UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MODELAGEM EM ALTO NÍVEL DO CONSUMO DE ENERGIA PARA SISTEMA EM CHIP EM REDES DE SENSORES SEM FIO

HEIDER MARCONI GUEDES MADUREIRA

ORIENTADOR: JOSÉ CAMARGO DA COSTA

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PGEA.DM - 455/11

BRASÍLIA/DF: NOVEMBRO – 2011

FICHA CATALOGRÁFICA

MADUREIRA, HEIDER MARCONI GUEDES
Modelagem em Alto Nível do Consumo de Energia para Sistema em Chip em Redes de Sensores sem Fio [Distrito Federal] 2011.
vi, 88p., 210 x 297 mm (ENE/FT/UnB, Mestre, Dissertação de Mestrado) – Universidade de Brasília. Faculdade de Tecnologia.
Departamento de Engenharia Elétrica
1.Sistemas em Chip 2.Modelagem
3.Redes de Sensores sem Fio 4.Energia
I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

MADUREIRA, H. M. G. (2011). Modelagem em Alto Nível do Consumo de Energia para Sistema em Chip em Redes de Sensores sem Fio. Dissertação de Mestrado em Engenharia Elétrica, Publicação PGEA.DM - 455/11, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 88p.

CESSÃO DE DIREITOS

AUTOR: Heider Marconi Guedes Madureira

TÍTULO: Modelagem em Alto Nível do Consumo de Energia para Sistema em Chip em Redes de Sensores sem Fio.

GRAU: Mestre ANO: 2011

_____

Heider Marconi Guedes Madureira
SQS 410 Bloco Q Entrada B Apartamento 201 – Asa Sul
70276-170 Brasília – DF – Brasil.

RESUMO

MODELAGEM EM ALTO NÍVEL DO CONSUMO DE ENERGIA PARA SISTEMA EM CHIP EM REDES DE SENSORES SEM FIO

Autor: Heider Marconi Guedes Madureira

Orientador: José Camargo da Costa

Programa de Pós-Graduação em Engenharia de Sistemas Eletrônicos e de Automação

Brasília, Novembro de 2011

Este trabalho apresenta a modelagem em nível de transações (TLM) do consumo de energia de um sistema em chip (SoC) para aplicação em redes de sensores sem fio. Os modelos desenvolvidos permitem a execução de software embarcado desenvolvido para o SoC em uma plataforma virtual tornando possível co-projeto hardware/software. A estratégia de modelagem descrita permite avaliar tanto o consumo de energia de cada componente de hardware dos nós quanto o consumo total de cada nó.

O desenvolvimento do projeto foi realizado usando a linguagem de descrição de hardware SystemC em nível de transações. Foram desenvolvidos estudos de caso que o SoC isolado executa o algoritmo de criptografia AES, uma rede de sensores sem fio em estrela e uma rede de sensores sem fio com multihopping. Os resultados mostram a viabilidade de usar modelagem em alto nível para estimativa de consumo de energia e projeto de redes de sensores sem fio.

ABSTRACT

HIGH LEVEL MODELING OF ENERGY CONSUMPTION OF SYSTEM ON CHIP IN WIRELESS SENSOR NETWORKS

Author: Heider Marconi Guedes Madureira

Supervisor: José Camargo da Costa

Programa de Pós-Graduação em Engenharia de Sistemas Eletrônicos e de Automação

Brasília, November 2011

This work describes the transction-level modeling of the energy consumption of a system on chip (SoC) for wireless sensor networks applications. The developed models allow the execution of the embedded software designed for the SoC in a virtual platform enabling hardware/software co-design. The modeling strategy described here allow the estimation of the energy consumption of each node's hardware component as well as the total node consumption.

The design was made using the hardware description language SystemC in transaction-level. Case studies presenting an isolated SoC running the AES cryptography algorithm, a wireless sensor network in star topology and a wireless sensor network with multihop. The results show the viability of using high level modeling to estimate energy consumption and design wireless sensor networks.

Dedicatória

A todos que eu chamo de "família". Agora um pouco maior.

Heider Marconi Guedes Madureira

Agradecimentos

Quero agradecer à Prof^{a}. Dr^{a}. Katia Obraczka e ao Prof. Dr. Magdy A. Bayoumi os depoimentos colhidos. Gostaria de agradecer ao meu orientador José Camargo da Costa o apoio, dedicação e conselhos que sempre vieram para o melhor. Agradecer a toda a equipe do LPCI o apoio e convivência sempre agradável. Aos amigos José Edil Guimarães de Medeiros e Gilmar Silva Beserra por toda a ajuda e paciência durante o trabalho. Quero agradecer a todos os meus amigos pela força nas horas mais duras e pela descontração nas horas mais tranquilas. Finalmente, a meus pais por todo o suporte e apoio ao longo de toda a minha vida.

Heider Marconi Guedes Madureira

RESUMO

Este trabalho apresenta a modelagem em nível de transações (TLM) do consumo de energia de um sistema em chip (SoC) para aplicação em redes de sensores sem fio. Os modelos desenvolvidos permitem a execução de software embarcado desenvolvido para o SoC em uma plataforma virtual tornando possível co-projeto hardware/software. A estratégia de modelagem descrita permite avaliar tanto o consumo de energia de cada componente de hardware dos nós quanto o consumo total de cada nó.

O desenvolvimento do projeto foi realizado usando a linguagem de descrição de hardware SystemC em nível de transações. Foram desenvolvidos estudos de caso que o SoC isolado executa o algoritmo de criptografia AES, uma rede de sensores sem fio em estrela e uma rede de sensores sem fio com *multihopping*. Os resultados mostram a viabilidade de usar modelagem em alto nível para estimativa de consumo de energia e projeto de redes de sensores sem fio.

ABSTRACT

This work describes the transction-level modeling of the energy consumption of a system on chip (SoC) for wireless sensor networks applications. The developed models allow the execution of the embedded software designed for the SoC in a virtual platform enabling hardware/software co-design. The modeling strategy described here allow the estimation of the energy consumption of each node's hardware component as well as the total node consumption.

The design was made using the hardware description language SystemC in transaction-level. Case studies presenting an isolated SoC running the AES cryptography algorithm, a wireless sensor network in star topology and a wireless sensor network with multihop. The results show the viability of using high level modeling to estimate energy consumption and design wireless sensor networks.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos

E	Energia	[J]
t_s	Tempo dentro do simulador	$[s_{sim}]$
$Q(t)_{Restante}$	Carga Restante no tempo t	[C]
I_k	Consumo de corrente do módulo k	[A]

Siglas

ADC	Analog-to-Digital Converter
ADL	Architecture Description Language
AMS	Analog-Mixed Signal
APS	Active Pixel Sensor
ASIC	Application Specific Integrated Circuit
CMOS	Complementary Metal-Oxide-Semiconductor
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
IP	Intellectual Property Block
ISA	Instruction Set Architecture
ISM	Industrial, Scientific and Medical Radio Bands
MIPS	Microprocessor without Interlocked Pipeline Stage
QPSK	Quadrature Phase Shift Keying
RF	Rádio-Frequência
RISC	Reduced Instruction Set Computer
RoSA	Reconfigurable Stream-Based Architecture
RSoC	Reconfigurable System-on-Chip
RSSF	Redes de Sensores Sem Fio
RTL	Register Transfer Level
SCNSL	SystemC Network Simulation Library
SoC	System-on-Chip
TLM	Transaction Level Modeling

1 INTRODUÇÃO

Redes de sensores sem fio (RSSF) são um tipo especial de redes *ad-hoc* e constituem um importante ramo de pesquisa no campo da computação ubíqua. Estas redes são compostas por dispositivos denominados nós sensores que são espalhados em uma região que se deseja monitorar e/ou controlar [8]. Estes nós sensores são responsáveis por coletar informações de interesse no ambiente em que estão depositados, processá-los e disseminar esta informação.

Uma RSSF tende a ter seu projeto voltado aos requisitos da aplicação a que ela se destina. Fatores como necessidade de tolerância a falhas, custo de produção, escalabilidade da rede, meios de transmissão de dados disponíveis e tempo de operação da rede impõem requisitos específicos no projeto dos nós sensores e em todas as camadas de software embarcado. Na camada física, esquemas de modulação, potência de transmissão, sensibilidade, taxas de transmissão e projeto de baixo consumo são alguns dos requisitos. Acesso ao meio, controle de colisões e de perdas de pacotes, estratégias de controle de erros e modos de operação para economia de energia são algumas das preocupações dos protocolos da camada de enlace. A camada de rede deve ser capaz de lidar com problemas como endereçamento dos nós, escalabilidade e interface com outras redes. Outras camadas de software devem permitir o processamento local de dados (fusão, agregação, compressão, etc), disseminação de dados e segurança no tráfego de informações para dentro e fora da rede. Tendo em vista todos esses pontos, o consumo de energia de uma RSSF é dependende da aplicação e, portanto, estimar o tempo de vida de uma rede não é tarefa trivial.

A utilização de sistemas em chip (SoC) tem sido proposta como solução para o desenvolvimento de RSSF devido a características como tamanho reduzido, baixo consumo de potência e adequada capacidade de processamento. Um SoC típico integra diversos blocos funcionais, incluindo processadores, barramento, memórias, circuitos analógicos e mistos e transceptores de RF.

O desenvolvimento de soluções em hardware por si já é uma tarefa difícil. A complexidade da atividade se torna maior quando são considerados os requisitos de software e de rede durante o desenvolvimento de circuitos integrados e placas para aplicações de RSSF. Abstração é uma técnica para o projeto e implementação de sistemas complexos. Abstrair determinados detalhes que se mostram desnecessários em uma certa etapa do fluxo de projeto permite lidar com a complexidade e tratar tais detalhes em etapas mais convenientes.

A falta de um ambiente de desenvolvimento unificado que permita a análise de um sistema em diversos níveis de abstração é um problema que ainda não foi satisfatoriamente solucionado. As ferramentas de desenvolvimento utilizadas atualmente [9] [10] permitem um fluxo de desenvolvimento de protocolos de rede desconectado do fluxo de desenvolvimento de hardware e de software, sem que seja possível traçar um caminho natural entre os dois. Dentre várias abordagens possíveis para o desenvolvimento deste tipo de ferramenta, destaca-se o uso de SystemC, uma biblioteca de C++ que implementa diversas primitivas úteis para a modelagem de hardware em diferentes níveis de abstração.

O núcleo de simulação de SystemC, por sua vez, não oferece primitivas adequadas à modelagem

de sistemas em rede. A fusão de diferentes núcleos de simulação foi proposta como forma de atacar este problema, às custas de aumento da complexidade na modelagem e de baixo desempenho das simulações [11].

O uso de modelagem em alto nível para simulações de RSSF já foi demonstrado em trabalhos como [3]. No entanto, esse trabalho não levou em consideração nenhum aspecto relativo ao consumo de energia nesse tipo de rede.

Este trabalho mostra para uma modelagem em alto nível de consumo de energia capaz de simular uma RSSF qualquer de forma que o consumo de energia de cada nó possa ser estimado. Ele é parte do esforço do LDCI - UnB para projetar um sistema em chip reconfigurável (RSoC) para aplicações em RSSF.

1.1 DEFINIÇÃO DO PROBLEMA

Estudar o consumo de energia em RSSF é uma tarefa bastante difícil. Além de depender da aplicação para qual a rede foi projetada, o gasto de energia depende da topologia da rede, localização física dos nós, características do ambiente como qualidade da comunicação no meio, rotas de comunicação escolhidas entre outras variáveis muitas vezes não controláveis. A dependência de um grande número de parâmetros dificulta a estimativa do consumo de energia para uma aplicação de RSSF genérica. Desta forma, o uso de simulador para essa tarefa se mostra adequado uma vez que permite o tratamento rápido e acurado de um grande número de variáveis.

Este trabalho tem como problema central o estudo de gasto de energia em RSSF.

1.2 OBJETIVOS

Frente aos problemas identificados, este trabalho busca demonstrar que a modelagem em alto nível pode servir como ferramenta para estimar o consumo de energia em RSSF para uma aplicação qualquer. Essa forma de modelagem deve ser capaz, portanto, de simular uma RSSF e apresentar dados de consumo de energia dos nós sensores individualmente assim como a evolução das baterias da RSSF como um todo.

1. Desenvolver modelos de SoC capazes de permitir estimativas de consumo de energia;
2. Validar os modelos por meio de estudos de caso.

1.3 APRESENTAÇÃO DO TEXTO

O Capítulo 2 apresenta uma breve revisão bibliográfica sobre os temas importantes para este trabalho como RSSF, modelagem, SystemC entre outros. O Capítulo 3 contém resumos de alguns

trabalhos semelhantes para efeitos de comparação e contextualização. No Capítulo 4, está comentada a forma de modelagem do sistema em chip, da RSSF e uma proposta de metodologia de projeto para RSSF que usa modelagem em alto nível para auxiliar o projeto dessas redes. A forma de modelagem de consumo de energia está mostrada no Capítulo 5. O módulo *bateria* está detalhado assim como o desenvolvimento matemático e as hipóteses usadas na construção do modelo. A implementação do sistema em chip usado nas simulações de RSSF está descrita no Capítulo 6. Os resultados de simulações de um sistema em chip isolado e de duas RSSF de topologias diferentes estão apresentados no Capítulo 7. As conclusões estão apresentadas no Capítulo 8. Em anexo, estão relatos de entrevistas com pesquisadores da área realizados para complementar as informações (escassas na literatura) sobre as metodologias de projeto de RSSF comumente utilizadas (Anexo I). O Anexo II apresenta resultados de caracterização de hardware para RSSF da National Instruments gerados com o objetivo de testar procedimentos de aquisição experimental de parâmetros de nós sensores. Uma breve pesquisa sobre plataformas de hardware para RSSF disponíveis comercialmente está mostrada no Anexo III. Uma primeira descrição de uma ferramenta de auxílio a projeto de RSSF é feita no Anexo IV. Por fim, algumas informações sobre o CC2420 da Texas Instruments para conferência se encontram no Anexo V.

