de seus elementos, principalmente a operação de substituição da bateria. Uma vez esgotada a energia da bateria de um determinado nó sensor, este se torna definitivamente inativo e é descartado da rede. Assim, uma vez "lançada" ou posicionada em um sistema de interesse, a "missão", o modo de operação e a vida útil da RSSF é determinada em função do seu consumo de potência e da energia disponível.

As RSSFs empregam estratégias como o escalonamento, a coordenação e a cooperação de determinadas atividades pelos nós sensores e também diversas técnicas para conservação de energia como, por exemplo, o controle do estado operacional do processador do nó sensor [52]. Resultados expressivos são verificados na prática quando são adotados procedimentos de otimização da comunicação ao longo da rede, motivando pesquisas e esforços de se conceber protocolos mais adequados, uma vez que a potência resultante requerida pela transmissão das informações pela rede é dispendiosa e determinante para a performance das RSSFs.

A natureza dinâmica das RSSFs requer procedimentos e a aplicação de mecanismos que dizem respeito à auto-organização e a autoconfiguração da rede. Uma RSSF deve se ajustar às condições de perda e inserção de nós sensores, que podem ocorrer em situações de esgotamento da energia de suas baterias (definitivamente descartados), quando eles tornam-se inativos para economizar energia (estado "adormecido") ou tornam-se operacionais a partir deste estado, quando não é possível estabelecer comunicação e quando outros dispositivos nós sensores são adicionados à rede. Outros aspectos a serem considerados no desenvolvimento das RSSFs estão relacionados à localização, sincronização de tempo, disseminação e processamento de dados.

Nas redes cabeadas, as requisições e o envio de informações entre os nós e demais elementos que compõem a sua estrutura, são baseadas em endereços. Esta forma de se ter acesso à informação não é aplicável no caso das RSSFs, uma vez que, os dados adquiridos pelos sensores dos nós são difundidos através da rede, de acordo com o tipo de dado solicitado ou a ser propagado – os nós sensores podem, em um determinado instante, iniciar a propagação de um dado de temperatura, pressão, umidade, etc. Por serem orientadas a dados, as RSSFs:

- demandam por um gerenciamento eficiente do fluxo de dados;
- requerem diferentes formas de efetuar a propagação de dados;
- realizam atividades de agregação de dados através do compartilhamento das informações entre nós vizinhos;
- estabelecem restrições e definem regras de direcionamento da transmissão.

Outra diferença significativa com relação às redes cabeadas, é que os protocolos criados e disponíveis para estas estruturas são inadequados para serem adotados nas aplicações baseadas em RSSF, devido a sua característica de restrição de potência, de memória e de roteamento.

## 2.1. Agrupamento ou Clusterização

Uma vez que uma RSSF pode conter uma grande quantidade de nós sensores, estes nós podem ser organizados em grupos ou *clusters*, dependendo da localização física de cada nó. Na maioria das vezes, a localização de um nó particular não pode ser previamente determinada, em virtude da natureza dinâmica e da maneira aleatória e irregular com que os nós são distribuídos em um RSSF. Assim, para se formar um *cluster* são utilizados diversos algoritmos específicos e conjuntos de regras próprias de clusterização.

Esta estratégia permite uma otimização do consumo de energia de uma RSSF. Isto porque na formação de um *cluster* é definido sempre um elemento central denominado líder de *cluster* ou *cluster head*. Uma RSSF pode possuir múltiplos *cluters* e, conseqüentemente, vários líderes de *clusters* (Figura 2.2).

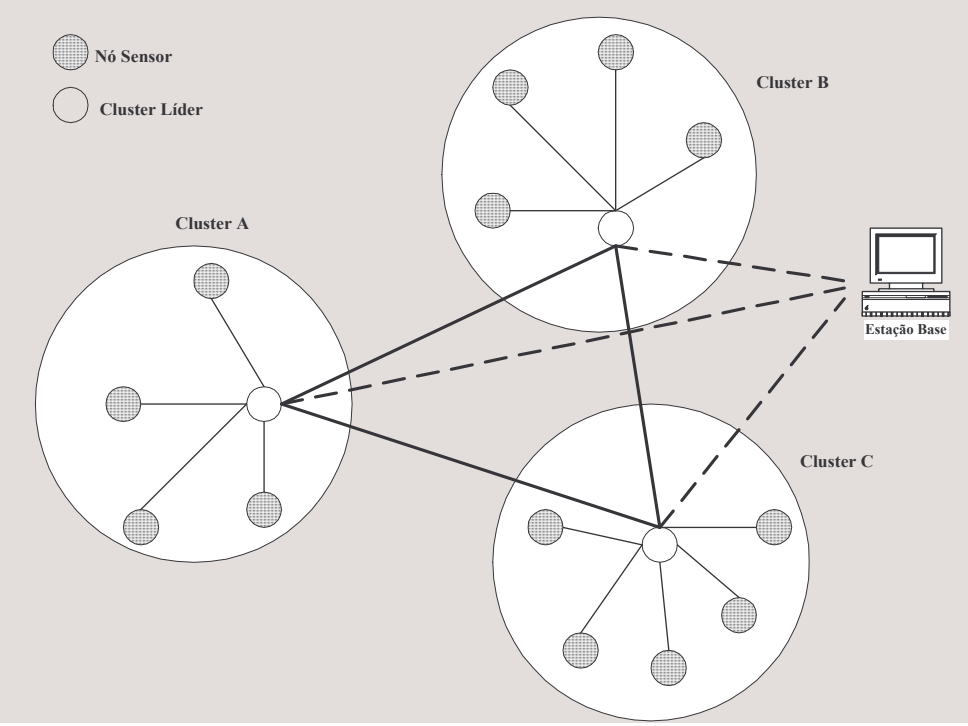



Figura 2.2: Clusterização de nós sensores.

O líder de *cluster* centraliza a comunicação e os dados transmitidos pelos nós membros, para então, enviá-los à estação base. Como normalmente à distância dos nós para o líder de *cluster* é pequena, a energia desprendida pela sua comunicação é menor, se comparada à situação na qual todos os nós tivessem que transmitir separadamente dados para a estação base.

É evidente ressaltar que, neste contexto, o recurso de energia do líder de *cluster* se esgota mais rapidamente em relação aos demais nós sensores. Alguns algoritmos de clusterização prevêem este comportamento e adotam procedimentos de escolha ou eleição de um novo líder, para assumir o papel do que foi descartado da rede.

## 2.2. Padrão IEEE 802.15.4/Zigbee

A "*Zigbee Alliance*" [40] é um consórcio de empresas (Sylvania, Zensys, AMI, Atmel, Microchip, Philips, Renesas, etc), criado com a finalidade de especificar e definir um conjunto de tecnologias aplicáveis na elaboração de um padrão de RSSF de baixo consumo,

para serem empregadas em diversos sistemas de monitoramento (alarmes de segurança, detectores de fumaça, sensores de presença, etc).

A grande expectativa é que a especificação ZigBee proporcione a unificação de padrões (existência atual de muitos padrões e de soluções proprietárias), a simplificação dos protocolos, o interfuncionamento, permita implementações globais e viabilize a produção de dispositivos sensores em grande escala com baixo custo (menor que três dólares).

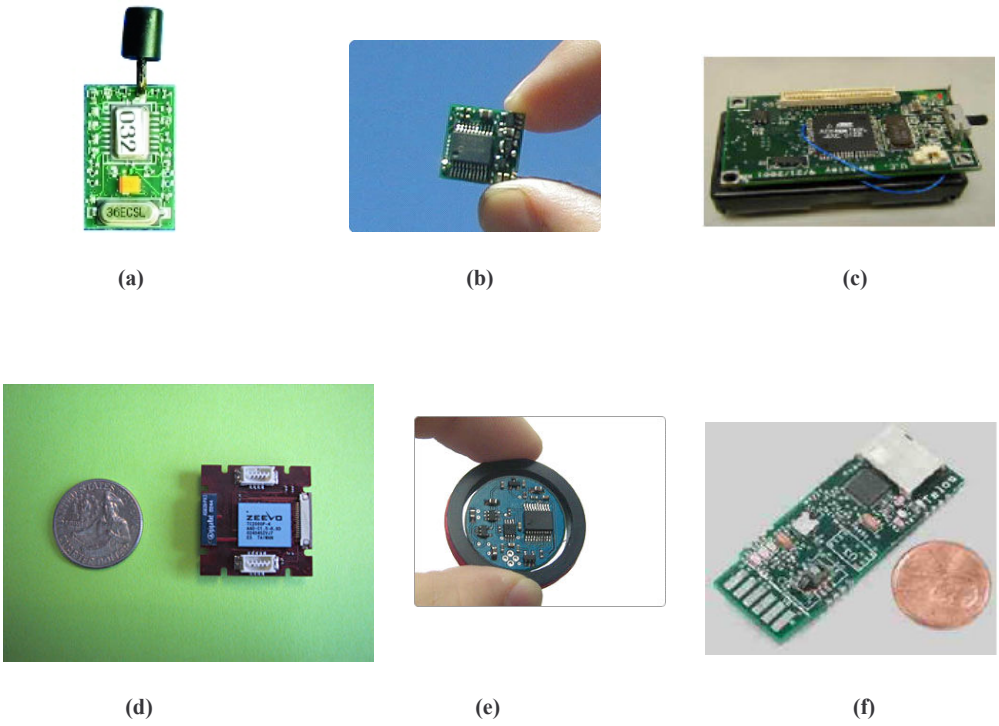
O padrão de rádio adotado pela Zigbee é o definido pela recomendação IEEE 802.15.4 [49]. Este padrão estabelece uma faixa de transferência de dados de 20 kbps a 250 kbps, dependendo da banda de freqüência adotada, especifica as camadas MAC (*Medium Access Control*) e PHY (*Physical Layer*) e adota a técnica de espalhamento espectral de freqüência DSSS (*Direct-Sequence Spread Spectrum*), o que reduz, significativamente, o consumo de potência envolvido na transmissão.

A razão para oferecer uma banda reduzida de transmissão, decorre do fato, de que, a maioria dos dispositivos nós sensores efetuam a transferência de poucos bytes, sendo, portanto, opostos em largura de banda, aos dispositivos e interfaces de rede sem fio do tipo Wi-Fi 802.11g (54Mbps) [21,22] e Bluetooth (1Mbps) [17-18, 20, 23], que são apropriados para atender às aplicações que requeiram transferência de grandes volumes de dados (áudio, correio, imagem, texto, etc) e em sistemas que suportam um custo maior de energia.

Uma outra característica da tecnologia Zigbee que proporciona economia de energia, é a redução dos requisitos de performance e de capacidade de memória exigidos do processador do dispositivo nó sensor. Isto porque, os protocolos empregados são bastante otimizados e a implementação das suas camadas ocupa poucos bytes de memória (<32kbytes).

## 2.3. Dispositivos Nós Sensores

Existem inúmeros projetos voltados para o desenvolvimento de dispositivos nós sensores [3, 5, 15, 19, 32, 63], com a finalidade de desenvolver plataformas de *hardware* e *software* e de gerar contribuições para a área de RSSF, que possibilitem a implementação de dispositivos eficientes, pequenos, auto-suficientes, de baixo custo e em grande escala.




(a) (b) (c)

(d) (e) (f)

Figura 2.3: Dispositivos nós sensores comerciais.
a) Millenial iB-5209
b) MicroStain Enbedsense RFID sense
c) Mica Mote
d) Intel Mote
e) MicroStain Enbedsense RFID sense
f) Telos

O projeto *Smart Dust* [4] desenvolvido na Universidade de Berkeley [3] é uma destas importantes iniciativas de pesquisa, com foco na implementação de RSSFs com dispositivos nós sensores extremamente miniaturizados, a fim de serem "pulverizados" em sistemas de interesse como se fossem verdadeiros grãos de poeira dotados de inteligente. Neste sentido, ao longo dos anos, o *Smart Dust* vem aprimorando a construção de modelos de dispositivos nós sensores, conhecidos como Motes, e de sistemas de *software*, como é o caso do sistema operacional TinyOS [16], desenvolvido para aplicações de RSSF e outros sistemas embutidos.

Os dispositivos Motes MICA2, MICAdot, MICAz, módulos de sensores e plataformas de desenvolvimento, são oferecidos comercialmente pela Crossbow [15], o que possibilita o desenvolvimento de algoritmos e aplicações baseadas em RSSF de forma rápida e exploratória.

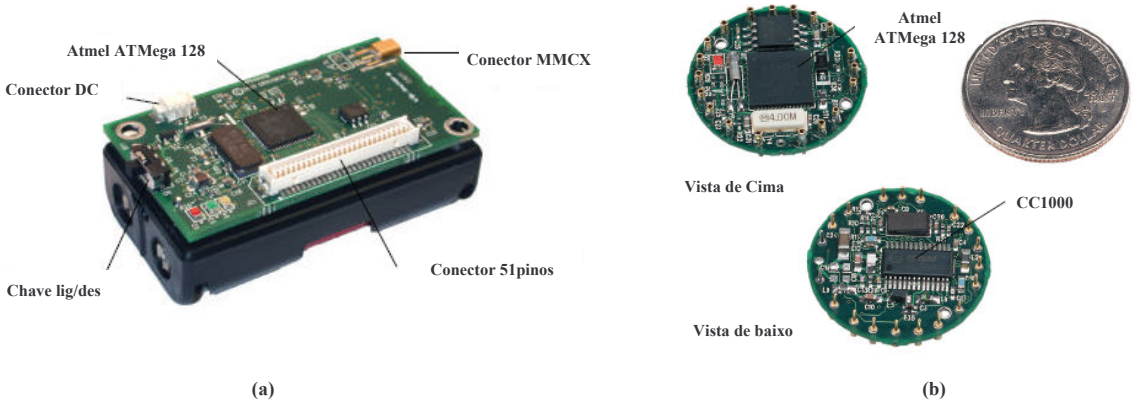



Figura 2.5: Dispositivos nós sensores comercializados pela Crossbow.
a) modelo MICA2 Mote
b) modelo MICAdot

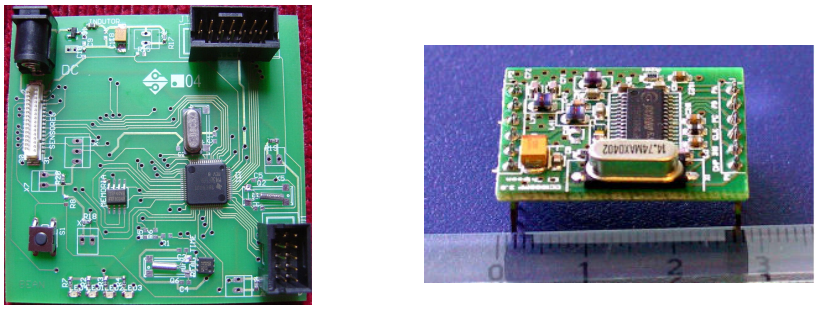
O MICA2 Mote, atualmente, é um dos dispositivos nós sensores que apresenta o maior número de unidades comercializadas. Sua arquitetura de *hardware* é formada por um microcontrolador ATmega 128L [81] (integrado memória, conversor AD, interface serial, SPI, I2C, timer e interfaces digitais de E/S), rádio transceptor CC1000 [44], bateria do tipo AA e conector de expansão de 51 pinos para adicionar módulos de sensores.

O SENSORNET [50] é um projeto de pesquisa coordenado pelo Departamento da Ciência da Computação (DCC) do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais (ICEX-UFMG), criado com o objetivo de elaborar pesquisas e gerar contribuições para a área de RSSF, tendo produzido diversos trabalhos relacionados, como é o caso do nó sensor BEAN [43] (*Brazilian Energy-Efficient Architectural Node*) e do sistema operacional YATOS [42] (*Yet Another Operating System*). O RANS-300 é mais uma iniciativa deste projeto voltado para a implementação de dispositivos nós sensores.

O desenvolvimento do BEAN surgiu para atender à necessidade do projeto SENSORNET de possuir um protótipo de nó sensor próprio, considerando que não existia no mercado nacional até aquele presente momento, uma plataforma computacional destinada ao desenvolvimento de aplicações de RSSF.

O BEAN possui uma arquitetura modular de *hardware* constituída por uma placa base de processamento baseada no microcontrolador MSP430F169 da Texas Instruments [51], um módulo agregado de comunicação CC1000PP (rádio transceptor integrado) da Chipcon [44] e um conector de expansão para permitir a incorporação de sensores. As aplicações de RSSF

implementadas no BEAN utilizam o suporte do sistema operacional YATOS, simultaneamente, desenvolvido para este dispositivo.



(a) (b)

Figura 2.4: Detalhes da implementação do nó sensor BEAN.
a) placa base baseada no MSP430F169
b) módulo de rádio CC1000PP agregado

Os componentes usualmente encontrados na arquitetura da maioria dos nós sensores são: microcontrolador, rádio, conector de expansão e bateria. Este conjunto de componentes eletrônicos forma o bloco núcleo básico de um nó sensor, como definido no Capítulo 1. Avaliando diversas implementações de dispositivos nós sensores, podemos observar que, cada projeto adota uma combinação própria destes componentes na tentativa de obter maior eficiência operacional e melhor relação custo/benefício. Do ponto de vista da arquitetura de *hardware*, poucas modificações são empregadas em relação ao modelo conceitual do núcleo básico. O RANS-300 se diferencia destes dispositivos (BEAN, MICA2 Motes, EYES, Telos, MicroStain e outros) por acrescentar um novo bloco de *hardware* (componentes reconfiguráveis e recursos de memória) ao núcleo básico.

Alguns processadores adotados em projetos de nós sensores dispõem de mecanismos que permitem reduzir expressivamente o seu consumo de potência, como por exemplo, a disponibilidade de registros internos de configuração que controlam o estado dos sinais de *clock* dos seus periféricos e definem modos de funcionamento da sua unidade de processamento (*core)*. A ativação ou redução da freqüência destes sinais de *clocks* representa relação direta com o consumo de potencia destes componentes eletrônicos.

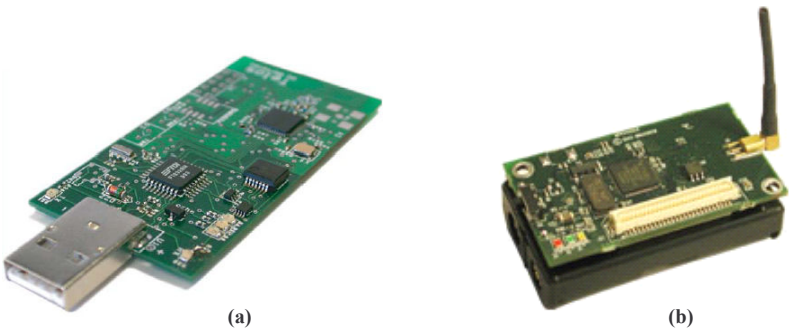
Para implementar a interface de comunicação de dados sem fio, muitos modelos de dispositivos nós sensores utilizam módulos comerciais de rádios, que são bastante integrados (poucos componentes externos requeridos) e robustos, destacando os modelos TR1000 [65] e

o CC1000 [44]. O rádio TR1000 pode operar com as modulações de RF OOK (*On-Off Keyed*) e ASK (*Amplitude-Shift Keyed*) [67] que suporta taxa de transmissão de dados de até 115.2 kbps. O consumo nesta taxa é da ordem de 15mW na recepção, 36mW na transmissão e de 15uW no estado adormecido (*sleep mode*). O rádio CC1000 opera com modulação FSK (*Frequency Shift Key*) na banda ISM (*Industry Science Medical*) e SRD (*Short Range Devices*) nas freqüências de 315, 433, 868 e 915 MHz. O consumo no modo recepção na banda de freqüência de 488/868 é da ordem de 22/28 mW, 220µW/288µW no modo recepção com *polling* e de 600nW quando o oscilador interno é desligado.


(a) (b) (c)

Figura 2.6: Modelos de rádio utilizados em RSSFs.
a) modelo TR1000 da RFM
b) modelo CC1000 da Chipcon
c) modelo CC2420 da Chipcon

Como mencionado na Seção 2.2, a Zigbee adota o padrão 802.15.4 de rádio especificado pelo IEEE [39]. Alguns fabricantes de semicondutores já disponibilizam modelos de rádio baseados neste padrão, como é o caso da Chipcon (Figura 2.5 (c)). O rádio CC2420 [44] oferece 16 canais de comunicação e opera na freqüência de na banda ISM na freqüência de 2.4GHz com 5MHz de separação entre canais. A taxa de transmissão oferecida é de 250kbps empregando a modulação O-QPSK [70]. O consumo em operação é de 30mW e de 1µW no modo adormecido. A tensão de alimentação é de 1.8V (1.6 a 2V) com o regulador interno desabilitado ou de 3.3V (2 a 3.6V) no modo habilitado. A tendência é que novos dispositivos sejam construídos adotando a especificação Zigbee, como é o caso surgimento recente dos modelos Telos [69] e MICAz [68] da Crossbow (Figura 2.6).



(a) (b)

Figura 2.7: Nós sensores utilizando rádios IEEE 802.15.4/Zigbee.
a) Telos
b) MICAz

| Nós Sensores | Microcontrolador | Rádio | Recursos de Memória | | | Sistema Operacional |
|---|---|---|---|---|---|---|
| | | | Flash | RAM | EEPROM | |
| MICA Mote | Atmega 103L | TR1000 | 128kB | 4kB | 4kB | TinyOS |
| MICA2 Mote | Atmega 128L | CC1000 | 128kB | 4kB | 4kB | TinyOS |
| MICAz Mote | Atmega 128L | CC2420 | 128kB | 4kB | 4kB | TinyOS |
| EYES | MSP430F149 | TR1001 | 60kB | 2kB | 1MB | PeerOS |
| Telos | MSP430F149 | CC2420 | 60kB<br>512kB | 2kB | --- | TinyOS |
| BEAN | MSP430F169 | CC1000PP | 60kB | 2kB | | YATOS |
| RANS-300 | MSP430F149 | CC1000 | 60kB<br>8MB(1)<br>4GB (2) | 2kB (1)<br>2MB (2) | 256B | YATOS |

**Nota 1**: Recursos de memória Flash e RAM implementados no bloco reconfigurável do RANS-300.
**Nota 2:** Recurso máximo de memória permitido pela unidade CompactFlash do bloco reconfigurável do RANS-300.

Tabela 2.1: Detalhes construtivos de alguns nós sensores.

A Tabela acima apresenta algumas características construtivas e os recursos de memória disponíveis na implementação do RANS-300 e dos nós sensores abordados neste capitulo.

# CAPÍTULO 3

# Descrição dos Componentes de *Hardware*

## 3.1. XC2S300E – FPGA da Família Spartan-IIE 1.8V

A Spartan-IIE [10] é uma família de FPGA *(Field Programmable Gate Array)* desenvolvida pelo fabricante Xilinx para permitir aplicações de alto desempenho, oferecendo recursos lógicos e um conjunto valioso de características com baixo custo. A densidade oferecida pelos componentes que compõem esta família varia de 50.000 a 600.000 portas lógicas e a freqüência de operação do sinal de *clock* interno pode atingir até 200MHz. Especificamente, o componente XC2S300E contém 300.000 portas lógicas.

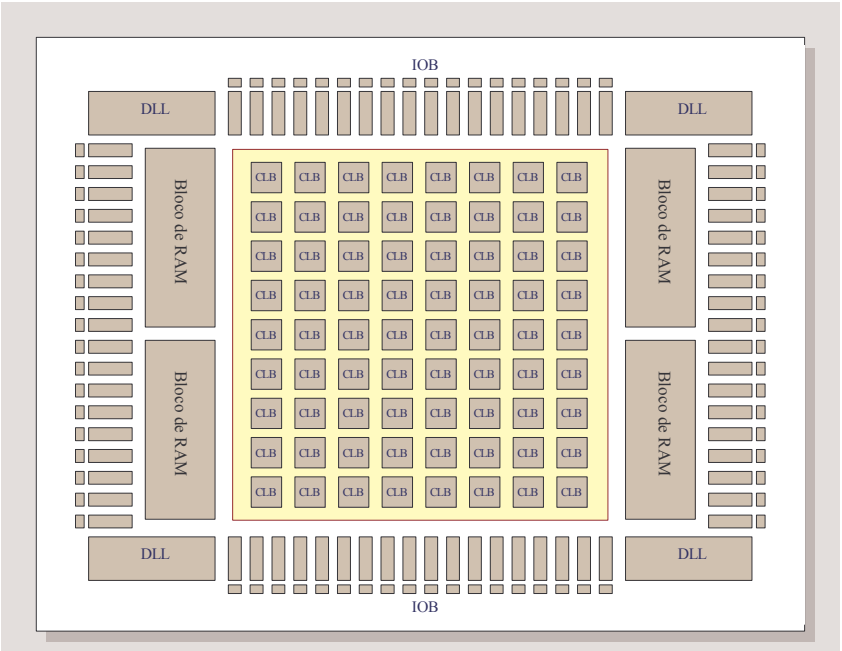



Figura 3.1: Estrutura construtiva da FPGA Spartan-IIE.

As FPGAs da família Spartan-IIE possuem uma arquitetura bastante flexível e composta por cinco principais blocos configuráveis:

- IOBs: estes blocos são responsáveis pela conexão dos pinos do encapsulamento aos blocos de lógica interna;
- CLBs: estes blocos incorporam os elementos funcionais destinados à implementação de funções lógicas;
- RAM: blocos dedicados de memória contendo cada um deles 4096 bits;
- DLLs: estes blocos são utilizados para realizar compensação interna de atraso do sinal de *clock* e sua geração interna em diferentes taxas;
- Estrutura versátil de interconexão com vários níveis de roteamento.

Adicionalmente, as FPGAs incorporam milhares de células de memória estática nas quais são carregados valores que controlam toda a configuração dos elementos lógicos presentes nos blocos descritos anteriormente e determinam de que forma as interconexões internas destes serão efetuadas. Estes valores devem ser carregados após a energização do componente e podem ser recarregados a qualquer momento a fim de modificar a sua funcionalidade.

**Blocos de entrada e saída**

Os IOBs são blocos que implementam as interfaces de entradas e saídas da FPGA Spartan II-E e podem ser configurados para atender a uma variedade de padrões elétricos. A Tabela 3.1 apresenta um resumo dos padrões suportados pela Spartan II-E, identificando a aplicação típica e, em alguns casos, o fabricante idealizador. Os detalhes mencionados de cada padrão podem ser encontrados nas especificações do JEDEC/EIA [72].

Alguns destes padrões necessitam que uma determinada tensão de referência seja aplicada em alguns terminais da FPGA e provisionada uma terminação externa (Vtt). A Tabela 3.2 mostra os padrões das interfaces elétricas que podem ser implementados a partir da configuração dos blocos IOBs.

| *Padrão Elétrico* | *Descrição* | *Espec.* | *Utilização e Idealizador* | *Eentrada* | *Saída* |
|---|---|---|---|---|---|
| LVTTL | Low-Voltage TTL | JESD8B | Uso genérico | LVTLL | Push-Pull |
| LVCMOS2 | Low-Voltage CMOS 2.5V | JESD8B | Uso genérico 2.5V | CMOS | Push-Pull |
| PCI | Peripheral Component Interconnect | PCA SIG | barramento PCI 33MHz-5V | LVTTL | Push-Pull |
| GTL | Gunning Transceiver logic | JESD8-3 | Barramento de alta velocidade; painel traseiro; Xerox | Amplificador diferencial | Open Drain |
| GTL+ | Gunning Transceiver logic Plus | JESD8-3 | Intel Pentium Pro | Amplificador diferencial | Open Drain |
| HSTL | High Speed Transceiver Logic | JESD8-6 | Hitachi SRAM; IBM | Amplificador diferencial | Push-Pull |
| SSTL3 | Stub Series Terminated Logic 3.3V | JESD8-8 | Barramentos de SRAM/ SDRAM; Hitachi e IBM | Amplificador diferencial | Push-Pull |
| SSTL2 | SSTL 2.5V | JESD8-9 | Barramento de memória; Fujitsu | Amplificador diferencial | Push-Pull |
| CTT | Center Tap Terminated | JESD8-4 | Barramento de memória; Fujitsu | Amplificador diferencial | Push-Pull |
| AGP | Advanced Graphics Port | AGP forum | Intel Pentium II, SRAM | Amplificador diferencial | Push-Pull |

Tabela 3.1: Padrões elétricos suportados pela FPGA Spartan-IIE.

| *Padrão da interface de Entrada/Saída* | *Tensão de referência de entrada* | *Tensão de entrada* | *Tensão de saída* | *Tensão da terminação de placa* |
|---|---|---|---|---|
| | (Vref.) | (Vcco) | (Vcco) | (Vtt) |
| LVTTL (2-24ma) | N/A | 3.3 | 3.3 | N/A |
| LVCMOS2 | N/A | 2.5 | 2.5 | N/A |
| LVCMOS18 | N/A | 1.8 | 1.8 | N/A |
| PCI (3V,33MHz/66MHz) | N/A | 3.3 | 3.3 | N/A |
| GTL | 0.8 | N/A | N/A | 1.2 |
| GTL+ | 1.0 | N/A | N/A | 1.5 |
| HSTL Class I | 0.75 | N/A | 1.5 | 0.75 |
| HSTL Class III | 0.9 | N/A | 1.5 | 1.5 |
| HSTL Class IV | 0.9 | N/A | 1.5 | 1.5 |
| SSTL3 Class I e II | 1.5 | N/A | 3.3 | 1.5 |
| SSTL2 Class I e II | 1.25 | N/A | 2.5 | 1.25 |
| CTT | 1.5 | N/A | 3.3 | 1.5 |
| AGP | 1.32 | N/A | 3.3 | N/A |
| LVDS | N/A | N/A | 2.5 | N/A |
| LVPECL | N/A | N/A | 3.3 | N/A |

Tabela 3.2: Descrição dos padrões elétricos suportados pela FPGA Spartan-IIE.

Os detalhes construtivos dos blocos IOBs são mostrados na Figura 3.2.

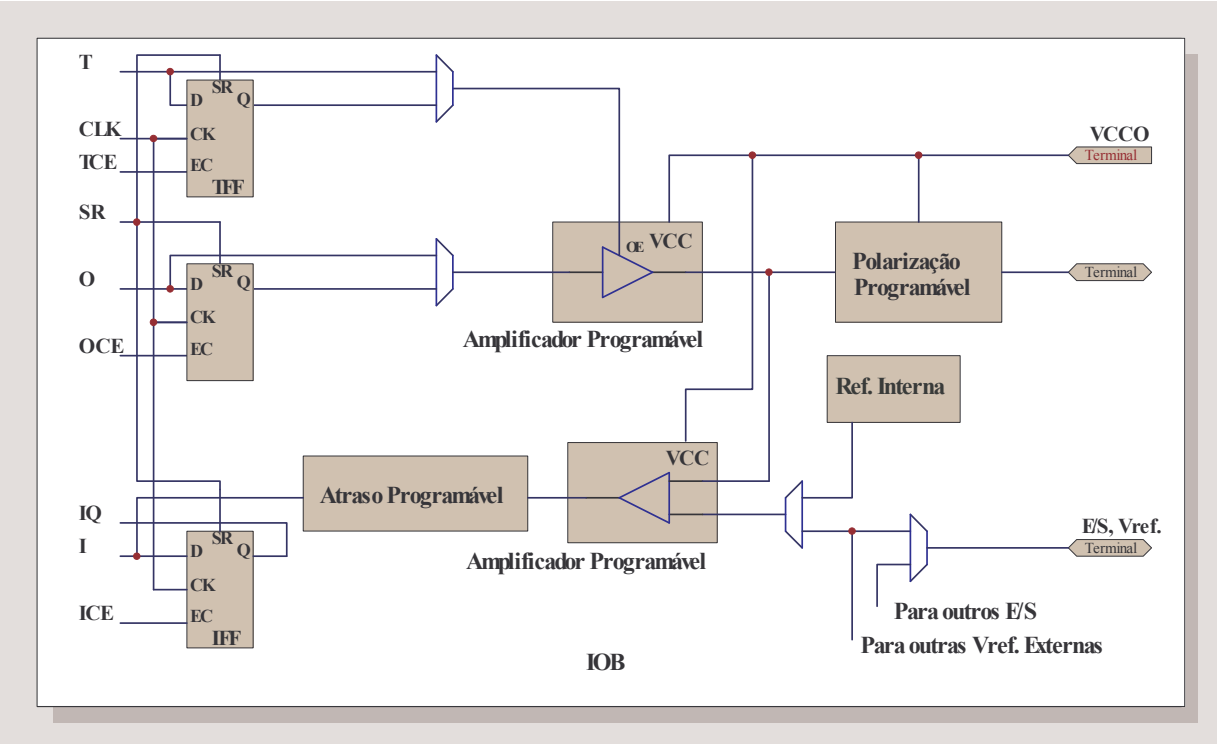



Figura 3.2: Detalhe do bloco IOB da FPGA Spartan-IIE.

Os três registros TFF, OFF e IFF do bloco IOB (Figura 3.2) podem operar como *flip-flop* do tipo D (sensíveis à borda do pulso de *clock*) ou como registros do tipo *latch* (sensíveis por nível lógico). Cada bloco IOB possui uma entrada de sinal de *clock* (CLK) e três entradas independentes (TCE, OCE e ICE) que permitem controlar a sua habilitação em cada registro. Adicionalmente, os três registros compartilham o mesmo sinal de *SR* (*Set/Reset*) que pode ser configurado como um sinal síncrono de *Reset*, um síncrono *Set*, um assíncrono *Preset* ou um sinal assíncrono de *Clear*.

Uma característica não mostrada na Figura 3.2 é que os amplificadores de saída e entrada e todos os sinais de controle do IOB possuem controles independentes de polaridade configurados por *software*.

O amplificador de entrada pode ser roteado diretamente para o bloco de lógica interna ou passar através de um *flip-flop*. Na entrada deste *flip-flop* pode ser, opcionalmente, empregado um elemento de compensação de tempo a fim de eliminar possíveis atrasos que possam ocorrer devido à propagação dos sinais entre os terminais do componente. Este elemento utiliza o sinal interno de *clock* como fonte para realizar esta compensação.

O amplificador de saída conduz o sinal proveniente da lógica interna para o terminal externo do componente. O sinal pode ser roteado diretamente ou passar através de um *flip-flop* com controle síncrono de ativação e desativação. Este controle permite que a saída deste amplificador opere com três estados lógicos sendo: nível zero, nível um e alta impedância.

Adicionalmente, é permitido configurar o IOB para conectar resistores internos de *pull-up* e *pull-down* em cada terminal do componente para definir os níveis lógicos de entrada e saída, quando estes terminais estão desconectados ou em alta impedância.

Como podem ser notados na Tabela 3.2, alguns destes padrões de interfaces de entrada e saída requerem tanto a tensão Vcco como a de Vref para o funcionamento.

**Bloco de lógica configurável**

Os blocos CLBs da Spartan II-E são formados por elementos construtivos básicos denominados células lógicas (LC). Uma LC é constituída por um gerador de funções lógicas de quatro entradas e implementado a partir de uma *look-up table* (LUT), por uma lógica de *carry* e por elementos de armazenamento baseados em *Flip-Flops* do tipo D. A utilização de LUTs torna o atraso de propagação dos sinais independente do tipo de função booleana implementada. Adicionalmente, cada LUT pode operar como unidade de memória RAM síncrona com capacidade de 16x1 bit. A combinação de duas LCs forma um *slice* e as suas LUTs combinadas podem oferecer uma capacidade de memória RAM síncrona de 16x2 bit, 32x1 bit ou 16x1 bit com dupla interface de acesso. Cada CLB contém quatro LCs ou dois *slices*. A Figura 3.3 mostra como é formado um *slice*.

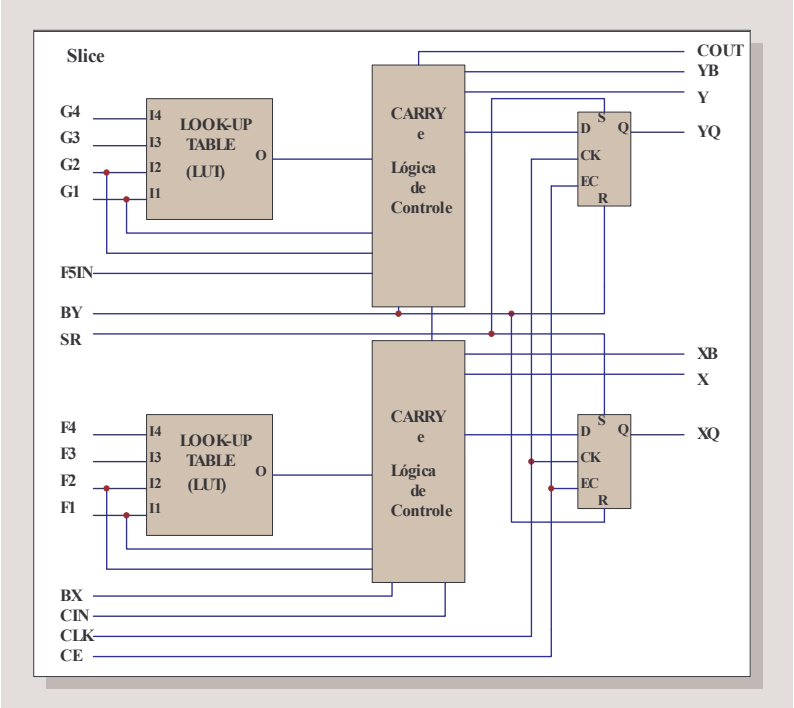



Figura 3.3: Detalhe construtivo de um Slice da FPGA Spartan-IIE.

Para formar um CLB são empregados alguns circuitos multiplexadores para combinar as saídas das LUTs de um determinado *slice* e outros para combinarem as saídas dos dois *slices*. A Figura 3.4 mostra o emprego destes multiplexadores no bloco CLB.

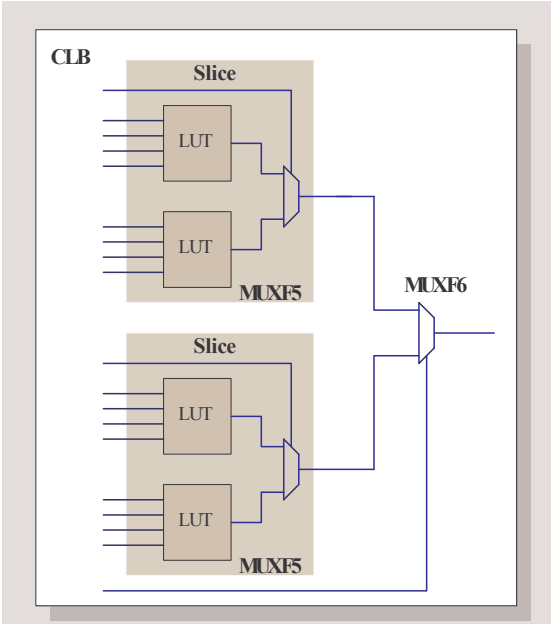



Figura 3.4: Detalhe construtivo de um CLB da FPGA Spartan-IIE

**Bloco de RAM**

As FPGAs Spartan-IIE incorporam diversos blocos dedicados de memória RAM síncrona, oferecendo uma capacidade complementar para a estrutura de RAM distribuída e oferecida pelas LUTs nos CLBs. Cada bloco de RAM oferece 4096 bits de capacidade organizados na forma de um porto de acesso duplo com controle individual para cada porto. Os dados de cada bloco de RAM podem ser configurados para atender a determinados comprimentos de dados requeridos pelos barramentos formados internamente.