2 CONCEITOS RELACIONADOS

Nesta seção, será feita uma breve revisão dos conceitos tratados neste trabalho. O primeiro assunto abordado será Redes de Sensores Sem Fio seguido de uma exposição breve sobre hardware disponível atualmente para este tipo de rede e de descrições dos circuitos importantes para este trabalho. São apresentados, então, conceitos sobre modelagem, SystemC, TLM (do inglês, *Transaction-Level Modeling*), a linguagem de descrição de arquiteturas ArchC e a biblioteca SCNSL para simulação de redes.

2.1 REDES DE SENSORES SEM FIO

Rede de Sensores Sem Fio (RSSF) é um tipo de rede *ad hoc* em que um grande número de nós, dezenas a milhares, colaboram para executar uma função como monitorar um dado ambiente ou até mesmo controlá-lo [8]. Este tipo de solução é altamente dependente da aplicação visada e pode ser empregada sempre que o problema tratado demandar comunicação sem fio. De fato, existem trabalhos sobre RSSF publicados em áreas diversas como medicina [12], monitoramento de atletas de remo [13], monitoramento de ambientes industriais [14][15], monitoramento e detecção de incêndios florestais [16], agricultura de precisão [17], monitoramento ambiental [18][19] entre muitas outras. Entre os trabalhos citados, pode-se observar:

- Áreas diversas de aplicação;
- Requisitos de performance variadas;
- Hardwares usados com poder computacional variado;
- Diversidade no perfil de consumo de energia, desde alguns miliwatts até mais de 1W.

Observando as características das RSSF citadas acima, nota-se uma enorme heterogeneidade entre os hardwares usados, desde hardwares simples como MICA [20] até sistemas personalizados capazes de tratar imagens [16]. Como consequência das escolhas de performance do hardware escolhido, uma grande variação de gastos de energia é observado, desde alguns miliwatts até mais de 1W.

2.2 HARDWARE PARA REDES DE SENSORES SEM FIO

Apesar de existirem exemplos de RSSF que não apresentam limitação quanto ao suprimento de energia - principalmente os exemplos de monitoramento industrial, como [15] -, a disponibilidade de energia é o principal desafio para aplicações em áreas onde não há outra infraestrutura disponível. Como em aplicações em campo o hardware é geralmente alimentado por baterias, componentes

que apresentam baixo consumo são preferidos. Como consequência do baixo consumo, podem apresentar performance relativamente baixa: baixo poder computacional, banda de comunicação estreita e pouca memória.

Neste trabalho, o termo *nó* se refere à entidade de rede enquanto o termo *mote* se refere ao conjunto de hardware que será usado.

Nós de RSSF que são alimentados por bateria são geralmente compostos por uma unidade de processamento, memória, transceptor RF, sensores, por ventura atuadores, todos escolhidos com base no consumo de energia como mostrado na Figura 2.1. Esta é uma solução de compromisso entre a complexidade das aplicações possíveis e o tempo de vida da rede que usará este hardware. De fato, as plataformas de hardware que serão mostradas neste trabalho seguem esta regra e entre elas está a CrossBow TelosB [21].

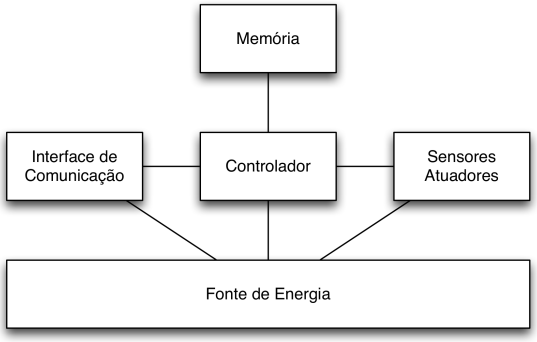

Figura 2.1: Principais componentes de um nó para RSSF [1].

A TelosB [21] é apresentada em diversos trabalhos como sendo a plataforma de hardware mais usada atualmente. Ela é composta por um processador Texas Instruments MSP430 [22], que dispõe de ADC e temporizadores integrados, e um transceptor Texas Instruments CC2420 [2] que opera em 2,4GHz com taxa de comunicação de até 250kbps. Mais detalhes deste componente estão comentados na Seção 2.2.2.

O MICAz [23] da CrossBow é composto por um processador ATmega 128L [24], que também dispõe de ADC e temporizadores integrados, e um transceptor Texas Instruments CC2420 [2]. O processador usado neste mote consome 8mA quando ligado, frente a algumas centenas de microamperes do MSP430.

A National Instruments possui uma solução para implementação rápida de RSSF. O hardware é composto por nós sensores com MSP430 e XBee [25]. A comunicação com o *gateway* é feita usando um módulo que usa um FPGA. Toda a programação do comportamento da rede é feita via LabView. Esta solução apresenta tempo de desenvolvimento curto mas pouco versátil uma vez que a maioria das configurações do nó sensor é feita automaticamente pelo LabView. Um trabalho de caracterização desse hardware está descrito no Anexo II.

A plataforma de hardware iSense [26] é composta por um controlador RISC de 32 bits de 16MHz e transceptor RF compatível com IEEE 802.15.4 fornecendo taxas de 250kBits/s. Apesar dos autores não apresentarem o modelo dos componentes usados, as especificações se encaixam com o Texas

MSP430 e o Texas CC2420.

Uma alternativa para monitoramento residencial e industrial é apresentada em [27]. Essa plataforma usa controlador Texas MSP430 e um transmissor na frequência de 27MHz. A comunicação sem fio se baseia em usar a fiação do edifício como antena. Com a frequência muito baixa, o consumo do hardware alcança 65μW permitindo alimentação com baterias tipo moeda.

O uso de plataformas de hardware como as comentadas acima representa uma solução de compromisso entre a otimização do projeto do hardware para a RSSF e o tempo de projeto.

2.2.1 Arquitetura MIPS

A palavra arquitetura define uma máquina abstrata, e não uma implementação real dessa máquina. Em geral, uma arquitetura de processador consiste em um conjunto de instruções (ISA, *Instruction Set Architecture*) e conhecimento acerca dos registradores definidos para a máquina. Atualmente a arquitetura MIPS é definida pelas arquiteturas MIPS32 e MIPS64 publicadas pela MIPS Technologies Inc. [28][29][30]. Neste trabalho, o processador usado para compor o hardware do nó de RSSF será um MIPS 32.

MIPS32 é baseada numa arquitetura *load/store*. Neste tipo de arquitetura, a maioria das instruções assume que seus operandos estão armazenados em registradores. As únicas instruções que acessam a memória são *load*, que move um dado de uma endereço de memória para um registrador, e *store*, que move um dado armazenado em um registrador para um endereço na memória.

2.2.1.1 Registradores

A arquitetura MIPS provê 32 registradores de propósito geral, um contador de programa (PC, *Program Counter*) e dois registradores de uso especial(HI e LO). Todos os registradores da arquitetura são de 32 bits e identificados como `$0, $1, $2` ... `$31` na linguagem *assembly* da máquina. Dois dos registradores de propósito geral são reservados para funções especiais [28]:

- O registrador `$0` armazena sempre o valor zero. Sempre que o programador necessitar da constante zero pode utilizar este registrador como fonte. Qualquer dado gravado neste registrador será descartado pelo processador.
- O registrador `$31` é usado para armazenar o valor de retorno da instrução de chamada de função (`jal`)

O registrador PC aponta para a próxima instrução que deve executar. A cada ciclo de execução, o processador lê na memória a instrução endereçada por PC. Depois de cada acesso, PC deve ser automaticamente incrementado. Como na arquitetura MIPS cada instrução tem exatamente 4 bytes de comprimento, o registrador PC é incrementado por 4. Este registrador não pode ser endereçado e somente é alterado pelo próprio processador ou pelas instruções de controle de fluxo.

Os registradores HI e LO são utilizados para armazenar os resultados das instruções de multiplicação e divisão que podem gerar inteiros maiores que 32 bits. Em operações de multiplicação, HI

e LO armazenam o resultado de 64 bits. HI armazena os 32 bits mais significativos e LO os 32 bits menos significativos. Em operações de divisão, o quociente de 32 bits é armazenado no registrador LO e o resto da operação é armazenado no registrador HI. Os registradores HI e LO também não são diretamente endereçáveis, devendo ser acessados por meio de instruções específicas [28].

Apesar de não ser um requisito da arquitetura, foi estabelecida uma convenção de uso dos registradores de propósito geral da arquitetura MIPS. Esta convenção permite que diferentes programadores/compiladores estruturem o código *assembly* de maneira consistente para promover interoperabilidade. A Tabela 2.1 resume a função de cada registrador da arquitetura [31].

Tabela 2.1: Registradores da arquitetura MIPS e sua convenção de uso.

Nome do registrador	Número	Uso	Preservado entre chamadas de funções
`zero`	0	Constante zero	N/A
`$at`	1	Reservado para o montador	Não
`$v0, $v1`	2, 3	Resultado de uma função	Não
`$a0` — `$a3`	4 — 7	Argumentos 1 — 4	Não
`$t0` — `$t7`	8 — 15	Temporários	Não
`$s0` — `$s7`	16 — 23	Temporários salvos	Sim
`$t8, $t9`	24, 25	Temporários	Não
`$k0, $k1`	26, 27	Reservados para o sistema operacional	Não
`$gp`	28	Ponteiro para a área de memória global	Sim
`$sp`	29	Ponteiro para o topo da pilha (*Stack Pointer*)	Sim
`$fp`	30	Ponteiro do quadro (*Frame Pointer*)	Sim
`$ra`	31	Endereço de retorno de uma função	N/A

O registrador `$sp` é utilizado para implementar um pilha na memória pois a arquitetura MIPS não provê instruções para implementação de uma pilha. O compilador MIPS não utiliza um ponteiro de quadro. Dessa forma, o registrador `$fp` é utilizado como temporário salvo `$s8`. O registrador `$ra` armazena o valor de retorno de uma chamada de procedimento. O registrador `$gp` aponta para a área de de memória que armazena constantes e variáveis globais. O registrador `$at` é geralmente usado pelo compilador para traduzir pseudo-instruções (instruções não definidas pela arquitetura mas suportadas pelo montador).

2.2.2 CC2420

O CC2420 é um transceptor RF que opera com portadora na banda ISM 2,4GHz e tem o protocolo de controle de acesso ao meio descrito no padrão IEEE 802.15.4 implementado em hardware [2]. Esse chip trabalha com estratégia de espalhamento espectral e modulação O-QPSK para prover até 250kbps. O circuito dispõe ainda de suporte em hardware para criptografia e indicador de qualidade do sinal. O acesso aos registradores de configuração e de dados é feito usando uma interface

SPI, o que permite fácil conexão com microcontroladores.

Este transceptor, fabricado pela Texas Instruments, tem aplicações em sistemas *ZigBee* [32], automação predial e residencial, controle industrial. Ele é amplamente usado em aplicações de RSSF devido ao seu baixo consumo de energia e baixo custo além de prover uma solução para comunicação quase totalmente integrada em um único componente. A Figura 2.2 mostra o diagrama de blocos desse transceptor.

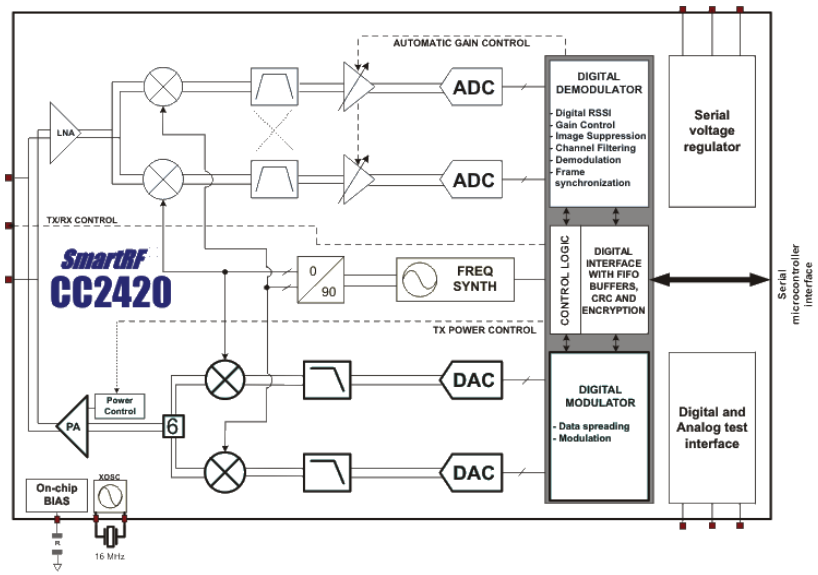

Figura 2.2: Diagrama de blocos do transceptor RF CC2420. [2].

O circuito dispõe de controle de potência de transmissão em oito níveis, desde -24dBm até 0dBm, e o estagio de recepção apresenta sensibilidade de -95dBm. Estão listados a seguir alguns dados de performance importantes do componente.

- Consumo de corrente no modo recepção: 18,8mA
- Consumo de corrente no modo transmissão para potência de 0dBm: 17,4mA
- Faixa de temperatura de operação: -40°C até 85°C
- Consumo de 0,02 μA quando desligado

Para o funcionamento do circuito são necessários alguns componentes passivos para fazer o casamento de impedância entre o circuito e a antena e um cristal de referência de 16MHz. No Anexo V estão disponíveis informações adicionais sobre esse circuito integrado.

2.2.3 Outros Blocos de Hardware Importantes para Este Trabalho

Circuitos integrados contendo memória e temporizadores podem ser úteis caso não estejam disponíveis em quantidade suficiente para a aplicação desejada apesar de microcontroladores atuais disporem desses recursos integrados.

Uma possibilidade para expansão de quantidade de memória SRAM é o chip HM62256A [33] fabricado pela Hitachi. Esse chip dispõe de 32kpalavras de 8 bits e é fabricado em tecnologia CMOS 0,8μm, bastante antiga para projetos de memória. Devido à tecnologia de fabricação, o chip apresenta consumo de 33mA e tempo de acesso de 85ns.

O circuito integrado ST M25P80 [34] apresenta uma opção mais moderna para o caso de necessidade de uso de memória externa. Esse componente dispõe de 8Mbits de memória flash podendo operar em frequências de até 25MHz. Por ser mais moderno, o consumo de corrente é de 15mA.