**Bloco DLL – *Delay Locked Loop* e a Distribuição de *Clock***

A distribuição de *clock* é feita através de roteamento próprio e constituí um recurso global fornecido pelo componente para proporcionar a utilização de altas taxas com reduzido atraso de propagação e dos tempos de inclinação da rampa (*skew*) de subida e descida do pulso de *clock*.

As FPGA's Spartan-IIE disponibilizam quatro entradas amplificadas para sinais globais de *clock*. Estas entradas dedicadas suprem um barramento interno que efetua a distribuição deste recurso para toda a lógica interna.

Um bloco digital denominado DLL (*Delay Locked Loop*) é associado a cada entrada de *clock* global a fim de eliminar possíveis inclinações da rampa de subida ou descida dos sinais de *clock* gerados e fornecidos externamente a estes terminais, condicionando-os, para então, serem aplicados à lógica interna da FPGA.

Adicionalmente, os DLLs podem eliminar o atraso de propagação dos sinais de *clock* através da sua rede interna de distribuição e fornecer um avançado controle de múltiplos domínios que permite dobrar a freqüência do *clock* ou dividi-lo pelos fatores de 1.5, 2, 2.5, 3, 4, 5, 8 ou 16.

**Requisitos de Potência**

As FPGAs Spartan-IIE definem alguns requisitos que devem ser atendidos pela fonte de potência que supre a tensão de alimentação Vccint para os blocos de lógica interna do componente.

Ao aplicar um potencial elétrico aos terminais Vccint da FPGA, ocorre um surto de corrente durante um curto período de tempo. Este efeito é conhecido como POS (*Power-On Surge)* e a quantidade mínima de corrente que flui pelos terminais Vccint é especificada no catálogo da FPGA como Iccpo (*total supply current required during power-on*). A natureza desta corrente de surto é atribuída ao fato de que todas as conexões internas da FPGA são definidas por células de memória RAM, que durante a fase de energização encontram-se em um estado aleatório, provocando múltiplas contenções internas. Adicionalmente, os amplificadores internos também drenam uma quantidade de corrente durante esta fase de energização do componente.

Para que a FPGA da família Spartan-IIE seja energizada corretamente é preciso garantir que:

- A fonte de alimentação forneça a corrente mínima de 500mA para Iccpo;
- O tempo da rampa de subida da tensão Vccint seja inferior a 50ms. A corrente de surto pode ser limitada a um determinado valor desde que não viole esta condição (Figura 3.5);
- A tensão Vccint aplicada em relação ao terminal de referência GND atinja o valor limite de 1.8V sem apresentar picos negativos em relação à referência.

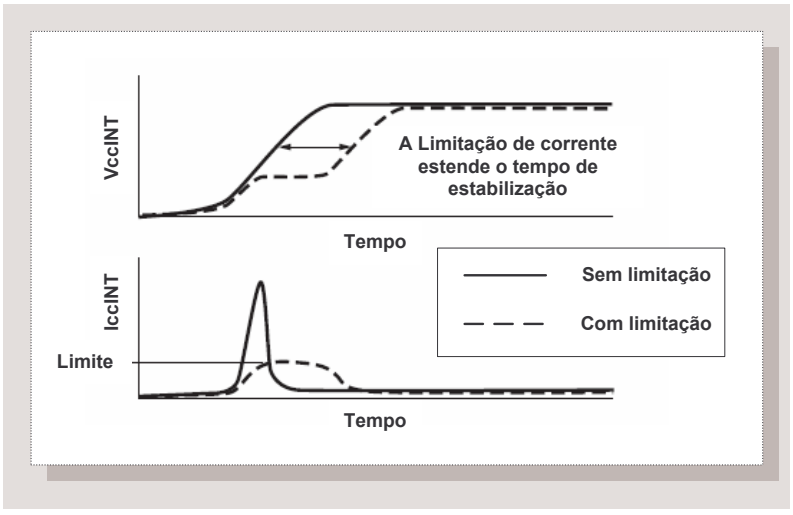


Figura 3.5: Detalhe da Rampa de alimentação da FPGA Spartan-IIE

**Métodos de Configuração da FPGA Spartan-IIE**

A configuração é o processo pelo qual um vetor de bits denominado *bitstream,* gerado previamente em um ambiente de desenvolvimento de aplicações de FPGA, é carregado para a memória interna do componente. Cada bit do vetor de configuração define o estado de uma célula de memória estática que controla um bloco interno da FPGA (LUTs, Flip-Flop, multiplexadores, DLL, etc) ou uma interconexão.

As FPGAs Spartan-IIE suportam que seja efetuada a carga do vetor de configuração através do modo serial mestre ou escravo, da interface JTAG [73] e do barramento paralelo do tipo escravo.

As FPGAs são configuradas seqüencialmente por blocos de dados que foram previamente concatenados em um arquivo de configuração. Estes arquivos apresentam tamanhos diferenciados de acordo com o tipo de componente utilizado da família Spartan-IIE. Estes tamanhos devem ser considerados a fim de definir o espaço de memória não-volátil necessário para o seu armazenamento.

As FPGAs Spartan-IIE destinam os terminais M0, M1 e M2 para que se efetue a programação do modo de carga do vetor de configuração, como mostra a Tabela 3.3.

| Modo de configuração | M0 | M1 | M2 | Sentido do clock | Largura do Barramento | DOUT Serial |
|---|---|---|---|---|---|---|
| Serial Mestre | 0<br>0 | 0<br>0 | 0<br>1 | saída | 1 | sim |
| Paralelo Escravo | 0<br>0 | 1<br>1 | 0<br>1 | entrada | 8 | não |
| JTAG | 1<br>1 | 0<br>0 | 0<br>1 | não aplicável | 1 | não |
| Serial Escravo | 1<br>1 | 1<br>1 | 0<br>1 | entrada | 1 | sim |

Tabela 3.3: Modos de carga de bitstream suportados pela FPGA Spartan-IIE

Um dos métodos de realizar a configuração da FPGA Spartan-IIE é utilizar uma memória PROM, previamente gravada com os dados do arquivo de configuração, conectada

externamente aos seus terminais dedicados. Este método apresenta uma forma rígida para armazenar o arquivo de configuração e que obriga à substituição da PROM, toda vez que, é necessário efetuar alguma alteração de funcionamento do componente. Este método é empregado em sistemas locais que não necessitam ser constantemente reconfigurados.

Existem alguns métodos que permitem controlar remotamente a carga de configuração tornando o sistema extremamente ágil e flexível. Estes métodos utilizam uma estratégica de programação conhecida como ISP [74] para possibilitar que a configuração seja efetuada pelo próprio circuito que integra a FPGA. A Figura 3.6 mostra um dos métodos possíveis de programação ISP.

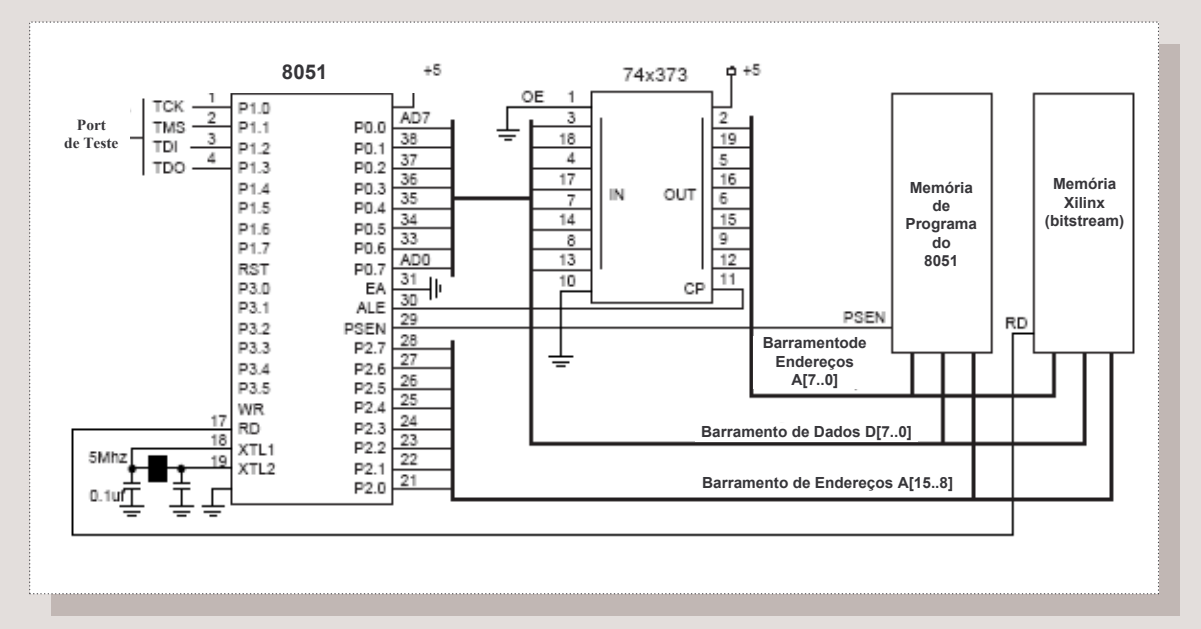



Figura 3.6: Circuito com implementação da configuração ISP (*in-system programming)*.

**Processo de Configuração**

A seqüência de eventos necessários para configurar uma FPGA Spartan-IIE é apresentada na Figura 3.7. Esta seqüência pode ser dividida em: fase inicial da configuração, apagamento da memória interna de configuração, carregamento do vetor de configuração e ativação da FPGA.

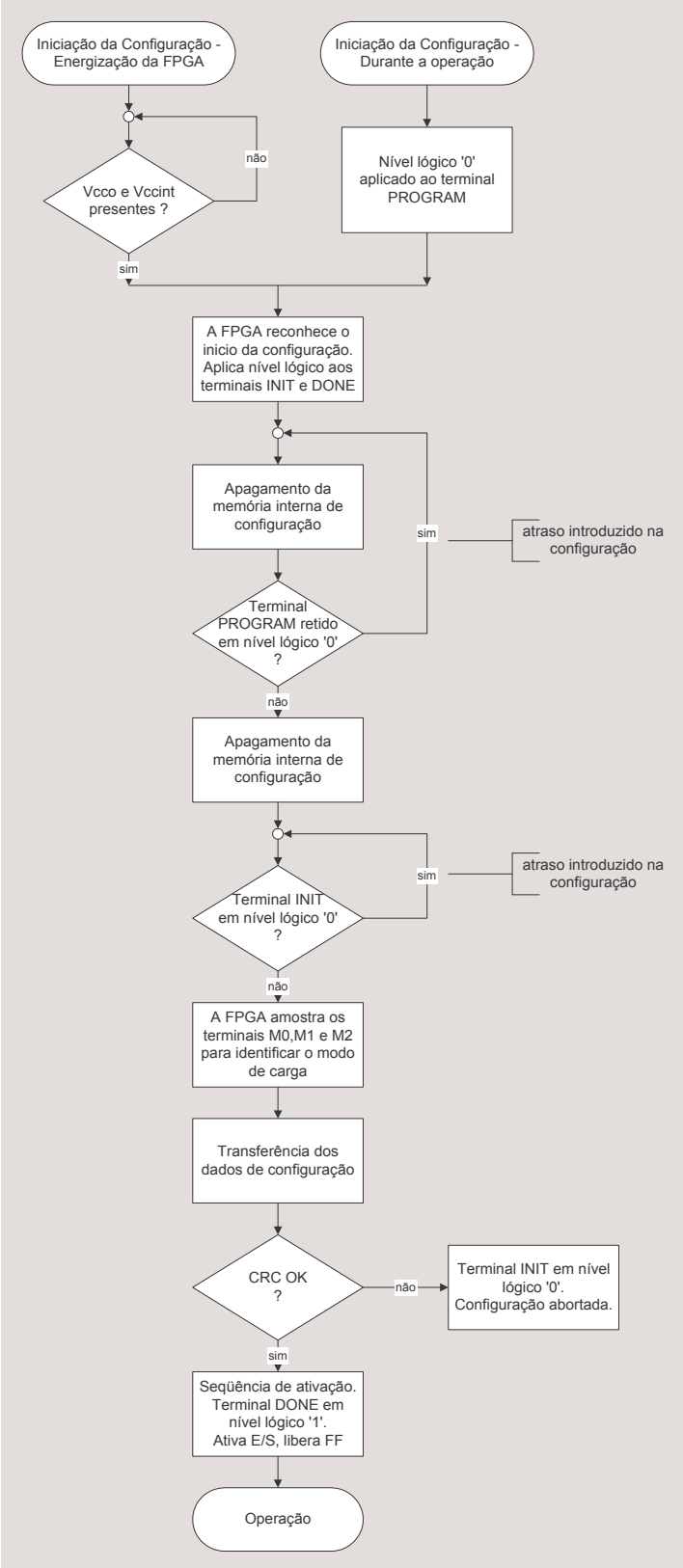



Figura 3.7: Fluxograma do processo de configuração da Spartan-IIE.

A fase inicial da configuração da FPGA é realizada durante a sua energização ou durante o seu funcionamento quando for necessário reconfigurar a sua lógica interna. Utilizando o terminal PROGRAM da FPGA, pode-se controlar o momento adequado para iniciar a sua configuração. A FPGA responde ao comando PROGRAM aplicando um nível lógico '0' ao terminal DONE, sinalizando que a fase de configuração foi iniciada e que em seguida será realizado o apagamento da sua memória interna de configuração. A FPGA sinaliza o inicio desta fase de apagamento aplicando um nível lógico '0' ao terminal INIT e em seguida o nível lógico '1', indicando o término do apagamento. Neste ponto, os dados contidos no arquivo de configuração devem ser transferidos para a sua memória interna utilizando um dos modos de carga.

O arquivo de configuração possui um valor inserido destinado à verificação de erro de CRC que é comparado ao valor do CRC calculado pela FPGA. Se os valores são diferentes é aplicado o nível lógico '0' ao terminal INIT indicando erro no processo de carga. Neste caso, um novo procedimento de carga deve ser iniciado acionando-se novamente o terminal PROGRAM. Se os valores dos CRCs são iguais, a FPGA entra na fase de ativação. Esta fase é caracterizada pela aplicação do nível lógico ao terminal DONE, pela ativação de todos os terminais de entrada e saída e pela liberação da operação dos Flip-Flops e das RAMs internas.

**Arquivos de Configuração**

Existem diversos formatos de arquivos que são empregados para a configuração das FPGAs, sendo cada um deles, mais adequado ao modo de carga adotado para o componente.

O arquivo no formato BIT [76, 79] (*Configuration bitstream*) é uma representação binária da implementação lógica da aplicação. Ele pode ser utilizado para criar um arquivo de gravação para uma memória PROM de configuração. Uma característica deste formato é o tamanho constante do arquivo que é gerado para uma dada FPGA, independente da lógica implementada por ela. Esta quantidade constante de bits deve-se ao fato de que cada bit da memória interna deve ser programado, mesmo que a lógica associada a este bit não seja utilizada pela aplicação. Este formato é aplicado aos modos serial mestre ou escravo e JTAG.

O arquivo no formato HEX [76,79] utiliza o código ASCII para formatação dos seus dados, onde cada algarismo hexadecimal representa quatro bits de um arquivo BIT. Este formato é utilizado em ambientes que utilizam o suporte de um microcontrolador ou microprocessador para efetuar a carga de configuração. Neste caso, o arquivo é armazenado

internamente na memória RAM ou FLASH destes componentes, para em seguida ser descarregado na memória de configuração da FPGA.

**Métodos de Desenvolvimento de Aplicações em FPGA**

Diversos ambientes destinados a desenvolver aplicações para FPGA permitem sintetizar modelos e circuitos, utilizando linguagens de descrição de *hardware* (HDL) como VHDL [24,79] e Verilog [37,77]. Esta forma de implementação garante a portabilidade e o reuso do código gerado, que possibilita que o mesmo código seja utilizado em diversas famílias e fabricantes de FPGA, além de permitir que sejam efetuadas modificações e correções com maior grau de segurança, rapidez e verificação formal por meio de asserções [75].

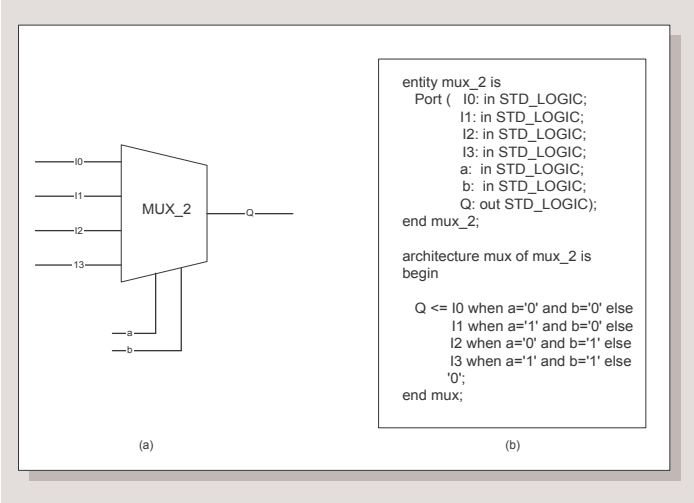



Figura 3.8: Exemplo de implementação utilizando HDL.

Uma outra forma comum de implementação de aplicações em FPGA é pela captura de esquemático lógico. Neste método, a implementação é feita utilizando as bibliotecas lógicas ou de funções disponibilizadas pelo ambiente de desenvolvimento da FPGA, para gerar um esquemático (ou desenho) lógico que represente graficamente a aplicação. A grande desvantagem da utilização deste método é que as ferramentas gráficas e as bibliotecas não são padronizadas, ou seja, estes recursos são dedicados e específicos para cada família e fabricante de FPGA. Este aspecto particular de implementação, não garante portabilidade, e o reuso da lógica fica restrito aos componentes suportados por aquela determinada família de FPGA.

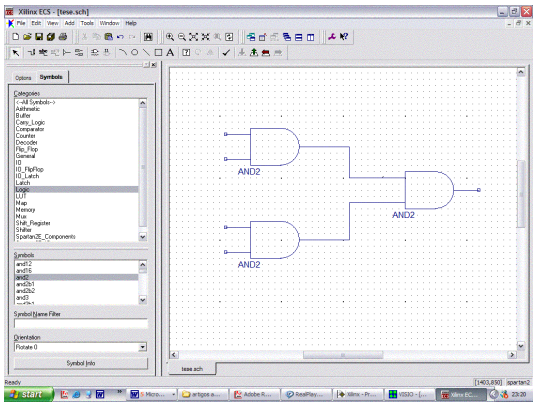



Figura 3.9: Exemplo de implementação utilizando captura de esquemático lógico.

## 3.2. CC1000 – Rádio Transceptor UHF

O CC1000 é um circuito integrado da Chipcon [44] de baixo consumo de potência e desenvolvido para ser utilizado em aplicações de comunicação sem fio. Foi concebido para operar nas bandas de freqüência ISM e SRD em 315, 433, 868 e 915MHz, podendo também ser programado para operar em outras bandas de freqüência de 300 a 1000MHz. O CC1000 possui as seguintes características:

- transceptor UHF completo;
- baixo consumo de potência;
- alta sensibilidade de recepção de sinal: típico de -110dbm em 2.4 kBaud;
- potência de sinal de saída programável de -20 a 10dbm;
- utiliza modulação FSK com taxa de transmissão de até 76.8 kBaud;
- possibilita a programação de compensação da variação da freqüência do cristal, de acordo com a temperatura, e em passos de 250Hz;
- opera com tensão reduzida de alimentação de 2.1 a 3.6V;
- requer poucos componentes externos;
- utiliza encapsulamento de tamanho reduzido: TSSOP-28;
- possui terminal para conexão de antena externa.

O CC1000 integra uma interface serial baseada em 3 terminais (PALE, PCLK e PDATA) pela qual é possível configurar os seus principais parâmetros de operação. Em um sistema típico, esta interface é conectada e controlada por um microcontrolador.

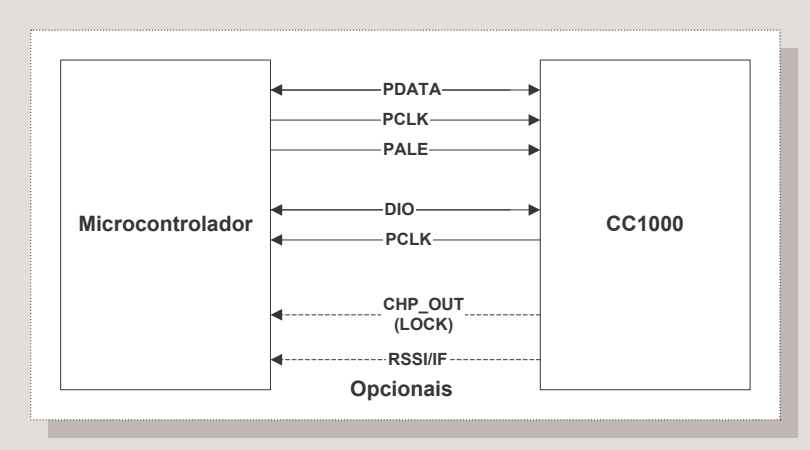



Figura 3.10: Detalhe funcional da interface do rádio CC1000.

Os parâmetros dos registros internos do CC1000 podem ser configurados para que ele obtenha diferentes níveis de performance. Podemos destacar a configuração dos seguintes parâmetros:

- modo recepção/transmissão;
- potência do sinal RF de saída;
- freqüência do sinal de RF de saída;
- modo desenergizado ou energizado;
- ativação/desativação do cristal oscilador;
- taxa de transmissão de dados;
- formato da codificação de dados: NRZ, Manchester ou modo UART;
- saída opcional RSSI ou de sinal IF.

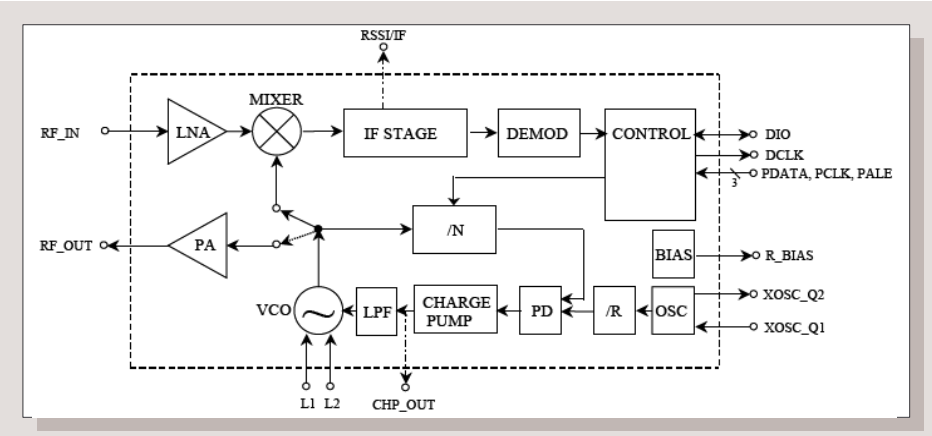



Figura 3.11: Diagrama de blocos da arquitetura do rádio CC1000.

De acordo com o diagrama de blocos do CC1000 mostrado na Figura 3.11, podemos verificar que:

- no modo de recepção, o CC1000 é configurado como um receptor do tipo super-heteródino. A entrada do sinal de RF (RF_IN) é amplificada por um amplificador de baixo ruído, o LNA, e reduzida pelo misturador (MIXER) para a freqüência intermediaria IF. No estágio de freqüência intermediaria (IF STAGE), este sinal é amplificado e filtrado antes de ser enviado ao estágio demodulador (DEMOD). Um sinal RSSI ou de IF pode ser opcionalmente enviado ao terminal de saída RSSI/IF do CC1000. Depois da demodulação, o sinal é convertido para o padrão digital e enviado para saída DIO. A sincronização da transmissão destes dados é feita internamente através do sinal de *clock* fornecido na entrada DCLK;

- no modo de transmissão, a saída do VCO é enviada para o amplificador de potência PA. A saída de RF tem sua freqüência chaveada pelos dados digitais recebidos na entrada DIO.

- o sintetizador de freqüência gera o sinal equivalente ao oscilador local que é enviado ao misturador (MIXER) do modo de recepção e ao amplificador de potência PA no modo transmissão. Este sintetizador é formado pelo cristal oscilador XOSC, pelo detector de fase (PD), pelo VCO, pelo circuito de CHARGE/PUMP e pelos divisores de freqüência (/R e /N).

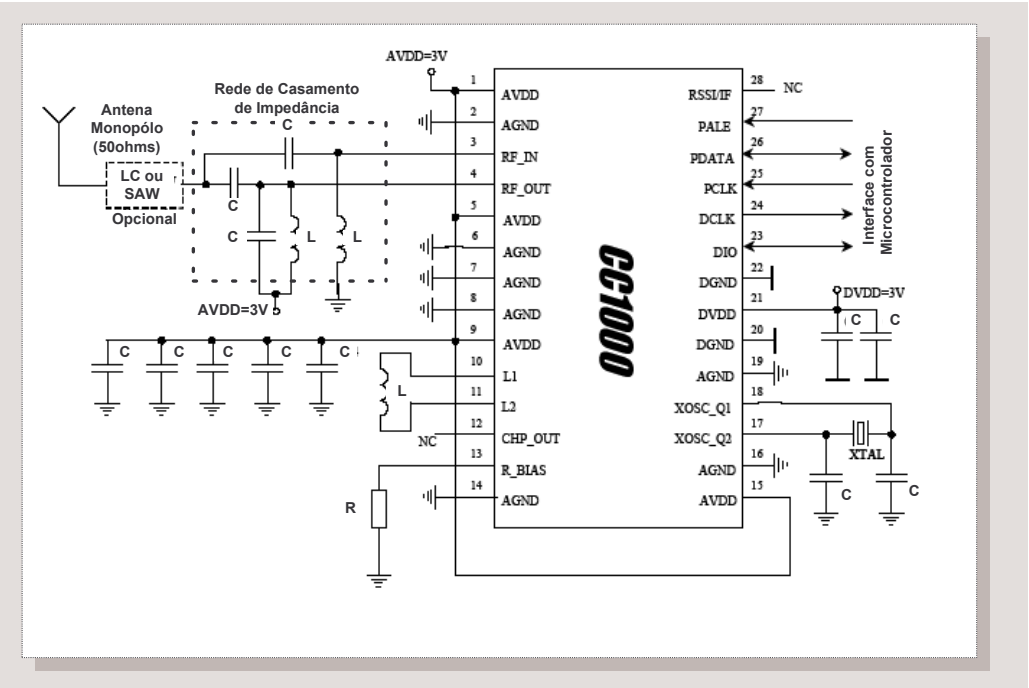



Figura 3.12: Implementação utilizando o rádio CC1000.

Os valores dos capacitores e indutores da rede conectada nas saídas RF_IN e RF_OUT devem ser ajustados para efetuar o casamento de impedância do receptor e transmissor em 50 ohms.

### 3.3. MSP430F149 – Microcontrolador

O MSP430F149 [78] faz parte da família MSP430 de microcontroladores de baixíssimo consumo de potência desenvolvida pela Texas Instruments. A arquitetura do MSP430 é otimizada e ideal para aplicações portáteis e com alimentação baseada em bateria. Ele permite cinco modos de operação de baixo consumo. A mudança rápida do estado de baixo consumo para o estado de ativação (tempo menor que 6μs) é obtida pela utilização de um oscilador controlado digitalmente (DCO).

O MSP430F149 possui as seguintes características:

- reduzido consumo de potência: 280μA@1MHz;
- arquitetura RISC de 16 bits com ciclo de instrução de 125 ns;
- conversor ADC integrado com tensão de referência interna;

- dois timers integrados de 16bits;
- comparador analógico interno;
- número de série programável;
- duas interfaces seriais universais e programáveis;
- 60k+256 bytes de memória Flash interna;
- 2kbytes de RAM interna.

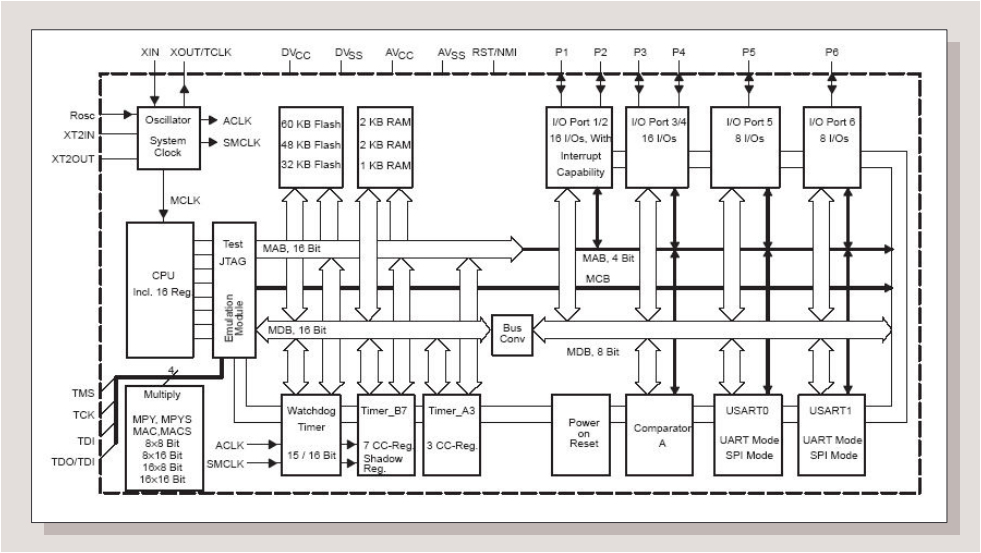



Figura 3.13: Diagrama de blocos da arquitetura interna do microcontrolador MSP430F149.

**CPU**

A unidade de processamento do MSP430, a CPU, é formada por 16 registradores com a função de reduzir o tempo de execução das instruções. Deste conjunto de registradores, 4 são dedicados para desempenhar as funções de contador de programa, apontador de pilha, registrador de status e registrador gerador de constante. Os restantes são de uso geral.

Uma característica operacional deste conjunto é que as operações do tipo registrador-registrador são executadas em apenas um ciclo de *clock*.

### Carregador Bootstrap

O MSP430 possui uma área de memória de código residente com a finalidade de possibilitar a programação da memória Flash e RAM do MSP430 através da interface serial.

### Memória Interna

A memória Flash interna pode ser programada através da interface JTAG, do carregador de *bootstrap* ou pela CPU. A CPU pode realizar operações de escritas de *bytes* ou *words* na memória Flash. A memória Flash do MSP430F149 possui as seguintes características:

- a área de memória principal é formada por n segmentos de 512 *bytes* e dois segmentos de informação (A e B) de 128 *bytes* cada;
- os segmentos de 0 a n podem ser apagados de uma única vez ou individualmente;
- os segmentos de informação A e B podem ser apagados em conjunto com os segmentos de 0 a n ou individualmente .

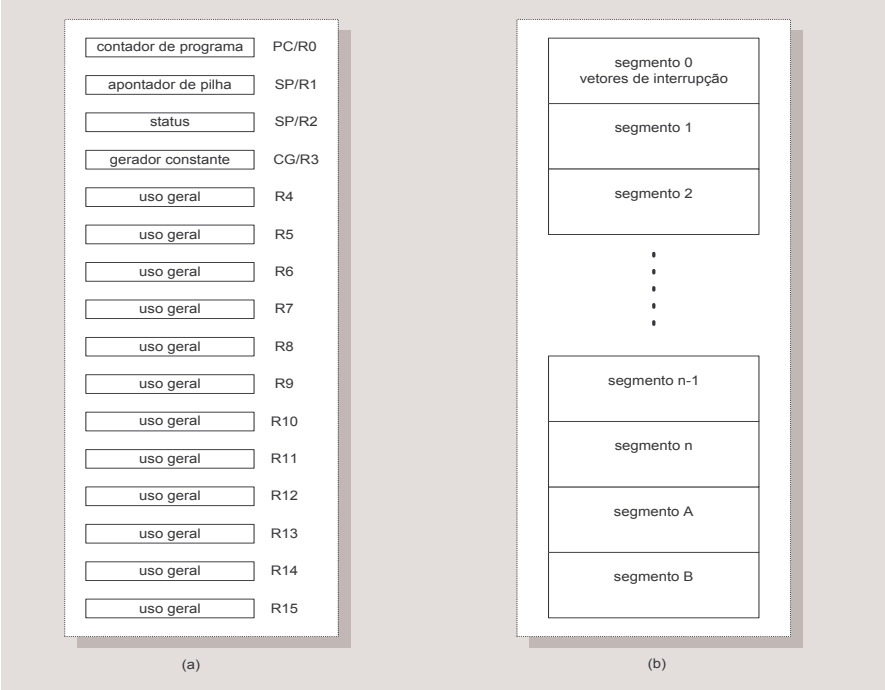



Figura 3.14: (a) registradores do MSP430F149. (b) organização da memória Flash interna.

### Watchdog Timer

O *Watchdog timer* tem a função de reiniciar a CPU quando ocorre um problema de execução de *software*. Um sinal de *reset* é gerado internamente, caso não ocorra regularmente acessos a este registro num intervalo de tempo inferior ao programado no *timer*. Cada acesso inibe e reinicializa o contador de tempo para o próximo acesso.

### Entradas e Saídas Digitais

O MSP430 dispõem de oito portas de entrada e saída com oito terminais cada, onde:

- todos os terminais individuais desta interface podem ser programados, independentemente entre si, como entrada ou saída;

- acessos de leitura e escrita aos registros de controle das portas de entrada e saída podem ser efetuados por todos os tipos de instruções.

### Oscilador e Sistema de *clock*

O módulo básico de *clock* inclui o cristal oscilador de relógio 32.768 kHz, o oscilador interno controlado digitalmente (DCO) e o cristal oscilador de alta freqüência. Este módulo foi desenvolvido para atender aos requisitos demandados pelos modos de baixo consumo do MSP430, possibilitando a estabilização rápida deste sinal durante a transição do estado inerte para o estado ativo.

### Multiplicador

O MSP430 integra um módulo multiplicador interno capaz de realizar as operações de multiplicação de 16x16, 16x8, 8x16 e 8x8 bit. Este módulo permite realizar estas operações de multiplicação com ou sem sinal. O resultado pode ser obtido, tão logo, ocorra a carga dos operandos nos registradores do multiplicador, sem a necessidade de ciclos extras de *clock*.

### Periféricos

Os barramentos internos de dados e endereços interconectam os periféricos à CPU e tem sua operação suportada por todos os tipos de instruções.

### Modos de Baixo Consumo

O MSP430 possui um modo de ativação e outros cinco modos de operação com baixo consumo. A Tabela 3.4 mostra os detalhes operacionais de cada modo.

| Modo de Operação | Estado da CPU | Estado do DCO | Cristal Oscilador | Sinais de clock | | |
|---|---|---|---|---|---|---|
| | | | | MCLK | ACIK | SMCLK |
| Modo Ativo | ativada | ativado | ativado | ativado | ativado | ativado |
| Modo baixo consumo 0 | desativada | ativado | ativado | desativado | ativado | ativado |
| Modo baixo consumo 1 | desativada | desativado | ativado | desativado | ativado | ativado |
| Modo baixo consumo 2 | desativada | ativado | ativado | desativado | ativado | desativado |
| Modo baixo consumo 3 | desativada | desativado | ativado | desativado | ativado | desativado |
| Modo baixo consumo 4 | desativada | desativado | parado | desativado | desativado | desativado |

Tabela 3.4: Modos de operação do MSP430F149.

## 3.4. E28F320J3A – Memória FlASH

O circuito integrado de memória Flash E28F320J3A [82] foi desenvolvido pelo fabricante Intel para aplicações de alta densidade de armazenamento de dados.

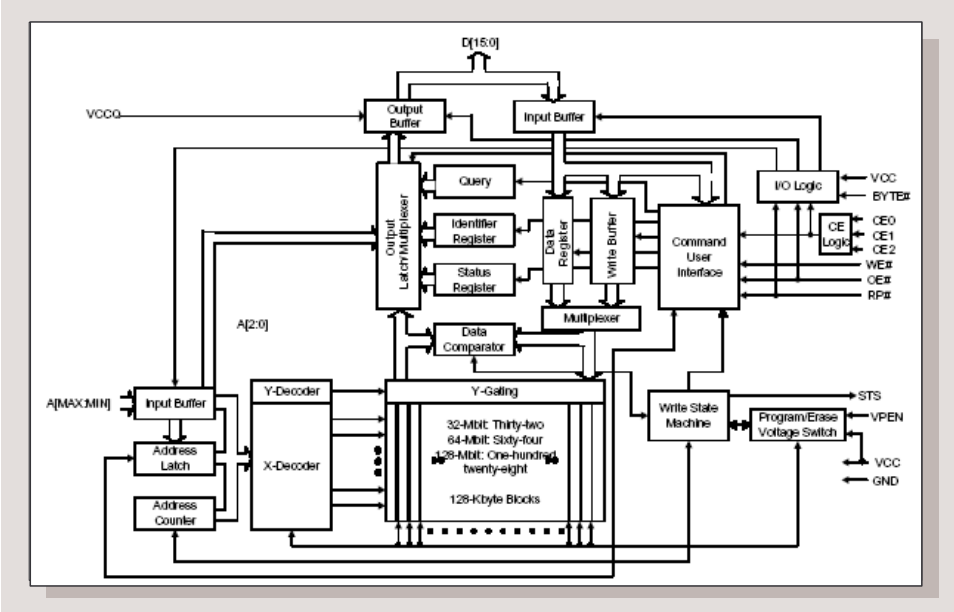



Figura 3.15: Diagrama de blocos da arquitetura interna da Flash E28F320J3A.

Este componente possui as seguintes características:

- 32M bits de memória organizados em 32 blocos simétricos de 128k bytes cada;
- acesso com comprimento de barramento de 8 ou 16 bits;

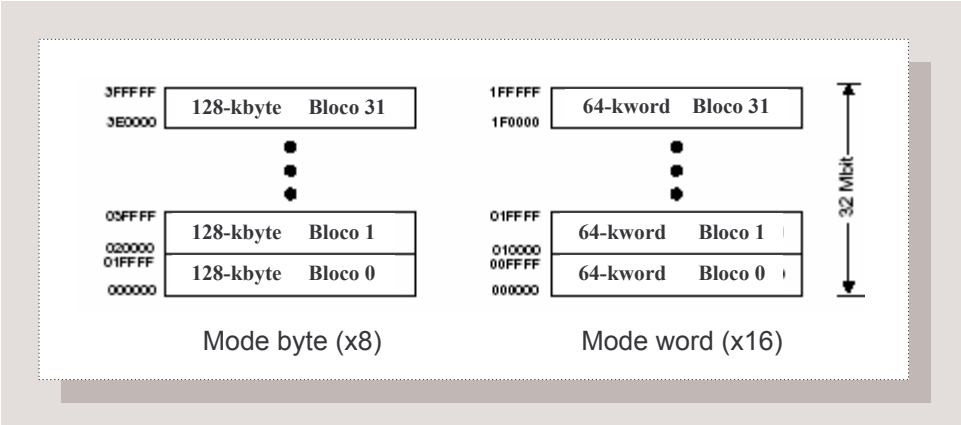



Figura 3.16: Organização interna da memória Flash E28F320J3A.