Para o caso da necessidade da adição de temporizador ao hardware uma possibilidade é o LMC555 [35] da National Semiconductors. Neste circuito, o período dos eventos gerados é definido por componentes externos e, portanto, fixo para um dado hardware. Este circuito apresenta consumo de 100μA sempre que ligado.

2.3 SISTEMA EM CHIP - SOC

Com o avanço da tecnologia de fabricação VLSI (Very Large Scale Integration), é possível integrar bilhões de transistores em um único chip. SoCs são caracterizados por conter uma grande variedade de componentes: lógica, memória, processadores, circuitos analógicos, etc. Seu projeto é desafiador tanto pela diversidade dos elementos quanto pela limitação de tamanho dos chips.

Para melhorar a produtividade, são utilizados módulos IP (Intellectual Property), que são componentes pré-projetados que podem ser de dois tipos: hard IP, que é um layout completo disponível, e soft IP, que é um módulo sintetizável modelado em uma linguagem de descrição de hardware como VHDL ou Verilog. Para isso, os IPs devem ser bem documentados e testados, e devem possuir uma interface padronizada que permita a sua integração ao SoC.

Processadores embarcados têm sido bastante utilizados em SoCs, já que muitas aplicações são implementadas em software. Além disso, muitos sistemas complexos usam sistemas operacionais para gerenciamento de arquivos e de rede.

Algumas características desejáveis para SoCs são tamanho reduzido, baixo consumo de energia, alto desempenho, baixo custo e robustez. Para tanto, a tecnologia CMOS é adequada devido às suas características, entre as quais podem-se citar: alta capacidade de miniaturização, tensão de alimentação baixa (gera menor consumo de potência), custo reduzido, e integração do sistema facilitada pelo fato de todos os componentes poderem ser fabricados na mesma pastilha.

A Universidade de Brasília participa do projeto de um sistema em chip reconfigurável (RSoC). Diz-se que um SoC é reconfigurável quando o sistema dipõe de componentes de hardware reconfiguráveis integrados [36]. O RSoC em desenvolvimento é composto por um processador, barramento, memória, seção reconfigurável, sensor de imagens, conversor A/D, transceptor RF, interfaces digitais entre outros componentes como ilustrado na Figura 2.3.

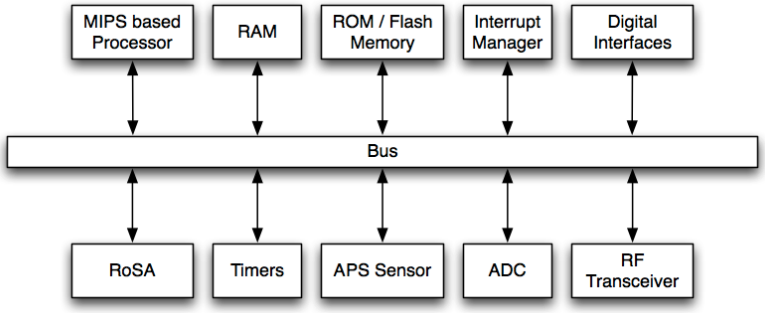

Figura 2.3: Diagrama de blocos do RSoC [3].

2.4 MODELAGEM

A descrição de sistemas em diferentes níveis de abstração constitui sempre um balanço entre velocidade e acurácia da simulação. É importante conhecer os extremos possíveis para a descrição do sistema, pois o fluxo de projeto, em geral, flui através desses extremos [6].

No início do fluxo de projeto encontra-se a modelagem em nível de algoritmos. Neste nível, não existe a noção de software ou hardware. Não são modelados registradores nem detalhes de sincronização relacionadas à arquitetura do sistema. Apenas a funcionalidade dos algoritmos do sistema é avaliada. Apesar de não permitir a execução do software embarcado nem a análise da arquitetura do sistema em desenvolvimento, a descrição neste nível de abstração permite melhor performance de simulação quando comparada com descrições mais detalhadas.

No extremo final do fluxo de projeto encontra-se a modelagem em nível RTL (do inglês *Register-Transfer Level*). A descrição neste nível é realizada em uma linguagem de descrição de hardware (HDL) e envolve a descrição fiel de todos os detalhes necessários para a síntese do hardware. A característica relevante desse nível de descrição é a análise minuciosa da arquitetura de hardware e avaliação precisa da performance final. O contraponto é o tempo necessário para simular o sistema tornando inviável a execução de software embarcado. Dessa forma, o hardware só pode ser integrado ao software para teste em uma etapa posterior quando um protótipo encontra-se disponível. Uma modificação no sistema nessa etapa será muito mais custosa do que as realizadas em etapas inicias do projeto.

Um nível de modelagem intermediário deve ser capaz de promover o desenvolvimento de software em etapas iniciais e permitir ao projetista de hardware explorar o espaço de projeto da arquitetura do sistema. Para isto, um modelo neste nível deve ser capaz de permitir a execução de milhões de ciclos em um espaço de tempo razoável que permita várias rodadas de execução de software durante um turno de trabalho. Deve também prover acurácia suficiente para a execução do software embarcado mas escondendo detalhes desnecessários para não tornar as simulações demasiadamente demoradas. Por fim, o esforço necessário para desenvolver este tipo de modelo deve ser considerado baixo em comparação ao desenvolvimento de modelos RTL. A modelagem em nível TLM (do inglês *Transaction-Level Modeling*) é considerada adequada para solução desses três requisitos [6].

TLM simplifica a tarefa de modelagem abstraindo os mecanismos de comunicações entre os

diferentes componentes do sistema. São utilizadas chamadas de funções entre os componentes para representar transações dentro do sistema em vez da da modelagem dos sinais e da temporização. Transações podem conter informações sobre o tempo de início e fim e os dados transmitidos [37].

A modelagem em nível TLM define como será realizada a comunicação entre os diversos componentes do sistema, comportando qualquer nível de acurácia na modelagem da funcionalidade do sistema. Dessa forma, cabe ao projetista definir a quantidade de informação disponível no modelo de acordo com o compromisso existente entre acurácia dos modelos, tempo de desenvolvimento e tempo de simulação. Neste trabalho, serão definidos dois níveis de acurácia para os modelos de componentes ligados em rede: o nível funcional e o nível arquitetural.

Definimos o nível funcional como aquele em que todos os detalhes de hardware e software são abstraídos em uma única entidade. Por exemplo, cada nó da rede é composto por processador, barramento, memória e outros blocos, além do software embarcado. No nível funcional, cada nó é modelado como uma entidade que realiza as tarefas programadas mas sem expor os detalhes do hardware ou do software. A comunicação entre os diversos nós é modelada por meio de chamadas de funções.

Definimos o nível arquitetural como aquele em que são descritos detalhes de hardware suficientes para execução do software embarcado. Há uma clara diferenciação entre hardware e software. A comunicação entre os diversos blocos de hardware é feita por meio de chamadas de função bem como a comunicação entre diversos nós de uma rede.

2.5 SYSTEMC

C++ é uma linguagem orientada a objetos que permite a abstração de dados e a elaboração de projetos hierárquicos. SystemC [38] é uma biblioteca do C++ utilizada para a descrição de hardware que contempla níveis sistêmicos de abstração de projeto. Por ter mecanismos que permitem a modelagem em níveis mais altos de abstração, é mais do que uma simples linguagem de descrição de hardware. O uso de SystemC permite a descrição de hardware e software em um ambiente homogêneo, facilitando o processo de compreensão, descrição e validação de projeto. Esta linguagem facilita o processo de modelagem de hardware por possuir características como tipos de dados próprios para definição de hardware (*sc_lv*), estruturas (*sc_module*) e processos (*sc_method*) que flexibilizam a descrição de paralelismo natural em hardware. A base fornece um núcleo de simulação orientado a objeto e qualquer programa pode ser compilado usando um compilador C++ e produzir um arquivo executável [37].

Uma boa maneira de entender a estrutura da linguagem é por meio de um exemplo. Para tanto, será utilizada a Figura 2.4 que mostra os principais componentes de SystemC utilizados neste trabalho.

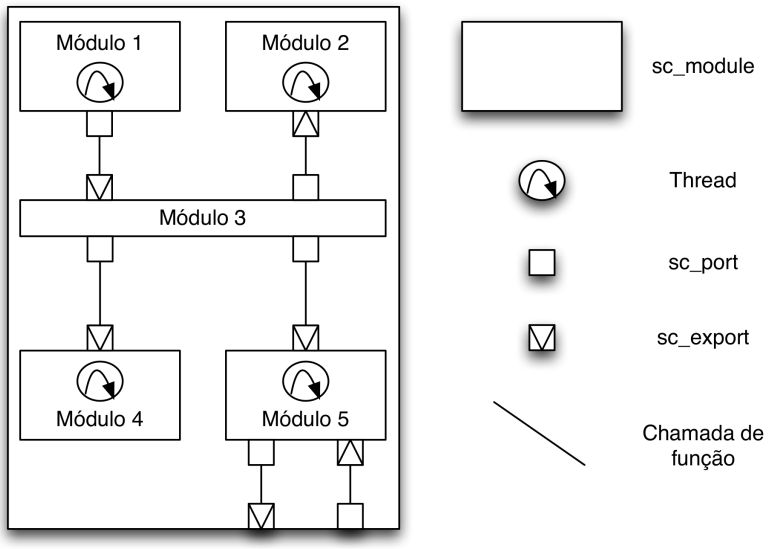

Figura 2.4: Principais componentes de SystemC [3].

2.5.1 Fluxo de simulação

Um simulador programado em SystemC tem duas fases principais de operação: elaboração e execução. Uma terceira fase pode ser definida no fim da execução de uma simulação como pós-processamento [4].

Como todo programa escrito em C++, um simulador escrito em SystemC inicia a execução do código pela função `main()`. Esta é definida pelo núcleo de simulação de SystemC e realiza diversas tarefas ligadas ao simulador. A execução do código do usuário inicia pela função `sc_main()`, chamada automaticamente pelo núcleo do simulador.

Todo o código executado antes da chamada da função `sc_start()` compreende a fase de elaboração do simulador. Esta fase é caracterizada pela declaração de instâncias dos módulos, estabelecimento da conectividade e preparação para a fase de execução.

A fase de execução do simulador se inicia com uma chamada ao método `sc_start()`, definido pelo núcleo de SystemC. A chamada a este método passa o controle da simulação para o núcleo de simulação de SystemC, responsável por organizar a execução dos processos e criar a ilusão de concorrência.

A Figura 2.5 mostra o fluxo de uma simulação em SystemC. Após a chamada a *sc_start()*, todos os processos são chamados durante a inicialização. A ordem de chamada é determinística mas não é especificada.

Após a inicialização, os processos executam quando ocorrem eventos aos quais eles são sensíveis. O núcleo de SystemC implementa um ambiente multi-tarefa não-preemptivo. Uma vez iniciado, um processo executa até que ele libere o controle novamente ao núcleo de simulação. Vários processos podem iniciar no mesmo instante de simulação. Neste caso, os processos são executados e as variáveis influenciadas por eles são atualizadas. Uma execução seguida por uma atualização é

chamada de ciclo-delta.

Se em um determinado instante de simulação nenhum processo precisar ser executado o tempo de simulação é incrementado. Quando não há mais processos pendentes, a simulação termina.

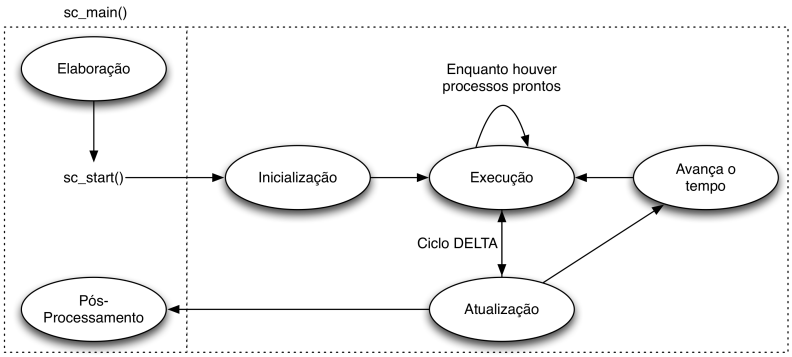

Figura 2.5: Fluxo de simulação em SystemC [4].

2.5.2 Módulos

Em SystemC, os módulos são os blocos básicos usados para particionar um projeto complexo. Um módulo típico contem portas para se comunicar com outros módulos, processos que descrevem sua funcionalidade, dados internos e outros módulos em níveis hierárquicos mais baixos.

A criação de um módulo requer herança da classe *sc_module* como mostrado no Trecho de Código 2.1. Como toda classe em C++, um módulo deve fornecer um método construtor que tenha o mesmo nome da classe. Este método será chamado na fase de elaboração da simulação e é apropriado que se ele faça a inicialização do estado interno do módulo. A macro `SC_HAS_PROCESS` deve ser utilizada para declarar o método construtor.

```
#include <systemc>
class modulo1 : public sc_module {
    public:
    SC_HAS_PROCESS(modulo1);
    void modulo1(){
        INICIALIZAÇÃO DO MÓDULO
        ...
    };
    ...
};
```

Trecho de Código 2.1: Declaração de um módulo em SystemC.

Na Figura 2.4, cada módulo teria uma declaração similar a mostrada no Trecho de Código 2.1.

2.5.3 Processos

Processos são utilizados para descrever as funcionalidades dos módulos e fornecem um mecanismo para simular o comportamento concorrente encontrado em sistemas eletrônicos. Eles constituem a unidade básica de processamento em SystemC. Os processos são métodos de C++ registrados no núcleo de simulação de SystemC e são chamados apenas por ele. Cada processo deve conter uma lista que enumera os eventos aos quais ele é sensível. Sempre que o evento listado ocorrer, a execução do processo se inicia.

Em SystemC existem 3 tipos de processos:

1. Métodos (*sc_method*): Iniciam sempre que ocorrer um evento da lista de sensibilidade. Não podem ser suspensos e executam toda a rotina contida no processo.

2. *Threads* (*sc_thread*): Iniciam no início da simulação e são executadas indefinidamente. Podem ser suspensas para permitir a execução de outros processos utilizando o método *wait()*.

3. *Clocked Threads* (*sc_cthread*): Estes processos são semelhantes às *threads*, mas são sensíveis a um evento especial (*sc_clock*). São utilizados para criar máquinas de estado implícitas, nas quais um estado é definido entre dois comandos *wait()*.

O registro de uma *thread* no núcleo de simulação de SystemC é realizado pelo uso da macro `SC_THREAD` no construtor do módulo, como mostrado no Trecho de Código 2.2.

```
#include <systemc>
class modulo1 : public sc_module {
  public:
  SC_HAS_PROCESS(modulo1);
  void modulo1(){
    SC_CTHREAD(thread);
    INICIALIZAÇÃO DO MÓDULO
    ...
  };
  void thread(){...};
  ...
};
```

Trecho de Código 2.2: Declaração de uma *thread* em SystemC.

Neste trabalho, foram utilizadas apenas *threads* para a modelagem dos módulos. A modelagem com este tipo de processo é, em geral, mais simples do que a modelagem utilizando métodos [4]. *Threads* iniciam sua execução no início da simulação e tipicamente permanecem em um laço até o fim da simulação. Isto é feito porque as *threads* executam apenas uma vez durante toda a simulação, isto é, uma vez terminadas, não tornam a executar.

Como descrito na Seção 2.5.1, o núcleo de simulação de SystemC não suspende a execução de um processo. Dessa forma, uma *thread* deve retornar o controle da execução para o núcleo de SystemC sob pena de que outros processos não possam executar. O retorno de controle por parte

da *thread* pode ser realizado de duas formas. Por um lado, a *thread* pode terminar sua execução definitivamente, simplesmente retornando um valor para o chamador. Por outro lado, a *thread* pode suspender sua execução através do método *wait()*.

2.5.4 Eventos

SystemC implementa um núcleo de simulação baseado em eventos. Eventos são notificações do tipo *sc_event* e acontecem em um instante específico de tempo, não tendo duração ou valor. Um vez ocorrido um evento, não há como rastrear sua ocorrência, exceto pela observação das suas consequências.

Para causar efeitos em um processo, um determinado evento deve estar sendo observado por um processo, seja por meio da lista de sensibilidade ou por meio do método *wait()*. Neste trabalho, os eventos foram observados com o uso do método *wait()* como explicado a seguir.

Ao suspender a sua execução, uma *thread* chama o método *wait()* tendo como argumento um evento. Apesar de comportar vários tipos de eventos, neste trabalho o método *wait()* é chamado de duas formas: `wait(sc_time)` ou `wait(sc_event)`.

Na primeira forma, `wait(sc_time)`, a *thread* é suspensa por um tempo determinado. Ao término desse tempo, a *thread* é marcada novamente para execução. Na segunda forma, `wait(sc_event)`, a *thread* é suspensa por tempo indeterminado. Quando o evento indicado ocorre, a *thread* é marcada para execução.

2.5.5 Portas e Interfaces

As portas são utilizadas para ligar um módulo a outro para que os mesmos possam se comunicar. As duas portas utilizadas neste trabalho foram *sc_port* e *sc_export*. *sc_port* consiste em um ponteiro para um canal fora do módulo. *sc_export* consiste em um ponteiro para um canal dentro de um módulo.

As interfaces de SystemC são classes abstratas de C++. Classes abstratas não podem ser usadas diretamente mas servem como modelo para criação de outras classes através do mecanismo de herança. O mecanismo de polimorfismo de C++ permite que uma classe seja referenciada através da classe da qual herda. Dessa maneira, SystemC define a interface básica *sc_interface* que deve ser herdada por qualquer outra classe que define uma interface. Este mecanismo é utilizado para definir as diversas interfaces utilizadas para modelagem em nível TLM em SystemC.

A declaração de uma porta em SystemC está sempre atrelada a uma interface. Dessa maneira, a interface serve como um contrato entre duas portas, definindo quais os métodos estão disponíveis para a comunicação entre os módulos. Quando é declarada uma instância de *sc_port*, a interface informa ao módulo quais os métodos estão disponíveis para comunicação através daquela porta. Ao se declarar uma instância de *sc_export*, a interface informa ao módulo quais os métodos ele deve implementar para permitir a comunicação com outros módulos.

O Trecho de Código 2.3 mostra a declaração de portas em SystemC, utilizando como exemplo

os módulos 1 e 3 da Figura 2.4.

```
#include <systemc>
class modulo1 : public sc_module {
    public:
    SC_HAS_PROCESS(modulo1);
    void modulo1(){
        SC_CTHREAD(thread);
        ...
    };
    void thread(){...};
    sc_port<interfaceA> porta1;
    ...
};

class modulo3 : public sc_module {
    public:
    SC_HAS_PROCESS(modulo1);
    void modulo3(){
        ...
    };
    sc_export<interfaceA> porta3;
    ...
};
```

Trecho de Código 2.3: Declaração de portas em SystemC.

Existem várias formas de conectar portas em SystemC. Neste trabalho preferiu-se conectar as portas por nomes. Neste estilo, a sintaxe utilizada é: `modulo->sc_port(modulo->sc_export)`. O exemplo da Figura 2.4 é mostrada no Trecho de Código 2.4.

```
...
modulo1->porta1(modulo3->porta3);
...
```

Trecho de Código 2.4: Conexão de portas em SystemC.

2.5.6 Interfaces TLM

Para permitir o uso da metodologia TLM, SystemC provê interfaces baseadas nos conceitos de *bloqueante* ou *não-bloqueante* e *unidirecional* ou *bidirecional*.

Os dois tipo básicos de processos em SystemC, *threads* e métodos, diferem pela possibilidade de suspensão de sua execução por meio de chamadas ao método *wait()*. A chamada ao método *wait()* em um método implica em erro em tempo de execução do modelo. Por outro lado, o uso de *threads* implica em mais mudanças de contexto dentro do simulador e consequente perda de desempenho. Dessa forma, SystemC define como bloqueante uma função que pode conter uma chamada ao método *wait()* e como não-bloqueante uma função que não contém chamadas ao método *wait()*. O uso de interfaces bloqueantes implica no uso de *threads* para a modelagem dos módulos.

Por outro lado, deve observar a natureza das transferências que ocorrem em um determinado sistema. Algumas são bidirecionais por natureza como uma operação de leitura por meio de um barramento, no qual o início da transação (requisição de um dado) está fortemente acoplado ao fim da transação (leitura do dado). Por outro lado, algumas transferências são naturalmente unidirecionais, como é o caso do envio de um pacote por uma interface de rede. Qualquer transação, entretanto, pode ser dividida em diversas operações uni e bidirecionais de acordo com o nível de detalhe da modelagem. Por esse motivo, SystemC provê os dois tipos de interface.

Neste trabalho foi utilizada somente a interface bloqueante bidirecional de SystemC, definida pela classe *tlm_transport_if*, mostrada no Trecho de Código 2.5.

```
template<REQ, RSP>
class tlm_transport_if: public sc_interface {
  public:
    virtual RSP transport(const REQ&) = 0;
};
```

Trecho de Código 2.5: Declaração da interface bidirecional bloqueante em SystemC.

Neste trabalho, os módulos que iniciam as transações são chamados mestres e os que respondem as transações são denominados escravos. É importante notar que um mesmo módulo pode atuar como mestre em determinados contextos e como escravo em outros.

Nesta interface, apenas um método deve ser implementado, o `RSP transport(const RES&)`. Ele será chamado pelo módulo mestre e deve ser implementado pelo módulo escravo. Os *templates* *REQ* e *RSP* são classes quaisquer, definidas pelo usuário. A primeira, *REQ* é a classe passada pelo mestre ao escravo e a segunda, *RSP* é a classe que o escravo retorna como resposta à transação.

2.5.7 A Linguagem de Descrição de Hardware ArchC

ArchC [39] é uma linguagem de descrição de arquiteturas (ADL) híbrida, de código aberto, baseada na HDL SystemC. Ela permite o desenvolvimento de modelos funcionais de processadores em SystemC a partir de informações estruturais e do conjunto de instruções. Seu principal objetivo é permitir aos desenvolvedores de sistemas explorar e verificar uma nova arquitetura. ArchC foi projetada tendo em mente os usuários da linguagem SystemC e, por isso, sua sintaxe é totalmente baseada em C++ e SystemC.

Nas descrições de comportamentos em ArchC, existe um método de comportamento para cada instrução. Este método pode conter uma descrição mono-ciclo, para descrições puramente funcionais, multi-ciclo ou dividido em estágios de pipeline. Uma vantagem de se utilizar C++ como base para o desenvolvimento da linguagem é que isso permite utilizar qualquer código C++ válido na descrição dos comportamentos. A partir dos modelos de processadores descritos utilizando ArchC, pode-se gerar simuladores interpretados ou compilados, montadores e outras ferramentas úteis para o desenvolvimento de software embarcado.

2.5.8 Biblioteca SystemC Network Simulation Library (SCNSL)

Em fluxos de projeto que utilizando SystemC, o sistema que está sendo projetado é modelado utilizando as primitivas da linguagem (módulos, processos, portas, eventos, etc). O modelo é então simulado utilizando a distribuição gratuita de SystemC ou utilizando uma ferramenta proprietária. Para realizar simulações de sistemas em rede, novas primitivas são necessárias. A biblioteca *SystemC Network Simulation Library* [5] (SCNSL) é uma extensão de SystemC que permite a modelagem de redes baseadas em pacotes. Sua principal vantagem é permitir a descrição do sistema e do ambiente de rede em uma linguagem comum, SystemC. Outras vantagens incluem a geração de simuladores eficientes que permitem a descrição do sistema em diferentes níveis de abstração e a disponibilidade do código-fonte para livre modificação e adaptação.

Como mostrado na Figura 2.6, SCNSL divide o ambiente de simulação de rede em dois domínios. O domínio do usuário é o que contém os modelos desenvolvidos pelo projetista de sistemas de acordo com as especificidades de seu projeto. O domínio do simulador contém as estruturas fornecidas pela biblioteca para integração dos elementos em um ambiente de rede.

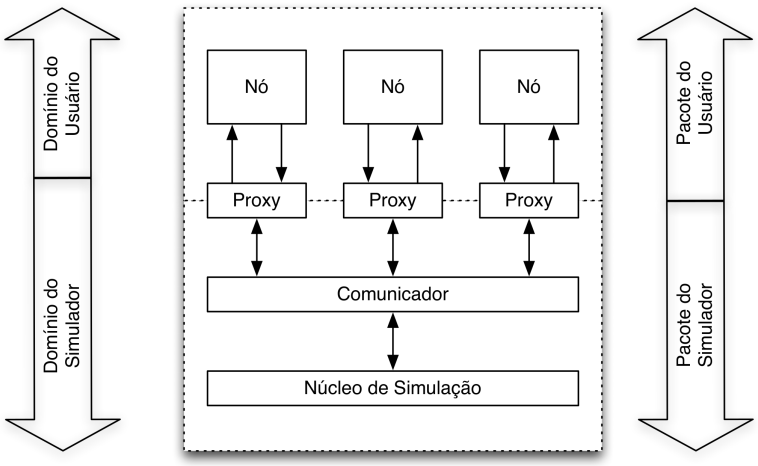

Figura 2.6: Principais componentes de SCNSL [5].

Em SCNSL, o núcleo do simulador é implementado pela classe *Network_if_t*. Essa classe é responsável por gerenciar todas as transmissões ocorridas no âmbito da rede, simular o atraso de transmissão de pacotes, atualizar parâmetros dos nós como posição, potência e taxa de transmissão, determinar quais os nós podem receber pacotes em um dado instante e determinar a ocorrência de colisões. Essa classe explora o escalonador de eventos de SystemC para realizar essas tarefas através do mapeamento de eventos da rede em eventos de System. Essa estratégia permite aproveitar todo o esforço de otimização do escalonador de SystemC e gerar simulações mais eficientes do que com a presença de dois escalonadores de eventos.

SCNSL provê a classe *NodeProxy_if_t*, responsável por desacoplar a implementação do simulador de rede da implementação dos nós. Esta classe provê uma interface bem definida entre os nós e o núcleo de simulação ao mesmo tempo que permite grande liberdade para a modelagem dos nós, permitindo ao projetista trabalhar com diferentes níveis de abstração.

SCNSL reproduz o comportamento do meio de transmissão no nível de pacotes para acelerar a simulação. É utilizada a classe *Communicator_if_t*. Esta classe pode ser estendida para modelar diversos comportamentos. De fato, os *proxies* e o próprio núcleo de simulação derivam desta classe mais fundamental. SCNSL foi desenvolvida de modo que pode-se criar uma cascata de comunicadores, cada um modelando um comportamento distinto (taxa de erro de bit, perda de pacotes, atenuação, etc). Esta abordagem permite estender o modelo da rede sem a necessidade de alterações no núcleo do simulador.

Outra divisão importante quando se utiliza SCNSL para a modelagem de sistemas em rede ocorre no conceito de pacote. Em geral, o formato dos pacotes que trafegam por uma rede depende dos protocolos utilizados e das funcionalidades providas por estes protocolos. O desenvolvimento de sistemas requer a descrição dos pacotes com acurácia de bit para teste e validação do software embarcado [6]. Simuladores de rede, por sua vez, podem abstrair estes pacotes e modelar somente endereços, comprimento dos pacotes e outras informações de maneira mais otimizada para o simulador, com campos diversos daqueles presentes no pacote real [40]. Além disso, quanto menos tipos de pacotes o simulador precisar encaminhar, mais otimizada pode ser sua implementação [5]. Dessa forma, SCNSL encapsula os pacotes definidos no domínio do usuário em um único pacote definido no domínio do simulador.

3 TRABALHOS CORRELATOS

RSSF tem aplicação potencial em diversas áreas como já discutido. No entanto, a tecnologia ainda carece de estabilidade e maturidade em muitos aspectos para que esse tipo de computação se torne popular. Atualmente, mais de uma década do início da pesquisa na área, o grande número de desafios técnicos dificulta o uso de RSSF para aplicações mais complexas do que simplesmente coletar e enviar dados [41].