- registro de segurança de 128 bits com suporte para identificação do componente (ID);
- modo paginado de leitura para obter melhor performance;
- *buffer* de escrita de 32 bytes para otimizar a transferência de dados do processador para a memória Flash. Este recurso melhora em mais de vinte vezes a performance se comparado com componentes que não o oferecem;
- a interface CFI permite a utilização de algoritmos comuns as outras famílias de Flash. Cada modelo de Flash possui uma identificação JEDEC e do fabricante que permite o reconhecimento do dispositivo;
- a interface CUI permite que o processador controle e supervisione a operação interna do componente. Uma seqüência de comandos na CUI inicia a operação automática do dispositivo. Uma WSM executa automaticamente os algoritmos e as temporizações necessárias para as operações de apagamento, programação e escrita de bit de configuração;
- uma operação de apagamento está sempre associada a um bloco de 128kbytes. Cada bloco permite 100.000 (cem mil) operações de apagamento;

- o terminal de saída STS permite uma avaliação rápida do estado operacional em que se encontra a WSM interna da memória Flash. O processador monitorando este sinal pode identificar se a memória Flash está ocupada ou livre para realizar novas operações de apagamento, programação, etc.

- o terminal de entrada #BYTE permite configurar a Flash para operar com comprimento de barramento de 8 ou 16 bits;

- o terminal #RP permite que o processador controle o estado de ativação do componente. Aplicando nível lógico '0' neste terminal o componente entra no estado de baixo consumo;

- a ausência da tensão de programação Vpen no terminal VPEN da memória Flash bloqueia as operações de apagamento e escrita. Este recurso garante proteção extra contra operações indevidas de apagamento e escrita no componente;

- tempo de acesso inicial de 110ns.

### 3.5. TC55W800 – Memória RAM

O circuito integrado de memória RAM TC55W800 [83], desenvolvido pela Toshiba, possui reduzido consumo de potência e alta velocidade de operação. Este componente possui as seguintes características:

- capacidade de 8,388,608 bit organizados em 524,288 x 16bit ou 1,048,576 x 8bit;

- faixa de tensão de operação de 2,3 a 3,3V;

- tempo de acesso inicial de 55ns;

- potência de operação de 9,9mW/MHz;

- corrente em repouso de 5µA@3V e 10µA@3,3V;

- faixa de tensão de retenção de dados de 1,5 a 3,3V;

- faixa de temperatura estendida de -40 a 85 Celsius;
- o terminal de entrada #BYTE permite configurar a RAM para operar com comprimento de barramento de 8 ou 16 bits.

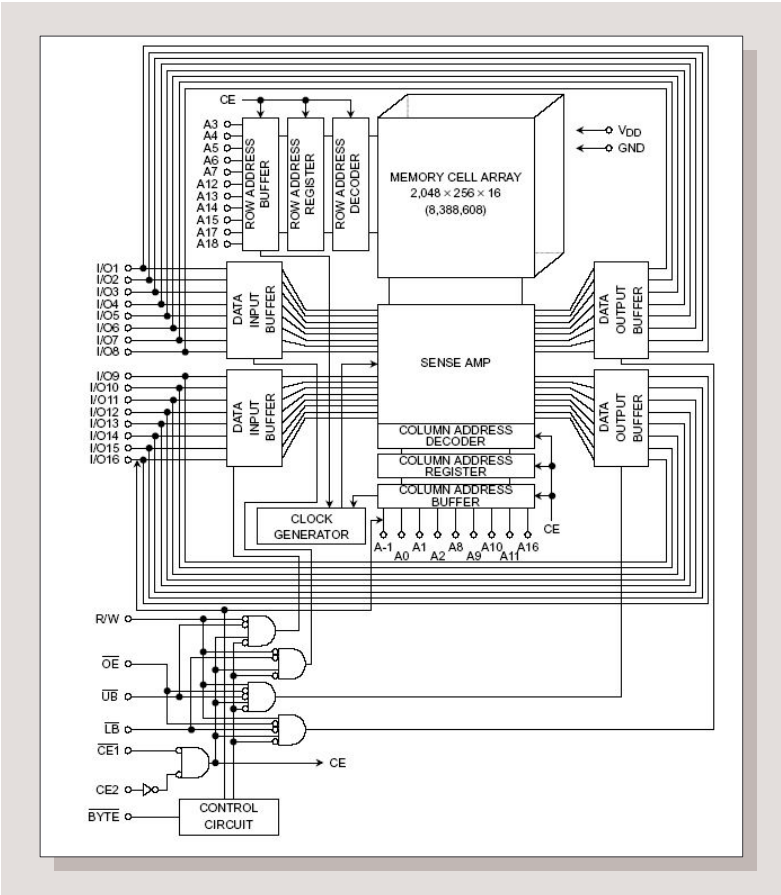



Figura 3.17: Diagrama de blocos da arquitetura interna da RAM TC55W800.

## 3.6. CompactFlash

O padrão CompactFlash [90] foi desenvolvido inicialmente com a finalidade de oferecer uma solução de captura, armazenamento e transporte de dados, áudio e imagem. Na época do seu surgimento, estes cartões empregavam exclusivamente memória do tipo Flash, como sugere o próprio nome do cartão. Atualmente, estes cartões também integram diversos dispositivos periféricos como interface Ethernet, Wireless, USB, Fax/Modem, discos magnéticos para armazenamento de dados, etc.

Estes cartões atendem ao conjunto de especificações técnicas e funcionais definidos pelo padrão PCMCIA [57], garantindo a compatibilidade entre sistemas digitais.

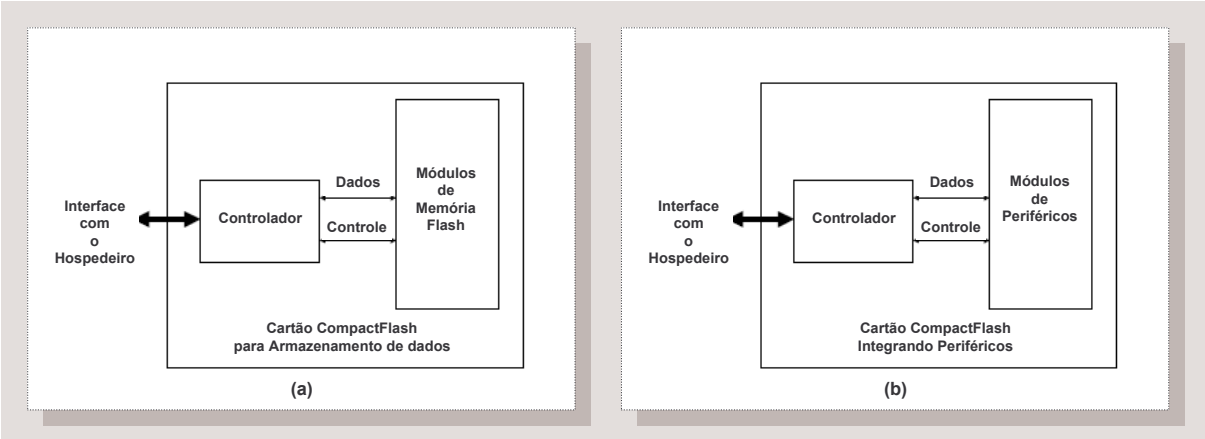



Figura 3.18: Diagramas de blocos das interfaces de cartões CompactFlash.
a) cartões de memória.
b) cartões de periféricos.

Os cartões CompactFlash podem operar no modo True IDE oferecendo compatibilidade com as unidades de disco magnético padrão IDE.

O controlador interno do CompactFlash controla e gerencia o protocolo da interface com o sistema hospedeiro, efetua armazenamento e recuperação dos dados com verificação de erro ECC, executa verificações de defeito e geração de diagnósticos, efetua o gerenciamento de potência e controle do sinal de *clock*.

O cartão CompactFlash pode operar em três modos:

- no modo PC CARD ATA usando o modo de entrada e saída;
- no modo PC CARD ATA usando o modo de memória;
- no modo True IDE compatível com as unidades de discos magnéticos.

A configuração do CompactFlash é realizada utilizando os registros definidos e alocados nos endereços atribuídos pela especificação PCMCIA. O espaço de endereçamento depende do modo de operação definido para cada tipo de cartão. Os sinais associados aos pinos do conector CF assumem significados distintos de acordo com cada modo. A Tabela 3.5 ilustra como os sinais são organizados no conector CF em cada modo de operação.

| PC Card Modo Memória | | | PC Card Modo E/S | | | Modo True IDE | | |
|---|---|---|---|---|---|---|---|---|
| **Pino** | **Sinal** | **Tipo de Pino** | **Pino** | **Sinal** | **Tipo de Pino** | **Pino** | **Sinal** | **Tipo de Pino** |
| 1 | GND | | 1 | GND | | 1 | GND | |
| 2 | D03 | E/S | 2 | D03 | E/S | 2 | D03 | E/S |
| 3 | D04 | E/S | 3 | D04 | E/S | 3 | D04 | E/S |
| 4 | D05 | E/S | 4 | D05 | E/S | 4 | D05 | E/S |
| 5 | D06 | E/S | 5 | D06 | E/S | 5 | D06 | E/S |
| 6 | D07 | E/S | 6 | D07 | E/S | 6 | D07 | E/S |
| 7 | CE1 | E | 7 | CE1 | E | 7 | CS0 | E |
| 8 | A10 | E | 8 | A10 | E | 8 | A10 | E |
| 9 | OE | E | 9 | OE | E | 9 | ATA SEL | E |
| 10 | A09 | E | 10 | A09 | E | 10 | A09 | E |
| 11 | A08 | E | 11 | A08 | E | 11 | A08 | E |
| 12 | A07 | E | 12 | A07 | E | 12 | A07 | E |
| 13 | VCC | | 13 | VCC | | 13 | VCC | |
| 14 | A06 | E | 14 | A06 | E | 14 | A06 | E |
| 15 | A05 | E | 15 | A05 | E | 15 | A05 | E |
| 16 | A04 | E | 16 | A04 | E | 16 | A04 | E |
| 17 | A03 | E | 17 | A03 | E | 17 | A03 | E |
| 18 | A02 | E | 18 | A02 | E | 18 | A02 | E |
| 19 | A01 | E | 19 | A01 | E | 19 | A01 | E |
| 20 | A00 | E | 20 | A00 | E | 20 | A00 | E |
| 21 | D00 | E/S | 21 | D00 | E/S | 21 | D00 | E/S |
| 22 | D01 | E/S | 22 | D01 | E/S | 22 | D01 | E/S |
| 23 | D02 | E/S | 23 | D02 | E/S | 23 | D02 | E/S |
| 24 | WP | S | 24 | IOSI16 | S | 24 | IOSI16 | S |
| 25 | CD2 | S | 25 | CD2 | S | 25 | CD2 | S |
| 26 | CD1 | S | 26 | CD1 | S | 26 | CD1 | S |
| 27 | D11 | E/S | 27 | D11 | E/S | 27 | D11 | E/S |
| 28 | D12 | E/S | 28 | D12 | E/S | 28 | D12 | E/S |
| 29 | D13 | E/S | 29 | D13 | E/S | 29 | D13 | E/S |
| 30 | D14 | E/S | 30 | D14 | E/S | 30 | D14 | E/S |
| 31 | D15 | E/S | 31 | D15 | E/S | 31 | D15 | E/S |
| 32 | CE2 | E | 32 | CE2 | E | 32 | CE2 | E |
| 33 | VS1 | S | 33 | VS1 | S | 33 | VS1 | S |
| 34 | IORD | E | 34 | IORD | E | 34 | IORD | E |
| 35 | IOWR | E | 35 | IOWR | E | 35 | IOWR | E |
| 36 | WE | E | 36 | WE | E | 36 | WE | E |
| 37 | RDY/BSY | S | 37 | IERQ | S | 37 | INTRQ | S |
| 38 | VCC | | 38 | VCC | | 38 | VCC | |
| 39 | CSEL | E | 39 | CSEL | E | 39 | CSEL | E |
| 40 | VS2 | S | 40 | VS2 | S | 40 | VS2 | S |
| 41 | RESET | E | 41 | RESET | E | 41 | RESET | E |
| 42 | WAIT | S | 42 | WAIT | S | 42 | WAIT | S |
| 43 | INPACK | S | 43 | INPACK | S | 43 | INPACK | S |
| 44 | REG | E | 44 | REG | E | 44 | REG | E |
| 45 | BVD2 | E/S | 45 | SPKR | E/S | 45 | DASP | E/S |
| 46 | BVD1 | E/S | 46 | BVD1 | E/S | 46 | PDIAG | E/S |
| 47 | D08 | E/S | 47 | D08 | E/S | 47 | D08 | E/S |
| 48 | D09 | E/S | 48 | D09 | E/S | 48 | D09 | E/S |
| 49 | D10 | E/S | 49 | D10 | E/S | 49 | D10 | E/S |
| 50 | GND | | 50 | GND | | 50 | GND | |

Tabela 3.5: Associação dos sinais e pinos da interface do cartão CompactFlash.

### 3.7. QS34X245 – 32 Chaves FET

O circuito integrado QS34X245 [86] integra um conjunto de 32 chaves FET destinado às aplicações que requeiram isolação e comutação de múltiplos sinais, como por exemplo, os barramentos digitais.

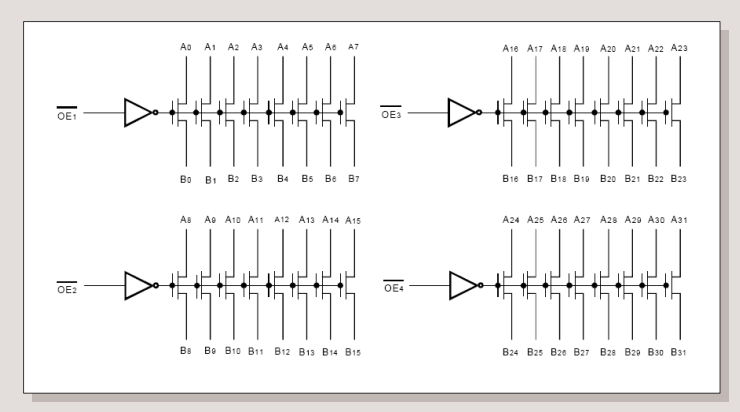



Figura 3.19: Organização interna das chaves FET do CI QS34X245.

Estas chaves são bidirecionais e inserem uma resistência série típica de 5 ohms quando ligadas. Este baixo valor de resistência garante que os sinais das entradas sejam conectados às saídas sem adicionar atraso de propagação.

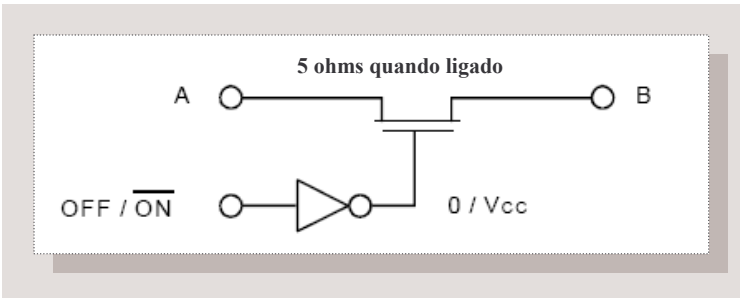



Figura 3.20: Detalhe elétrico e funcional de uma chave FET do QS34X245.

A tensão de saída segue o nível da tensão de entrada até atingir o limite de aproximadamente Vcc-1V. A partir deste ponto, qualquer variação do nível de entrada entre Vcc-1 e Vcc, não produz variação correspondente no nível de saída que se mantêm fixo em Vcc-1V. Este efeito é devido ao fato da implementação da chave utilizar um transistor FET canal N em “seguidor de emissor”.

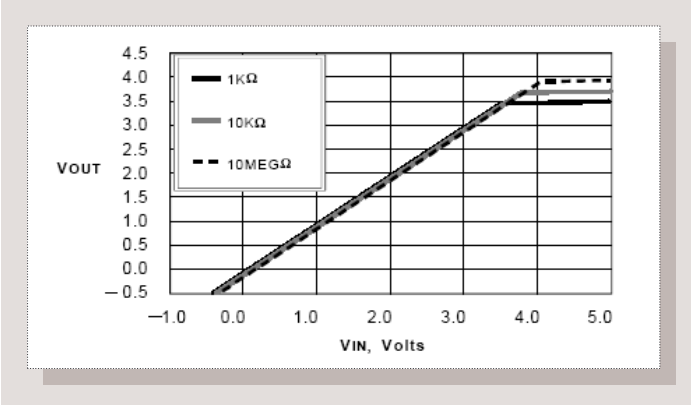



Figura 3.21: Características de entrada e saída da chave FET QS34X245.

## 3.8. MAX3243 – Interface Elétrica RS-232

O MAX3243 [85] é um circuito integrado que integra três interfaces de transmissão (*drivers*) e cinco de recepção (*receivers*), compatíveis com as especificações TIA/EIA-232 [52] e padrão ITU V.28 [56]. Este componente opera com fonte de alimentação simples na faixa de 3 a 5,5V e requer alguns capacitores externos para a geração das tensões e níveis elétricos adequados ao padrão RS-232. Adicionalmente, o MAX3243 possui controle de ativação para redução do consumo de potência e do estado de repouso.

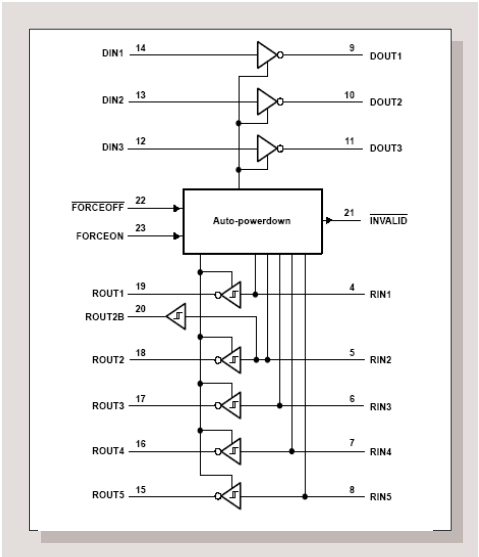



Figura 3.22: Diagrama de blocos da arquitetura interna do CI MAX3243.

## 3.9. MC34063 – Controlador para Conversor DC-DC

O controlador MC34063 [88] incorpora um conjunto de funções que simplificam a implementação de diversos tipos de conversões DC-DC como: *step-up*, *step-down* e tensão invertida. Este controlador possui internamente:

- referência de tensão com compensação térmica;
- oscilador com controle da relação de intervalo de ciclo – *duty cycle*;
- circuito limitador de pico de corrente;
- comparador de tensão;
- *driver* de saída com alta capacidade de corrente.

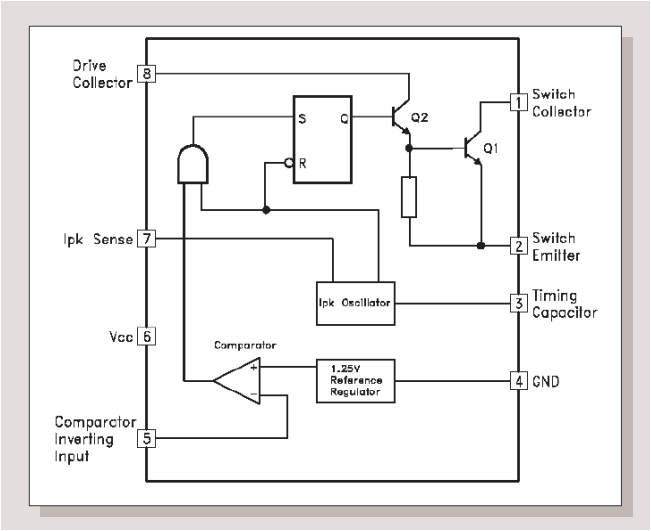



Figura 3.23: Diagrama de blocos do sensor de temperatura LM70.

## 3.10. TPS60130 – Conversor e regulador DC-DC

O TPS60130 [87] é um conversor *step-up* e regulador DC-DC que produz na saída uma tensão de 5V com tensão de entrada variando entre 2,7 e 5,4 V.

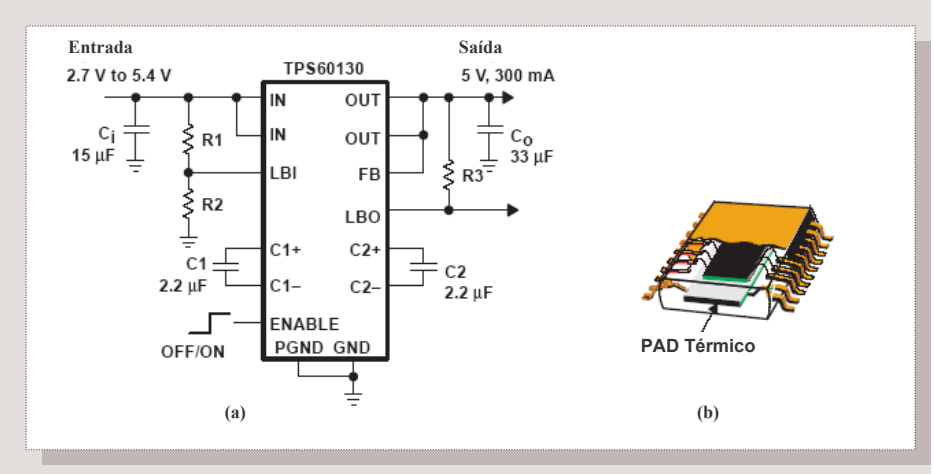



Figura 3.24: Solução de implementação utilizando o CI TPS60130.

Este conversor possui as seguintes características:

- excelente eficiência de conversão (>90%);
- capacidade de corrente de saída de 300mA;
- saída de tensão regulada em 5 V +/- 5%;
- 60 µA de corrente quiescente de operação;
- 0,05µA de corrente no estado desativado;
- incorpora PAD térmico (área metálica no encapsulamento) para auxiliar na dissipação de calor;
- não requer indutores externos: baixa emissão eletromagnética;
- requer somente quatro componentes externos.

### 3.11. LM70 – Sensor de Temperatura

O LM70 [60] incorpora um sensor de temperatura, um conversor analógico/digital Delta-Sigma e interface serial compatível com os padrões SPI [64] e MICROWIRE [62].

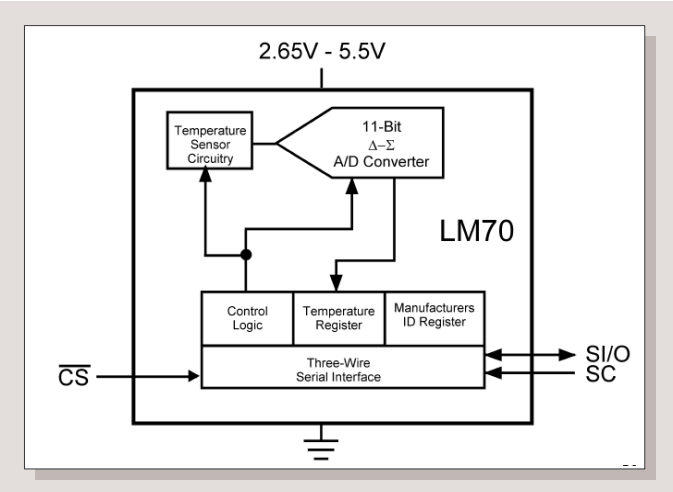



Figura 3.25: Diagrama de blocos do sensor de temperatura LM70.

No LM70, o sinal do sensor de temperatura é convertido internamente através de um conversor A/D de 10bit, oferecendo uma resolução de 0,25 Celsius por LSB numa faixa de temperatura que vai de -55 a +150 Celsius. O valor da temperatura pode ser obtido acessando-se o registro interno definido para essa finalidade no LM70. Este acesso é feito através da interface serial do componente, tipicamente, efetuado por um microcontrolador.

## 3.12. Elementos de Bateria

As baterias são classificadas em dois tipos: primárias (não-recarregáveis) e secundárias (recarregáveis). Além desta divisão, são normalmente identificadas de acordo com o tipo de material eletroquímico utilizado na sua construção. A combinação de diferentes elementos químicos produz uma variedade de modelos de células com diferentes parâmetros construtivos e de performance, tais como: capacidade de energia, nível de tensão, resistência interna, ciclo de carga e descarga, estabilidade, faixa de temperatura de operação, formato, tamanho e vida útil. Em equipamentos eletrônicos usualmente são empregados os seguintes tipos de baterias [94]: Ni-MH, NiCad, Lithium, Zinc-air, Alkaline (Manganese dioxide) e Lithium-Ion. O tipo adequado de bateria deve ser escolhido de acordo com os requisitos de operação e performance exigidos para cada aplicação.

# CAPÍTULO 4

# Arquitetura Desenvolvida

A arquitetura do RANS-300 que será apresentada neste trabalho foi idealizada com o objetivo de ampliar o horizonte de aplicações utilizando RSSFs e oferecer recursos computacionais não previstos pelos nós sensores atuais que possibilitem a realização de monitoramento avançado de sistemas.

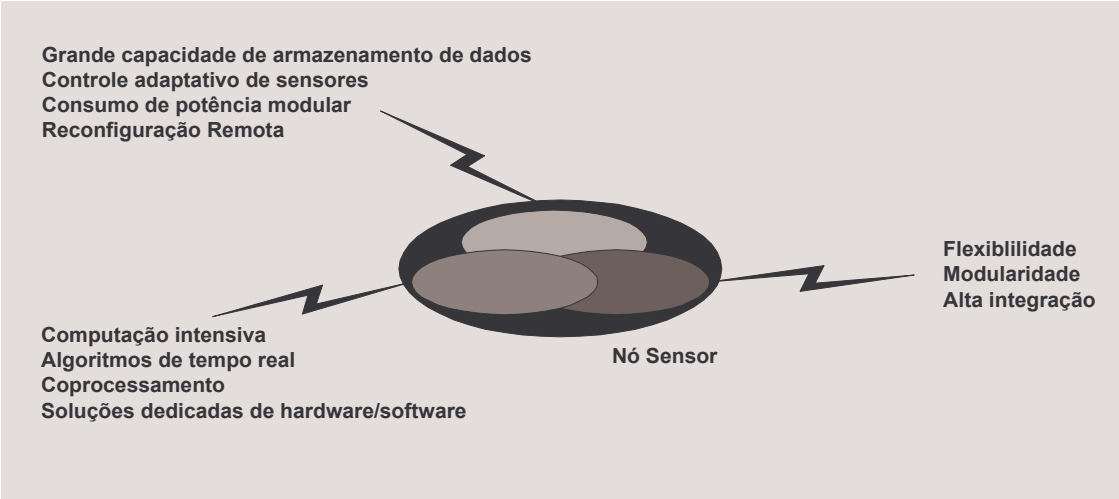



Figura 4.1: Soluções baseadas na utilização do RANS-300.

De uma maneira geral, cada aplicação possui uma forma mais eficiente e otimizada para ser implementada, podendo ser realizada por *hardware*, *software* ou pela combinação destes. Possibilitar estas alternativas requer que a plataforma utilizada para a implementação possua aspectos construtivos especiais, dinâmicos e configuráveis de *hardware* e *software*. Desenvolver uma plataforma que proporcione atender a estes aspectos, requerer que a arquitetura de *hardware* possua partes flexíveis, adaptativas, programáveis e baseadas em arquiteturas reconfiguráveis [54, 55]. O desenvolvimento do RANS-300 foi orientado para possibilitar a utilização destes recursos reconfiguráveis de *hardware* pelas aplicações de RSSFs.

## 4.1. Requisitos Técnicos e Funcionais

A atividade de levantamento dos requisitos técnicos e funcionais (Figura 1.4) produziu um conjunto de diretrizes gerais e de requisitos técnicos, para orientar a elaboração da arquitetura do RANS-300. Estas diretrizes definiram que a arquitetura deveria prever:

- a divisão elétrica do *hardware* em blocos de acordo com o consumo de potencia;
- a utilização de diferentes tipos de fonte de energia (bateria, painéis solares, etc); Ou seja, não deve ser incorporado fisicamente e nem limitado que o circuito utilize um tipo especifico de fonte de potência;
- um conector de entrada de alimentação que possua polarização mecânica a fim de eliminar o risco de inversão da polaridade da fonte externa;
- a regulação e a filtragem da alimentação de entrada para possibilitar que o circuito opere numa faixa flexível de tensão;
- modos para efetuar o monitoramento do consumo de potência do circuito. Esta característica permitirá conhecer o comportamento de consumo em diversas situações reais de funcionamento. As informações de consumo poderão ser utilizadas para esboçar o mapa energético e operacional da RSSF em funcionamento ou, por exemplo, permitir que se realize um gerenciamento dinâmico do recurso de energia no sentido de obter a máxima autonomia e desempenho;
- modos para efetuar a medição da tensão da fonte. Para as aplicações que utilizarem bateria como fonte de energia, esta medida permitirá identificar a evolução da curva de descarga e tornar possível a geração de alarme de bateria fraca e a utilização de procedimentos de controle do consumo de potência;
- um conector para permitir a conexão de módulos auxiliares e de sensores controlados pelo microcontrolador. Não é mandatório que o dispositivo possua integrado ao seu circuito nenhum conjunto de sensores;
- a implementação de uma interface padrão CompactFlash para a instalação de dispositivos comerciais de armazenamento de dados e comunicação;

- a implementação de uma FPGA de alta densidade de elementos lógicos com capacidade mínima de 300k portas lógicas;

- que um conjunto de terminais de E/S da FPGA sejam disponibilizados em um conector de expansão. Este recurso permitirá o interfaceamento com módulos de *hardware* externos;

- a implementação de bancos de memória volátil e não-volátil com capacidade mínima de 1 Mbytes e interligados à FPGA. Estas memórias deverão ser destinadas às aplicações que necessitem de áreas para armazenamento temporário e permanente de dados;

- o controle dinâmico da ativação da alimentação da FPGA, da interface CompactFlash e dos bancos de memória Flash e RAM para otimização do consumo de potência;

- um conector para disponibilizar a interface JTAG da FPGA para permitir a carga de configuração e auxiliar na depuração;

- a utilização de um microcontrolador versátil, eficiente e de baixo consumo para efetuar o controle e o processamento central;

- que os barramentos da FPGA e demais periféricos possam ser emulados pelos terminais de E/S do microcontrolador;

- que o protocolo de comunicação permita transferir arquivos binários de configuração da FPGA e também dados destinados ao armazenamento nos bancos de memória;

- um conector para interface JTAG para permitir a gravação do *software* aplicativo na memória interna do microcontrolador;

- a utilização de rádio transceptor com parâmetros operacionais configuráveis por *software* e que disponha de mecanismos de controle da potência de transmissão e de estados de conservação de energia;

- que o barramento de comunicação do rádio seja baseado na interface RS-232C e possibilite a sua conexão com um microcomputador externo;

- a identificação física do dispositivo. Deve permitir que a gravação da identificação seja efetuada na área de memória não-volátil do microcontrolador ou em memória externa removível;

- a utilização de sistemas operacionais aplicados às RSSFs;

- uma interface de monitor baseada no padrão RS-232C para permitir a conexão externa de microcomputador pessoal;

- interfaces visuais (leds, display, etc) para auxiliarem na depuração dos aplicativos de *software*;

- que a elaboração do projeto do PCB seja efetuada utilizando somente duas faces para roteamento dos sinais elétricos;

- a utilização preferencial de componentes com encapsulamentos SMD.

## 4.2. Justificativa para o Emprego de Arquiteturas Reconfiguráveis

O desenvolvimento e a popularização de circuitos integrados baseados em arquiteturas reconfiguráveis de *hardware,* como as CPLDs e as FPGAs, tem proporcionado a migração e direcionamento de implementações, antes utilizando componentes discretos e ASICs, para esta classe de componentes. Estes componentes combinam flexibilidade, performance e custo efetivo. Sua utilização proporciona a implementação de uma variedade de alternativas customizadas de circuitos lógicos, oferecendo a facilidade de se realizar atualizações (*updates*), melhoramentos (*upgrades*), correções à distância e apresentando excelente portabilidade (reuso de lógica). Estas características tornam ideal a utilização destes componentes para implementar soluções otimizadas e eficientes para aplicações que requeiram computação intensiva.

A sua flexibilidade permite que qualquer aplicação baseada nestes componentes possa ser facilmente atualizada através do procedimento de carga de um arquivo de configuração para a sua memória interna. Este procedimento elimina em grande parte a necessidade de ter que efetuar correções e adequações físicas de *hardware*. No caso específico das FPGAs, os dados de configuração são guardados na sua memória interna, o que proporciona um número quase ilimitado de ciclos de reprogramação.

A arquitetura do RANS-300 apresentará uma solução que combina microcontrolador e FPGA. A justificada deve-se aos níveis de performance e flexibilidade pretendidos para cada tipo de aplicação. O microcontrolador pode, por exemplo, realizar aplicações sistêmicas e gerenciais, enquanto, a FPGA, pode executar um conjunto de tarefas complementares, programáveis, de coprocessamento e adicionar diversos periféricos.

O emprego de arquiteturas reconfiguráveis em nós sensores tornam estes dispositivos melhores adaptados para:

- processar dados em tempo real e garantir que os eventos sejam corretamente analisados;
- modificar o processamento pela simples reconfiguração do dispositivo e para consertar problemas de funcionamento (*bugfix*);
- modificar o processamento dinamicamente de acordo com as informações enviadas pela aplicação ou à partir de resultados analisados;
- adapta-se a um conjunto de especificações sistêmicas;
- prover um conjunto diferente de interfaces e circuitos;
- fornecer novas aplicações por qualquer meio de transferência de arquivos e através de programação local (*in_sytem programming*);
- implementar máquinas de estados em *hardware*;
- implementar sistemas operacionais em hardware – ex: Sierra 16 RTOS [63].

## 4.3. Arquitetura de Hardware

Para o desenvolvimento do RANS-300 foi utilizada uma FPGA de grande densidade lógica e de alguns periféricos como, uma unidade CompactFlash, memória RAM e memória Flash. Estes elementos específicos de *hardware* devem ser considerados pelas aplicações como recursos computacionais complementares e de uso temporário. Isto porque, a ativação contínua destes elementos comprometeria a autonomia de operação do dispositivo nó sensor, uma vez que, nas RSSFs a energia é restrita e sua utilização deve ser bem justificada e

otimizada ao máximo. Para tornar possível o emprego destes elementos de *hardware* foram implementados alguns mecanismos de controle da ativação da sua alimentação e da comutação dos sinais de acesso. Estes mecanismos serão descritos detalhadamente mais adiante.

A arquitetura de *hardware* desenvolvida para o RANS-300 pode ser representada por três blocos funcionais: *fonte*, *núcleo básico* e *reconfigurável*.

O bloco *fonte* é responsável pela conversão, comutação, adaptação e fornecimento das tensões de alimentação para todo o circuito deste dispositivo nó sensor.

A criação dos blocos *núcleo básico* e *reconfigurável* satisfaz o requisito de possibilitar a separação elétrica dos circuitos de *hardware* baseada no consumo de potência. Esta separação foi idealizada levando em consideração a aplicabilidade de cada bloco no contexto das redes de sensores. A idéia básica foi de controlar o fornecimento de energia destinado ao bloco *reconfigurável* para que o seu consumo de potência seja quase nulo quando os seus recursos não estejam sendo utilizados.

A Figura 4.2 mostra o diagrama de blocos da arquitetura de *hardware* desenvolvida para o RANS-300. Como pode ser notado nesta figura, o *núcleo básico* controla a ativação da alimentação do bloco *reconfigurável*.

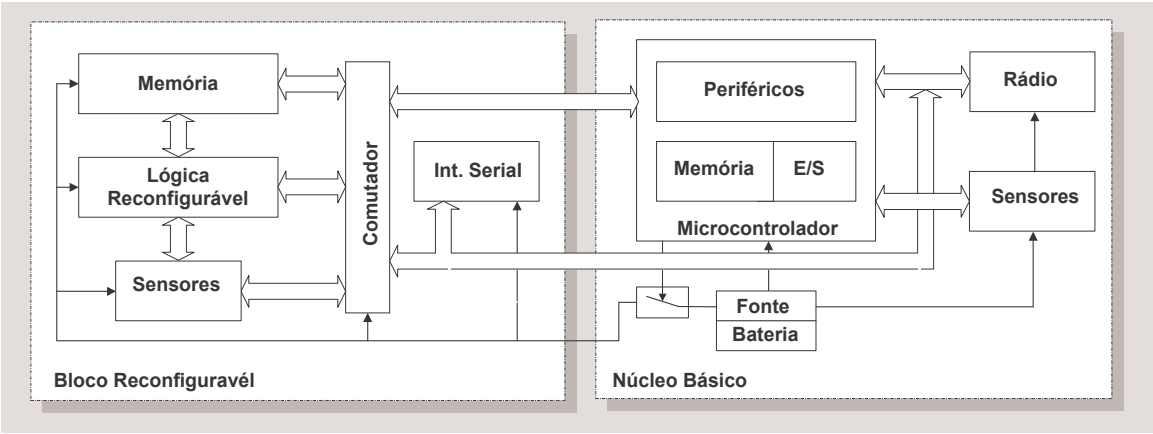



Figura 4.2: Diagrama de blocos da arquitetura do RANS-300.

### 4.3.1. Núcleo Básico

O núcleo básico foi desenvolvido considerando os aspectos construtivos tradicionais encontrados em diversos nós sensores. A implementação do seu *hardware* foi baseada no emprego de elementos clássicos como: microcontrolador, sensores e rádio transceptor. Este tipo de construção proporciona excelente integração, com consumo reduzido de potência e baixo custo.

Este bloco oferece um conjunto de recursos de *hardware e software* necessários para atender aos requisitos funcionais requeridos pela grande maioria das aplicações atualmente idealizadas para RSSFs. Ele é responsável por:

- efetuar o controle e processamento das informações relativas aos sensores;
- estabelecer a comunicação com os demais nós;
- gerenciar os recursos de energia;
- efetuar o tratamento dos eventos gerados pela rede;
- controlar o bloco *reconfigurável* (alimentação e o funcionamento das interfaces);
- monitorar as informações de consumo da fonte de alimentação.