Um desses desafios é a limitação quanto às fontes de energia em RSSF, comumente baterias, lançadas em ambientes onde nenhuma outra infraestrutura está disponível, o que não ocorre em [15], trabalho que descreve uma RSSF num ambiente industrial. Da limitação de energia surgem outras limitações como a baixa capacidade computacional e estreita banda de comunicação. Tendo em vista que o progresso na tecnologia de baterias foi muito mais lento do que as demandas por processamento e comunicação, a execução de operações eficientes do ponto de vista do gasto energético em RSSF se torna essencial [41]. O trabalho [41] foi publicado na Revista *Proceedings of The IEEE* Volume 98 Número 11 em Novembro de 2010. Este volume foi especialmente compilado com trabalhos sobre RSSF servindo de boa fonte bibliográfica para o assunto.

A seguir serão apresentados alguns trabalhos relevantes dos últimos 3 anos (de 2009 até 2011) na área de modelagem e simulação de RSSF. Esses trabalhos visam, como este, estudar o gasto de energia em nós de RSSF e foram encontrados modelos de consumo de energia para RSSF em diversos níveis, desde o nível de rede até trabalhos de modelagem de consumo para sistemas em chip.

O primeiro trabalho estudado trata de uma abordagem geral para estimativa de performance e consumo de energia em modelos em alto nível para sistemas em chip [42]. A ferramenta FUNTIME é apresentada e se baseia em modelos funcionais sem noção de tempo. Essa ferramenta se baseia na hipótese de existência de uma biblioteca de IPs que disponham de dados precisos de performance e consumo. De acordo com os autores, o reuso de IPs para projeto de SoCs cada vez mais complexos reduzem o tempo de desenvolvimento assim como o esforço de projeto. Num primeiro passo, simulações detalhadas dos IPs que formarão o SoC são feitas com o objetivo de levantar informações sobre performance e consumo dos módulos IP usados. Em seguida, a ferramenta FUNTIME é alimentada com dados sobre quais IPs serão usados e como eles serão conectados e informações da aplicação desejada. Com base nas informações adquiridas das simulações dos IPs e esses dados, uma simulação em nível de algoritmo é realizada e estimativas de consumo e performance são feitas. Esta ferramenta é usada para estimar consumo e performance de sistemas em chip e não pode ser usada diretamente em ambientes de RSSF. Além disso, comparações com simulações em nível de transistor mostram diferenças de 15% apesar de grandes ganhos em performance de simulação.

Já no âmbito de simulação de redes, em [43] está apresentado um modelo matemático para o gasto de energia de uma RSSF para monitoramento de gases tóxicos em uma refinaria. Esse trabalho apresenta um modelo do ponto de vista da rede em que apenas o gasto energético da comunicação é contabilizado. Um estudo de conectividade para uma rede aleatoriamente lançada é feito para

estimar a distância esperada entre os nós. O modelo é elaborado a partir desse valor, de parâmetros como tamanho dos pacotes, consumo de energia em cada modo de operação, ciclo de trabalho entre outros. No entanto, esse tipo de modelo é bastante específico e pouco preciso uma vez que todo o hardware é abstraído e apenas a comunicação estudada. Esse tipo de modelo pode ser usado em etapas iniciais de projeto de uma RSSF enquanto a rede propriamente dita está sendo projetada deixando o detalhamento do hardware para etapas mais avançadas.

Mak e Seah [44] apresentam uma discussão sobre tempo de vida da rede. Nesse trabalho, o simulador de rede GloMoSim é usado e novamente apenas o consumo da comunicação é levado em consideração. Os autores justificam que, dada a tecnologia atual, o consumo da comunicação sem fio é ordens de grandeza superiores ao da computação, o que não é verdade para todas as aplicações e nem para todos os hardwares [27]. Diversos protocolos de comunicação conhecidos, como o LEACH e o difusão direcionada, são analisados e comparados sob diversas condições e os resultados apresentados. Os autores afirmam que não existem grandes diferenças no tempo de vida para os protocolos estudados e fazem uma discussão sobre a definição de tempo de vida.

Uma metodologia para estudo de consumo de potência de um nó de uma RSSF é apresentada em [45]. Os autores alegam que a metodologia proposta é compatível com Projeto Baseado em Plataforma [46]. Esse trabalho usa SystemC para modelar um nó representado como uma máquina de estados finita cujos estados correspondem a modos de operação possíveis. A aplicação simulada no estudo de caso é a aquisição periódica de um dado do ambiente e transmissão do pacote para uma estação base e o hardware modelado é o TelosB [21] da CrossBow que é composto por um microcontrolador MSP430 e um transceptor CC2420. Os dados de energia e performance usados foram obtidos a partir da documentação do hardware ou de medidas elétricas. Esse trabalho implementa ainda um modelo de circuito para a bateria usando PSpice.

A abordagem de usar um modelo funcional, descrito como uma máquina de estados finitos, é repetido em [47]. Os autores usam o NS-2 para simular uma RSSF computando o consumo de energia novamente apenas da comunicação. A implementação de um modelo de processador é difícil devido a problemas de compatibilidade entre o NS-2 e o SystemC. As simulações são feitas baseadas no *iMote2* que dispõe de um processador Marvell PXA271 e um transceptor CC2420. A aplicação consiste em transmitir um *stream* de audio usando o protocolo IEEE 802.15.4 e comparar a performance frente a diversos tipos de codificação. A conclusão do trabalho mostra que o número de nós da rede influencia o gasto energético e, à medida que o número de nós cresce a codificação torna a aplicação mais otimizada. No entanto, não é possível avaliar o gasto de energia para codificar o áudio na simulação executada.

O simulador IDEA1 é apresentado em [48]. Esse simulador usa SystemC, TLM e a biblioteca SCNSL [5] para modelar nós ligados em rede. O uso da biblioteca SCNSL permite ganhos de performance em simulação de cerca de 2 ordens de grandeza frente ao NS-2. Nesse trabalho, os nós também são modelados como máquinas de estados finitos e o comportamento de um dado nó dividido em tarefas como processamento, configuração do ADC entre outras. Nesse modelo, cada tarefa é associada a um gasto de corrente constante e as informações sobre consumo de energia são obtidas com base em *datasheets*. O artigo traz por fim, uma lista de parâmetros desejáveis em simuladores e RSSF como escalabilidade, possibilidade de monitoramento de gasto de energia, compatibilidade

com o fluxo de projeto de sistemas embarcados entre outras. Ainda usando o simulador IDEA1, o trabalho [49] usa dados da plataforma de hardware MICAz da CrossBow, composta por microcontrolador ATMega 128 e transceptor CC2420, para fazer um estudo de caso de avaliação de gasto de energia em RSSF em topologia estrela com oito nós e um coordenador. Apresenta uma comparação de consumo de energia alterando um dos parâmetros do protocolo IEEE 802.15.4.

Um ambiente de simulação construído em SystemC nível TLM é descrito em [50]. Nesse artigo, os autores discutem que diferentes blocos de um mote de RSSF podem ser modelados em níveis diferentes e que, portanto, o uso de modelos de consumo de energia diferenciados para cada bloco de hardware modelado é uma boa alternativa para modelar o consumo do mote como um todo. A aplicação simulada consiste em medir a temperatura periodicamente e mandar um aviso somente se a temperatura superar um dado limite. A rede é composta por quarenta e nove nós, responsáveis por adquirir os dados e retransmitir pacotes de outros nós, e uma estação base, onde todos os pacotes devem chegar. Nesse trabalho, o transceptor é modelado como uma máquina de estados finitos e dados de consumo de energia são retirados de *datasheets* enquanto o processador é modelado usando um simulador de conjunto de instruções (ISS). A modelagem do processador é feita nesse nível pois o consumo depende do número de instruções executadas e acessos à memória, por exemplo. A acurácia do modelo é maior mas esse modelo apresenta maior custo computacional.

No artigo [51], um simulador para RSSF com possiblidade de estimativa de consumo de energia criado em SystemC nível TLM que usa a biblioteca SCNSL [5] é apresentado. O simulador apresentado permite a simulação de nós em vários níveis de abstração como o nível funcional, em que apenas o comportamento é modelado, e o nível de arquitetura onde os componentes de hardware são separados claramente. Em nível de arquitetura, o processador é simulado em nível ISS, que executa software de aplicação e fornece boa precisão, e o transceptor é modelado como uma máquina de estados finita. O modelo é baseado no hardware AquiGrain-2 da Philips. Esse mote é composto de um sistema em chip Texas CC2430 com núcleo de Intel 8051 e um transceptor nos moldes do CC2420. Diferenças sobre o trabalho [51] e este trabalho serão apresentadas na Seção 7.3, após a apresentação do modelo criado.

Entre todos os trabalhos analisados nota-se a forte dependência do consumo de energia com o hardware utilizado, de forma que todos os trabalhos buscam dados de consumo de algum circuito integrado conhecido e bem documentado ou obtém-se esses dados de medidas elétricas. No caso de projeto de sistemas em chip, o consumo de energia será dependente da tecnologia usada no projeto. Portanto, no projeto de uma RSSF, o consumo de energia começa a ser definido no momento da escolha da tecnologia de fabricação a ser usada nos componentes do mote e termina com a definição detalhada da aplicação. Essa percepção torna possível compreender a dimensão do problema de estudar o consumo de energia em RSSF.

Este trabalho trata de modelagem de um nó de RSSF que dispõe de infraestrutura para estimativa de gasto de energia e simulação de RSSF arbitrárias. O nó modelado neste trabalho usa:

- Um processador modelado em nível de conjunto de instruções gerado com a auxílio da ADL ArchC;
- Periféricos, como temporizador e transceptor RF, modelados como máquinas de estados fini-

tos;

- Um módulo *bateria* responsável por toda a estimativa de consumo de energia.

A decisão de modelar o hardware nesse nível foi tomada de modo a modelar consumo de energia e performance em um nível próximo do real. Apesar de exigir esforço para o desenvolvimento do modelo, esta abordagem fornece ganhos claros sendo as principais: (i)a criação de um modelo de referência para o projeto de hardware e (ii)uma plataforma de desenvolvimento de software. Uma proposta de uso desse nível de modelagem no projeto de uma RSSF para uma aplicação genérica está apresentada na Seção 4.3.

A Tabela 3.1 mostra uma síntese de características relevantes dos trabalhos acima apresentados.

Tabela 3.1: Comparação entre os trabalhos apresentados.

Trabalho	Alvo da Modelagem	Nível de Modelagem	Modelagem de Consumo	Ferramenta usada
[42]	SoC	Diversos níveis (RTL, comportamental)	Baseado em simulações RTL	FUNTIME
[43]	RSSF	Nível de rede	Apenas consumo da comunicação (Modelo estatístico)	-
[44]	RSSF	Nível de rede	Apenas consumo da comunicação	GloMoSim
[45]	RSSF	Modela hardware como uma máquina de estados (comportamental)	Modela consumo dos estados da máquina	SystemC
[47]	RSSF	Modela hardware como uma máquina de estados (comportamental)	Modela consumo dos estados da máquina	NS-2
[49]	RSSF	Modela hardware como uma máquina de estados (comportamental)	Modela consumo dos estados da máquina	SystemC SCNSL
[50]	RSSF	Modela hardware de forma heterogênea de acordo com o bloco	Modela consumo do hardware de acordo com o nível de modelagem	SystemC
[51]	RSSF	Modela hardware de forma heterogênea de acordo com o bloco	Modela consumo do hardware de acordo com o nível de modelagem	SystemC SCNSL
Trabalho Proposto	RSSF	Modela hardware de forma heterogênea de acordo com o bloco	Modela consumo do hardware de acordo com o nível de modelagem	SystemC SCNSL

4 METODOLOGIAS

Neste capítulo está apresentada a metodologia de projeto de circuitos integrados, usada no projeto de sistemas em chip. Um detalhamento da etapa de modelagem alto nível é feito dado o foco deste trabalho. Uma proposta de metodologia de projeto para RSSF será proposta.

4.1 METODOLOGIA DE PROJETO DE SOC

O projeto de sistemas embarcados, em geral, é bastante complexo e muitas decisões e compromissos devem ser tratados. Ao longo do tempo, estabeleceu-se um fluxo em que o projeto passou a ser feito com etapas bem determinadas de forma a tratar aspectos específicos com a atenção necessária. Em resposta ao aumento de complexidade dos sistemas em chip e redução do intervalo de tempo entre início do projeto e lançamento no mercado, um passo de modelagem em alto nível foi adicionado. Este fluxo de projeto modificado está mostrado na Figura 4.1 [6]. A adição de modelagem em alto nível proporciona ganhos de produtividade já que decisões de projeto importantes, o particionamento entre hardware e software e a validação, por exemplo, podem ser feitas com menor esforço de projeto, uma vez que modelos TLM são construídos de forma mais simples do que seus correspondentes RTL além de evitar alterações em etapas avançadas do projeto.

O projeto se inicia na etapa de especificação em que a funcionalidade e requisitos mínimos são definidos. Estas especificações são comumente comportamentais com poucos detalhes claros de como o sistema deve ser implementado. Um primeiro particionamento entre hardware e software é seguido por uma modelagem TLM do sistema. A modelagem TLM tem a função de confrontar a especificação comportamental e as características técnicas do sistema proposto. Além de validar o particionamento entre hardware e software, a modelagem traz as especificações iniciais para um nível mais próximo do necessário para a construção do sistema. Não é raro, no entanto, a necessidade de iterações nestas etapas para solução das decisões de projeto. O sistema uma vez modelado em nível TLM adquire dois papéis fundamentais no restante do projeto: (i)servir de modelo de referência para o desenvolvimento de hardware e (ii) servir como plataforma virtual de desenvolvimento de software antes que um protótipo do hardware esteja disponível.

O projeto de hardware, no caso de sistemas em chip, segue o fluxo comum de circuitos integrados. Circuitos analógicos seguem o caminho chamado *full custom* [52] enquanto circuitos digitais seguem o chamado *standard cell* [52]. Após o projeto de hardware e software, o sistema é integrado, prototipado e testado. Após os testes, o sistema como um todo é enviado para produção em escala.