O núcleo básico foi idealizado para desempenhar todas estas funções com um consumo extremamente reduzido de energia. Isto porque, este bloco deverá operar continuamente e mesmo adotando procedimentos de economia de energia o ideal é obter o menor consumo a fim de estender ao máximo a vida útil deste dispositivo nó sensor. A escolha dos componentes eletrônicos e das interfaces que são disponibilizadas seguiu rigorosamente este requisito. Os principais componentes utilizados na implementação do *hardware* foram:

- o CI MSP430F149 (Texas Instruments). Microcontrolador baseado em arquitetura RISC de 16 bits operando na freqüência de 1MHz;
- o CI CC1000 (Chipcon). Rádio transceptor para comunicação operando na banda ISM/SRD de 315, 433, 868 e 915MHz configurada através da interface serial;

- o CI Q34X245 (Quality Semiconductor). Incorpora 32 chaves FET;
- o CI MAX3243 (Maxim). Interface elétrica RS-232;
- o CI NM 93CS56 (Atmel). Memória serial EEPROM de 2 k bits;
- o CI NM74LVS273 (National). Incorpora oito Flip-Flop tipos D com clear geral;
- módulo oscilador de 14.7456 MHz;
- cristal de 32.768kHz;
- o CI LM70 (National). Sensor de temperatura de -55 a +150 Celsius;
- O CI 74HCT4016 (National). Incorpora oito chaves analógicas;
- LEDs bicolores;
- conector de expansão de 50 pinos (Hirose);
- conector Header 2x7, passo 2,54mm (Molex);
- conector Header 2x5, passo 2,54mm (Molex);
- conector Header 1x5, passo 2,54mm (Molex).

Os detalhes funcionais destes componentes foram descritos no capitulo 3.

A Figura 4.3 ilustra a arquitetura do núcleo básico identificando os componentes eletrônicos utilizados. Como relação a esta figura podemos verificar que:

- os terminais de entrada e saída do microcontrolador MSP430 foram utilizados para emular alguns barramentos paralelos, sinais de controle de acesso e de habilitação de alguns periféricos externos. A Tabela 4.1 ilustra a alocação funcional efetuada para estes terminais e identifica o propósito de cada um deles na implementação do *hardware* do RANS-300. Alguns terminais possuem alocação fixa como é o caso do conversor ADC e da interface JTAG;

- os sinais do MSP430 que são enviados para o bloco *reconfigurável* passam antes por um arranjo de chaves FET, para permitir efetuar a separação elétrica destes blocos. Estas chaves evitam que ocorra um consumo desnecessário de corrente quando a parte reconfigurável encontra-se desenergizada. Neste estado, a maioria dos circuitos integrados apresenta baixa impedância interna, e, caso os sinais entre estes blocos fossem diretamente conectados, o bloco *reconfigurável*, drenaria uma determinada corrente do *núcleo básico* que tenderia, neste caso, fornecer a alimentação para todo o circuito elétrico do bloco *reconfigurável* através dos terminais do MSP430. Esta corrente indesejável afetaria diretamente a autonomia da bateria do dispositivo e poderia causar danos aos estágios de saída do MSP430. Para a implementação do arranjo de chaves foram utilizados dois CIs QS34X245;

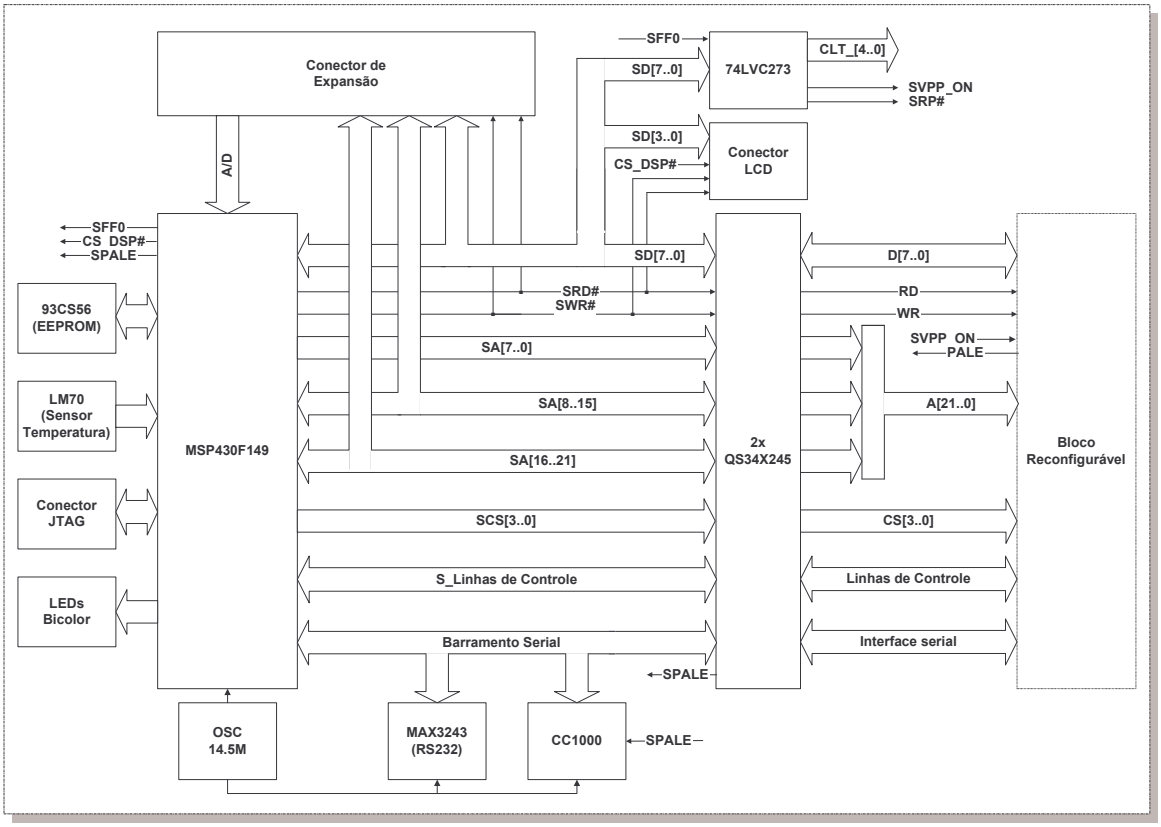



Figura 4.3: Diagrama de blocos do núcleo básico do RANS-300.

| Portos MSP430 | Sinais | Sentido | Função Emulada |
|---|---|---|---|
| P1[7..0] | SD[7..0] | Bidirec. | Barramento de dados |
| P2.0 | SCS0 | Saída | Comando de habilitação da memória Flash |
| P2.1 | SCS1 | Saída | Comando de habilitação da memória RAM0 |
| P2.2 | SCS2 | Saída | Comando de habilitação da memória RAM1 |
| P2.3 | FPGA1 | Bidirec. | Sinal de controle genérico para a FPGA |
| P2.4 | FPGA0 | Bidirec. | Sinal de controle genérico para a FPGA |
| P2.5 | SRD# | Saída | Sinal de controle de leitura de dados |
| P2.6 | SWR# | Saída | Sinal de comando de escrita de dados |
| P2.7 | SCS_DSK# | Saída | Comando de hab. da interface CompactFlash |
| P3.0 | SFF0 | Saída | Comando de hab. do registro auxiliar externo |
| P3.1 | SPALE | Saída | Comando de hab. do rádio CC1000 |
| P3.2 | SFPGA_ON | Saída | Comando de hab. do comutador/barramento |
| P3.3 | SOSC0 | Entrada | Clock padrão (14.57 M para a interface serial) |
| P3.4 | TXD | Saída | Sinal de trans. de dados da interface serial |
| P3.5 | RXD | Entrada | Sinal de recep. de dados da interface serial |
| P3.6 | IO_CMD | Saída | Comando de hab. da conFigura da FPGA |
| P3.7 | ALM_ON | Saída | Comando de hab. do bloco reconfigurável |
| P4[7..0] | AS[7..0] | Saída | Barramento de endereços |
| P5[7..0] | SAD[15..8] | Bidirec. | Barramento multiplex. de dados e endereços |
| P6[5..0] | SA[21..16] | Saída | Barramento de endereços |
| P6.6 | ICC0 | Ent.(A/D) | Amostragem da corrente total do circuito |
| P6.7 | ICC1 | Ent. A/D) | Amostragem da tensão de entrada (bateria) |
| TCK | TCK | Entrada | Sinal de clock da interface JTAG |
| TMS | TMS | Entrada | Sinal de estados da interface JTAG |
| TDI | TDI | Entrada | Sinal de entrada de dado JTAG |
| TDO | TDO | Saída | Sinal de saída de dado JTAG |
| Vref+ | Vref+ | Entrada | Tensão de referência do conversor A/D |
| VeREF+ | VeREF+ | Entrada | Tensão de referência do conversor A/D |
| Vref+/VeREF+ | Vref+/VeREF+ | Entrada | Tensão de referência do conversor A/D |

Tabela 4.1: Alocação dos sinais do MSP430F149.

- os sinais do MSP430 que são enviados para o bloco *reconfigurável* passam por um arranjo de chaves FET (implementado pelo CI QS34X245) que permite separar eletricamente estes blocos. Estas chaves evitam que ocorra um consumo desnecessário de corrente quando a parte reconfigurável encontra-se desenergizada. Neste estado, a maioria dos circuitos integrados apresenta baixa impedância interna, e, caso os sinais entre estes blocos fossem diretamente conectados, todo o circuito do bloco *reconfigurável* seria alimentado através dos terminais do MSP430 (Figura 4.4). Esta

corrente drenada afetaria diretamente a autonomia da bateria e poderia causar danos aos estágios de saída do MSP430.

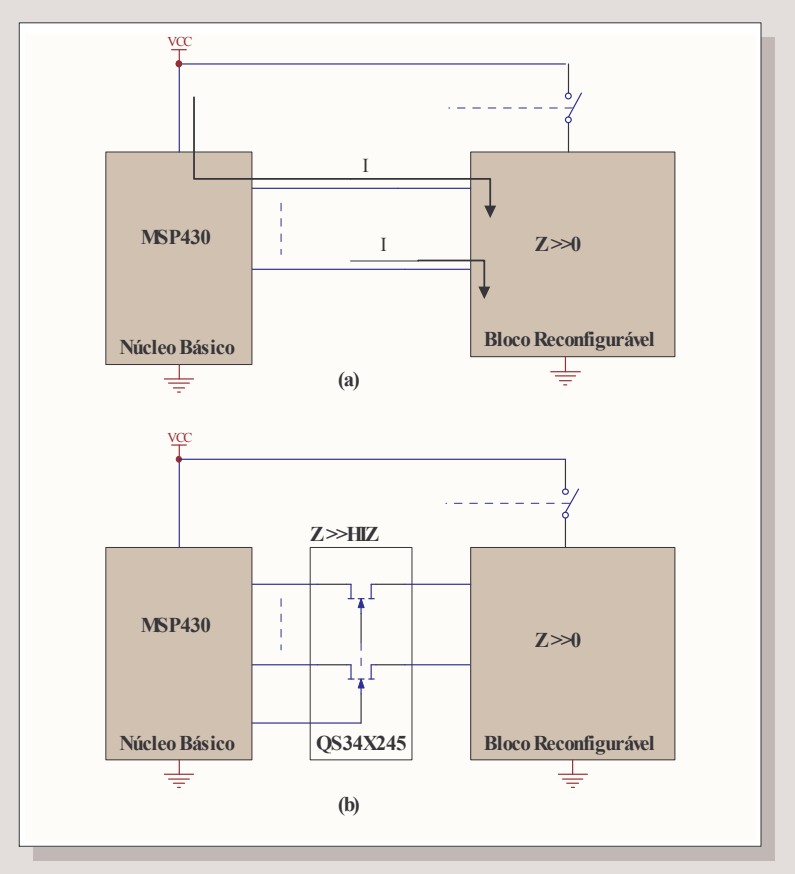



Figura 4.4: Interconexão dos sinais elétricos do núcleo básico e bloco reconfigurável.
(a) A corrente (I) flui dos terminais do MSP430 para o bloco reconfigurável.
(b) As chaves FET impedem o dreno de corrente drenada.

- o barramento serial formado pelas linhas LinA e LinB permite realizar diferentes interconexões entre as interfaces seriais do rádio CC1000, do MSP430, do driver RS232 (MAX3243) e da FPGA. A combinação destas interfaces pode ser configurada dinamicamente por *software*. O MSP430 utiliza o CI 74LVC273 (8 x Flip-Flops) para armazenar a configuração das chaves analógicas que conectam os sinais destas interfaces ao barramento serial. A Figura 4.5 mostra de que forma as chaves analógicas conectam os sinais de TX e RX de cada interface. A Tabela 4.2 apresenta a programação destas chaves e as interconexões que são permitidas;

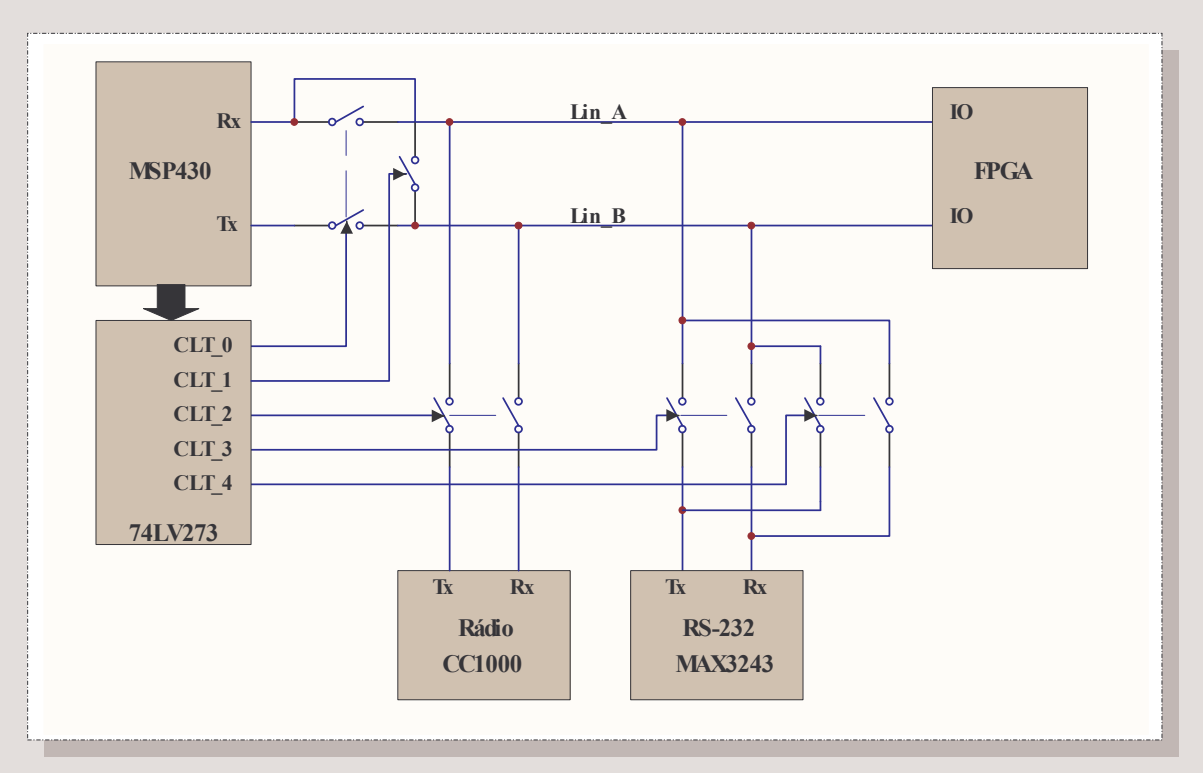



Figura 4.5: Detalhe da estrutura do barramento serial interno do RANS-300.

| Sinais de Controle (74LVS273) | | | | | Interfaces Conectadas | MSP430 - RX |
|---|---|---|---|---|---|---|
| CS4 | CS3 | CS2 | CS1 | CS0 | | |
| 0 | 0 | 0 | 0 | 0 | Todas Desconectadas | Não |
| 0 | 0 | 1 | 0 | 1 | Rádio CC1000 & MSP430 | Não |
| 0 | 0 | 1 | 0 | 0 | Rádio CC1000 & FPGA | Não |
| 0 | 0 | 1 | 1 | 0 | Rádio CC1000 & FPGA | Monitorando |
| 1 | 0 | 1 | 0 | 0 | Rádio CC1000 & MAX 3243 | Não |
| 1 | 0 | 1 | 1 | 0 | Rádio CC1000 & MAX 3243 | Monitorando |
| 0 | 0 | 0 | 0 | 1 | MSP430 & FPGA | Não |
| 0 | 1 | 0 | 0 | 1 | MSP430 & MAX 3243 | Não |
| 0 | 1 | 0 | 0 | 0 | FPGA & MAX 3243 | Não |
| 0 | 0 | 0 | 1 | 0 | FPGA & MAX 3243 | Monitorando |

Tabela 4.2: Configurações das interconexões do barramento serial do RANS-300.

- o oscilador fornece a freqüência de *clock* padrão de 14.7456MHz para a geração das taxas de transferência (*baud rates*) especificadas pela comunicação serial RS-232 (de 300 a 115200bps);

- a memória EEPROM (NM93CS56) possibilita o armazenamento de dados de configuração inicial e de uso das aplicações;
- o LM70 é um sensor destinado a informar a medida de temperatura do ambiente em que o nó sensor opera;
- o conector de expansão permite cascatear diversos módulos externos que são destinados aos sensores e controlados pelo MSP430;
- o conector JTAG possibilita a conexão externa de módulo auxiliar JTAG/PC para configurar,depurar e programar o microcontrolador MSP430;
- o conector LCD possibilita a conexão e o controle de um display de cristal líqüido de 16x1 (16 caracteres x 1 coluna) ou 16x2 para auxiliar na depuração e desenvolvimento de aplicações;
- os LEDs bicolores são para uso genérico e podem ser utilizados para auxiliar na depuração e desenvolvimento das aplicações.

### 4.3.2. Bloco Reconfigurável

O bloco reconfigurável adiciona ao núcleo básico diversos elementos dinâmicos e reconfiguráveis de *hardware* com o objetivo de auxiliar no tratamento e desenvolvimento de aplicações especiais em redes de sensores. Estes recursos, entretanto, consumem dezenas de vezes mais potência que os oferecidos pelo núcleo básico e, por isso, sua utilização deve ser bem gerenciada.

Para a implementação deste bloco, adotou-se uma FPGA Xilinx como elemento central da arquitetura reconfigurável aproveitando a versatilidade e performance operacional oferecida por este componente. A FPGA permite obter soluções eficientes para aplicações que necessitam realizar processamento intensivo e com tempo de resposta crítico que podem ser melhores atendidas por implementações dedicadas de *hardware*. Estas características poderão ser exploradas pelas aplicações em RSSFs, uma vez que, esta arquitetura disponibiliza estes recursos. O carregamento remoto de vetores de bits de configuração, denominados por

*bitstreams*, permitirão criar estas unidades de processamento dedicadas ou circuitos auxiliares com extraordinário potencial computacional.

O bloco reconfigurável conta ainda com outros elementos de *hardware* como: memória RAM, FLASH e uma unidade de acesso para cartão CompactFlash. Estes elementos podem juntamente com a FPGA formar um subsistema de processamento paralelo ao MSP430, através da implementação de outro processador ou coprocessador na FPGA (conhecidos como *softcores)*.

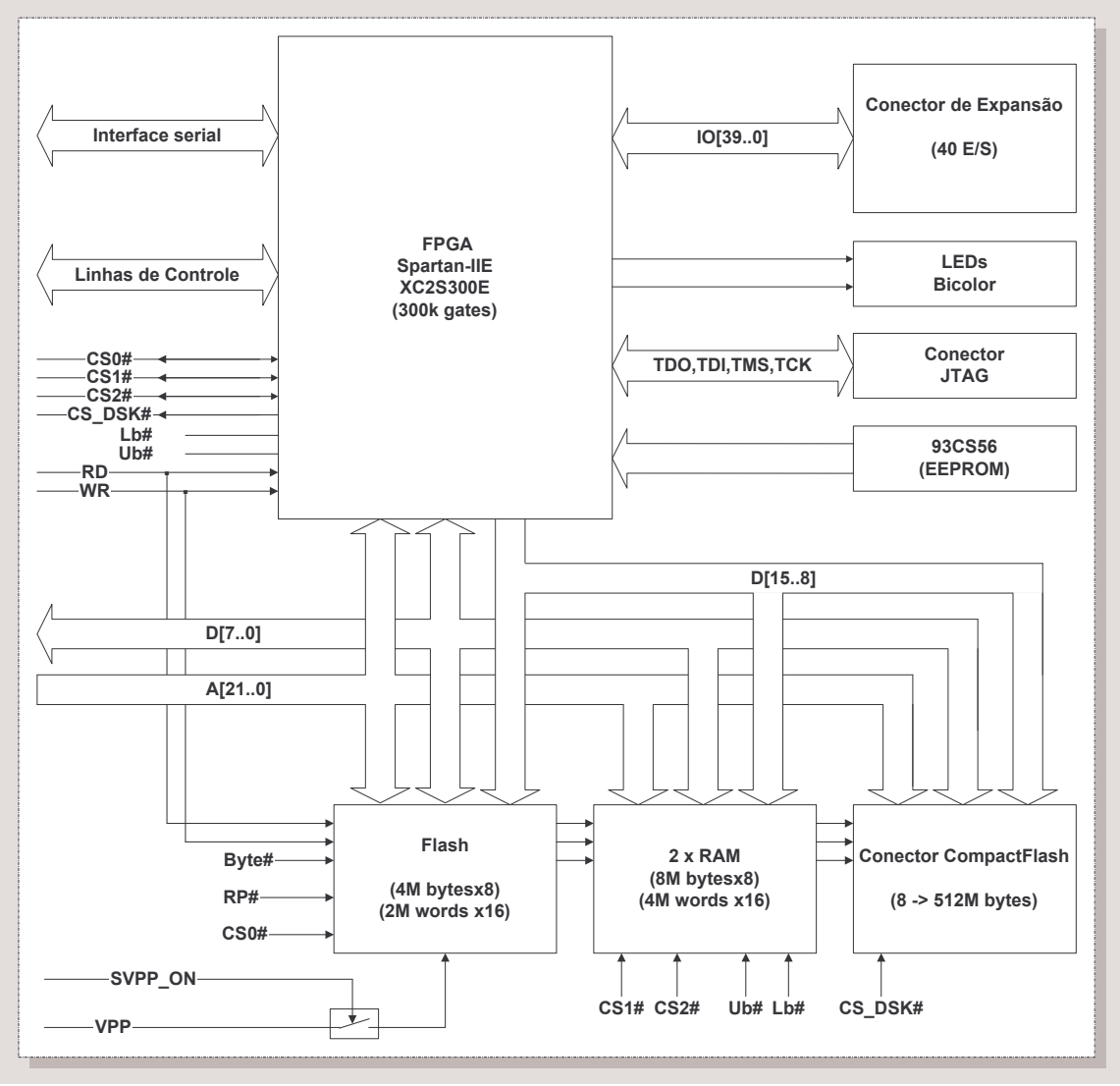



Figura 4.6: Diagrama de blocos da arquitetura do bloco reconfigurável.

A Figura 4.6 ilustra a arquitetura do bloco reconfigurável que utilizou os seguintes componentes para a sua implementação:

- CI XC2S300E. FPGA da família Spartan-IIE como capacidade de 300kgates;
- CI E28F320S3. Memória Flash de 32Mbits configurada para permitir acessos com comprimento de barramento de 8 ou 16bits;
- CI TC55W800. Memória RAM estática de 8Mbits utilizada para formar um banco de 16Mbits podendo ser configurada para permitir acessos com comprimento de barramento de 8 ou 16bits;
- conector para cartões de interface CompactFlash tipo I de uso geral;
- conector de 50 pinos para módulo de expansão controlado pela FPGA;
- soquete 8 pinos para instalação opcional da memória PROM 17LV002 [61] para carga de *bitstream* da FPGA;
- LED bicolores;
- conector Header 2x7, passo 2,0mm;

Ainda com relação à Figura 4.6 podemos observar que:

- os terminais de entrada e saída da FPGA foram interligados aos barramentos paralelos gerados pelo MSP430 do núcleo básico, aos sinais de controle de acesso e de habilitação das memórias RAM, Flash e do cartão CompactFlash. Estes componentes podem ser controlados tanto pelo MSP430 do núcleo básico como pela FPGA;
- o sinal SVPP_ON controla a ativação de uma chave FET para efetuar o fornecimento da tensão de alimentação (VPP) que habilita a programação da memória Flash. Na ausência desta tensão a memória Flash não aceita que sejam realizadas modificações nos dados armazenados internamente, ficando bloqueada para as operações de apagamento e escrita de dados;
- o conector JTAG possibilita a conexão externa de um emulador e programador para controlar a interface JTAG da FPGA, para auxiliar no desenvolvimento de aplicações em laboratório;

- o conector de expansão possibilita a conexão de diversos módulos cascateados a fim de adicionar sensores específicos, quando à aplicação exigir que o tratamento dos dados seja implementado na FPGA ou para adicionar outros circuitos eletrônicos complementares;

- foi previsto um soquete para permitir a instalação de uma memória PROM dedicada para configuração da FPGA. Esta memória necessita ser previamente gravada e pode ser utilizada quando for necessário implementar uma aplicação fixa na FPGA. Este recurso é útil para efetuar teste e verificação formal em laboratório. No caso de utilização em redes de sensores, pode-se prever a instalação opcional de uma memória EEPROM no seu lugar. Este tipo de memória permite ser reprogramada pelo MSP430 do núcleo básico. Entretanto, o processo preferencial de carga e armazenamento da configuração da FPGA deverá utilizar os recursos de armazenamento da memória RAM, Flash e do cartão CompactFlash;

- foi previsto leds bicolores para auxiliar na depuração das aplicações que são executadas na FPGA; A FPGA também pode acessar a interface para o display de cristal liquido prevista no bloco núcleo básico, uma vez que, o barramento de controle é comum aos dois blocos. Deve-se, neste caso, tratar estas operações de forma excludentes.

**Modos de carga de *bitstream* para a FPGA**

A FPGA XC2S300E ajusta-se ao modo de carga de *bitstream* de acordo com o nível lógico definido nos seus terminais M1, M2 e M3. No caso especifico deste nó sensor, esta programação é feita por *hardware* (instalando-se ou removendo-se resistores de pull-up ou pull-down nos terminais M1, M2 e M3).

Na elaboração da arquitetura do RANS-300, o MSP430 foi definido como sendo responsável pelo procedimento de carga do *bitstream* para a FPGA. A definição deste modo de carga deverá ser previamente estabelecida, para que a aplicação de *software* executada no microcontrolador possa controlar corretamente a configuração da FPGA. Os modos de carga que o MSP430 pode efetuar são: serial escravo, JTAG e paralelo escravo.

**Aplicações do Bloco Reconfigurável**

O comprimento do vetor de bits ou *bitstream* para configurar todas as interconexões internas da FPGA XC2S300E é de 1.875.684 x 1bit, ou 234.456 k x 8 bits se convertido para byte. Este é o tamanho mínimo requerido de memória para se efetuar o armazenamento de um único *bitstream*, e, podemos concluir facilmente, que a maioria dos microcontroladores atualmente disponíveis não dispõem deste recurso internamente. O MSP430F149 utilizado neste desenvolvimento, por exemplo, possui a capacidade de memória RAM interna de 2 kbytes.

Uma alternativa para realizar o armazenamento destes dados é utilizar áreas temporárias da memória RAM ou blocos permanentes da memória Flash ou do cartão CompactFlash do bloco reconfigurável. O mesmo procedimento deverá ser adotado no caso de transferências de arquivos relativos a outros dados como: código binário que será executado em processadores do tipo *Softcore* em FPGA, volumes de amostras de sensores, etc.

**Exemplo 1: Transferência remota de arquivo de *bitstream.***

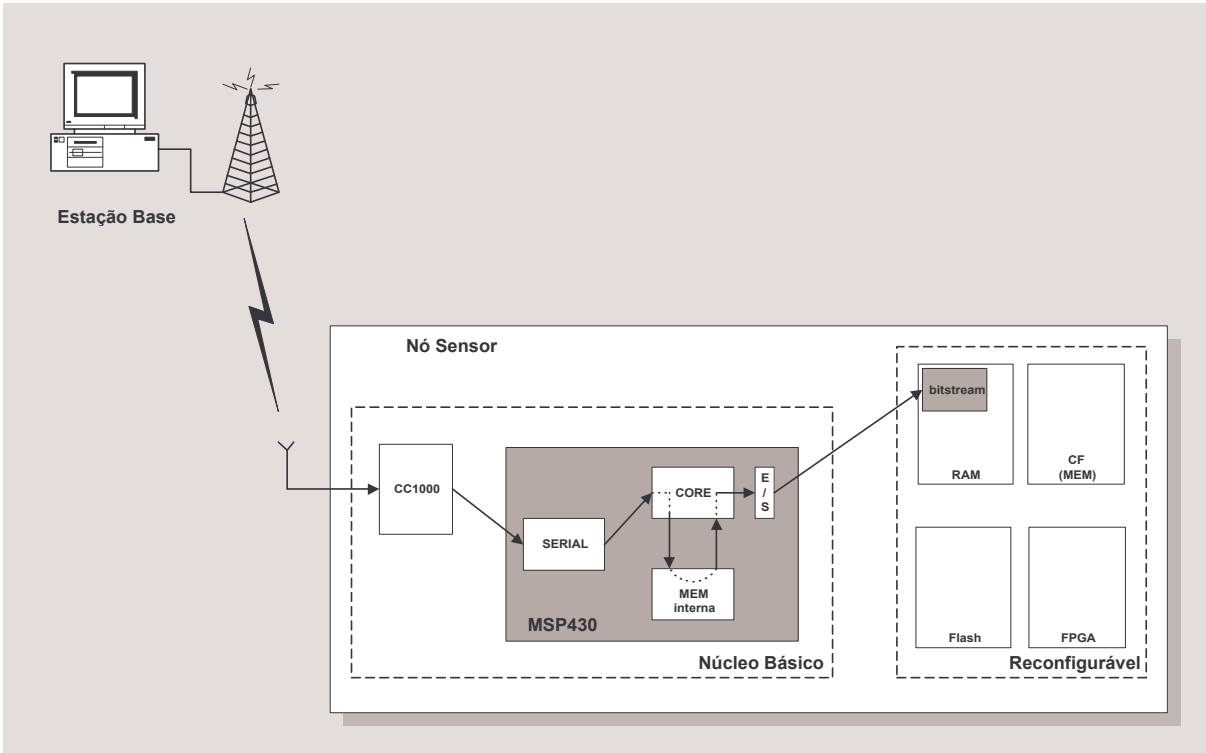



Figura 4.7: Aplicação de transferência remota de *bitstreams*.

Neste exemplo, a estação base comunica-se com o nó sensor e envia o arquivo de *bitstream* utilizando o enlace de rádio provido pelo CC1000. Os dados recebidos neste enlace são transferidos serialmente para um buffer circular temporário implementado na memória interna do MSP430. Por último, os dados armazenados no buffer são transferidos para a memória RAM via barramento paralelo emulado pelo MSP430.

**Exemplo 2: Transferência de arquivo *bitstream* da RAM para a memória Flash ou CF.**

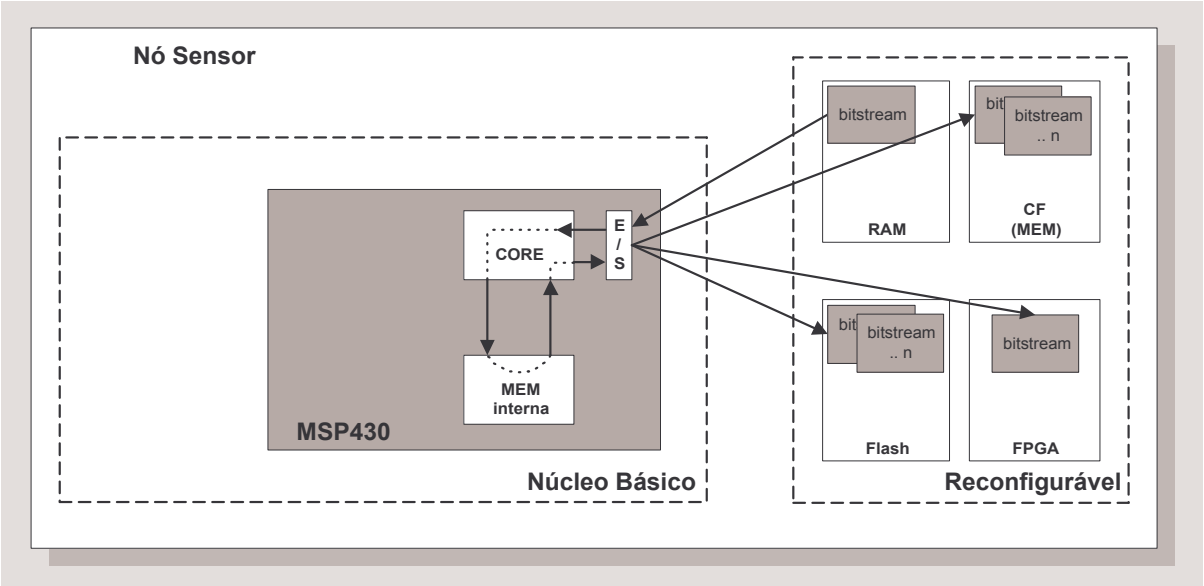



Figura 4.8: Aplicações transferências internas de *bitstreams.*

O arquivo de *bitstream* transferido e armazenado na RAM (forma descrita no exemplo anterior) pode ser transferido para a memória Flash ou para o CF caso seja necessário efetuar o seu armazenamento permanente. Este procedimento é ideal para minimizar as transferências de *bitstream* de uso freqüente pelas aplicações através da rede sem fio. Assim, diversas soluções de algoritmos e implementações dedicadas para RSSFs e baseadas em FPGA podem ser carregadas de acordo com o contexto de cada aplicação.

**Exemplo 3: Armazenamento de dados coletados.**

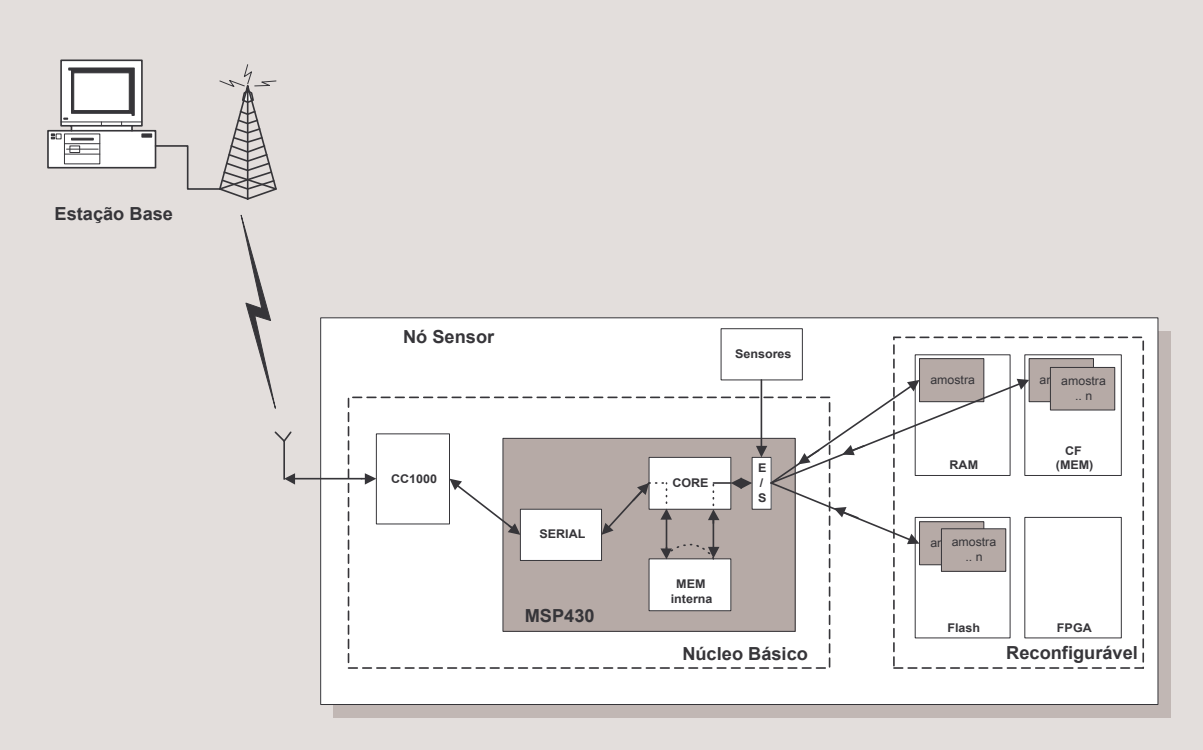



Figura 4.9: Aplicação de armazenamento de dados coletados.

Neste exemplo, o MSP430 acessa as informações do módulo de sensores e utiliza os recursos do bloco reconfigurável para efetuar o armazenamento temporário ou permanente dos dados. Este procedimento pode proporcionar, por exemplo, que eventos em determinados momentos sejam avaliados com uma maior taxa de amostragem que exigiriam uma maior área de armazenamento de amostras.

Uma outra aplicação é atuar como elemento concentrador de dados da RSSF (líder de *cluster*). Os demais nós podem descarregar seus dados para serem armazenados ou tratados pelo RANS-300, e, posteriormente, enviados para a estação base.

**Exemplo 4: Utilizando Interface 802.11.**

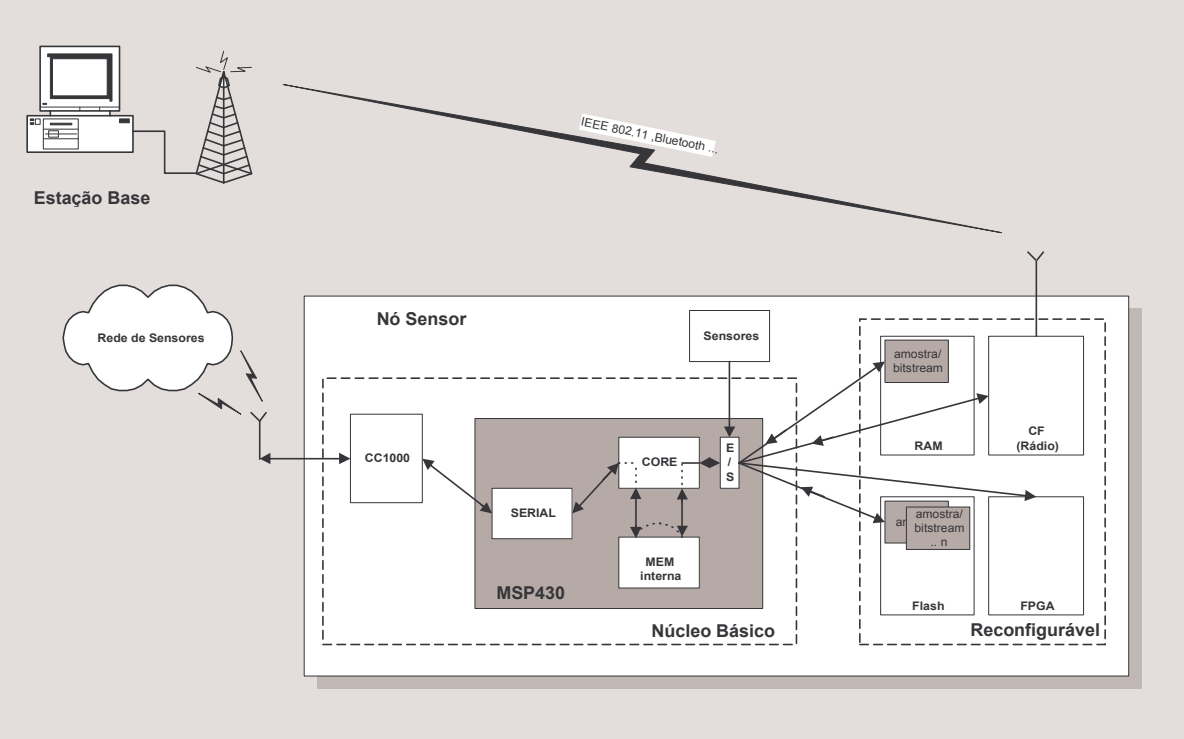



Figura 4.10: Aplicação baseada em interface IEEE 802.11.

A interface CompacFlash permite instalar um cartão de interface padrão IEEE 802.11 (*Wireless Ethernet* de 11 Mbit/s) implementando um elemento concentrador com capacidade de oferecer uma alta taxa de transmissão de dados para a estação base. Os dados da rede de sensores são recebidos pelo rádio CC1000, armazenados, tratados e enviados para a estação base através da interface IEEE 802.11 em rajadas.

Uma outra aplicação imaginada para este tipo de configuração é permitir transmitir imagens coletadas por câmaras sensores. As imagens adquiridas, por exemplo, podem sofrer compactação efetuada por algoritmos executados em FPGA antes de serem enviadas para a estação base através da interface de 11 Mbits/s.

**Exemplo 5: Subsistema de Processamento.**

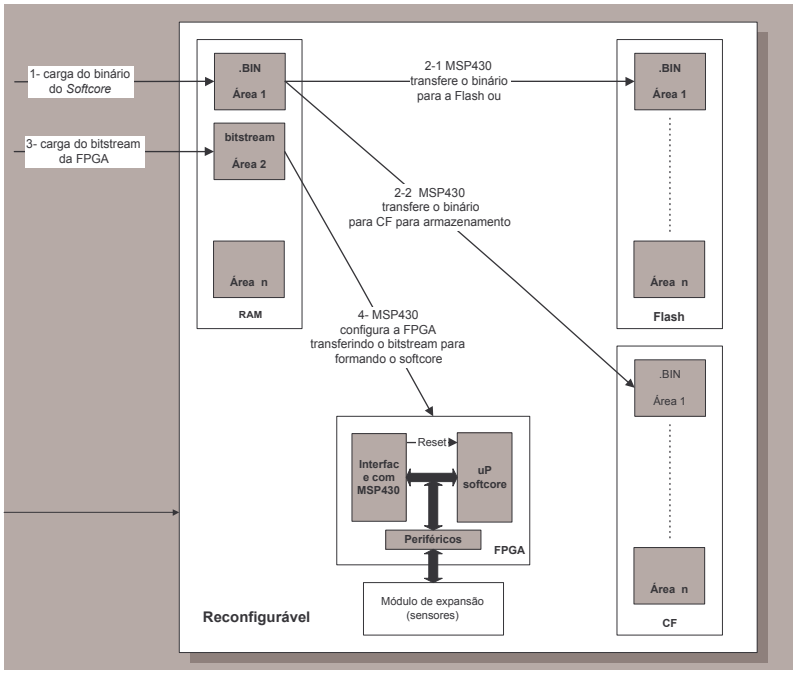



Figura 4.11: Aplicação de subsistema de processamento.

Neste exemplo, o bloco reconfigurável é utilizado para formar um subsistema de processamento complementar ao do núcleo básico. O conceito dos *softcores* (processadores implementados por *software* em FPGA) oferece uma excelente alternativa para controlar e monitorar eventos dedicados, eliminando a carga do processador do núcleo básico. Estes processadores são usualmente implementados utilizando linguagens de descrição de *hardware* do tipo VHDL ou Verilog, o que garante a sua portabilidade em uma variedade de aplicações.

Diversos tipos de processadores podem ser sintetizados na FPGA XC2S300E do RANS-300, como por exemplo, os tradicionais 8051, Z80 e PIC16xx, bem como, os MicroBlaze e PicoBlaze de propriedade da Xilinx. Outros processadores podem ser desenvolvidos para atender a requisitos específicos e dedicados de performance e tamanho de lógica.

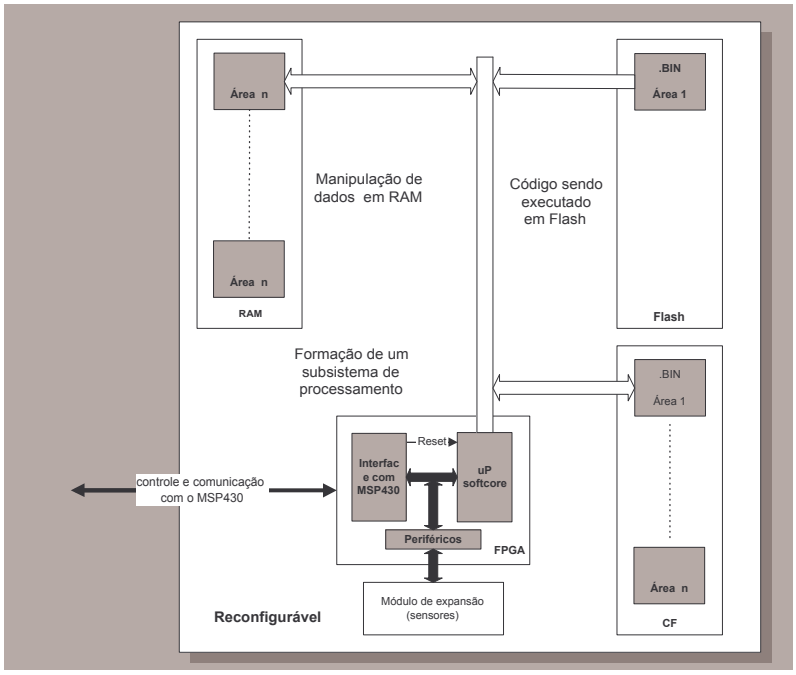



Figura 4.12: Aplicação baseada em softcores.

A idéia básica é que o *bitstream* que implementa estes processadores e seus periféricos na FPGA seja transferido de acordo como a necessidade de processamento da aplicação.

O código binário que será executado pelos processadores de *software* deve ser transferido para uma área de memória de programa em RAM ou Flash. A área de dados pode utilizar os blocos de memória interna da FPGA ou a memória RAM do bloco reconfigurável.

O controle do sinal de *Reset* que libera a execução do programa destes processadores, neste caso, é feito pelo MSP430 do núcleo básico. Quando este sinal é liberado, forma-se o subsistema de processamento. Os barramentos de acesso às memórias são formados e algumas interfaces de comunicação permitem que os processadores do núcleo básico e do processador implementado por *software* permutarem dados entre si.

### 4.3.3. Fonte de Alimentação

O bloco *Fonte de Alimentação* é responsável pela geração de todas as tensões internas utilizadas pelos componentes eletrônicos do circuito do RANS-300 e foi idealizada com a finalidade de:

- proporcionar a redução das perdas decorrentes do uso de reguladores lineares destinados à conversão de tensões e, conseqüentemente, obter um melhor aproveitamento da energia da bateria;

- permitir estender a faixa da tensão de entrada para possibilitar a utilização de um número maior de células de bateria aumentando a energia resultante disponível. Ex: utilizar 3 x 1,5V/ 1500mA/h baterias tipo AA;

- permitir uma regulação de tensão para o circuito, eliminando os danos causados por transientes e oscilações da tensão nominal de entrada. Esta característica favorece o emprego de fontes alternativas de energia como: geradores, painéis solares, etc.

- proporcionar a medição de consumo sem a utilização de *shunt série* de corrente. Estes elementos provocam perdas de energia e são inadequados e imprecisos quando o circuito sofre grandes variações de corrente;

- facilitar a comutação das tensões de alimentação destinadas ao bloco reconfigurável.

| Principais Componentes | Faixa de Alimentação (V) |
|---|---|
| MSP430 | 1,8 a 3.6 |
| CC1000 | 2.1 a 3.6 |
| XC2S300E (VCCINT)<br>(VCCO) | 1.8 a 2.0<br>2.0 a 4.0 |
| QS34X245 | 2.0 a 7.0 (*) |
| CompactFlash | 3.0 a 5.5 |
| TC55W800 | 2.0 a 3.3 |
| E28F320S3 | 2.7 a 3.6 |
| MAX 3243 | 3.0 a 3.6 |

(*) Vcc (min) requerido = 1V + Vcc do MSP430

Tabela 4.3: Faixa de variação de alimentação dos componentes eletrônicos adotados.

A Tabela 4.3 mostra as faixas das tensões de alimentação permitida pelos principais componentes eletrônicos utilizados nesta implementação. Estas faixas definiram que a fonte deveria prever três níveis de tensões de alimentação da seguinte forma:

- o primeiro para atender o requisito do *Core* da FPGA (Vccint) em 1.8V;

- o segundo para atender o requisito das chaves FET do QS34X245 em 5V (mínimo de 1V > Vcc do MSP430);

- o terceiro para atender os demais componentes do circuito em 3.3V.

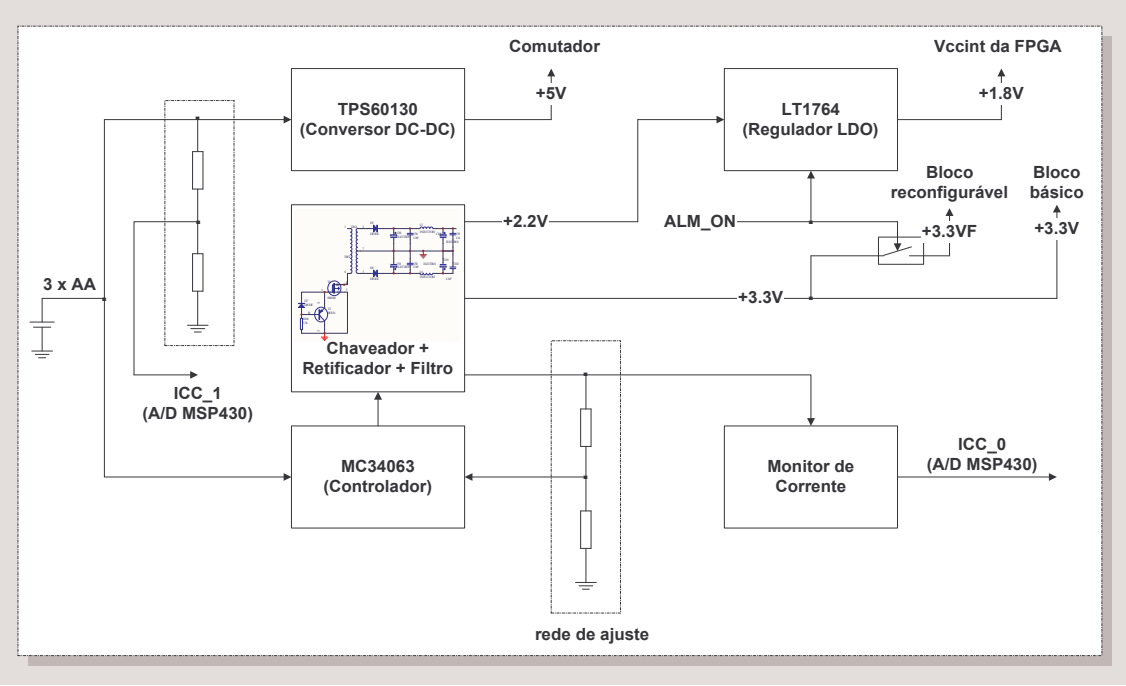



Figura 4.13: Diagrama de blocos da fonte de alimentação do RANS-300.

Com relação à Figura 4.13 podemos ilustrar que:

- a fonte utiliza um regulador chaveado, o MC32063, para implementar um conversor dc-dc do tipo flyback com saídas de +2.2V e +3.3V;

- a regulação das tensões é obtida através da rede de ajuste localizada em uma saída de amostragem dedicada do bloco chaveador;

- a variação de tensão da saída de amostragem é função da variação da carga nas alimentações de +2.2V e +3.3V. O bloco monitor de consumo trata estas variações e produz na saída ICC_0 uma tensão proporcional ao consumo de corrente e com nível elétrico adequado para que seja enviada a uma das entradas do conversor ADC do MSP430;

- o TPS60130 (*Charge Pump*) converte a tensão de entrada e produz uma saída regulada de +5V requerida para alimentar as chaves FET do comutador de barramentos QS34X245;

- a tensão de alimentação de 3.3V é utilizada para alimentar tanto os componentes do bloco núcleo básico como os do bloco reconfigurável. No caso especifico do bloco reconfigurável, a tensão de +3.3VF é fornecida através de uma chave implementada com transistor FET. O estado operacional desta chave pode ser controlado pelo *software* aplicativo que é executado no microcontrolador MSP430 utilizando o sinal ALM_ON para ativar/desativar o fornecimento de energia para este bloco;

- A tensão de +1.8V é gerada por um regulador LDO (*Low-Drop Output*) que regula a tensão de +2.2V da saída do enrolamento dedicado do transformador de potência.

### 4.3.4. Conectores de Expansão

O RANS-300 disponibiliza dois conectores de expansão, um deles localizado no núcleo básico e o outro no bloco reconfigurável, para permitir a conexão de módulos de sensores externos. No caso do conector do núcleo básico, os sinais disponibilizados são terminais de entrada e saídas do MSP430 que efetuam o controle e o acesso às informações provenientes dos sensores implementados no módulo auxiliar.

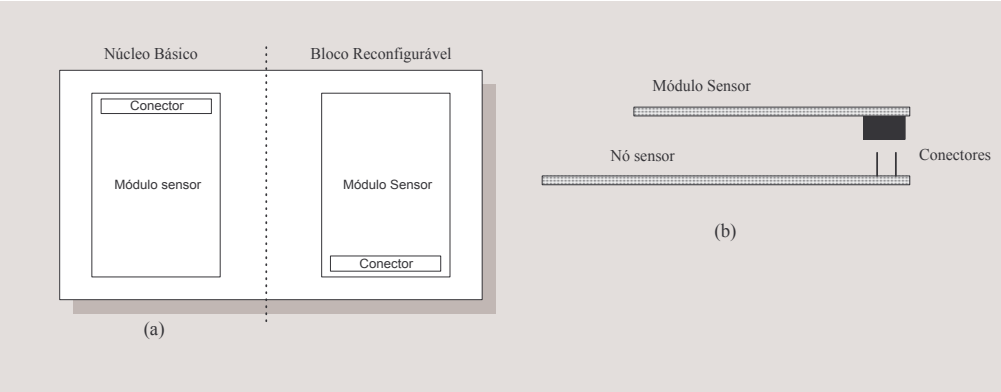



Figura 4.14: Detalhe da conexão de módulos de expansão.

No caso do bloco reconfigurável, os sinais do conector estão conectados a um conjunto de terminais de entrada de saída da FPGA, possibilitando que as informações dos sensores sejam tratadas por soluções de implementação baseadas em *hardware*. As Figuras C1 e C2 mostram os sinais previstos em cada um destes conectores.

### 4.4. Consumo de Potência

Em circuitos digitais, a freqüência do sinal de *clock* e o nível da tensão de alimentação são parâmetros que influenciam, significativamente, no consumo de potência. Qualquer dispositivo eletrônico pode ter o seu consumo de potência representado pelas componentes:

$$I(cc) = I(p) + I(v) \qquad (1)$$

Onde:

- I(p) representa a componente fixa da corrente requerida pela polarização das estruturas eletrônicas internas e pela corrente de carga das terminações externas dos sinais;

- I(v) representa a componente de corrente consumida pela carga e pelas capacitâncias internas em condições de operação dinâmica – decorrentes das transições de sinais provocados pelo *clock*. Esta componente representa a parcela mais significativa do consumo e também a mais complicada de se determinar, devido ao grande número de variáveis envolvidas.

Muitos dispositivos eletrônicos permitem controlar seu modo de funcionamento através da configuração de parâmetros relativos a ativação ou desativação de funções, periféricos e interfaces internas e de alterar a taxa do *clock* de forma dinâmica. Estas funcionalidades são implementadas buscando minimizar o impacto representado pela componente I(v) no consumo de potência.

No desenvolvimento de dispositivos nós sensores em que a comunicação sem fio é baseada em rádio transceptor de RF, a transmissão é a parte que representa o maior consumo de potência e determina a sua autonomia de operação.

No caso especifico do bloco reconfigurável, a FPGA, é o componente que apresenta o maior consumo de potência. Este consumo é dependente da forma de como os elementos

lógicos internos são configurados, da ativação das interfaces de entrada e saída e da taxa do *clock* de operação. Para não comprometer a autonomia de funcionamento do nó sensor, a alimentação deste bloco é aplicada somente quando as suas funcionalidades são requeridas pela aplicação.

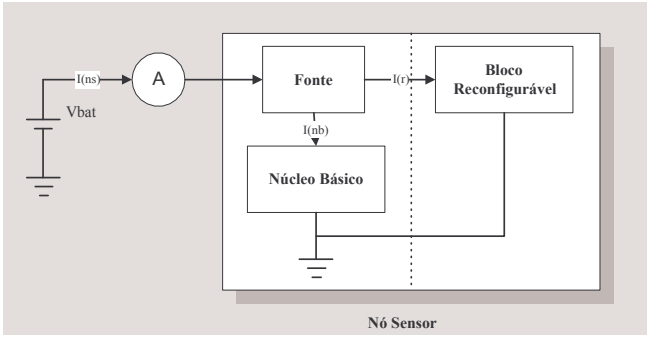



Figura 4.15: Diagrama simplificado da divisão do consumo de corrente do RANS-300.

O consumo global apresentado pelo dispositivo nó sensor pode ser decomposto também em duas componentes:

$$I(ns) = I(nb) + I(r) \qquad (2)$$

Onde:

- I(ns) representa o consumo total do nó sensor;
- I(nb) representa a componente da corrente devido ao funcionamento do bloco núcleo básico;
- I(r) representa a componente da corrente devido ao funcionamento do bloco reconfigurável.

No caso do bloco núcleo básico:

- o consumo do microcontrolador MSP430 pode ser obtido através das expressões:

$$I(AM) = I(AM)\ [1\ MHz] \times f(clock)\ [MHz] \qquad (3)$$

$$I(AM) = I(AM)\ [3V] + 175uA/V \times (VCC - 3\ V) \qquad (4)$$

Onde:

- I(AM) é a corrente consumida quando com todos os sinais de *clock* ativos. A expressão (3) mostra a relação do consumo em função da freqüência do *clock*, enquanto a expressão (4) representa a corrente em função da tensão de alimentação.

- o consumo de corrente do rádio CC1000 pode ser programado a fim de aumentar a sensibilidade e a potência de transmissão.

| Núcleo básico | Consumo de Corrente | |
|---|---|---|
| Componentes | Ativo | Estado Repouso |
| MSP430 (Xtal 32.768kHz) | 560uA modo AM | 3.9uA modo LPM3 |
| CC1000 | 7.4/9.6mA | 0.2uA |
| LM358 | 500uA | 500uA |
| LM70 | 260uA | 12uA |
| NM93CS56 | 500uA | 10uA |
| 74HCT4016 | 100uA | 5uA |
| NM74LVC273 | 500uA | 10uA |
| 2x QS34x245 | 0.5mA | 24uA |
| TPS60130 | 60uA | 0.05uA |
| MC34063 | 2.5mA | 2.5mA |
| MAX 3243 | 0.3mA | 1uA |
| Osc 14.5MHz | 2mA | 500uA |
| Consumo Máximo | 16.65mA | |
| Consumo Repouso | | 3.56mA |
| | | |
| Bloco Reconfigurável | Consumo de Corrente | |
| Componentes | Estado Ativo | Estado Repouso |
| XC2S300E | 500mA | 12mA |
| TC55W800 | 50mA | 1uA |
| E28F320J3A | 15mA | 500uA |
| CF- 128Mbytes | 50mA | 700uA |
| Consumo Máximo | 550mA | |
| Consumo Repouso | | 13.3mA |

**Nota 1**: CPU habilitada, todos os clocks internos habilitados.
**Nota 2:** CPU desabilitada, clocks MCLK e SMCLK desabilitados, DCO desabilitado e ACLK habilitado.
**Nota 3:** O consumo representado pela ocupação de 100% da lógica disponível.

Tabela 4.4: Levantamento de consumo do RANS-300.

A Tabela 4.4 apresenta uma estimativa de consumo dos blocos núcleo básico e reconfigurável, de acordo com os dados de catálogo dos seus componentes.

## 4.5. Levantamento de Custo

O levantamento de custo foi baseado na aquisição dos componentes e partes através do fornecedor Digikey [119] e apresentado nas Tabelas 4.5 a 4.7 para cada bloco funcional do RANS-300. O custo total verificado para cada protótipo foi de US$ 171.38.

| Quant. | Componentes do Núcleo básico | Custo em Dólar FOB (USA) |
|---|---|---|
| 1 | MSP430F149 | $10.00 |
| 1 | CC1000 | $9.00 |
| 1 | CMX-309FB 14.7456MHz | $1.24 |
| 1 | LM70 | $1.80 |
| 1 | 74LVS273 | $0.55 |
| 1 | 74HCT4016 | $0.60 |
| 1 | MAX3243I | $1.70 |
| 1 | AT93S56 | $0.26 |
| 1 | Cristal 32.768kHz | $0.13 |
| 1 | Conector Hirose 50 pinos | $2.80 |
| 1 | Componentes discretos | $2.00 |
| 1 | Área de PCB: 10% | $2.99 |
| | **Total:** | **$27.09** |

Tabela 4.5: Custos de componentes relativos à implementação do núcleo básico.

| Quant. | Componentes da Fonte | Custo em Dólar FOB (USA) |
|---|---|---|
| 1 | TPS60130 | $2.93 |
| 1 | LT1764 EQ-1.8 | $3.56 |
| 1 | LM358 | $0.45 |
| 2 | 75N05HD | $3.24 |
| 1 | Trafo Ferrite | $5.00 |
| 1 | Receptáculo para bateria | $0.74 |
| 1 | Componentes discretos | $2.00 |
| 1 | Área de PCB: 30% | $8.99 |
| | **Total:** | **$26.78** |

Tabela 4.6: Custos de componentes relativos à implementação da fonte de alimentação.

| Quant. | Componentes do bloco Reconfigurável | Custo em Dólar FOB (USA) |
|---|---|---|
| 1 | XC2S300E | $38.60 |
| 1 | E28F320J3A110 | $9.30 |
| 2 | TC55W800FT55M-ND | $18.00 |
| 1 | Cartão CompactFlash 128Mbytes | $19.99 |
| 1 | Conector CompactFlash | $8.11 |
| 1 | Conector Hirose 50 pinos | $2.80 |
| 2 | Qs34x245 | $2.80 |
| 1 | Área de PCB: 60% | $17.99 |
| | **Total:** | **$117.59** |

Tabela 4.7: Custos de componentes relativos à implementação do bloco reconfigurável.

## 4.6. Ambiente de Desenvolvimento

Para auxiliar no desenvolvimento da plataforma de *hardware* do RANS-300 foram utilizadas as seguintes ferramentas:

- Protel DXP [66]: ambiente de desenvolvimento para elaboração de esquemático eletrônico e detalhamento de placa de circuito impresso (biblioteca de componentes, roteador, produção de ART Master);
- DesignLab/MicroSim [80]: ambiente para simulação de circuitos digitais e analógicos;
- ModelSim XE II V5.6 [71]: ambiente de simulação lógica para implementações baseadas em linguagem de descrição de hardware – VHDL para FPGA Xilinx;
- Xilinx ISE 6 [58]: ambiente integrado com o ModelSim para desenvolvimento de aplicações baseadas nas FPGAs da Xilinx (editor, compilador, síntese lógica, geração de *bitstream*).

## 4.7. Decisões de Projeto

Para o desenvolvimento do *hardware* do RANS-300 foram consideradas algumas diretrizes de projeto no sentido de minimizar o reforço de implementação, de simplificar o processo de fabricação, a montagem e para otimizar os custos envolvidos na fase de prototipação.

### Seleção dos Componentes

A seleção de cada componente eletrônico baseou-se nos critérios técnicos e nos requisitos mínimos especificados na Seção 4.1, bem como nos critérios comerciais de aquisição de pequenas quantidades, frete, custo e prazo de entrega. Os principais elementos da arquitetura foram selecionados considerando os seguintes aspectos:

- o MSP430F149 da Texas Instruments é, atualmente, um dos mais populares microcontroladores empregados em projetos de equipamentos portáteis de medição e de uso geral que utilizam bateria como principal fonte de energia. Seu consumo de corrente é da ordem de micro-ampères (*ultra-low-power*) quando operando na faixa de freqüência de clock de 1 MHz, o que o coloca em destaque quando comparado aos outros microcontroladores similares como: ATmega128 da Atmel (utilizado no Mica2 Mote), ADUC834 da Analog Devices, 80C51 da Intel e Infineon, família HC05 e HC11 da Motorola, etc. Esta principal característica aliada a sua arquitetura RISC, aos seus recursos computacionais, periféricos, sua disponibilidade, a facilidade de aquisição e ao seu baixo custo fizeram do MSP430 a opção adequada para compor e implementar o controle do núcleo básico do RANS-300. Outro aspecto favorável, é que a Texas disponibiliza gratuitamente o ambiente de desenvolvimento (IAR *Embedded Workbench for MSP430*), o que permite implementar aplicações de *software* com baixo custo destinadas às RSSFs;

- o rádio CC1000 da Chipcon apresenta performance funcional superior aos equivalentes encontrados comercialmente. Para realizar a transmissão de dados, o CC1000 emprega a modulação FSK, o que proporciona uma maior economia de energia, característica essencial em se tratando de um sistema alimentado por bateria, além de oferecer uma maior imunidade a ruídos e, conseqüentemente, uma maior qualidade do enlace de comunicação. Seu barramento serial de controle adiciona flexibilidade operacional permitindo efetuar o controle de ganho e a configuração da

freqüência de transmissão/recepção, dos modos de baixo consumo e de outros parâmetros configuráveis, além de simplificar o interfaceamento com o microcontrolador;

- para implementar as funcionalidades previstas para o bloco reconfigurável, observando a quantidade mínima de lógica definida, foi selecionada a FPGA XC2S300E da Xilinx por apresentar a melhor relação custo/beneficio em relação ao componente equivalente EP1C6F256 da família Cyclone da Altera. Esta avaliação de custo foi realizada considerando a aquisição realizada através dos distribuidores Digikey e Arrow para a FPGA Xilinx (US$38.60) e para a FPGA Altera (US$40.60) respectivamente. Outro aspecto considerado, a exemplo do MSP430, foi a disponibilidade gratuita das ferramentas de desenvolvimento de aplicações (*ISE* e simulação *ModelSim*) oferecidas pela Xilinx;

- foram adotados materiais e componentes passivos comuns (resistores, capacitores, indutores, conectores), tendo sido especificados de acordo com as faixas de valores e tolerâncias padronizadas e disponibilizadas comercialmente. Também foi adotada a redução da variedade de componentes o que proporcionou uma simplificação do processo de aquisição e custo.

**Placa de Circuito Impresso - PCB**

Os aspectos gerais que orientaram o projeto e a aquisição da placa de circuito impresso do RANS-300 foram:

- o projeto foi elaborado tendo como pré-requisito a utilização de apenas duas camadas de cobre (camada inferior e superior) para realizar o roteamento dos sinais elétricos, descartando-se, portanto, processos de fabricação utilizando multicamadas ($\geq$ 4 camadas) o que agregaria um custo excessivo ao nó sensor;

- os componentes eletrônicos foram posicionados utilizando as duas faces da placa de circuito impresso o que proporcionou uma maior integração e redução física do dispositivo nó sensor;

- para a confecção das placas de circuito impresso foram levados em consideração os preços praticados por diversos fabricantes e seus respectivos lotes econômicos de

produção. Este processo de aquisição proporcionou uma melhor distribuição do custo fixo de fabricação na composição do custo unitário de cada dispositivo nó sensor.

O conjunto de regras de leiaute (Figura 4.16) adotado no projeto do PCB foi elaborado a partir de consultas técnicas realizadas aos fabricantes de placa de circuito impresso como as descritas abaixo:

- via de transposição com anel mínimo de 0.6 mm e furo de 0.3 mm de diâmetro;
- espessura mínima da trilha de sinal de 0.15 mm;
- espaçamento mínimo entre trilhas de sinais de 0.15 mm (*clearence*);
- placa em fibra de vidro FR4 com espessura de 1.6 mm.

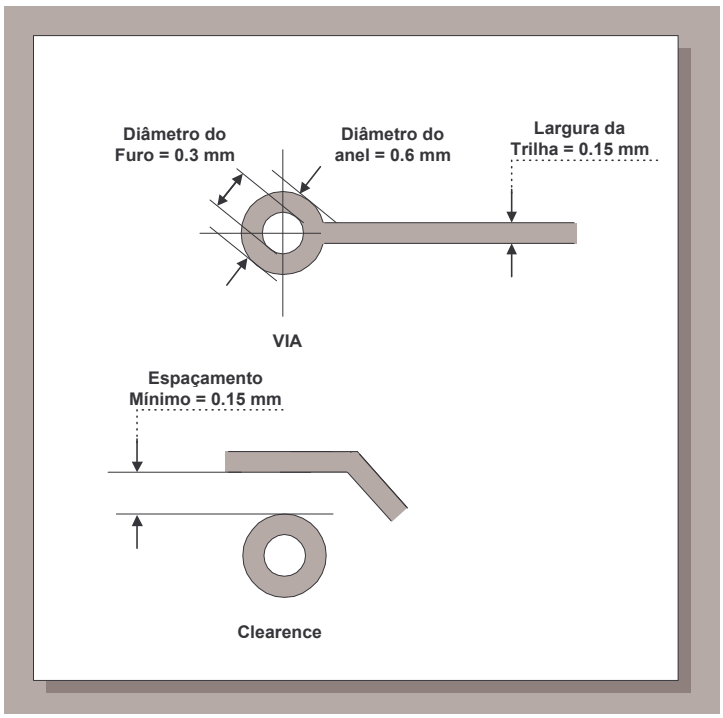



Figura 4.16: Regras de leiaute adotadas para o roteamento dos sinais do PCB.

# CAPÍTULO 5

# Validação e Resultados Obtidos

Os resultados que serão apresentados referem-se aos dados obtidos durante a realização dos testes elétricos e funcionais da plataforma de *hardware* desenvolvida para o RANS-300. Os testes foram estruturados e especificados no sentido de validar o modelo proposto, tornando a plataforma do RANS-300 apropriada e operacional para geração de trabalhos futuros.

## 5.1. Requisitos e Infra-Estrutura de Teste

Para auxiliar na depuração e validação da plataforma de *hardware* do RANS-300 foi necessário elaborar um programa de teste para ser carregado na memória interna do microcontrolador MSP430, com a finalidade de exercitar e controlar partes individuais do circuito. Este programa integra um conjunto de funções básicas de acesso (escritas em linguagem C), de configuração e controle dos periféricos do MSP430 e dos demais recursos de *hardware* do RANS-300.

O processo de depuração exigiu recursos laboratoriais, ferramentas para desenvolvimento de *software* e módulos adaptadores externos, destacando:

- módulo Xilinx Parallel Cable IV: este módulo implementa uma interface padrão JTAG para transferência de *bitstreams* para a FPGA (Figura F1(a));
- módulo de interface padrão JTAG: necessário para efetuar a transferência e gravação do programa executável na memória interna do MSP430F149 (Figura F1(b));
- módulo LCD: conectado na interface prevista pelo RANS-300 com a finalidade de permitir a exibição de mensagens de depuração (Figura F1(c));
- IAR Embedded Workbench [51]: ambiente para desenvolvimento de programas baseados nos microcontroladores da família MSP430 (compilador, simulador, etc);

- MSP430FET: aplicativo de depuração e programação da memória Flash interna do microcontrolador MSP430;

- Equipamentos de testes: multímetro, osciloscópio, analisador lógico, analisador de espectro de freqüência, fonte de alimentação, etc.

## 5.2. Validação da Plataforma *Hardware*

### Fonte de Alimentação

A fonte de alimentação do RANS-300 foi implementada empregando-se a topologia de conversor dc-dc do tipo "*fly-back*". A depuração do conversor consistiu na confecção e ajuste do transformador de saída, da definição dos valores dos componentes externos ao controlador MC34063 e da malha de ajuste de realimentação, a fim de obter as tensões nominais definidas no projeto eletrônico. Foram necessários diversos ensaios de variação da tensão de entrada, simulando a curva de descarga das baterias Ni-MH e Alkalina e situações de variação transitória de carga, devido à ativação e desativação do circuito do bloco reconfigurável. A eficiência obtida na conversão das tensões foi de 80%.

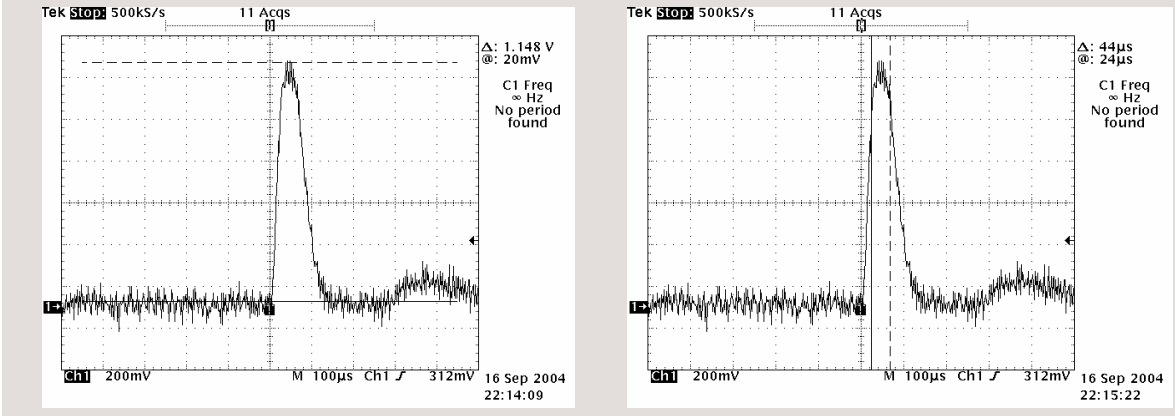



Figura 5.1: Surto de corrente durante a energização do bloco Reconfigurável.

A Figura 5.1 mostra o surto de corrente da ordem de 1.1A/44µs verificado durante o teste de energização do bloco reconfigurável. Este surto é decorrente da característica construtiva da FPGA Spartan-IIE da Xilinx e ocorre sempre que o bloco núcleo básico comanda a

ativação do bloco reconfigurável. A natureza desta corrente foi descrita na Seção 3.1 em requisitos de potência.

O consumo verificado devido a energização do bloco reconfigurável foi de 19 mA na condição das memórias Flash e o Cartão de 65Mbytes desabilitados e com a FPGA programada através da interface JTAG executando um aplicativo que emulava acessos seqüenciais de memória RAM, com temporização baseada no sinal de *clock* externo de 20MHz. A corrente drenada da fonte com o bloco reconfigurável desenergizado foi praticamente igual a zero.

**Núcleo Básico**

Os testes realizados para a validação do *hardware* do núcleo básico foram:

- carregamento do código executável para a memória interna do MSP430F149 via interface JTAG;
- emulação da interface de acesso e envio de caracteres para visualização no display LCD;
- emulação da interface serial de configuração dos registros internos do rádio CC1000;
- emulação da interface serial de acesso ao sensor de temperatura LM70;
- comando de ativação da alimentação do bloco reconfigurável pelo MSP430F149;
- ativação/desativação das chaves FET de isolação dos sinais de barramento entre o núcleo básico e o bloco reconfigurável.

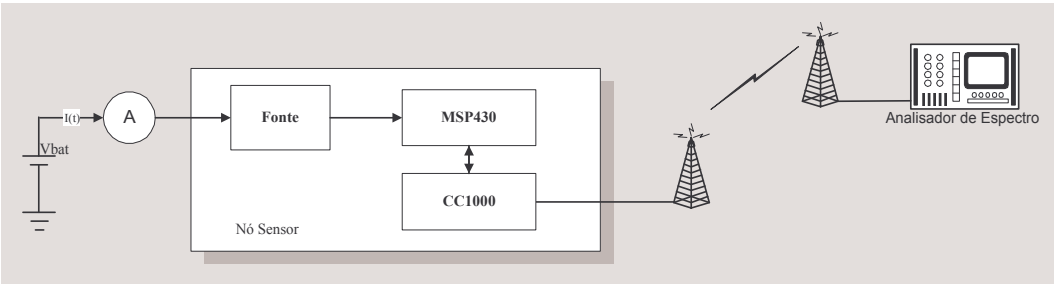



Figura 5.3: Detalhe do ambiente de teste do rádio CC1000.

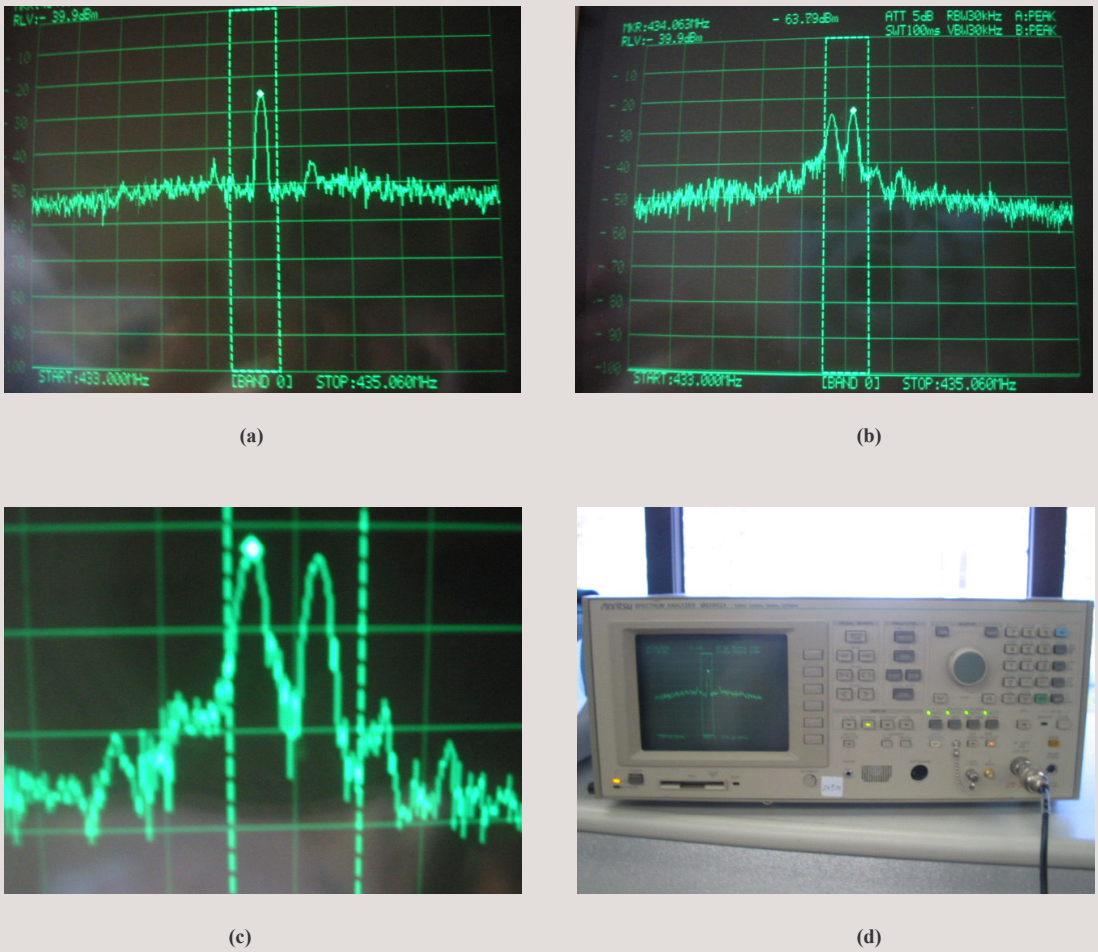



Figura 5.4: Sinal de transmissão do rádio CC1000.
a) transmissão de nível lógico ‘0’.
b) transmissão de ‘1s’e ‘0s’.
c) detalhe da modulação FSK.
d) analisador Anritsu MS 2802.