Este trabalho se concentra na parte de modelagem TLM de um sistema em chip e sua simulação em ambiente de RSSF. O modelo TLM gerado permite a simulação do hardware assim como observação do comportamento da rede dando informações sobre gasto de energia do hardware e levantamento da evolução da carga restante nas baterias dos nós da rede. É possível ainda a exploração de novas arquiteturas, tanto de hardware quanto topologias de rede. A abordagem mostrada neste

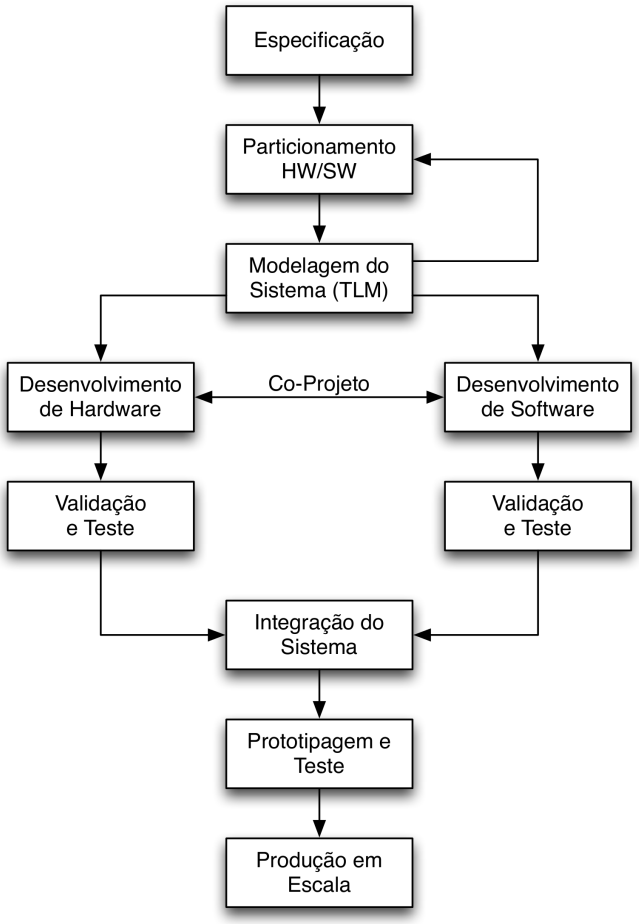

Figura 4.1: Fluxo de projeto de sistemas em chip [6].

trabalho é geral e pode ser usada para qualquer aplicação de RSSF uma vez que qualquer RSSF pode ser simulada usando a metodologia apresentada.

4.2 METODOLOGIA DE MODELAGEM DO SISTEMA

A Figura 4.2 mostra em detalhe a etapa de modelagem TLM do fluxo de projeto de SoC mostrado anteriormente. A escolha da arquitetura do sistema é definida após o particionamento entre hardware e software. Uma vez definidos os blocos que constituirão o sistema em chip completo é feita a padronização das interfaces de comunicação. Isto define o formato das transações a serem implementadas e tipo de dados a serem trocados.

Com as interfaces definidas, tem início o desenvolvimento de módulos. Nesta etapa, módulos modelados e validados previamente podem ser adaptados para reduzir o esforço e tempo no desenvolvimento do modelo completo. Neste trabalho, todos os códigos usados e gerados foram escritos usando a biblioteca SystemC. Foram usadas ainda a ADL ArchC [39] para geração do modelo de processador usado e a biblioteca SCNSL [5] para simulação da rede.

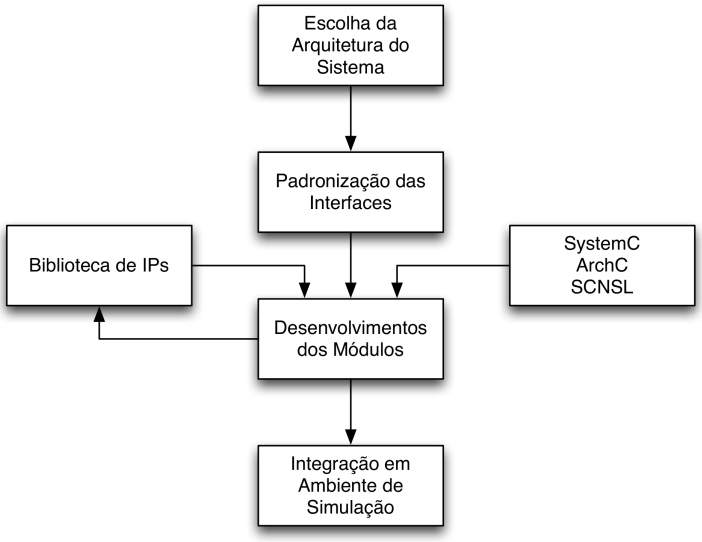

Figura 4.2: Fluxo adotado para modelagem de SoC [3].

Neste trabalho, todos os modelos de SoC usados foram criados de forma que os blocos de hardware são claramente separados. Este nível de modelagem de hardware é chamado aqui de nível arquitetural.

Os softwares embarcados nos nós das RSSF simuladas foram cross-compilados usando a *tool-chain* para o processador MIPS disponibilizada pela ADL ArchC. Todas as simulações foram feitas em uma máquina AMD Turion x2 1,9GHz, 2GB de memória RAM e rodando linux Ubuntu 10.10.

4.3 METODOLOGIA DE PROJETO DE RSSF

Nesta seção, a forma de projeto de RSSF usada atualmente será apresentada e criticada. Em seguida, uma proposta de metologia de projeto para RSSF que usa modelagem de sistemas em alto nível será apresentada.

4.3.1 Metodologia Atual de Projeto de RSSF

O projeto de uma RSSF para uma dada aplicação apresenta desafios de áreas da engenharia elétrica como eletrônica, processamento de sinais e redes de comunicação. Apesar de apresentar desafios multidisciplinares, esse tipo de solução é abordada com ferramentas que não são capazes de tratar todos eles de forma adequada. Um exemplo, é o uso comum do NS-2, que não é capaz de simular o comportamento do hardware ou o tratamento dos sinais, visto que o NS-2 é um simulador de rede. A existência de uma ferramenta de auxílio a projeto mais específicas para aplicações em RSSF levaria a uma melhor compreensão dos desafios da implementação dessas redes.

É comum projetar camada de rede, que lida com toda a comunicação da RSSF, com simuladores de rede como o NS-2. Após a camada de rede estar projetada para a aplicação, parte-se para o projeto

do hardware e software a ser embarcado. Outra característica do projeto atual de RSSF é o uso de hardware de prateleira que restringe as aplicações possíveis.

No Anexo I estão narrados depoimentos de alguns pesquisadores da área que foram consultados sobre a forma de projeto de suas aplicações com RSSF sendo identificadas falhas devido à metodologia usada no projeto.

Modelagem de sistemas em alto nível surge como uma possível solução para a inexistência de uma ferramenta capaz de tratar RSSF de forma estruturada. A implementação de simuladores para situações específicas permite estimar o comportamento da rede antes do início do projeto de qualquer protótipo.

4.3.2 Proposta de Metodologia de Projeto de RSSF

A metodologia de projeto comentada acima tem limitações quanto à compreensão das soluções de compromisso envolvidas. É difícil avaliar, por exemplo, se aplicações que exigem operações complexas sobre os dados coletados são viáveis para um dado hardware já que se desconhece a priori o custo computacional do tratamento. A adição de modelagem em alto nível e simulação da RSSF traz a possibilidade de compreender melhor os compromissos a serem resolvidos, aperfeiçoando as especificações do sistema e permitindo análises importantes.

Como mencionado no Capítulo 2, RSSF são sistemas altamente dependentes da aplicação para a qual foram projetadas e dado que o demandador da solução é, não raro, um leigo na área de engenharia, as primeiras especificações construídas são, geralmente, comportamentais. Partindo de uma especificação comportamental, é bastante complicado projetar o sistema como um todo devido à grande variedade de soluções possíveis e poucas condições de contorno para balizar o problema, ou seja, especificações comportamentais, sem nenhum refinamento, não são suficientes para a construção do sistema.

O projeto do hardware para a RSSF exige especificações bastante detalhadas. O fluxo descrito nesta seção tem o objetivo principal de servir como ponte para refinamento dessas especificações comportamentais fornecidas pelo demandador da solução. A Figura 4.3 mostra o fluxo descrito nesta seção.

Dado que a solução de engenharia a ser implementada com RSSF é fortemente dependente da aplicação, é natural esperar que a metodologia de projeto siga algo como o *top-down* para circuitos integrados,ou seja, parte-se do comportamento do sistema como um todo e o nível de detalhe aumenta ao longo do projeto. Desta forma, projetar a rede, isto é, a forma de comunicação, topologia da rede, densidade dos nós, formato da resposta da rede (monitoramento de variáveis ou de eventos) entre outros quaisquer parâmetros comportamentais, é o primeiro passo mostrado como *Simulação de Rede com nós funcionais*. Isso permite implementar, em simulador, o comportamento da RSSF. Nesta etapa, decisões importantes, como protocolos de rede, formato dos pacotes de dados que transitarão pela rede, entre outras serão tomadas. Uma vez que todo o hardware está abstraído e apenas o seu comportamento está modelado, todos os problemas do hardware como tamanho de memória e frequência do processador podem ser abtraídos. Detalhes como latência, rotas ou até mesmo

esquemas adaptativos de localização de rotas devem ser estabelecidos nesta etapa.

Esta etapa pode ser feita num simulador de rede como o NS-2 ou mesmo em SystemC usando nós modelados em nível comportamental com auxílio da biblioteca SCNSL como feito em [3]. As simulações podem apontar para estimativas de consumo do transceptor RF já que a comunicação é de fato simulada. Tais previsões de consumo não serão bastante acuradas (já que está simulação não leva em consideração o gasto de energia com processamento) mas poderão dar aos projetistas a ordem de grandeza dos valores tratados, uma vez que o hardware de fato não foi escolhido mas a ordem de grandeza do consumo não varia muito para circuitos semelhantes.

Esta simulação tem o objetivo de especificar a camada de rede num nível tal que toda a parte de comunicação do software que será embarcado nos motes esteja definida, basicamente protocolos de comunicação, roteamento de dados ou qualquer outra decisão no nível de rede. Separando em classes os diversos nós da rede (por exemplo, nós, *sinks* ou estações de campo) é possível estabelecer requisitos para o hardware de cada classe.

Uma vez que a rede esteja projetada e supondo a existência de uma biblioteca de modelos de hardware para montagem dos motes (como uma biblioteca de IPs), parte-se para uma primeira modelagem dos nós da rede em nível arquitetural, como será mostrado neste trabalho nos Capítulos seguintes, mostrado na Figura 4.3 por *Projeto dos Nós*. Nessa etapa, todos os requisitos de comunicação e de processamento necessários para a implementação da rede já estão definidos já que os protocolos já foram previamente definidos. Assim pode-se encarar o problema de projetar hardware para RSSF como tantos projetos de hardware mais simples quantas forem as classes identificadas. Essa etapa se assemelha ao projeto de sistemas embarcados em que blocos mais simples são agregados para formar o sistema. Isso se assemelha ao *bottom-up* de circuitos integrados. O desenvolvimento de software embarcado se inicia quando um modelo arquitetural para o hardware está disponível. Nesse software, deve estar descrito o comportamento desejado para o nó na *simulação de rede* além de processamento sobre os dados.

A conexão entre os dois níveis de abstração é bastante natural quando o SystemC e a SCNSL são usados já que a modelagem de hardware em diversos níveis de abstração é permitida. Dessa forma, é possível substituir na mesma simulação um nó descrito de forma comportamental por outro descrito de forma arquitetural sempre observando o comportamento da rede como um todo, como mostrado na Figura 4.4. Quando um nó arquitetural é adicionado à rede sem causar mudanças em seu comportamento é feita a validação do novo modelo. Na Figura 4.3, uma nova simulação da rede é feita, agora chamada *Simulação da Rede com Nós Funcionais e Arquiteturais*.

Por razões de performance de simulação, não é interessante substituir todos os nós comportamentais por nós arquiteturais. Dessa forma, substituir apenas alguns nós representativos de cada classe que tragam informações sobre o hardware de fato é aconselhável. Vale lembrar que o modelo em nível arquitetural descrito neste trabalho deve executar software embarcado.

Nesta etapa de *Simulação da Rede com Nós Funcionais e Arquiteturais*, decisões como forma de tratamento dos dados, compromisso entre processamento local ou transmissão dos dados podem ser atacados já que os custos de performance e de energia podem ser mais bem quantificados. A existência de uma biblioteca de modelos se mostra importante nesta etapa para reduzir o esforço e

tempo para desenvolvimento dos modelos.

Detalhes como ciclo de trabalho do hardware, tamanho da memória necessária e poder computacional podem ser especificados já que os requisitos da aplicação para comunicação, aquisição, processamento e, consequentemente gasto de energia, são conhecidos e bem compreendidos.

A simulação *Simulação da Rede com Nós Funcionais e Arquiteturais* deve ser feita num ambiente capaz de lidar com hardware, como o SystemC associado à biblioteca SCNSL. O uso de SystemC traz uniformidade ao fluxo de projeto evitando o uso de outras ferramentas.

Ao fim dessas simulações, a equipe de projeto de hardware dispõe de modelos de hardware detalhados para cada classe de nós necessários para construção da rede. Nesse ponto, o projeto de hardware para RSSF se torna o projeto dos hardwares das classes definidas anteriormente. Uma vez que o nível arquitetural pode dispor inclusive de modelos ISA do processador e modelos detalhados dos periféricos, a escolha do hardware para a dada aplicação é tomada. No caso de projeto de motes com SoC, o uso de IPs pode ser usado para acelerar o projeto do chip, enquanto no caso de circuitos de prateleira, basta proceder com a integração do sistema. O modelo arquitetural nesta etapa assume o papel de *modelo de referência* para validação do hardware.

A equipe de projeto de software, de posse do modelo arquitetural, pode proceder o projeto de software a ser embarcado nos nós mesmo que um protótipo do hardware não esteja disponível. Isso acelera o projeto e a etapa de depuração, diminuindo o tempo até o lançamento da rede.