O rádio CC1000 foi configurado para trabalhar na faixa de freqüência de 433MHz e o sinal de transmissão foi monitorado por um analisador de espectro de freqüência da Anritsu. A Figura 5.4 mostra os detalhes dos sinais monitorados na situação de repouso, transmitindo nível lógico ‘0’ (apenas portadora), e da modulação FSK quando transmitindo ‘1s’ e ‘0s’ lógicos (freqüências *f1* e *f2)*.

**Bloco Reconfigurável**

Para validar a implementação do circuito da FPGA e dos barramentos externos de dados, endereços e controle das memórias RAM e Flash, foi desenvolvido um circuito Emulador de barramento em VHDL com a função de gerar os ciclos de acesso para estas memórias. Para a geração destes sinais foi utilizada a temporização especificada pela memória Flash por ser o componente que requer o maior tempo de acesso (150ns).

Para a implementação do Emulador foi criado um gerador seqüencial e ajustável de endereços, 19 bits para a memória RAM (A0-A18) e 22 bits para memória Flash (A0-A21), para permitir o endereçamento de toda a capacidade de memória disponível em cada tipo de dispositivo. Equivalente aos endereços, uma seqüência de 8 bits foi gerada para o barramento de dados, a fim de que cada posição de memória fosse preenchida com um valor diferente (valor anterior + 1). Assim, o dado varia de 00 a FF (H) e é zerado a cada 256 posições de endereços, conforme ilustrado na Figura 5.5.

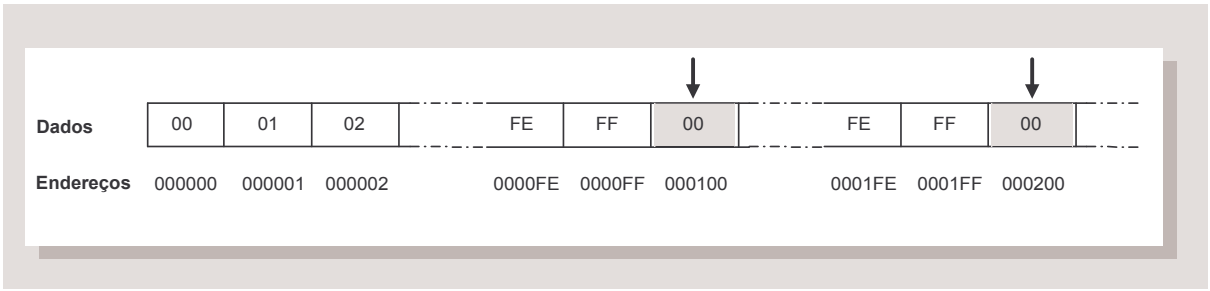



Figura 5.5: Seqüência gerada pelo Emulador de barramento.

O gerador de seqüência envia um evento de início (*start*) toda vez que o endereço é incrementado. Este sinal dispara uma máquina de estados para produzir a seqüência de eventos externos a FPGA, emulando o ciclo de barramento de acesso de memória com a temporização apropriada. Depois de iniciada, a máquina de estados envia um sinal de ocupado (*busy*) para que o gerador de seqüência aguarde a finalização de todo o ciclo de acesso de memória para gerar o próximo endereço. O teste é executado ciclicamente até que se

configure novamente a FPGA. A Figura 5.6 mostra detalhes da estrutura funcional do Emulador de barramento e o código em VHDL associado a esta implementação se encontra reproduzido no Apêndice G.

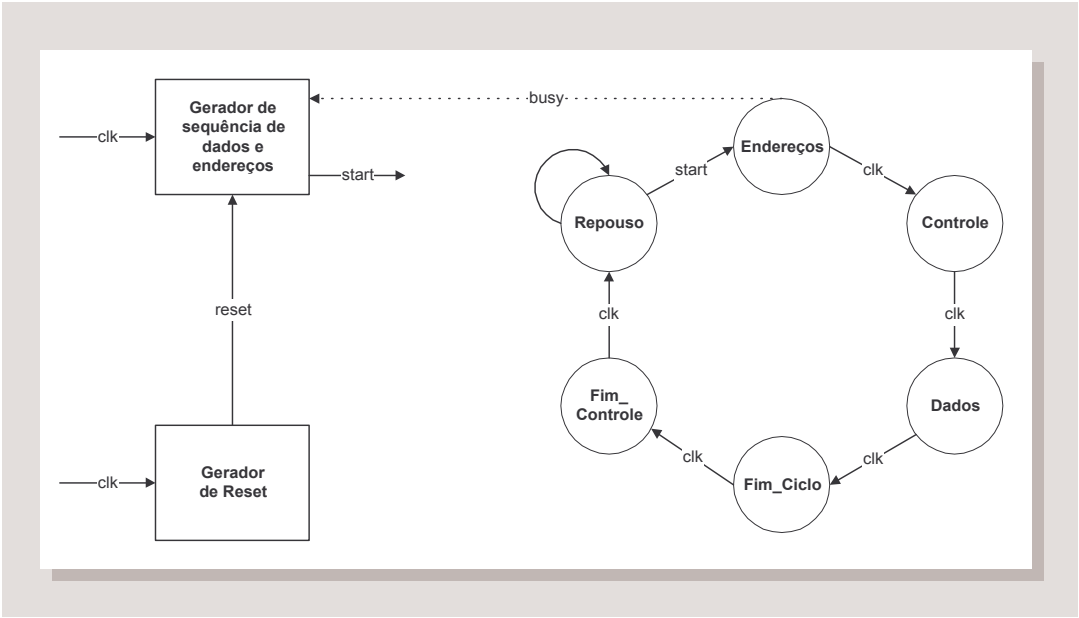



Figura 5.6: Diagrama Funcional do Emulador de barramento.

A Figura 5.7 mostra o resultado da simulação realizada no ambiente ModelSim relativo ao circuito lógico do Emulador que foi sintetizado pelo aplicativo ISE 6 da Xilinx. A Figura 5.8 mostra os sinais de seleção, escrita, linha de dado D0 e linha de endereço A1, monitorados nos pinos externos da FPGA utilizando um osciloscópio digital de quatro canais da Tektronics (TDS 644B). Como podem ser notados, os sinais elétricos mostrados no osciloscópio reproduzem a temporização prevista pela simulação.

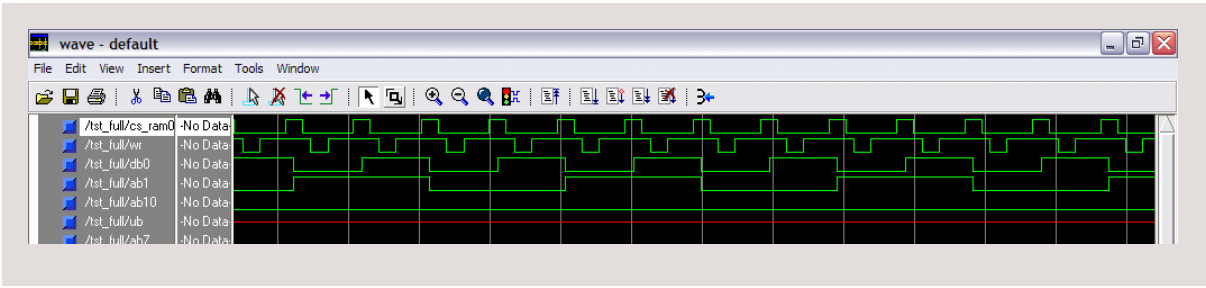



Figura 5.7: Detalhe dos sinais de seleção, escrita, linha D0 e A1 gerados na simulação.

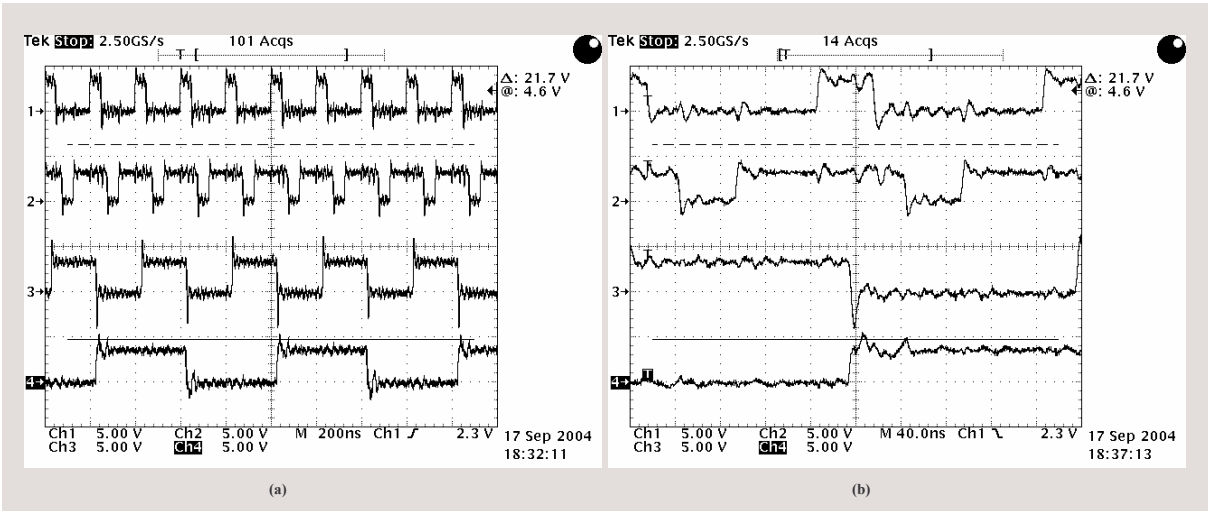



Figura 5.8: Detalhe dos sinais elétricos de seleção, escrita, linhas D0 e A1.

A alocação de recursos (contadores, flip-flops, interface de E/S, etc) que foram consumidos da FPGA para implementar o Emulador é apresentado de forma resumida da Figura G.3 do Apêndice. A implementação consumiu 1% do total de lógica disponível pela FPGA Spartan-IIE. A Figura 5.9 mostra um trecho de código VHDL do Emulador e o circuito lógico equivalente sintetizado pelo ISE 6.3 da Xilinx.

```
-------------------------------------------------
-- Arquitetura Gera
-------------------------------------------------

architecture gera of gera_teste is

signal sadress: unsigned(18 downto 0);
signal sdata: unsigned(7 downto 0);

begin

  increment: process (clock,start)
  variable dados_gerados: integer;

  begin
       if (start = '0')then
          dados_gerados:= 0;
          sdata<= "00000000";
          sadress<= "0000000000000000000";
          start_out<= '1';
   elsif (clock = '1' and clock'event) then
          if (busy = '0') then
       sadress <= sadress + 1;
            sdata <= sdata + 1;
                start_out<= '1';
            if (dados_gerados = 255) then
              sdata<= "00000000";
                   dados_gerados:= 0;
            else
              dados_gerados:= dados_gerados + 1;
                   sdata <= sdata + 1;
            end if;
          end if;
    end if;
  end process;

 adress <= std_logic_vector(sadress);
 data <=  std_logic_vector(sdata);

end gera;
```

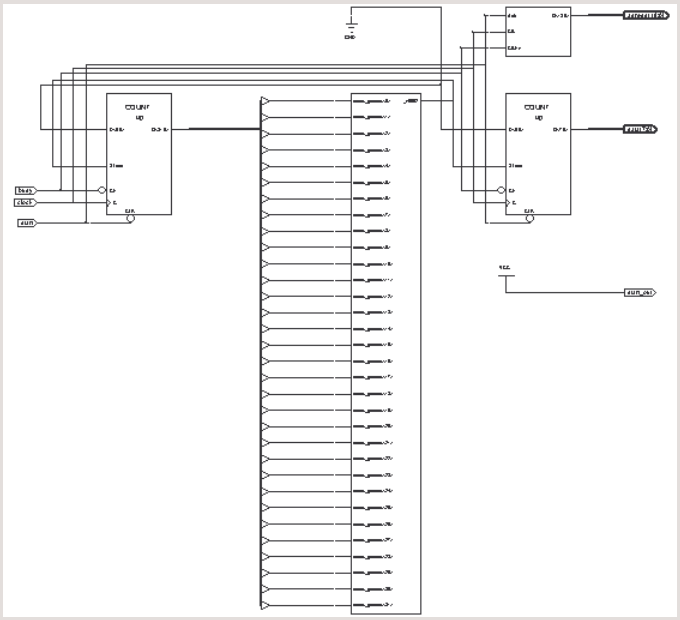


Figura 5.9: Detalhe de parte do código VDHL do Emulador
e o correspondente circuito digital sintetizado.

# CAPÍTULO 6

# Considerações Finais

## 6.1. Conclusões

O RANS-300 é um nó sensor que se diferencia, dentre a variedade de dispositivos desenvolvidos para a área de RSSF, pela simples razão de possibilitar soluções de alta performance computacional baseadas nos recursos oferecidos pelas arquiteturas reconfiguráveis de *hardware*. Esta característica aprimora e expande o horizonte de possibilidades para efetuar o monitoramento de sistemas baseados em RSSFs. O estágio tecnológico deste desenvolvimento é bastante atual e apresenta aspectos funcionais que servirão para o desenvolvimento de aplicações avançadas nesta área.

A característica de alocação sob demanda dos recursos computacionais do bloco reconfigurável provou ser viável a utilização de elementos de arquitetura reconfigurável de *hardware* em dispositivos nós sensores sem que a performance de consumo de potência fosse demasiadamente comprometida.

A funcionalidade oferecida pelo RANS-300 de permitir que sejam realizadas atualizações remotas de *bitstreams* flexibiliza e potencializa o monitoramento efetuado pelo bloco Reconfigurável. Sua característica adaptativa possibilita que sejam efetuados ajustes em função de determinadas condições de funcionamento e correções de falhas identificadas através de asserções [75] no código da aplicação.

A utilização do rádio CC1000 permite o interfuncionamento do RANS-300 com os dispositivos MICA2 Mote, MICAdot e o BEAN, possibilitando desenvolver aplicações integradas e complementares de monitoramento.

A corrente consumida devida à operação contínua do núcleo básico atende ao requisito de baixo consumo requerido para dispositivos nós sensores. De acordo com a avaliação de consumo da Seção 4.4, a transmissão de dados representa 57.65% do consumo máximo, sendo que, o percentual relativo ao microprocessador é de 0.03%.

Como esperado, a energização do bloco reconfigurável altera significativamente o consumo de potência do RANS-300. As aplicações previstas utilizando os recursos deste bloco devem considerar o custo adicional de energia requerida pela sua ativação e execução, o que representa uma perda substancial de autonomia de operação do dispositivo.

Uma grande quantidade de energia é consumida devido ao surto inicial de corrente que ocorre durante a fase de transição da alimentação da FPGA. Apesar de instantânea, a energia consumida por este evento é significativa e as aplicações devem considerar sempre o custo/benefício de se ativar/desativar este bloco. A corrente total consumida pela FPGA em operação pode atingir 500mA dependendo da ocupação dos seus elementos lógicos. Este valor representa uma ordem de grandeza de 33 vezes o valor da corrente consumida pelo bloco núcleo básico. Na aplicação do Emulador de barramento o consumo verificado foi de 19mA com uma ocupação de 1% da lógica total disponível.

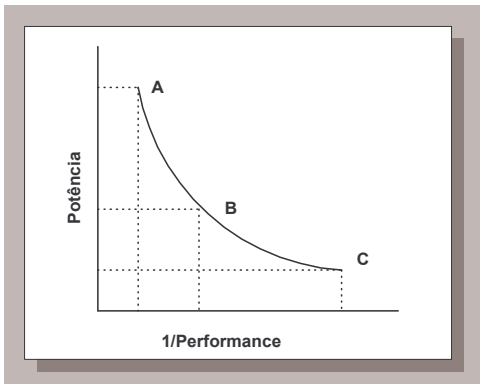



Figura 6.1: Relação Potência x Performance do RANS-300.

A Figura 6.1 ilustra o comportamento do consumo de potência em função da alocação dos recursos computacionais (processamento, memória e comunicação) do núcleo básico e bloco reconfigurável do RANS-300. A alocação dinâmica destes recursos possibilita, por exemplo, que o monitoramento de um determinado sistema seja inicialmente, ou constantemente, realizado no modo de baixo consumo de potência (ponto “C”) – somente recursos do núcleo básico – e alterne para os modos de alta performance computacional (pontos “A” e “B”). Neste contexto, é possível intensificar a avaliação de eventos a fim de determinar com maior precisão a sua natureza (raridade, situações de emergência) através da ativação de modos de alta performance toda vez que os sensores instalados no núcleo básico identificarem uma determinada condição de exceção, estabelecendo para isso, novos perfis de capacidade computacional em detrimento do consumo potência.

O levantamento de custo apresentado na Seção 4.5 e sumarizado na Tabela 6.1 esboça a contribuição que cada bloco da arquitetura representa no custo total do RANS-300. Este levantamento foi realizado considerando a aquisição de insumos para a confecção de poucos protótipos (de 1 a 25 unidades). O custo do bloco núcleo básico representa uma implementação de um nó sensor tradicional equipado com um sensor de temperatura. Os demais custos, fonte e bloco reconfigurável, são atribuídos à implementação dos recursos especiais de *hardware* previstos pela arquitetura do RANS-300. O custo total do protótipo do RANS-300 mostra que, mesmo adicionando os recursos reconfiguráveis de *hardware*, este dispositivo se justifica no cenário das RSSFS, quando ele é comparado com o preço de comercialização de alguns modelos de nós sensores, como é o do MICA2 Mote (em torno de duzentos dólares FOB).

| **Blocos da Arquitetura** | **Custo USD (FOB)** | **Percentual em relação ao custo total** |
|---|---|---|
| Núcleo Básico | $27.09 | 15.80% |
| Fonte | $26.78 | 15.60% |
| Bloco Reconfigurável | $117.59 | 68.60% |
| **Custo Total do Protótipo:** | **$171.38** | |

Tabela 6.1: Custo total do RANS-300.

Os testes elétricos e funcionais preliminares realizados no protótipo do RANS-300 produziram resultados esperados e evidenciam o funcionamento de determinados circuitos implementados. Durante a execução dos testes, não foi identificado nenhum problema de implementação que afetasse o funcionamento de algum recurso previsto na arquitetura do dispositivo. Entretanto, a complexidade deste desenvolvimento requer a realização de testes mais elaborados e amplos que permitam conhecer melhor o comportamento elétrico e funcional deste nó sensor.

## 6.2. Restrições de Projeto

### Padrão de Encapsulamento

No desenvolvimento do RANS-300 foram desconsiderados o emprego de circuitos integrados e outros componentes com encapsulamentos complexos e de alta densidade de integração como, por exemplo, o padrão BGA, devido a dificuldade de se realizar a sua montagem manual no PCB e de se efetuar testes e reparos. Estes padrões de encapsulamentos

são apropriados para montagem em máquinas de posicionamento e inserção automáticas e requer que o processo de solda seja efetuado em forno do tipo *reflow*.

**Aquisição de Componentes**

A especificação de cada componente eletrônico e parte (chave, conectores, suporte para bateria, etc) utilizados no projeto do RANS-300, além dos critérios técnicos, foi baseada em critérios comerciais como prazo de entrega, custo e facilidade para aquisição. Muitos componentes, apesar de serem baratos e eficientes, não estão disponíveis para aquisição em pequena quantidade, alguns deles somente são vendidos em lotes de mil ou mais unidades, o que é inviável para a construção de poucos protótipos. Outro aspecto é que muitos fabricantes não possuem uma boa rede de distribuição (no mercado brasileiro e mesmo no país de origem) e nem disponibilizam formas ágeis para aquisição de seus componentes, sendo necessário estabelecer um processo formal de importação e, neste caso, além de ser dispendioso em custo, muitas vezes não atendem em prazo.

A estratégia utilizada foi de basear a especificação e o processo de aquisição nos estoques de fornecedores internacionais especializados em venda de componentes pela WEB, como é o caso da Digikey [59] e a Farnell [84]. A disponibilidade e o preço de cada componente era verificado antes que ele fosse adotado no projeto. Este procedimento de adequação ocasionou na utilização de componentes similares aos tecnicamente pretendidos. No entanto, procurou-se na substituição, preservar ao máximo a compatibilidade funcional, minimizando as perdas de funcionalidades e otimizando ao máximo a integração.

**Projeto do PCB**

Para otimizar os custos de fabricação dos protótipos do RANS-300, foi descartado o emprego de placa de circuito impresso em multicamada, devido ao alto custo fixo envolvido no processo de fabricação e da aquisição das unidades de protótipos. Embora a complexidade do projeto de *hardware* justificasse o emprego deste tipo tecnologia, foi definido que o roteamento dos sinais elétricos seria realizado utilizando apenas duas faces de fiação (PCB dupla face). Esta restrição dificultou bastante a elaboração do PCB e introduziu um atraso considerável no prazo previsto para a execução desta atividade.

## 6.3. Trabalhos Futuros

A concepção da plataforma de *hardware* do dispositivo RANS-300 possibilita o desenvolvimento de novas aplicações em RSSFs e amplia a perspectiva para a elaboração de dos seguintes trabalhos futuros:

- realização de testes de desempenho, a fim de determinar os parâmetros reais de operação e consumo de potência;
- realização de estudos para incorporar ao dispositivo sistemas operacionais como o TinyOs ou YATOS;
- implementação dos procedimentos local e remoto de configuração da FPGA através do controle e envio de *bitstream*s pelo microcontrolador MSP430;
- desenvolvimento de módulos de expansão com sensores básicos;
- desenvolvimento de módulos de expansão com sensores elaborados e destinados a aplicações como aquisição de imagem;
- desenvolvimento de aplicações integradas com nós sensores que possuam compatibilidade de transmissão;
- desenvolvimento de módulos de expansão com rádio IEEE 802.15.4/Zigbee;
- desenvolvimento de aplicações utilizando cartões CompactFlash equipados com interface de comunicação IEEE 802.11;
- desenvolvimento de aplicações utilizando os recursos de memória, interface e de síntese lógica oferecidos pelo bloco reconfigurável;
- geração de publicações na área de RSSF e de sistemas embutidos para divulgar os conceitos e detalhes de implementação da arquitetura deste novo dispositivo;
- construção de outras unidades de protótipos a fim de implementar uma RSSF baseada neste novo dispositivo;
- geração de ferramentas e aplicativos de *software* para auxiliar no desenvolvimento de aplicações.

# APÊNDICE A - Diagramas Elétricos

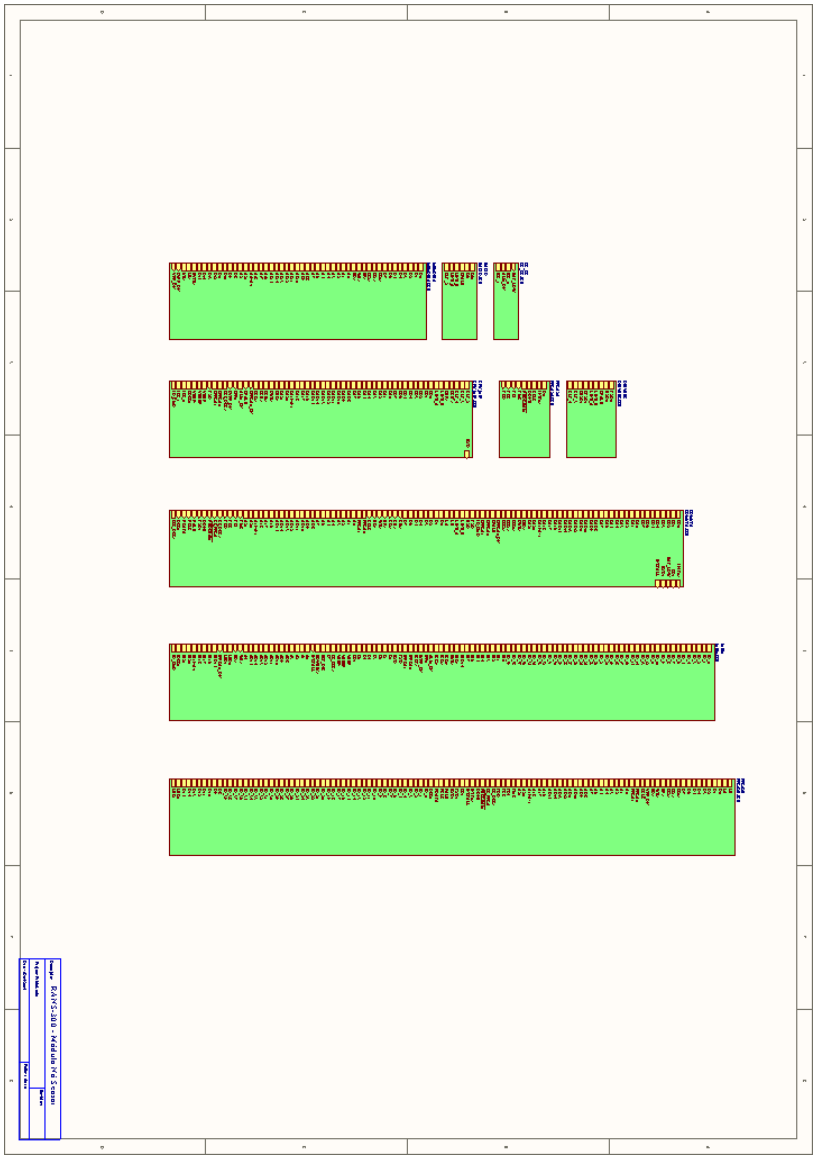


Figura A.1: Esquemático Hierárquico de Projeto.

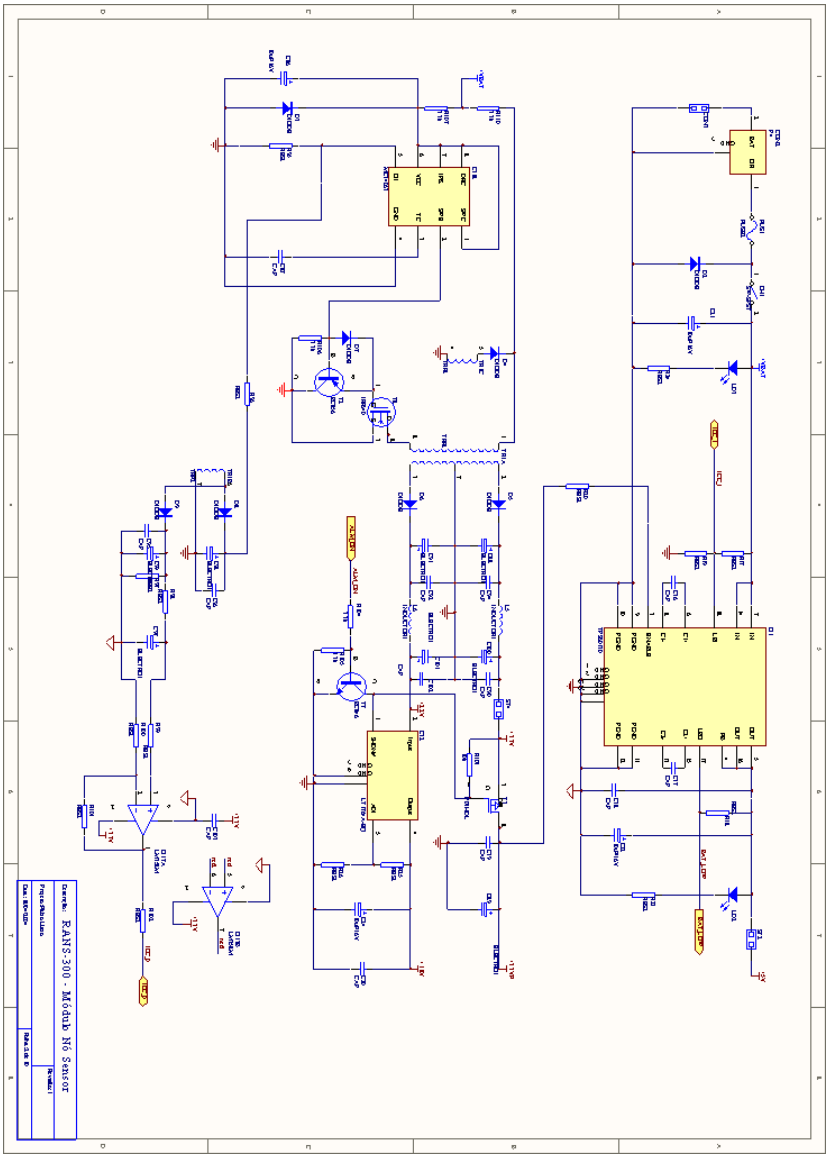


Figura A.2: Esquemático Elétrico da Fonte de Alimentação.

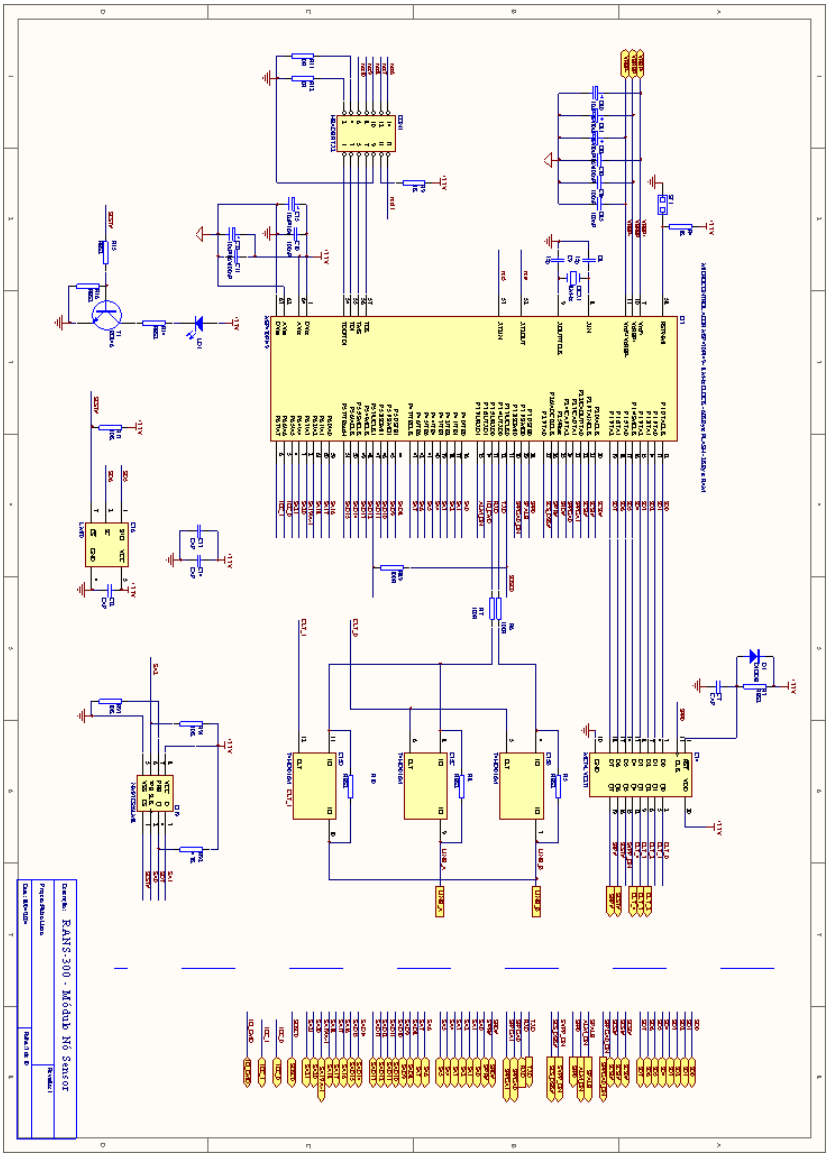



Figura A.3: Esquemático Elétrico da CPU MSP430F149.

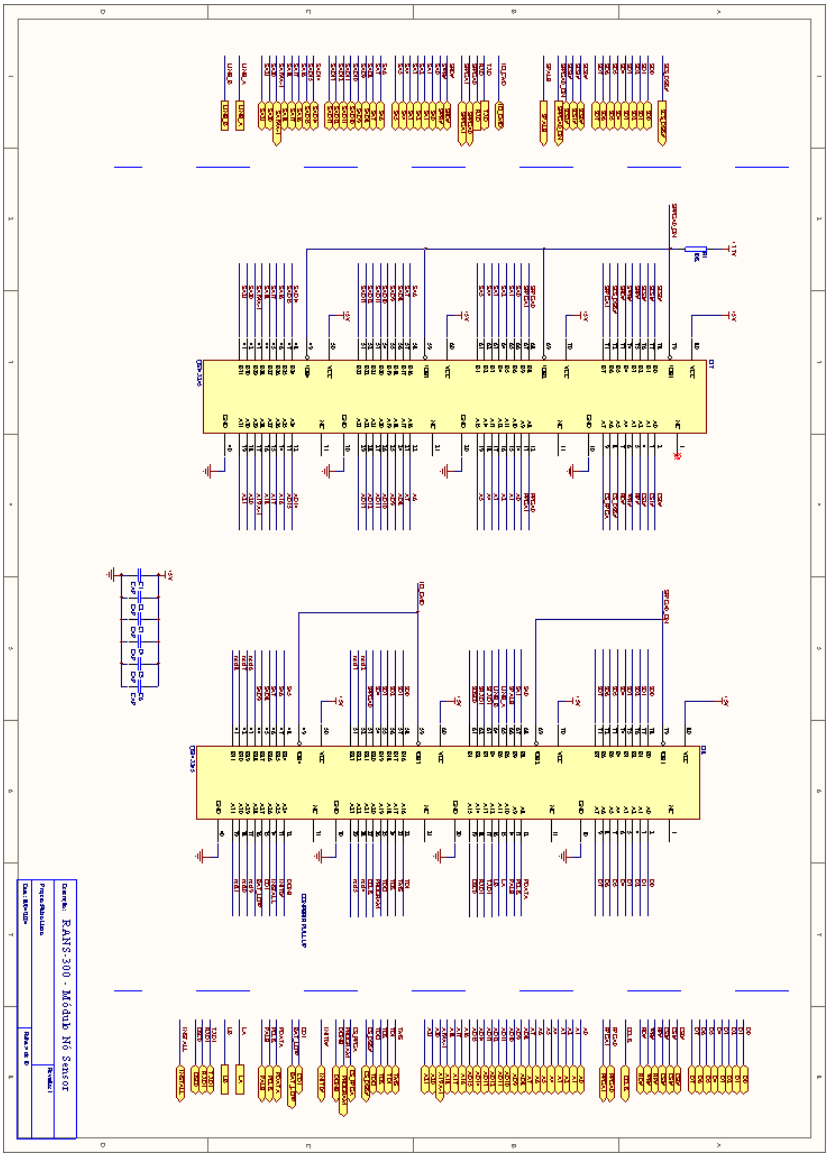



Figura A.4: Esquemático Elétrico da Comutação de Barramentos.

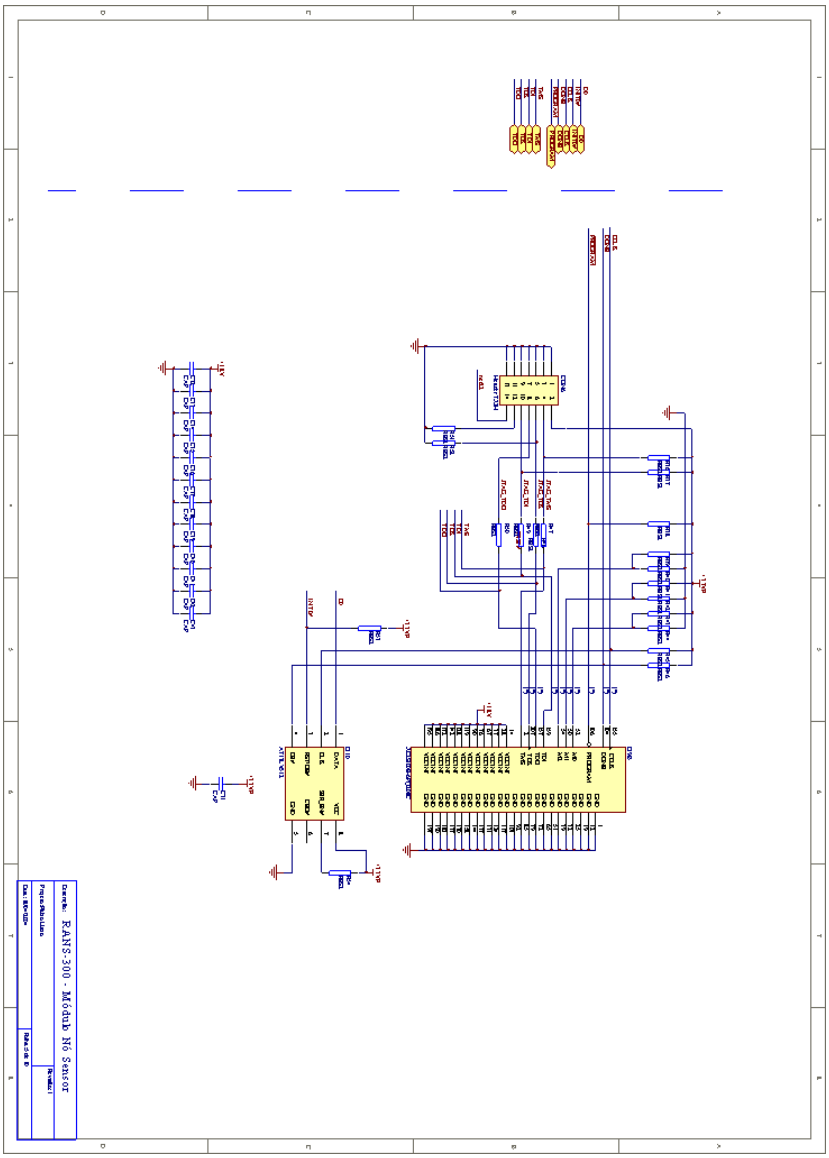



Figura A.5: Esquemático Elétrico da Alimentação e Configuração da FPGA.