Em nível de rede, é possível informar ao usuário da solução diversas caracteristicas como performance e tempo de vida em um nível comportamental apresentado inicialmente por ele facilitando, portanto, a comunicação.

Após a *Simulação da Rede com Nós Funcionais e Arquiteturais*, caso a rede e os hardwares se comportem como desejado na definição da aplicação, pode-se partir para a construção de nós sensores, já que os blocos de hardware que os compõem estão validados por simulação e listados. O software a ser embarcado nos nós sensores estará em estágio avançado de desenvolvimento podendo ser embarcado sem grandes mudanças.

Após a construção de nós sensores em número suficiente para a execução da aplicação, parte-se para a deposição da RSSF.

Após o completo desenvolvimento tanto de hardware em número suficiente para a aplicação como software, a solução implementada em RSSF pode ser coloca em funcionamento de fato.

A metodologia apresentada carece de uma ferramenta que a implemente e é dependente da existência de uma biblioteca de modelos de simulação para seu uso correto. Tanto a ferramenta como a biblioteca de modelos não estão construídas.

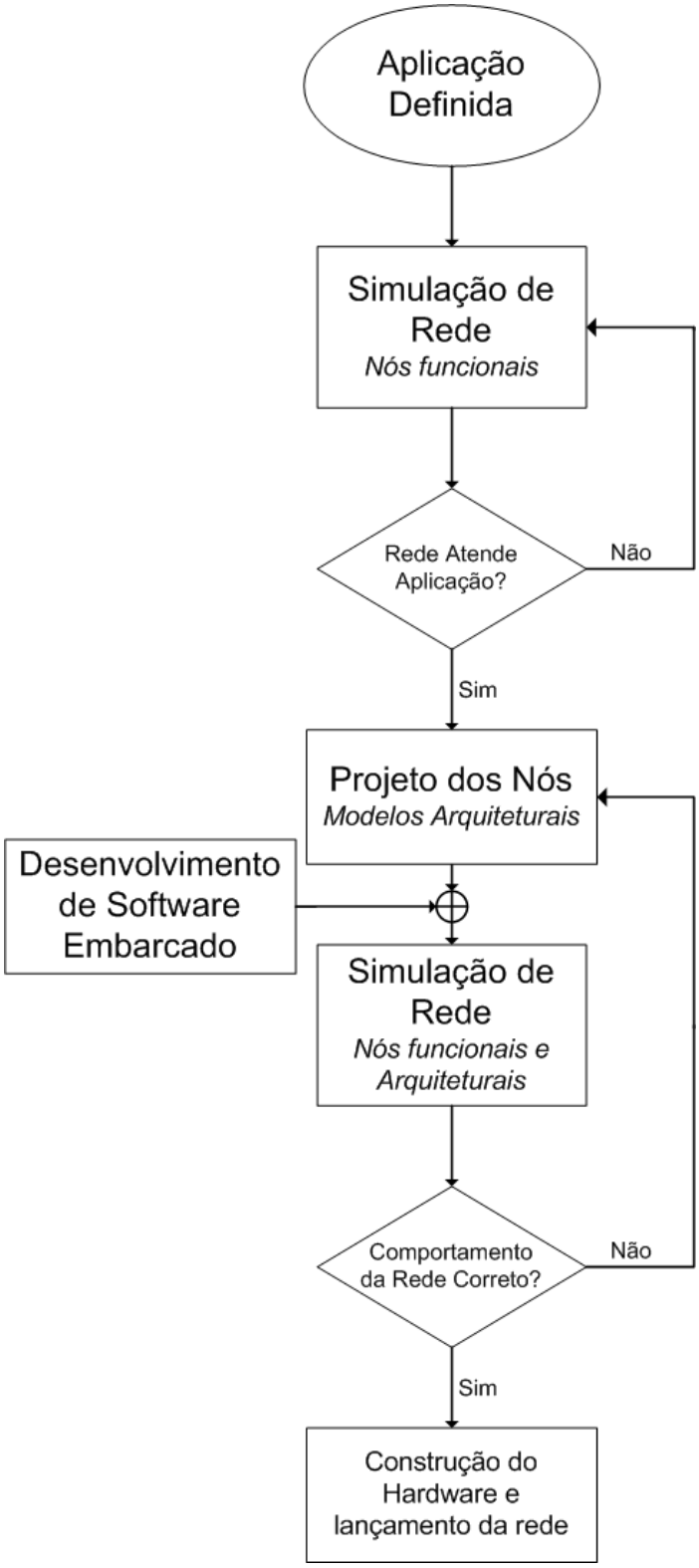

Figura 4.3: Fluxo da metodologia de projeto de RSSF proposta.

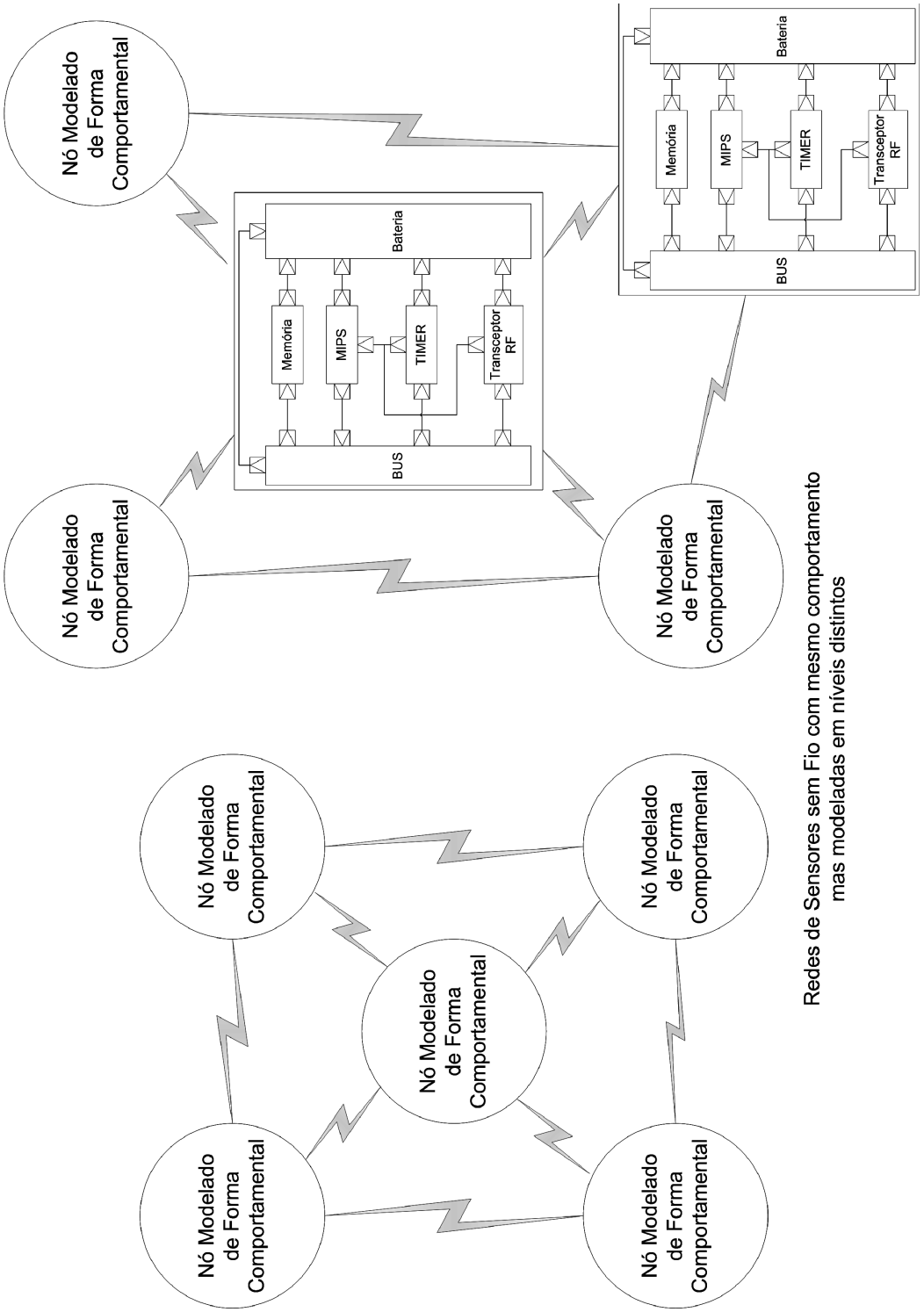

Figura 4.4: Substituição de nós funcionais por nós arquiteturais sem mudança de comportamento da rede.

5 MODELAGEM DE ENERGIA

Neste capítulo, o modelo de energia criado será detalhado e a bateria, modelada. O embasamento matemático do modelo de energia é apresentado e discutido, assim como as hipóteses assumidas. A forma de implementação em SystemC é comentada e exemplificada por meio de diagramas e trechos de código.

5.1 MODELAGEM DE GASTO DE ENERGIA

Como já comentado, RSSF são sistemas cujo projeto depende fortemente da aplicação. O projeto de uma RSSF, portanto, deve passar por etapas de (i)projeto do hardware de todos os nós desde a escolha dos sensores até a antena no caso de comunicação RF e (ii)projeto do software a ser embarcado que lida com tratamento dos dados coletados até esquema de roteamento da rede. Tendo estes dois pontos em mente, não é difícil entender que o consumo de energia da rede também depende da aplicação. E além de depender da aplicação, depende da topologia da rede e até da disposição física dos nós. Sendo assim, uma solução para estimar consumo de energia em RSSF é usar simuladores já que modelagens estatísticas deveriam levar em consideração muitos parâmetros além de serem demasiado específicas.

Em RSSF com limitações de energia, os nós são alimentados com baterias e pilhas. Esses componentes se baseiam em reações químicas para gerar eletricidade. Devido aos baixos custos relativos e volume, baterias são comumente usadas como uma fonte de energia para o hardware de RSSF. Além da característica não-linear como mostrado na Figura 5.1, seu comportamento elétrico é extremamente difícil de prever por ter forte dependência com variáveis ambientais, como temperatura e condições de uso.

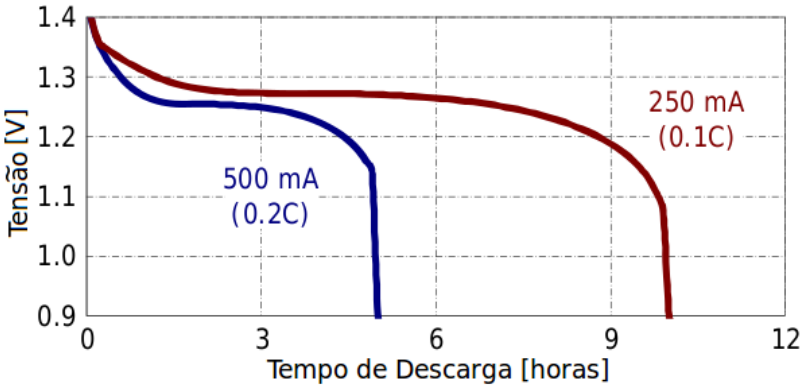

Figura 5.1: Curva de descarga para uma bateria NiCd de 2500mAh à 21°C [7].

Modelos matemáticos empíricos que caracterizam efeitos bastante particulares, como carga e descarga em diversas temperaturas, foram levantados mas limitações quanto à precisão dos modelos e tipo de dado gerado (no caso, tensão *versus* tempo) inviabilizam o uso para simulação de

circuitos[45]. Modelos eletroquímicos são complexos e demandariam grande esforço computacional [45].

Para simulação de uma RSSF alimentada por bateria, observar o comportamento da rede à medida que os nós se desligam por falta de energia traz uma nova possíbilidade no projeto de RSSF. Permite, por exemplo, observar como os dados trafegam numa rede mesh (topologia de rede em que os nós são responsáveis por encaminhar dados de outros nós) com a saída de um nó ou informar ao projetista que uma determinada área está agora descoberta pela rede. Uma vez que os modelos comentados dispõem apenas de informação entre tensão e tempo, frequentemente para descarga constante, outro tipo de modelo deve ser criado.

Esse novo modelo deve ser capaz de simular o comportamento de uma bateria do ponto de vista do hardware conectado a ela: servir como uma fonte de energia para as operações enquanto houver energia suficiente armazenada. Assim que a energia disponível se mostrar insuficiente para uma dada operação, o hardware tem sua funcionalidade afetada, no limite, tornando-o totalmente inoperante.

Sabendo que cada operação executada pelo hardware demanda alguma energia e sabendo que

$$E = \int P dt \tag{5.1}$$

onde E representa a energia e P, a potência, para conhecer a energia que uma dada operação necessita para ser executada basta conhecer a potência gasta pelo módulo e o tempo em que o dado módulo fica ligado. Sendo assim,

$$E = \int P dt = \int V \cdot I dt \tag{5.2}$$

onde V é a tensão de alimentação e I a corrente drenada pelo circuito. Segundo a definição de corrente elétrica

$$I = \frac{dQ}{dt} \tag{5.3}$$

onde Q representa a carga elétrica. Substituindo 5.3 em 5.2, obtem-se

$$E = \int V \cdot \frac{dQ}{dt} dt \tag{5.4}$$

Assumindo que a tensão gerada pela bateria é constante, chegamos à

$$E = V \int \frac{dQ}{dt} dt = V \cdot Q \tag{5.5}$$

A hipótese de tensão constante pode ser vista como consequência do uso de um circuito regulador de tensão entre a bateria propriamente dita e o circuito a ser alimentado. Reguladores de tensão são circuitos que mantêm a tensão de saída constante mesmo frente a variações na tensão de alimentação [53].

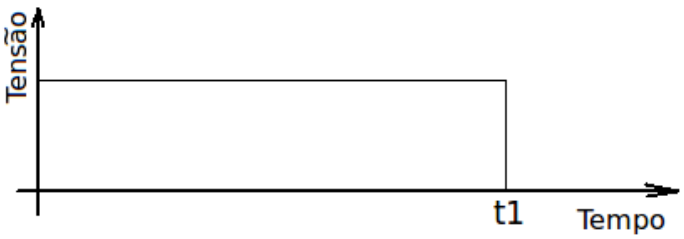

Figura 5.2: Curva de descarga para a bateria modelada.