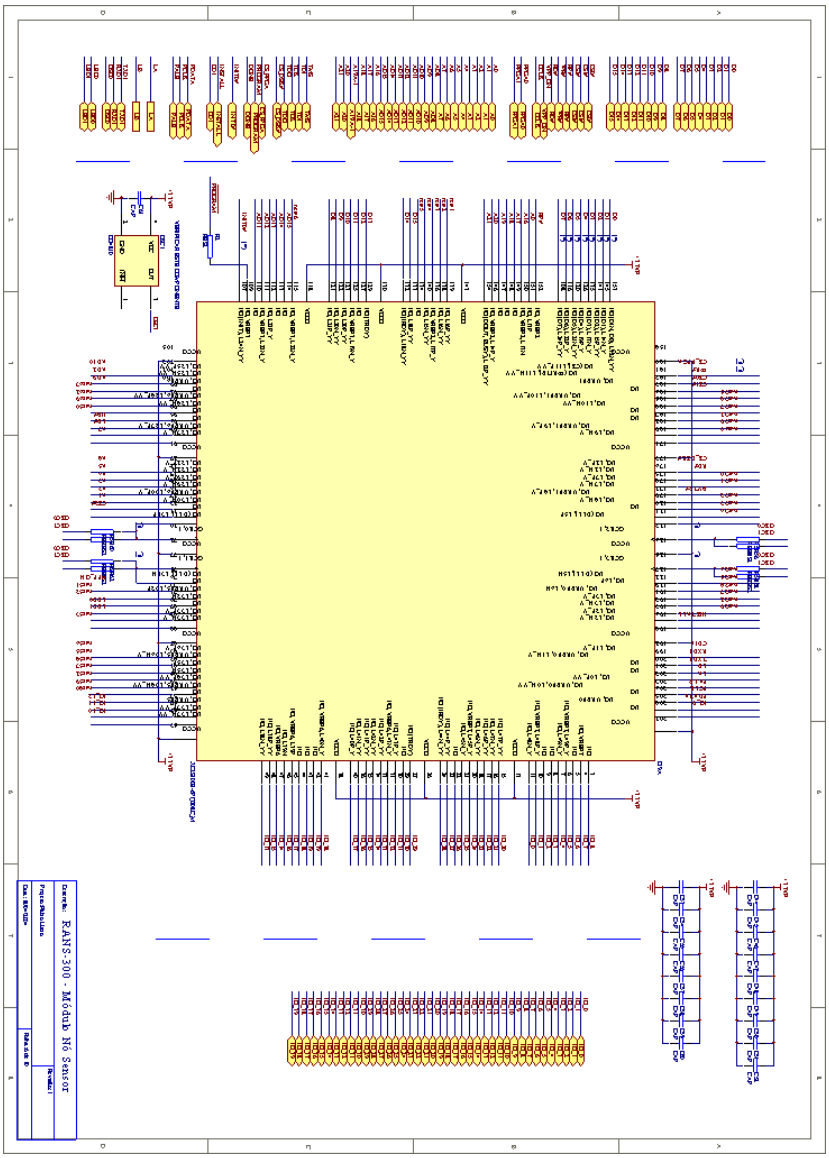



Figura A.6: Esquemático Elétrico da FPGA XC2S300E.

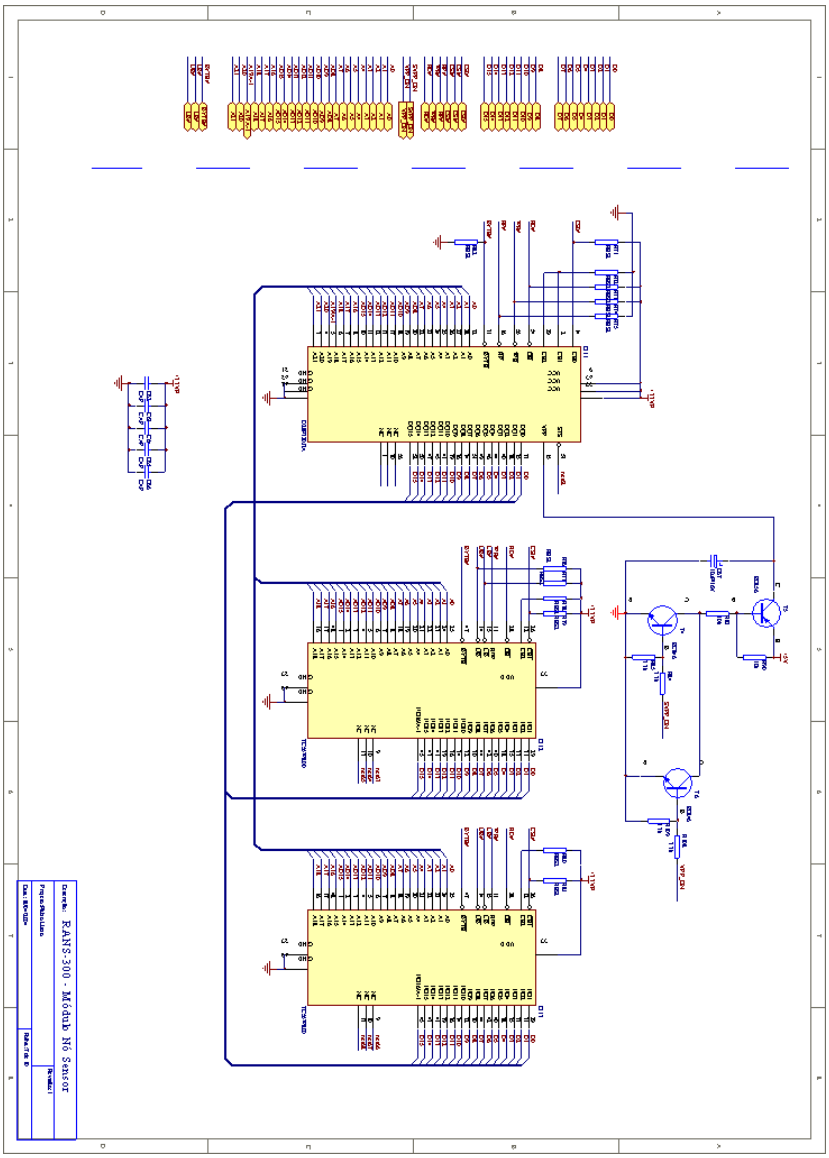



Figura A.7: Esquemático Elétrico da Unidade de Memória Flash e RAM.

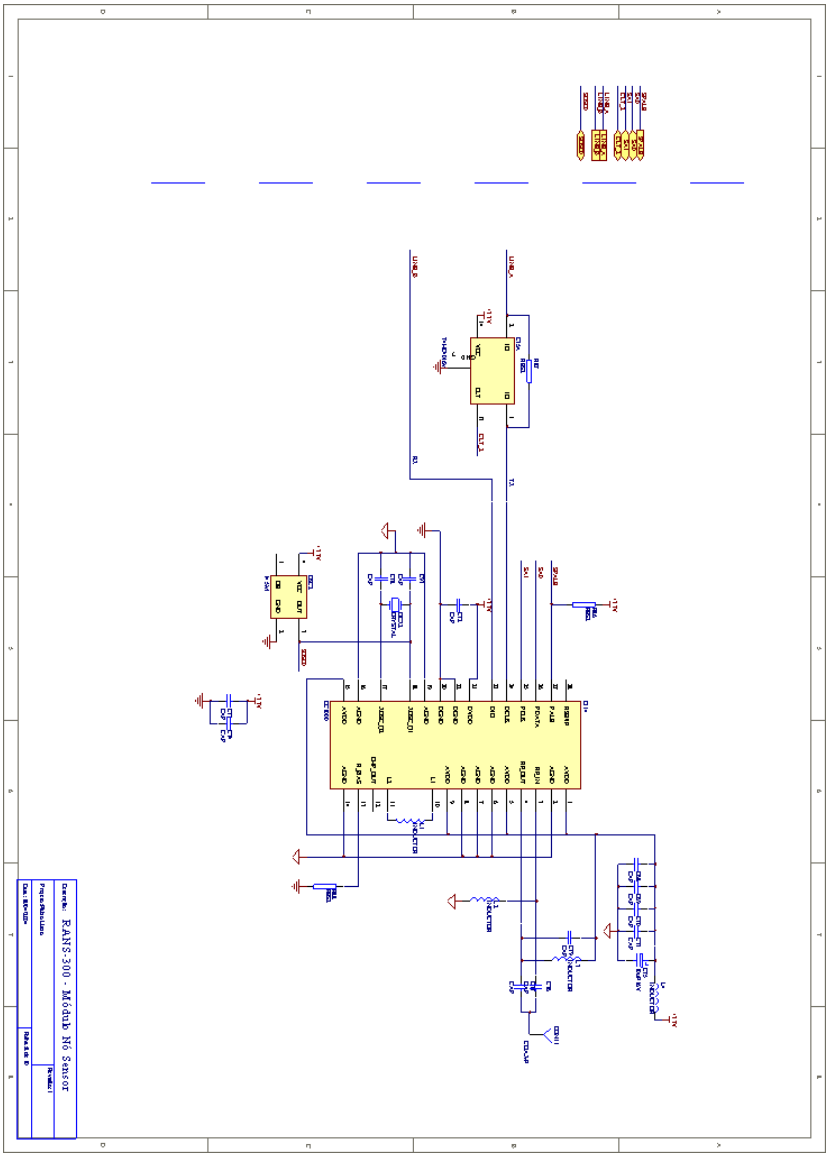



Figura A.8: Esquemático Elétrico do Rádio CC1000.

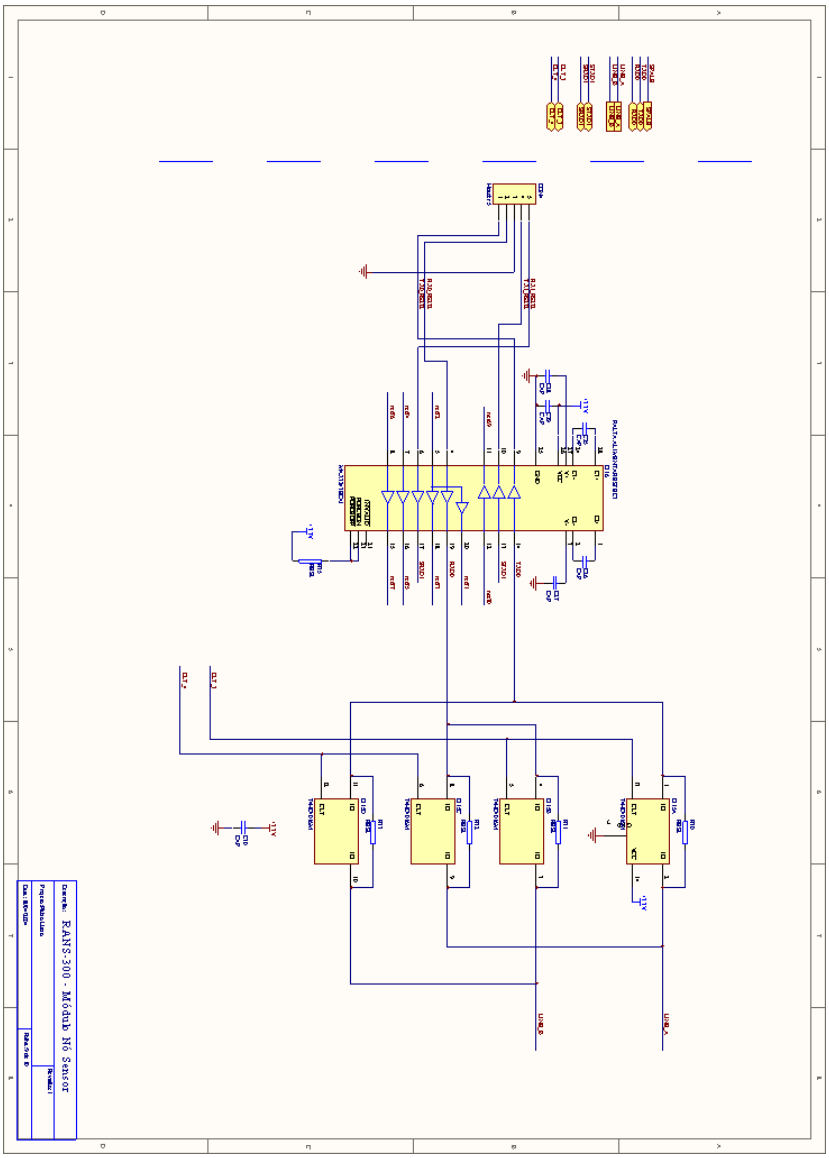



Figura A.9: Esquemático Elétrico da Interface Serial Padrão RS-232.

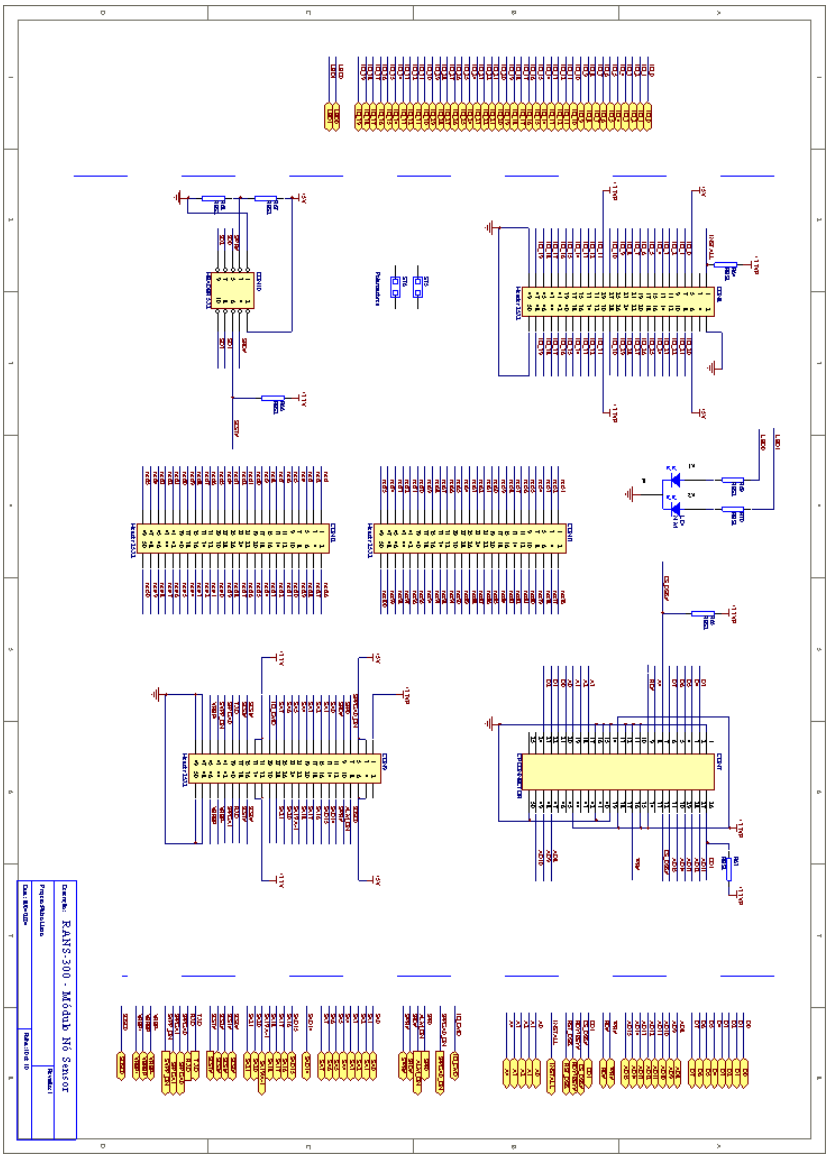
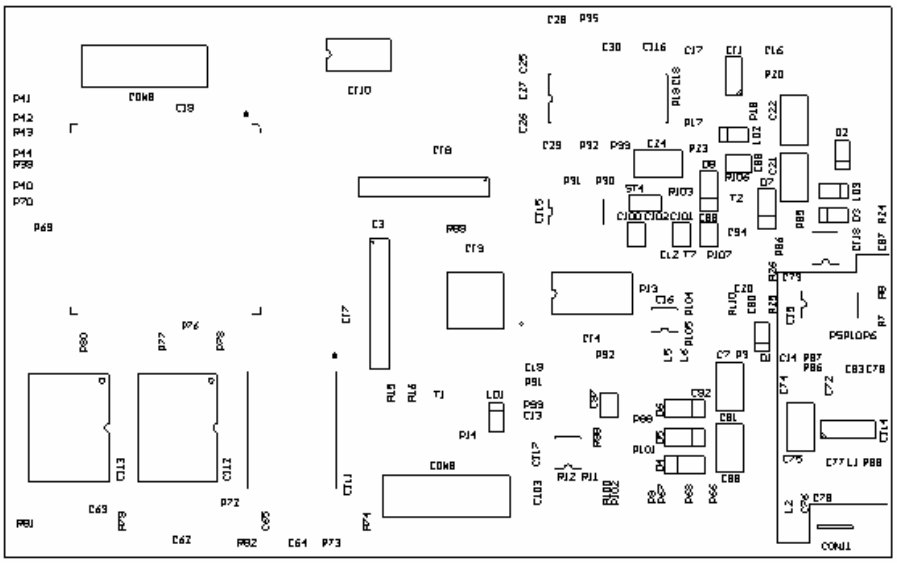
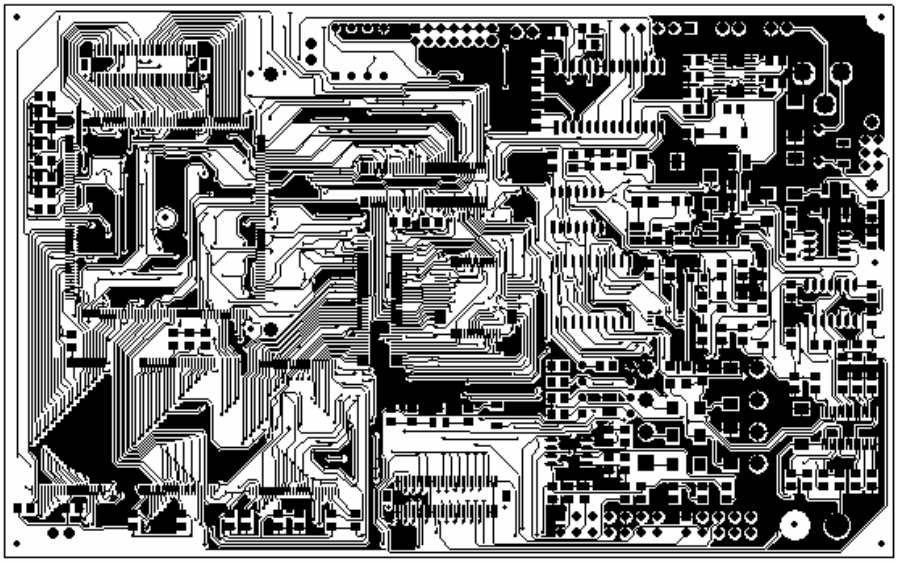


Figura A.10: Esquemático Elétrico dos Conectores de Acesso e Expansão.

# APÊNDICE B - Detalhes do projeto do PCB

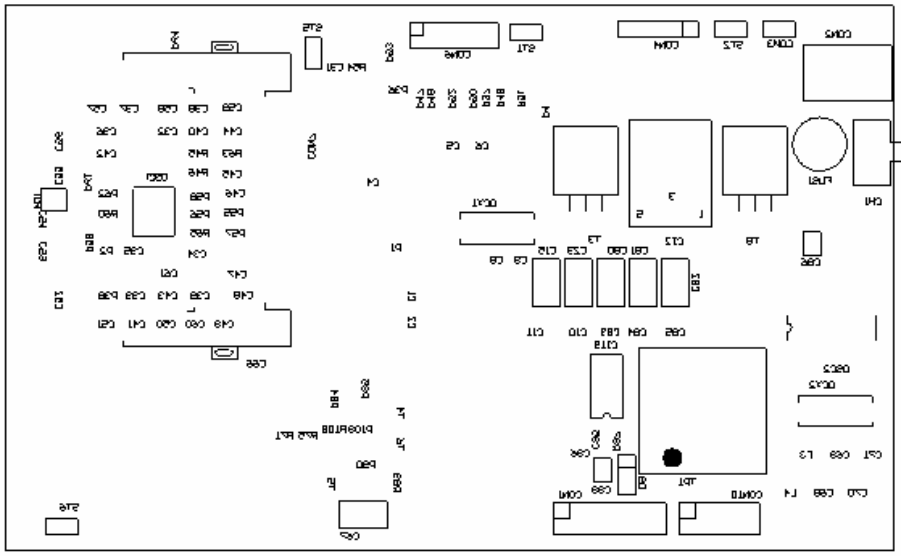


**(a)**

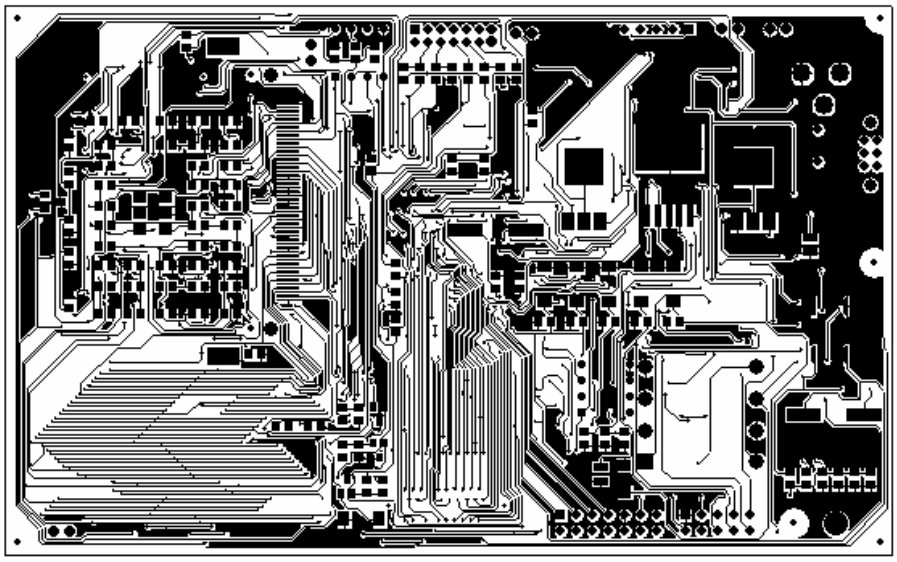


**(b)**

Figura B.1: Detalhe do PCB lado de componente.
a) simbologia.
b) roteamento dos sinais.

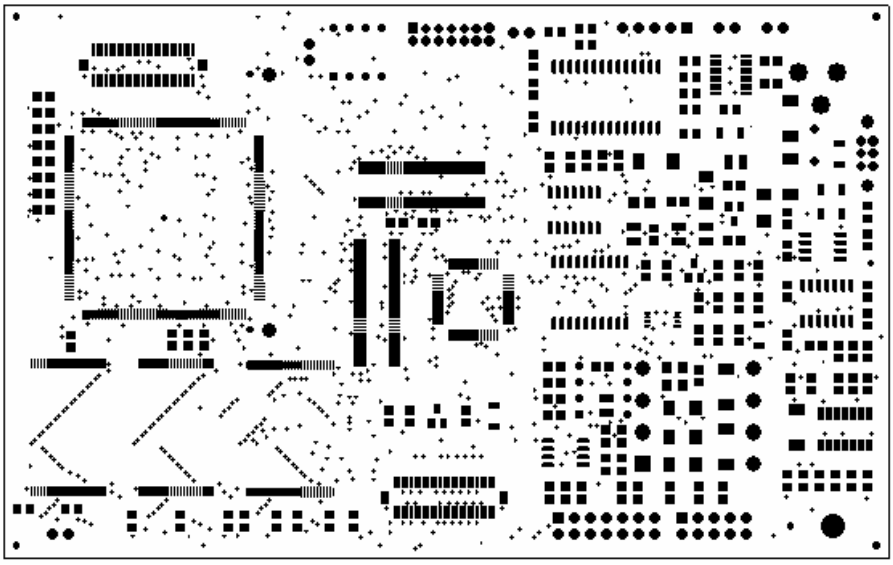


**(a)**

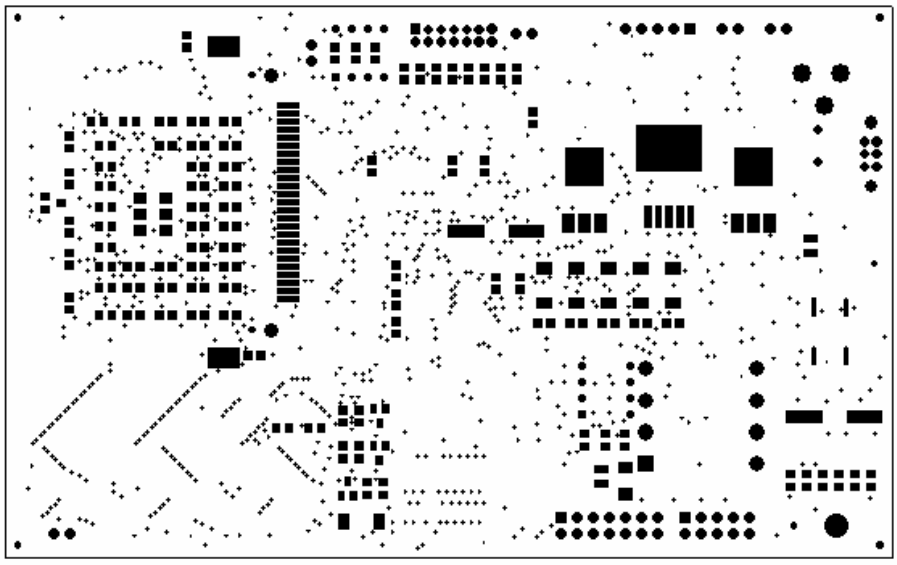


**(b)**

Figura B.2: Detalhe do PCB lado de solda.
a) simbologia.
b) roteamento dos sinais.



**(a)**



**(b)**

Figura B.3: Detalhe do PCB máscara de solda.
a) lado de componente.
b) lado de solda.

# APÊNDICE C - Pinagem dos Conectores de Expansão

| Pino | Sinal | Pino do MSP 430 |
|---|---|---|
| 1 | 3.3VF | |
| 2 | VAGO | |
| 3 | 5V | |
| 4 | 5V | |
| 5 | 5V | |
| 6 | 5V | |
| 7 | SPFGA0_ON | P3.2/SOMIO |
| 8 | SOSC0 | P3.3/UCLK0 |
| 9 | SFFO | P3.0/STE0 |
| 10 | ALM_ON | P3.7/URXD1 |
| 11 | SRD# | P2.5/Rosc |
| 12 | SWR# | P2.6/ADC12CLK |
| 13 | SA0 | P4.0/TB0 |
| 14 | SAD14 | P5.6/ACLK |
| 15 | SA1 | P4.1/TB1 |
| 16 | SAD15 | P5.7/TboutH |
| 17 | SA2 | P4.2/TB2 |
| 18 | SA16 | P6.0/A0 |
| 19 | SA3 | P4.3/TB3 |
| 20 | SA17 | P6.1/A1 |
| 21 | SA4 | P4.4/TB4 |
| 22 | SA18 | P6.2/A2 |
| 23 | SA5 | P4.5/TB5 |
| 24 | SA19/A-1 | P6.3/A3 |
| 25 | SA6 | P4.6/TB6 |
| 26 | SA20 | P6.4/A4 |
| 27 | SA7 | P4.7/TBCLK |
| 28 | VAGO | |
| 29 | IO_CMD | P3.6/UTXD1 |
| 30 | VAGO | |
| 31 | 3.3V | |
| 32 | 3.3V | |
| 33 | 3.3V | |
| 34 | 3.3V | |
| 35 | SCS1# | P2.1/TAINCLK |
| 36 | SCS0# | P2.0/ACLK |
| 37 | SCS2# | P2.2/CAOUT/TA0 |
| 38 | SCST# | |
| 39 | TXD | P3.4/UTXD0 |
| 40 | RXD | P3.5/URXD1 |
| 41 | SFPGA0 | P2.4/CA1/TA2 |
| 42 | SFPGA1 | P2.3/CA0/TA1 |
| 43 | SVPP_ON | |
| 44 | VREF- | VREF- |
| 45 | VREF+ | VREF+ |

| Pino | Sinal | Pino do MSP 430 |
|---|---|---|
| 46 | VEREF | VEREF |
| 47 | GND | |
| 48 | GND | |
| 49 | GND | |
| 50 | GND | |

Figura C.1: Detalhe da pinagem do conector de expansão do núcleo básico.

| Pino | Sinal /FPGA |
|---|---|
| 1 | INSTALL |
| 2 | GND |
| 3 | VAGO |
| 4 | VAGO |
| 5 | 5V |
| 6 | 5V |
| 7 | IO_0 |
| 8 | IO_20 |
| 9 | IO_1 |
| 10 | IO_21 |
| 11 | IO_2 |
| 12 | IO_22 |
| 13 | IO_3 |
| 14 | IO_23 |
| 15 | IO_4 |
| 16 | IO_24 |
| 17 | IO_5 |
| 18 | IO_25 |
| 19 | IO_6 |
| 20 | IO_26 |
| 21 | IO_7 |
| 22 | IO_27 |
| 23 | IO_8 |
| 24 | IO_28 |
| 25 | IO_9 |
| 26 | IO_29 |
| 27 | IO_10 |
| 28 | IO_30 |
| 29 | 3.3VF |
| 30 | 3.3VF |
| 31 | IO_11 |
| 32 | IO_31 |
| 33 | IO_12 |
| 34 | IO_32 |
| 35 | IO_13 |
| 36 | IO_33 |
| 37 | IO_14 |
| 38 | IO_34 |
| 39 | IO_15 |

| Pino | Sinal /FPGA |
|---|---|
| 40 | IO_35 |
| 41 | IO_16 |
| 42 | IO_36 |
| 43 | IO_17 |
| 44 | IO_37 |
| 45 | IO_18 |
| 46 | IO_38 |
| 47 | IO_19 |
| 48 | IO_39 |
| 49 | GND |
| 50 | GND |

Figura C.2: Detalhe da pinagem do conector de expansão do bloco reconfigurável.

# APÊNDICE D - Dimensões Físicas

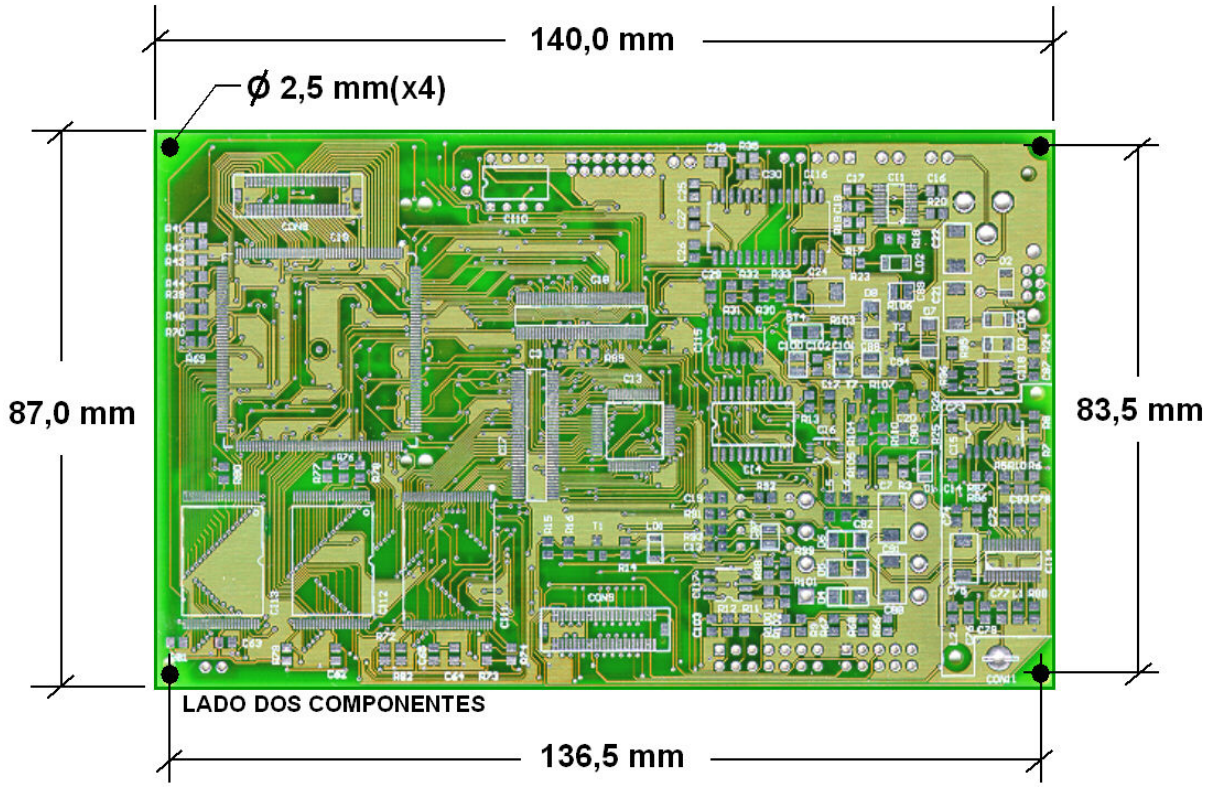
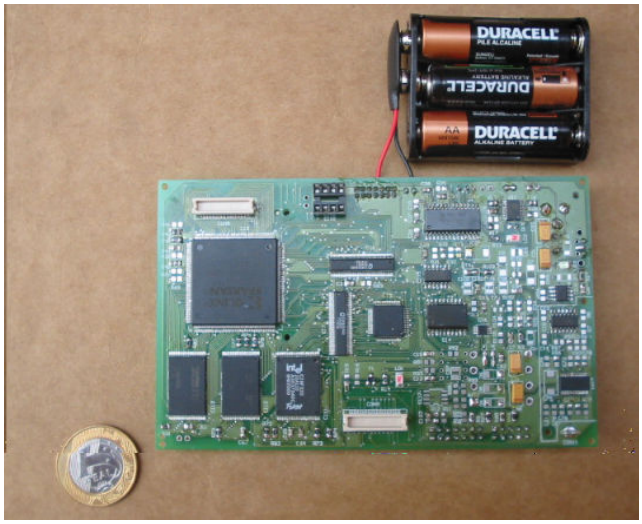



Figura D.1: Detalhe das dimensões físicas do PCB do RANS-300.

# APÊNDICE E - Detalhes da Arquitetura de Hardware



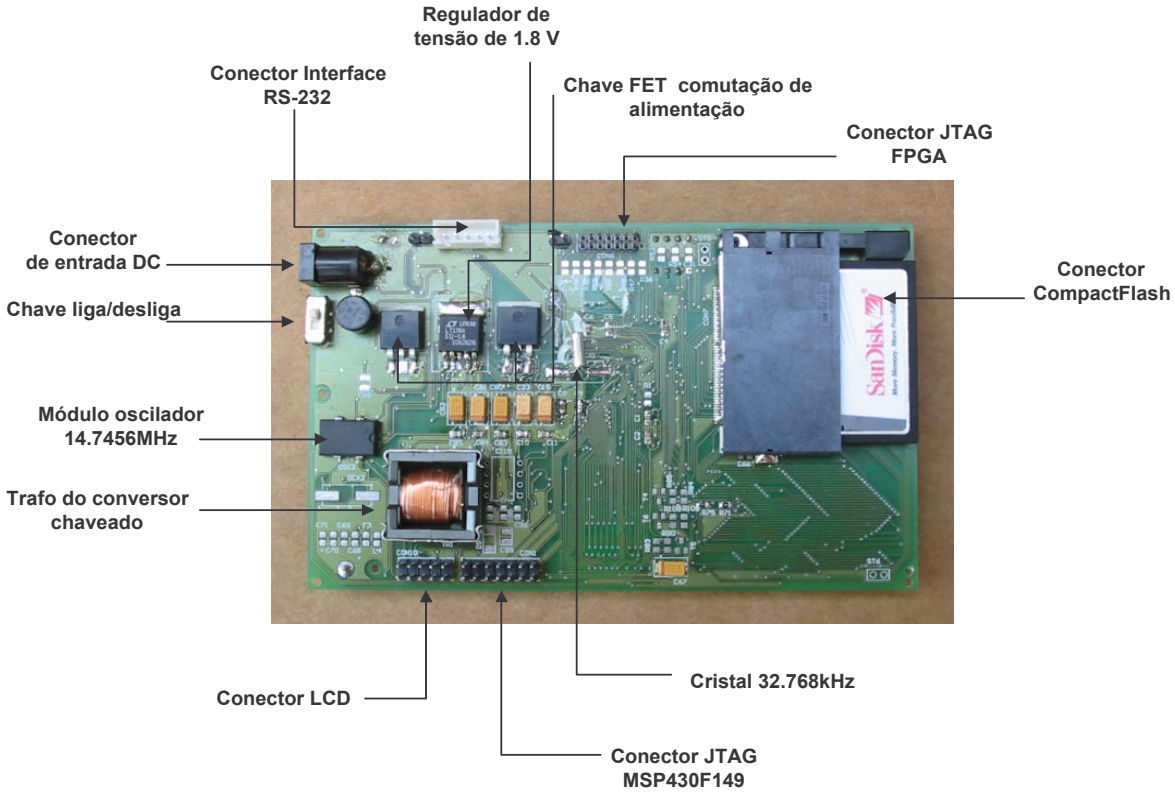
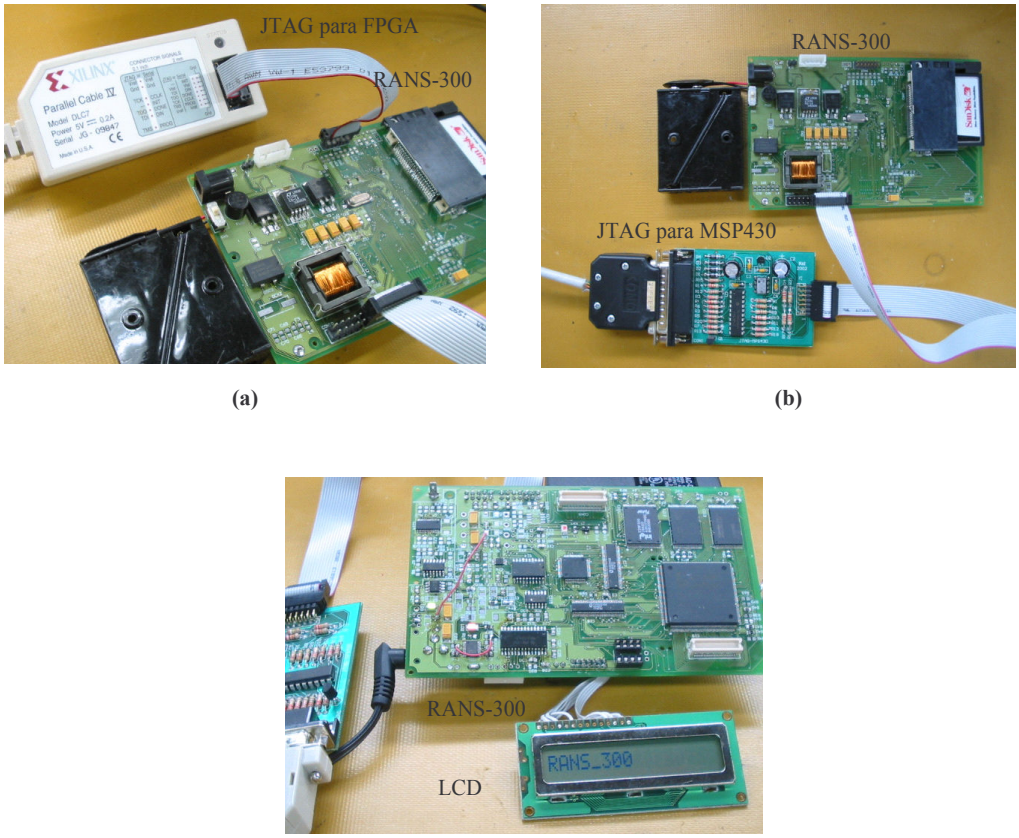




Figura E.1: Detalhes de implementação da arquitetura do RANS-300.

# APÊNDICE F - Ambiente de Depuração




(a) (b)

(c)

Figura F.1: Detalhes do ambiente de depuração.

a) interface JTAG para configuração da FPGA.
b) interface JTAG para programação do MSP430F149.
c) interface LCD para visualização de mensagens.

# APÊNDICE G - Simulações

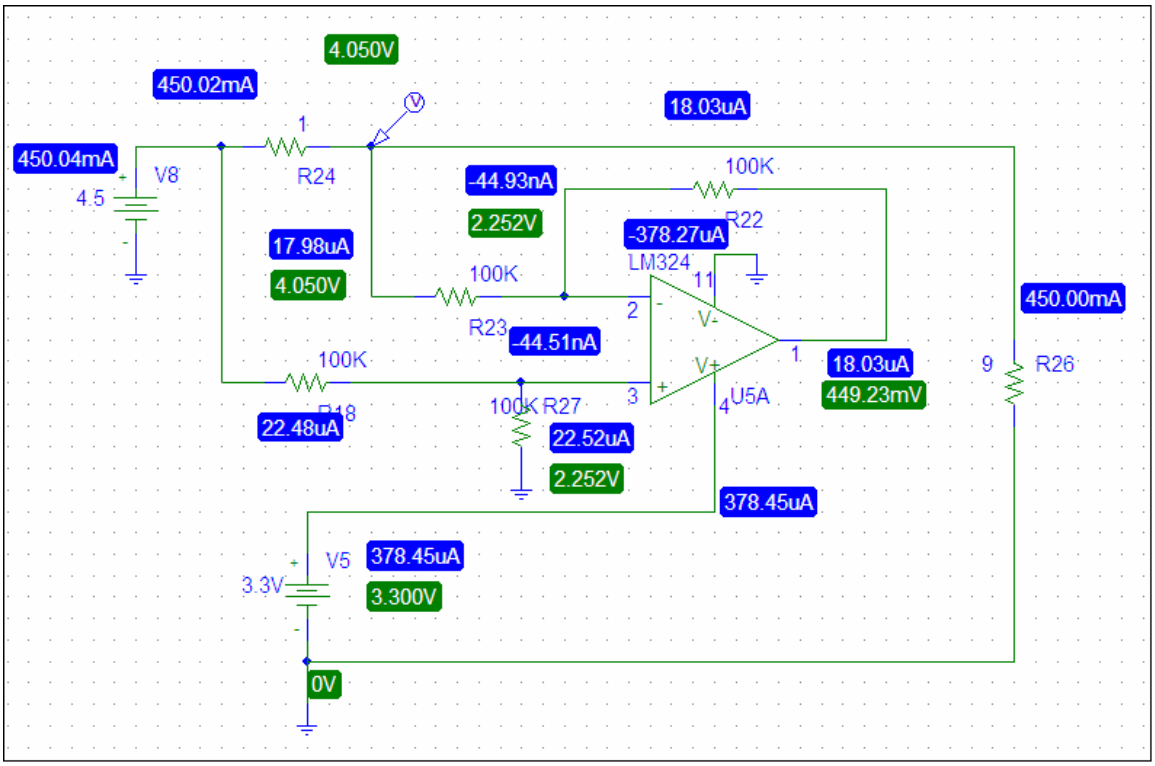



Figura G.1: Simulação do circuito de medição de consumo.

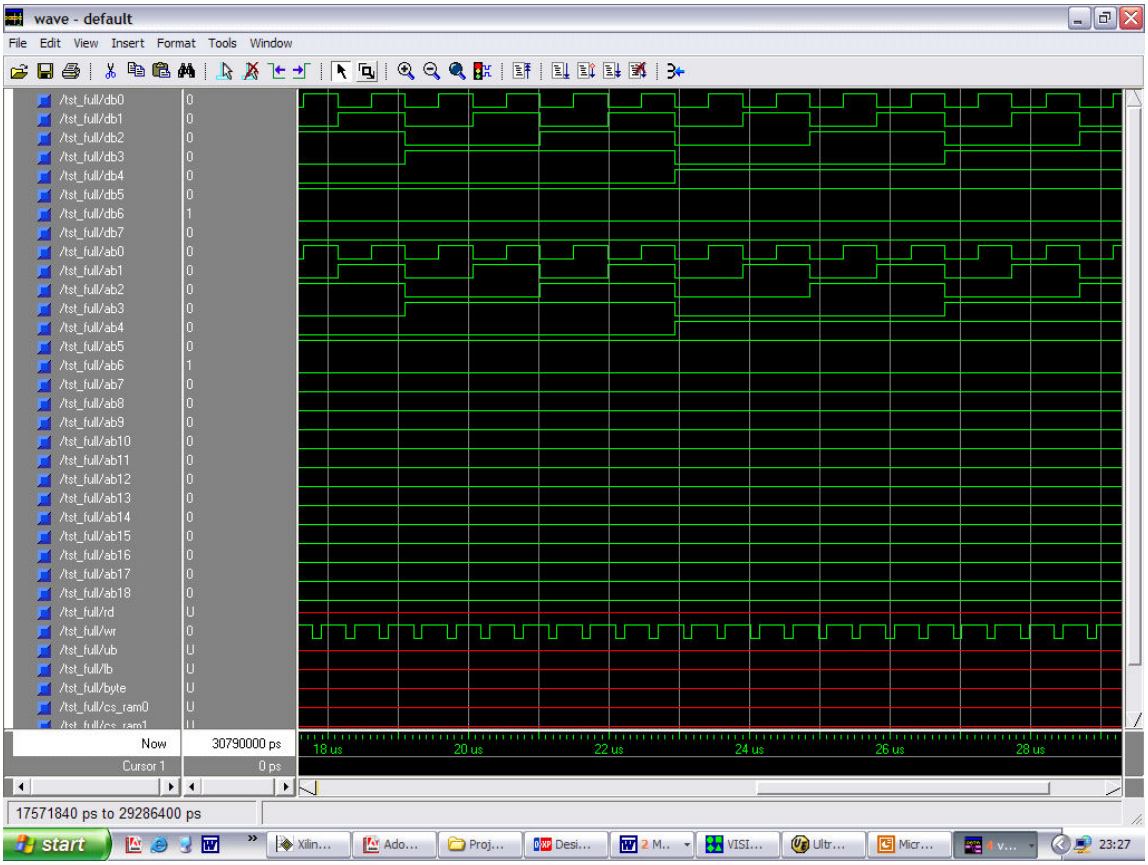



Figura G.2: Simulação dos sinais de barramento da RAM gerados pela FPGA.

```
=========================================================================
*                  Synthesis Options Summary                  *
=========================================================================
---- Source Parameters
Input File Name                    : main_rans.prj
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO
Verilog Include Directory          :

---- Target Parameters
Output File Name                   : main_rans
Output Format                      : NGC
Target Device                      : xc2s300e-7-pq208


=========================================================================
HDL Synthesis Report

Macro Statistics
# FSMs                             : 1
# Registers                        : 1
  1-bit register                   : 1
# Counters                         : 3
  32-bit up counter                : 1
  8-bit up counter                 : 1
  19-bit up counter                : 1

=========================================================================


=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2s300epq208-7

 Number of Slices:                  42  out of   3072     1%
 Number of Slice Flip Flops:        68  out of   6144     1%
 Number of 4 input LUTs:            72  out of   6144     1%
 Number of bonded IOBs:             29  out of    146    19%
 Number of GCLKs:                   1  out of      4    25%


=========================================================================
```

Figura G.3: Relatório simplificado gerado pelo ISE6.3 relativo à síntese lógica do Emulador.

# APÊNDICE H - Código VHDL do Circuito Emulador

```
-------------------------------------------
-- Modulo main_rans.vhd
-- Descricao: integracao dos modulos de teste
-- do no sensor RANS-300
-- Escrito por: Fabio L.C. Junior
-- Data: 10 Set 2004
-- DCC/UFMG
-------------------------------------------
-------------------------------------------
-- Bibliotecas
-------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
use ieee.std_logic_unsigned.all;
library UNisIM;
use UNisIM.Vcomponents.ALL;
-------------------------------------------
-- Entidade main_RANS
-------------------------------------------
entity main_rans is
   port (
-- barramento de acesso aos bancos de memoria externa
-- dados
     db0: inout STD_LOGIC;
          db1: inout STD_LOGIC;
          db2: inout STD_LOGIC;
          db3: inout STD_LOGIC;
          db4: inout STD_LOGIC;
          db5: inout STD_LOGIC;
          db6: inout STD_LOGIC;
          db7: inout STD_LOGIC;
-- sinais de controle
-- enderecos
     ab0: out STD_LOGIC;
          ab1: out STD_LOGIC;
          ab2: out STD_LOGIC;
     ab3: out STD_LOGIC;
     ab4: out STD_LOGIC;
          ab5: out STD_LOGIC;
          ab6: out STD_LOGIC;
     ab7: out STD_LOGIC;
     ab8: out STD_LOGIC;
          ab9: out STD_LOGIC;
          ab10: out STD_LOGIC;
     ab11: out STD_LOGIC;
     ab12: out STD_LOGIC;
          ab13: out STD_LOGIC;
          ab14: out STD_LOGIC;
          ab15: out STD_LOGIC;
     ab16: out STD_LOGIC;
```

```
     ab17: out STD_LOGIC;
          ab18: out STD_LOGIC;
          rd: out STD_LOGIC;
     wr: out STD_LOGIC;
     ub: out STD_LOGIC;
          lb: out STD_LOGIC;
     byte: out STD_LOGIC;
     cs_ram0: out STD_LOGIC;
     cs_ram1: out STD_LOGIC;
     cs_flash: out STD_LOGIC;
     clock: in STD_LOGIC);
-------------------------------------------
-- Pinout xc2s300E-Q208
-------------------------------------------
   attribute LOC : string ;
-- sinais de controle e barramento
   attribute LOC of db0: signal is "P153";
   attribute LOC of db1: signal is "P145";
   attribute LOC of db2: signal is "P141";
   attribute LOC of db3: signal is "P135";
   attribute LOC of db4: signal is "P126";
   attribute LOC of db5: signal is "P120";
   attribute LOC of db6: signal is "P116";
   attribute LOC of db7: signal is "P108";

   attribute LOC of ab0: signal is "P152";
   attribute LOC of ab1: signal is "P151";
   attribute LOC of ab2: signal is "P150";
   attribute LOC of ab3: signal is "P149";
   attribute LOC of ab4: signal is "P148";
   attribute LOC of ab5: signal is "P147";
   attribute LOC of ab6: signal is "P146";
   attribute LOC of ab7: signal is "P154";
   attribute LOC of ab8: signal is "P139";
   attribute LOC of ab9: signal is "P138";
   attribute LOC of ab10: signal is "P136";
   attribute LOC of ab11: signal is "P140";
   attribute LOC of ab12: signal is "P134";
   attribute LOC of ab13: signal is "P133";
   attribute LOC of ab14: signal is "P132";
   attribute LOC of ab15: signal is "P129";
   attribute LOC of ab16: signal is "P127";
   attribute LOC of ab17: signal is "P125";
   attribute LOC of ab18: signal is "P123";

   attribute LOC of rd: signal is "P99";
   attribute LOC of wr: signal is "P161";
   attribute LOC of ub: signal is "P87";
   attribute LOC of lb: signal is "P88";
   attribute LOC of byte: signal is "P86";
   attribute LOC of cs_ram0: signal is "P101";
   attribute LOC of cs_ram1: signal is "P100";
   attribute LOC of cs_flash: signal is "P162";
   attribute LOC of clock: signal is "P80";
end main_rans;
-------------------------------------------
-- Arquitetura main_1
-------------------------------------------
architecture integra of main_rans is
   signal sdata: STD_LOGIC_vector (7 downto 0);
   signal ssdata: STD_LOGIC_vector (7 downto 0);
```

```
   signal sadress: STD_LOGIC_vector (18 downto 0);
   signal sgera_bus: STD_LOGIC;
   signal sbus_busy: STD_LOGIC;
   signal shab_adress: STD_LOGIC;
   signal shab_data: STD_LOGIC;
   signal swr: STD_LOGIC;
   signal sum: STD_LOGIC;
   signal szero: STD_LOGIC;
   signal sstart: STD_LOGIC;
   signal sd1: STD_LOGIC;
   signal sd2: STD_LOGIC;
   signal sshab_data: STD_LOGIC;
-----------------------------------------
-- Componentes instanciados
-----------------------------------------
    component gera_teste is
    Port ( clock: in  std_logic;
           start: in  std_logic;
      busy: in  std_logic;
           data: out std_logic_vector (7 downto 0);
           adress: out std_logic_vector (18 downto 0));
    end component;
-----------------------------------------
    component tpo_write_ram is
       Port (hab_data: out STD_LOGIC;
             hab_adress: out STD_LOGIC;
        wr: out STD_LOGIC;
             rd: out STD_LOGIC;
        bus_busy: out STD_LOGIC;
        cs_ram: out STD_LOGIC;
        clock: in STD_LOGIC;
        gera_bus: in STD_LOGIC;
        start: in STD_LOGIC);
    end component;
-----------------------------------------
   attribute BOX_TYPE : string;
   component IBUF
      port ( I :  in STD_LOGIC;
             O :  OUT  STD_LOGIC);
   end component;
   attribute BOX_TYPE of IBUF : component is "BLACK_BOX";
-----------------------------------------
   component IBUF_GTLP
      port ( I :  in STD_LOGIC;
             O :  OUT  STD_LOGIC);
   end component;
   attribute BOX_TYPE of IBUF_GTLP : component is "BLACK_BOX";
-----------------------------------------
--   signal INT_SIG, T_ENABLE: std_logic;
   component OBUFT
      port (I, T: in std_logic; O: out std_logic);
   end component;
-----------------------------------------
   component FD
--   generic(INIT : BIT := '0');
      port ( C :  in STD_LOGIC;
             D :  in STD_LOGIC;
             Q :  out  STD_LOGIC);
   end component;
   attribute BOX_TYPE of FD: component is "BLACK_BOX";
```

```
-------------------------------------------
begin
-------------------------------------------
sum<= '1';
szero<= '0';

-- zero habilita buffer
sshab_data<= '0';


    U0: OBUFT port map (I => sdata(0),
                        T => sshab_data,
              O => db0);


    U1: OBUFT port map (I => sdata(1),
                        T => sshab_data,
              O => db1);

    U2: OBUFT port map (I => sdata(2),
                        T => sshab_data,
              O => db2);

    U3: OBUFT port map (I => sdata(3),
                        T => sshab_data,
              O => db3);

    U4: OBUFT port map (I => sdata(4),
                        T => sshab_data,
              O => db4);

    U5: OBUFT port map (I => sdata(5),
                        T => sshab_data,
              O => db5);

    U6: OBUFT port map (I => sdata(6),
                        T => sshab_data,
              O => db6);

    U7: OBUFT port map (I => sdata(7),
                        T => sshab_data,
              O => db7);
-------------------------------------------
    gera_padroes: gera_teste
    Port map ( clock=>clock,
               start=>sstart,
          busy=>sbus_busy,
               data=>sdata,
               adress(0)=>ab0,
          adress(1)=>ab1,
          adress(2)=>ab2,
          adress(3)=>ab3,
          adress(4)=>ab4,
          adress(5)=>ab5,
          adress(6)=>ab6,
          adress(7)=>ab7,
          adress(8)=>ab8,
          adress(9)=>ab9,
          adress(10)=>ab10,
          adress(11)=>ab11,
          adress(12)=>ab12,
```

```
          adress(13)=>ab13,
          adress(14)=>ab14,
          adress(15)=>ab15,
          adress(16)=>ab16,
          adress(17)=>ab17,
          adress(18)=>ab18);
-------------------------------------------
    inter_write: tpo_write_ram
    Port map ( hab_data=> shab_data,
               hab_adress=>open,
          wr=>wr,
               rd=> open,
          bus_busy=>sbus_busy,
        cs_ram=>cs_ram0,
          clock=>clock,
          gera_bus=>sum,
          start=>sstart);
-------------------------------------------
   ff0: fd
   port map(C=>clock,
            D=>sum,
            Q=>sd1);
   ff1: fd
   port map(C=>clock,
            D=>sd1,
            Q=>sd2);

   ff2: fd
   port map(C=>clock,
            D=>sd2,
            Q=>sstart);
end integra;


-------------------------------------------
-- Modulo gerador.vhd
-- Descricao: Gerador de padroes.
-- Escrito por: Fabio Lucio
-- Data: 13 SET 2004
-- DCC/UFMG
-------------------------------------------
-------------------------------------------
-- Bibliotecas
-------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-------------------------------------------
-- Entidade Gera_teste
-------------------------------------------

entity gera_teste is
    Port ( clock: in  std_logic;
           start: in  std_logic;
      busy: in  std_logic;
      start_out: out std_logic;
           data: out std_logic_vector (7 downto 0);
           adress: out std_logic_vector (18 downto 0));
end gera_teste;
```

```
-------------------------------------------
-- Arquitetura Gera
-------------------------------------------

architecture gera of gera_teste is

signal sadress: unsigned(18 downto 0);
signal sdata: unsigned(7 downto 0);

begin

   increment: process (clock,start)
   variable dados_gerados: integer;

   begin
   if (start = '0')then
      dados_gerados:= 0;
      sdata<= "00000000";
      sadress<= "0000000000000000000";
      start_out<= '1';
     elsif (clock = '1' and clock'event) then
      if (busy = '0') then
            sadress <= sadress + 1;
         sdata <= sdata + 1;
       start_out<= '1';
         if (dados_gerados = 255) then
            sdata<= "00000000";
         dados_gerados:= 0;
         else
            dados_gerados:= dados_gerados + 1;
         sdata <= sdata + 1;
         end if;
      end if;
     end if;
   end process;

  adress <= std_logic_vector(sadress);
  data <=  std_logic_vector(sdata);

end gera;

-------------------------------------------
-- Modulo tpo_RAM
-- RAM externa
-- Descricao: gera temporizacao de acesso de escrita
--            para memoria RAM externa
-- Escrito por: Fabio L.C. Junior
-- Data: 10 Set 2004
-- DCC/UGMG
-------------------------------------------
-------------------------------------------
-- Bibliotecas
-------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use WORK.NPACK2.ALL;
-------------------------------------------
-- Entidade tpo_write_RAM
```

```
-------------------------------------------
entity tpo_write_ram is
    Port (hab_data: out STD_LOGIC;
          hab_adress: out STD_LOGIC;
     wr: out STD_LOGIC;
          rd: out STD_LOGIC;
     bus_busy: out STD_LOGIC;
     cs_ram: out STD_LOGIC;

-- sinais de entrada
     clock: in STD_LOGIC;
     gera_bus: in STD_LOGIC;
     start: in STD_LOGIC);
end tpo_write_ram;
-------------------------------------------
-- Arquitetura tpo_write_ram
-------------------------------------------
architecture tpo of tpo_write_ram is
   signal current_state,next_state: machine1;
   signal sgera_bus: STD_LOGIC;
   signal sbus_busy: STD_LOGIC;
   signal shab_adress: STD_LOGIC;
   signal shab_data: STD_LOGIC;
   signal swr: STD_LOGIC;
   signal stroca: STD_LOGIC;
   signal scs_ram: STD_LOGIC;


begin
    bus_busy<= sbus_busy;
    hab_adress<= shab_adress;
    hab_data<= shab_data;
    wr<= not swr;
    cs_ram<=scs_ram;

-------------------------------------------
-- Processo gera_ciclo_write
-- Este processo realiza:
-- gera o ciclo de barramento de escrita para
-- a memoria RAM
-------------------------------------------
   gera_ciclo_write: process (current_state,clock)
      variable n_ciclos: integer;
    variable write: STD_LOGIC;
    variable busy: STD_LOGIC;
    variable data: STD_LOGIC;
    variable adress: STD_LOGIC;
   begin
      case current_state is
         when repouso =>
       n_ciclos:=0;
          write:= '0';
             busy:= '0';
       data:= '0';
       adress:= '0';

       swr<= write;
       sbus_busy<= busy;
       shab_data<= data;
       shab_adress<= adress;
```

```
        next_state<= endereco;
          when endereco =>
           write:= '0';
              busy:= '1';
        data:= '0';
        adress:= '1';

        swr<= write;
        sbus_busy<= busy;
        shab_data<= data;
        shab_adress<= adress;

                next_state<= controle;
          when controle =>
           write:= '1';
              busy:= '1';
        data:= '1';
        adress:= '1';
        swr<= write;
        sbus_busy<= busy;
        shab_data<= data;
        shab_adress<= adress;
--      if (n_ciclos=2)then
--         n_ciclos:=0;
           next_state<=fim_controle;
--               else
--         n_ciclos:= n_ciclos+1;
--                  next_state<= controle;
--               end if;
          when dados =>
           write:= '1';
              busy:= '1';
        data:= '1';
        adress:= '1';
        swr<= write;
        sbus_busy<= busy;
        shab_data<= data;
        shab_adress<= adress;

                next_state<= fim_controle;
          when fim_controle =>
           write:= '0';
              busy:= '1';
        data:= '1';
        adress:= '1';
        swr<= write;
        sbus_busy<= busy;
        shab_data<= data;
        shab_adress<= adress;

        next_state<=fim_ciclo;
          when fim_ciclo =>
           write:= '0';
              busy:= '0';
        data:= '0';
        adress:= '0';
        swr<= write;
        sbus_busy<= busy;
        shab_data<= data;
        shab_adress<= adress;
```

```
            next_state<= endereco;
          end case;
   end process;
-------------------------------------------
-- Processo states_eval:
-- Este processo realiza:
--    - reset assincrono;
--    - gera os estados para a maquina de estados:
--      repouso,
--      enderecos,
--      controle e
--      dados
-------------------------------------------
   states_eval: process(clock,start,gera_bus)
      variable troca: STD_LOGIC;

   begin
      if (start = '0') then
       troca:= '1';
       stroca<= troca;
       sgera_bus<= gera_bus;
         current_state<= repouso;
-- considera a subida do clock
      elsif (clock='1'and clock'event)then
         current_state<= next_state;
      end if;
   end process;

   states_eval1: process(clock,start)
   begin
      if (start = '0') then
          scs_ram<='1';
-- considera a subida do clock
      elsif (clock='0'and clock'event)then
       if(sbus_busy= '1') then
            scs_ram<='0';
         else
          scs_ram<='1';
         end if;
      end if;
   end process;

end tpo;
```

# Referências Bibliográficas


[1] D. Estrin, R.Govindan, Jheidemann, USC/Information Sciences Intitute, "*Scalable Coordinationin sensor Networks*", //www.lecs.cs.ucla.edu/~estrin/,1999.

[2] J. M. Kahn, R. H. Katz, K. S. Pister, EECS, U. C. *"Emerging Challenges Mobile Networking for Smart Dust"*.

[3] Berkeley, //www.eecs.berkeley.edu/~jmk/,1999.

[4] Warneke, Brett, Bhave, Sunil, Berkeley Sensor and Actuator Center. *"Smart Dust Mote Core Architecture"*.

[5] UC. Berkeley Research Group directory *"Largest Tiny Network Yet",* webs.cs.berkeley.edu/800demo/.

[6] S.,Biagioni Edoardo, Brigdes, K. W, *"A Remote Ecological Micro-Sensor Network"*, Departament of information and Computer Sciences, University of Hawaii at Manoa and Department of Botany, University at Manoa.

[7] ChemSense team *"A Proposal to Develop and Evaluate a Gas Sensor Network System for the Protection of Public Spaces*", //www.isr.ummd.edu/Labs/CACSE/chemsense/.

[8] S.,Sasa, P., Miodrag,*"Power Efficient Organization of Wireless Sensor Networks"* Computer Science Department, Univertsity of California, Los Angeles.

[9] Chandraksan, Anantha, *"Dynamic Power Management in Wireless Sensor Networks"* Massachusetts Intitute Techonology.

[10] Spartan-IIE datasheet, Xilinx components, http://www.xilinx.com.

[11] B. Atwood, B. Warneke, K.S.J. Pister, *"Preliminary Circuits for Smart Dust",* Proceeding of the 2000 Southwest Symposium on Mixed-Signal Design, San Diego, California, February 27-29,200.

[12] J. M. Kahn, R. H. Katz, K. S. J. Pister, *"Mobile Networking for Smart Dust"*, ACM/IEEE Intl. Conf. On Mobile Computing and Networking, Seattle, WA, August 17-19,1999.

[13] A. L. Zuquim, L. F. M. Vieira, M. A. Vieira, H. S. Carvalho, J. A. Nacif, C. N. Caludionor, D. C. Silva, A. O. Fernandes, "*Power Management for Communication Intensive Real-Time embeddes Systems",* Computer Science Department, Universidade Federal de Minas Gerais, MG, Brazil.

[14] N.McKay, *"Debugging Techniques for Dynamically Reconfigurable Hardware"*, Dept. Computing Science, The University of Glasgow, UK and S.Singh, Xilinx Inc San Jose, California,U.S.A.

[15] Crossbow Smarter Sensors in silicon: http://www.xbow.com/Products/Wireless_Sensor_Networks.htm.

[16] *"TinyOS",* Berkeley WEBS: Wireless Embedded Systems, http://webs.cs.berkeley.edu.

[17] *"Specification of the Bluetooth System",* Volume 1, Core, Version 1.1. Bluetooth SIG, February 22 2001, http://www.bluetooth.com.

[18] *"Specification of the Bluetooth System",* Volume 2, Core, Version 1.1. Bluetooth SIG, February 22 2001, http://www.bluetooth.com.

[19] *"MOTES - Tiny, Wireless, Networked Bluetooth Sensors Series"*, CROSSBOW and UC Berkeley EECS Dept., USA 2001, on http://tinyos.millennium.berkeley.edu/.

[20] *"Specifications of the Bluetooth System ",* Bluetooth Special Interest Group,, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Profiles",December 1999.

[21] K Negus, A Stephens and J Lansford, *"Wireless LAN MAC and PHY Specifications"*, 1999, IEEE 802.11 standard.

[22] "*HomeRF: Wireless Networking for the Connected Home"*, IEEE Personal Communications, Feb 2000, pp. 20-27

[23] J Haartsen, *"The Bluetooth Radio System"*, IEEE Personal Communications, Feb 2000, pp. 28-36.

[24] *"Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)",* IEEE Standard 1164.

[25] G. Koch, U.Kebschull, W. Rosenstiel,*"Debugging of Behavioral VHDL Specifications by Source Level Emulation"*.

[26] "HOBO® Micro Station Data Logger", Onset Computer Corporation, http://www.onsetcomp.com.

[27] *"HOBO Weather Station Soil Moisture Smart Sensor",* Onset Computer Corporation, http://www.onsetcomp.com.

[28] *"Sensirion Humidity & Temperature Sensmitter SHT1x/SHT7x",* Onset Computer Corporation, http://www.onsetcomp.com.

[29] *"PT2X Submersible Pressure/Temperature Smart Sensor",* INW Instrumentation Nortwest, Inc, http://www.inwusa.com.

[30] *"Smart Absolute Pressure Transducer LG1237",* Honeywell, http://www.ais.honeywell.com.

[31] *"Smart Sensor Systems Technology",* Battelle, http://coretechnology.battelle.org/smart.stm.

[32] *"Heterogeneous Sensor Netwoks",* Intel Tecnology, http://www.intel.com/research/exploratory.

[33] M. Vacher, D. Istrate, L. Besacier, E. Casteli, J. François, *"Smart Audio for Telemedice"*, CLIPS- IMAG, International Researchy Center MICA, Vietnan.

[34] H. Kopetz, M. Holzmann, W. Elmenreich, *"A universal Smart Transducer Interface: TTP/A",* Insitut fur Technische Informatik, Technische Universitat Wien, Austria.

[35] J. Feng, Computer Science Dept, University of California, M. Potkonjak, Computer Science Dept. University Of California, Los Angeles, "*Power Minimization by Separation of Control and Data Radios"*.

[36] E. J. Shriver and K. A. Sakallah, *"RAVEL: assigned-delay compiled-code logic simulation"* in Proceedings of the 1992 IEEE International Conference on Computer-Aided Design, pp. 364-368, Nov. 1992.

[37] S. L. Coumeri and D. E. Thomas, "*Benchmark descriptions for comparing the performance of Verilog and VHDL Simulato"*, in Proceedings of the 1994 International Verilog HDL Conference, pp. 14-16, Mar. 1994.

[38] Prakash Rashinka, *"System-on-a-chip Verification. Methodology & Techniques",* et al, Kluwer Academic Publishers, 2001.

[39] IEEE Standard Association. http:// www.standards.ieee.org.

[40] Zigbee Alliance. Zigbee Standard : http:// www.zigbee.com.

[41] NASA : http:// www.sensorwebs.jpl.gov.

[42] Luiz Felipe Menezes Vieira, "*Midleware para sistema embutidos e redes de sensores"*, Tese de dissertação de mestrado, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil, 2004.

[43] Marcos Augusto M Vieira, “Bean: a Computer Plataform for Wireless Sensor Network”, Dissertação de Mestrado, UFMG, 2004.

[44] Chipcon : http:// www.chipcon.com.

[45] Processadores ARM: http://www.arm.com.

[46] Processadores MIPS: http://www.mips.com.

[47] Processadores Sparc: http://www.sparc.com.

[48] Bhandarkar, Dileep and Douglas W. Clark. *"Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization”*, September 1991: 310-319.

[49] IEEE Standards Association: http://standards.ieee.org/getieee802/802.15.html.

[50] SENSORNET: http://www.sensornet.dcc.ufmg.br.

[51] Texas Instruments: http://www.ti.com.

[52] TIA Standard: http://www.tiaonline.org.

[53] P.J. Ashenden, *“The Designers Guide to VHDL”*, Morgan Kaufmann Pusblishers INC, 1996.

[54] Jim Burns, Adan Donlin, Jonathan Hogg, Satnam Singh, Mark de Wit, The Department of Computing Science, University of Glasgow, *“A Dynamic Reconfiguration Run-time System”*.

[55] Nicholas McKay, Dept. Computing Science, The University of Glasgow, UK, Satnam Singh, Xilinx Inc., São José, Califórnia, USA, *“Debugging Techniques for Dynamically Reconfigurable Hardware”*.

[56] Uyless D. Black, *"The V Series Recommendations: Standards for Data Communications over the telephone Network"*.

[57] PCMCIA specification: www.pcmcia.org.

[58] Xilinx Integrated Software Environment (ISE) 6.1i software, www.xilinx.com.

[59] Digikey: www.digikey.com.

[60] LM70, SPI/MICROWIRE 10-bit plus Sign Digital Temperatura Datasheet, www.national.com.

[61] AT17LV002, FPGA Configuration Memory Datasheet, Ver. 3020 A-CNFG-0502, www.atmel.com.

[62] MICROWIRE Serial, Application Note 452, January 1992, www.national.com.

[63] Tommy Klevin, *"Get RealFast RTOS with Xilinx FPGAs"*, Xcell Journal Online Article, Feverary 2003. www.xilinx.com.

[64] David Kalinsky, Roee Kalinsky, *"Introduction to Serial Peripheral Interface"* – SPI, www.embedded.com.

[65] RFM RF Monolithics, Inc: http://www.rfm.com.

[66] Protel: http://www.protel.com.

[67] Anthes John, *"OOK, ASK and FSK Modulation in the Presence of an Interfering signal"*, RF Monolithics, Dallas, Texas.

[68] Crossbow, Mica2 Mote: http://www.crossbow.com.

[69] Telos, módulo nó sensor baseado no padrão IEEE802.15.4: http://www.moteiv.com.

[70] Steven A. Groenemeyer, Alan L. bride, *"MSK and Offset QPSK Modulation"*, IEEE Transactions on Communications, vol.24, no. 8, August 1976 pp. 809-820.

[71] ModelSim XE II V5.6, Model Technology, revision 2003.03, March 2003.

[72] JEDEC: www.jedec.org.

[73] *"Standard Test Access Port and Boundary-Scan Architecture",* IEEE Std 1149.1-1990, www.ieee.org.

[74] *"Xilinx In-System Programming using an Embedded Microcontroller",* Application Note, XAPP058 (v3.0) January 15, 2001.

[75] Stolzman Richard G., *"Understanding Asseretion-Based Verification",* Verplex Systems, Inc., www.techonline.com.

[76] *"Data Generation and Configuration for Spartan Series FPGA",* Application Note, XAPP126 (v1.2) July 22, 2003.

[77] Cadence, Verilog family, Reference Manual, Vol 1, Version 1.6, May 1991.

[78] MSP430x, Mixed Signal Microcontroller Datasheet, SLAS272E, July 2000, www.ti.com.

[79] *"The Low-Cost, Eficient Serial Configuration of Spartan FPGAs",* Application Note, XAPP098 (v 1.0) November 13, 1998.

[80] MicroSim Schematic, Evaluation Version 8.0, July 1997, MicroSim Corporation.

[81] Atmel: www.atmel.com.

[82] E28F320J3A, 3 Volt Intel StrataFlash Memory Datasheet, www.intel.com.

[83] TC55W800, Toshiba Memory Datasheet, www.toshiba.com.

[84] Farnel: www.farnell.com.

[85] MAX3243, 3-V to 5.5-V Multichannel RS-232 Line Driver/Receiver Datasheet, SLLS350, April 1999, www.maxim-ic.com.

[86] QS34X245, High-speed CMOS Quickswitch 32-Bit Multiwidth Bus Switches Datasheet, Integrated Device Tecnology – IDT, Inc.

[87] TPS60130, High Efficiency Charge Pump DC/DC Converters Datasheet, SLVS258, November 1999, www.ti.com.

[88] MC34063, *"Theory and Application of MC34063 and μA78S40 Switching Regulator Control Circuits"* Application Note, AN920/D, www.onsemi.com.

# Glossário

| | |
|---|---|
| ACLK | Auxiliary Clock |
| ADC | Analog to Digital Converter |
| ASCII | American Standard Code II |
| ASIC | Application Specific Integrated Circuit |
| ASK | Amplitude Shift Key |
| ATA | AT Attachment |
| BIT | Bitstream |
| BGA | Ball Grid Array |
| bps | Bits por Segundo |
| CI | Circuito Integrado |
| CISC | Complex Instruction Set Computing |
| CF | CompactFlash |
| CFI | Common Flash Interface |
| CLB | Configurable Logic Block |
| CLK | Clock |
| CMOS | Complementary Metal Oxide Semiconductor |
| CPLD | Complex Programmable Logic Device |
| CPU | Central Processing Unit |
| CRC | Cyclic-Redundacy Check |
| DCLK | Data Clock |
| DCO | Digital Controlled Oscillator |
| DIN | Data Input |
| DIO | Data Input/Output |
| DLL | Delay Locked Loop |
| DOUT | Data Output |
| DSSS | Direct-Sequence Spread Spectrum |
| ECC | Error Correcting Code |
| EIA | Eletronic Industry Association |
| E/S | Entrada e Saída |
| EPROM | Eraseble and Programmable ROM |
| EEPROM | Electrically Erasable Programmable ROM |

| | |
|---|---|
| FET | Field Effect Transistor |
| FPGA | Field Programmable Gate Array |
| FSK | Frequency Shift Key |
| GPS | Global Positioning System |
| HDL | Hardware Description Language |
| HEX | Hexadecimal |
| IDE | Integrated Drive Electronics |
| IEEE | Institute of Electrical and Electronic Engineers |
| IF | Intermediate Frequency |
| IOB | Input/Output Blocks |
| ISM | Industrial Scientif and Medical |
| ISP | In-System Programming |
| JEDEC | Joint Electronic Devices Engineering Council |
| JTAG | Joint Test Action Group IEEE 1149.1 |
| LC | Logic Cell |
| LCD | Liquid Crystal Display |
| LDO | Low-Drop Output |
| LNA | Low-Noise Amplifier |
| LSB | Least Significant Bit e Least Significant Byte |
| LUT | Look-Up Table |
| MAC | Medium Access Control |
| MCLK | Main Clock |
| MENS | Micro Electro Mechanical Systems |
| MSB | Most Significant Bit e Most Significant Byte |
| MSOP | Mounted Mini Small Outline Package |
| Ni-Cd | Nickel-Cadmium |
| Ni-MH | Nickel Metal Hydride |
| NRZ | Non Return to Zero |
| OOK | On/Off Key |
| O-QPSK | Offset-Quadrature Phase Shift Keying |
| PA | Power Amplifier |
| PALE | Programming Adress Latch Enabled |
| PCB | Printed Circuit Board |
| PCLK | Programming Clock |

| | |
|---|---|
| PCMCIA | Personal Computer Memory Card International Association |
| PDATA | Programming Data |
| PHY | Physical Layer |
| PLL | Phase Locked Loop |
| POS | Power-On Surge |
| PROM | Programmable Read Only Memory |
| RAM | Random Access Memory |
| RANS | Reconfigurable Architecture for Network Sensor |
| RISC | Reduced Instruction Set Computing |
| RF | Radio Frequency |
| ROM | Read Only Memory |
| RSSI | Receive Signal Strength Indicator |
| RSSF | Rede de Sensores Sem Fio |
| RX | Receptor |
| SCK | Serial Clock |
| SMCLK | Sub-Main Clock |
| SMD | Surface Mounted Devices |
| SOP | Small Outline Package |
| SPI | Serial Peripheral Interface |
| SRD | Short Range Devices |
| TTL | Transistor-Transistor Logic |
| TSOP | Thin Small Outline Package |
| TSSOP | Thin Shrink Small Outline Package |
| TX | Transmissor |
| UART | Universal Asynchonous Receiver/Transmitter |
| USB | Universal Serial Bus |
| UHF | Ultra-High Frequency |
| VCO | Voltage Controlled Oscillator |
| VHDL | VHSIC Hardware Description Language |
| WSM | Write State Machine |