Segundo a Expressão 5.5, uma quantidade de energia está associada à uma quantidade de carga por meio de uma constante, neste caso a tensão. Portanto, a bateria pode entregar energia para outros blocos enquanto houver carga elétrica disponível. Podemos entender uma bateria, segundo este modelo, como um reservatório de carga. No instante em que a bateria não dispuser de carga elétrica, ela não poderá entregar energia aos módulos que, por consequência, pararão de funcionar.

Como consequência do modelo, a curva de descarga da bateria usada neste trabalho é mostrada na Figura 5.2.

Como estamos tratando a bateria como um depósito de carga elétrica, gastos de energia podem ser convertidos em gastos de carga elétrica. Desta forma, o cálculo de quanta carga ainda está disponível na bateria pode ser feito segundo a Equação 5.6.

$$Q(t + \Delta t_k)_{restante} = Q(t)_{restante} - I_k \cdot \Delta t_k \tag{5.6}$$

Na Equação acima, $Q(t)_{restante}$ é a carga restante na bateria no instante t, I_k é o consumo de corrente de um dado bloco de hardware e Δt_k é o intervalo de tempo que esse dado módulo drena corrente da bateria. Por exemplo, I_{CPU} é o consumo de corrente da CPU do sistema enquanto I_{mem} é o consumo de corrente do chip de memória usado no mote.

Uma das vantagens deste modelo é permitir o equacionamento da energia restante na bateria de forma objetiva, neste caso como carga elétrica, diferente da abordagem apresentada em [43] que apenas estima o consumo. Os dados necessários para estimar o consumo e simular uma rede com um dado hardware são a corrente consumida pelo bloco e o tempo que tal bloco permanece drenando corrente.

Para circuitos digitais, como microcontroladores e memórias, as informações sobre o consumo de corrente e tempo em que os circuitos drenam corrente estão sempre disponíveis em *datasheets* e, no caso de blocos IP que dispõem de modelos detalhados inclusive em nível de transistor, essa informação pode ser obtida via simulação para quaisquer modos de operação permitindo uma grande versatilidade e acurácia do modelo. Para este tipo de circuitos, é impossível conhecer com acurácia o tempo de execução de uma dada rotina ou o número de instruções a serem executadas sem uma simulação em nível de instrução. Portanto, modelagens do hardware em níveis de abstração mais alto levam à estimativas mais grosseiras do que as mostradas neste trabalho.

Para circuitos analógicos, como temporizadores e transceptores, o consumo de corrente é comumente documentado nos *datasheets* e a informação pode ser levantada em caracterização elétrica.

No entanto, o tempo em que o circuito fica ativo é fortemente dependente da aplicação. Desta forma, uma maneira de estimar o tempo em que esses circuitos ficam ativos é usar a noção de tempo interna ao simulador para calcular esse tempo. Em SystemC, o uso da função *sc_time_stamp()* retorna o instante t_s em que a função foi chamada. Com a chamada de *sc_time_stamp()* no momento em que o bloco é ativado e outra chamada no momento em que o bloco é desligado, é possível calcular o intervalo de tempo que o bloco ficou ativo.

De forma a unificar a forma de cálculo do tempo para circuitos digitais e analógicos, o modo que usa duas chamadas de *sc_time_stamp()*, calcula o intervalo de tempo e usa esse dado para calcular o consumo de carga elétrica, foi adotado.

Essa maneira de estimar o consumo exige uma modelagem dos blocos tal que a passagem do tempo dentro do simulador esteja modelada. Em SystemC, a passagem de tempo é simulada com o uso de *threads* chamando funções *wait()*. Sendo assim, o simulador é capaz de estimar o tempo gasto por cada bloco para executar uma dada operação. Apesar de exigir maior esforço para geração do modelo e exigir maior peso computacional do que as abordagens puramente comportamentais usadas por [43] e [44], já discutidas, o nível de detalhe dos modelos usados neste trabalho permite simulação do consumo de cada bloco de hardware separadamente, além de permitir execução de código embarcado.

A possibilidade de executar código de aplicação embarcado no nó da rede permite avaliar soluções de compromisso como ciclo de trabalho *versus* tempo de transmissão, no caso de compactação dos dados no processador. Este tipo de possibilidade viabiliza redes com operação mais flexível e abre caminho para aplicações mais complexas do que simplesmente coletar dados e enviá-los à estação base.

Uma vez que grande parte dos dados sobre consumo são obtidos a partir de documentação, pode-se levantar a questão de acurácia destes dados. No entanto, em [54] é apresentada uma plataforma de caracterização de consumo de potência para motes de RSSF. Os resultados obtidos pelos autores confirmam os dados da documentação dos componentes mostrando que os dados lidos de *datasheets* são confiáveis e representam o hardware sob análise.

A bateria modelada neste trabalho não dispõe de um sistema de *energy harvesting*. Apesar de facilmente implementavel neste modelo de SoC, a obtenção de energia do ambiente, por fonte solar por exemplo, depende da modelagem do ambiente no qual a RSSF está imersa e tal modelo de ambiente não está disponível no trabalho atual. O autores de [41] comentam que a insolação varia muito entre uma aplicação de monitoramento de floresta densa e outra de monitoramento da qualidade da água de um lago, como é esperado. Este resultado reforça a necessidade de modelar o ambiente para implementar adequadamente *energy harvesting*.

5.1.1 Modelo de Bateria

A bateria foi implementada como um módulo SystemC que recebe no construtor um nome, uma carga inicial em mAh e os consumos de corrente dos blocos nos diversos modos e operação modelados como mostrado no trecho de código a seguir.

```
//Battery((1)module name, (2)initial charge [mAh], (3)mips current consumption
    in NORMAL mode[A],
//(4)mips current consumption in LOW SPEED mode[A], (5)mips current consumption
    in IDLE mode[A],
//(6)bus current consumption[A], (7)transceiver current consumption in TX mode @
    0dBm[A],
//(8)transceiver current consumption in RX mode[A], (9)timer current consumption
    [A],
//(10)memory current consumption[A].

    battery = new Battery("battery", 2500, 11.6e-3, 90.625e-6, 0, 0, 17.4e-3,
        19.7e-3, 100e-6, 33e-3);
```

Trecho de Código 5.1: Construtor do módulo Bateria.

Com base na carga inicial e nos consumos de corrente informados o módulo bateria implementa a Equação 5.6.

```
      //Consumo de energia para o barramento
2     if (request.dev_id == 1) {
         if (request.data == 0) {
4           bus_t1 = sc_time_stamp();
            //cout << "BUS t1 = " << bus_t1 << endl;
6        }
         if (request.data == 1) {
8           bus_t2 = sc_time_stamp();
            //cout << "BUS t2 = " << bus_t2 << endl;
10          bus_deltaT = bus_t2 - bus_t1;
            BUS_CONSUMPTION = BUS_POW * bus_deltaT.to_seconds();
12          //cout << "BUS deltaT = " << bus_deltaT << endl;
            //cout << "BUS_CONSUMPTION = " << BUS_CONSUMPTION << endl;
14          BUS_TOTAL = BUS_TOTAL + BUS_CONSUMPTION;
            REMAINING_CHARGE = REMAINING_CHARGE - BUS_CONSUMPTION;
16       }
      }
```

Trecho de Código 5.2: Trecho de código que implementa a Equação 5.6 para o barramento.

Para exemplificar a implementação, foi escolhido o módulo barramento devido à sua simplicidade. A identificação de cada bloco é feita usando o campo *request.dev_id*. No caso, quando esse campo tem valor igual a 1 significa que houve um aviso do barramento para a bateria. O campo *request.data* informa se é o aviso correspondente ao início ou fim de atividades do módulo. No caso de um primeiro aviso (*request.data* = 0), a bateria salva o instante em que o aviso foi dado (linha 4 do Trecho de Código 5.2). No caso do segundo aviso (*request.data* = 1), a bateria salva o instante em que este aviso foi feito (linha 8 do Trecho de Código 5.2), calcula o intervalo de tempo que foi necessário ao módulo (linha 10), calcula a quatidade carga usada na operação executada (linha 11), acumula o gasto de carga usado na operação executada ao montante ja executado pelo módulo (linha 14) e atualiza a carga restante na bateria (linha 15).

Todos os módulos modelados seguem a mesma forma de estimativa de energia com diferenças de consumo associadas aos modos de operação, como o processador ou o transceptor. A seleção de consumo de energia para cada modo de operação é feita alterando o valor do campo *request.data*.

No início da execução de uma operação qualquer num dado módulo, uma transação é enviada à bateria por meio do método *transport()* da interface bidirecional bloqueante *tlm_transport_if*, informando à bateria qual módulo do sistema em chip vai executar uma operação e em qual modo de operação o referido módulo se encontra. Ao receber a transação, a bateria guarda o instante em que o módulo iniciou a execução. Ao fim da execução, uma nova transação é enviada à bateria informando que o módulo terminou a execução da tarefa. Ao receber esta segunda transação e de posse do instante de tempo em que a execução foi iniciada, a bateria calcula o intervalo de tempo, o multiplica pelo gasto de corrente e atualiza a carga restante como na Equação 5.6.

O módulo Bateria foi criado de forma que toda a estimativa de consumo de energia é feita por ele mesmo. O objetivo dessa estratégia é evitar grandes alterações em blocos já validados. Apenas uma porta e duas chamadas de transação para a bateria precisam ser implementadas, sendo que cada uma adiciona 4 linhas de código ao módulo que se deseja estimar o consumo.

A Figura 5.3 mostra o diagrama de tempo para um processo de leitura na memória com o objetivo de exemplificar e aclarar o procedimento de simulação de consumo de energia.

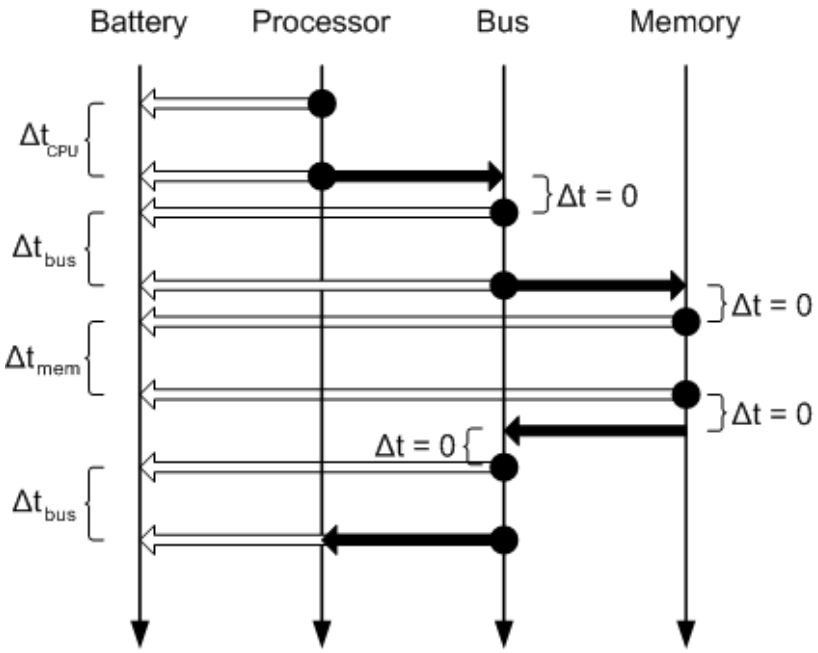

Figura 5.3: Diagrama de fluxo de um acesso à memória tal como modelado neste trabalho.

Na Figura 5.3, os pontos representam início ou fim da atividade em um módulo. O processo de leitura se inicia no processador. No instante em que a execução da instrução se inicia, o processador envia um aviso para a bateria. No fim do trabalho do processador, outro aviso é enviado para a bateria que calcula o intervalo de tempo usado pelo processador Δt_{CPU}. De posse do consumo de corrente do processador, informado para a bateria na fase de elaboração, é possível calcular o consumo de carga para a tarefa executada segundo a Equação 5.6.

Ao receber a informação vinda do processador, o barramento inicia sua atividade de direcionar a informação para a memória. De maneira análoga, o barramento envia avisos para a bateria no início e fim da sua atividade permitindo à bateria calcular o tempo levado para encaminhar o dado do processador até a memória. Como o consumo de corrente do barramento é conhecido e foi informado à bateria, esta calcula o consumo de carga da atividade executada pelo barramento.

Uma vez feito o pedido de acesso à memoria, esta envia avisos indicando início e fim do acesso. Novamente a Equação 5.6 é usada para calcular o consumo de carga do acesso à memória já executado.

O barramento é ativado novamente para entregar o dado ao processador e, novamente, o fluxo de estimativa de consumo de carga do barramento, já descrito, é percorrido até a entrega do dado ao processador.

Este fluxo funciona de maneira análoga para todos os módulos modelados, sempre usando dados de corrente e tempo. No instante em que a carga na bateria se esgota, um evento que interrompe todas as *threads* do nó é gerado, simulando seu desligamento.

6 IMPLEMENTAÇÃO DO MODELO DE SOC

O SoC modelado neste trabalho é uma extensão do modelo apresentado em [3] em que o objetivo era simular a comunicação entre nós de uma RSSF. Portanto, mudanças foram implementadas para permitir a estimativa de consumo de energia. Neste capítulo, o modelo de sistema em chip com possibilidade de consumo de energia será descrito.

6.1 MODELO DO SOC

Um sistema em chip composto de processador, memória, barramento, temporizador, transceptor de RF e bateria para simulações de RSSF foi desenvolvido. Este sistema está mostrado na Figura 6.1 e foi contruído para proporcionar infraestrutura para simulações de RSSF. Com este modelo, tanto informações da aplicação e rede (latência e conectividade da rede), tal como apresentado em [3], quanto informações do hardware (consumo de energia e ciclo de trabalho) podem ser avaliados.

O diagrama mostrado na Figura 6.1 apresenta um sistema em chip modelado de forma que os blocos de hardware são claramente separados. Neste trabalho, essa forma de modelagem de SoC será chamada de *nível arquitetural* em contraposição ao *nível funcional* em que apenas o comportamento do SoC é observado.

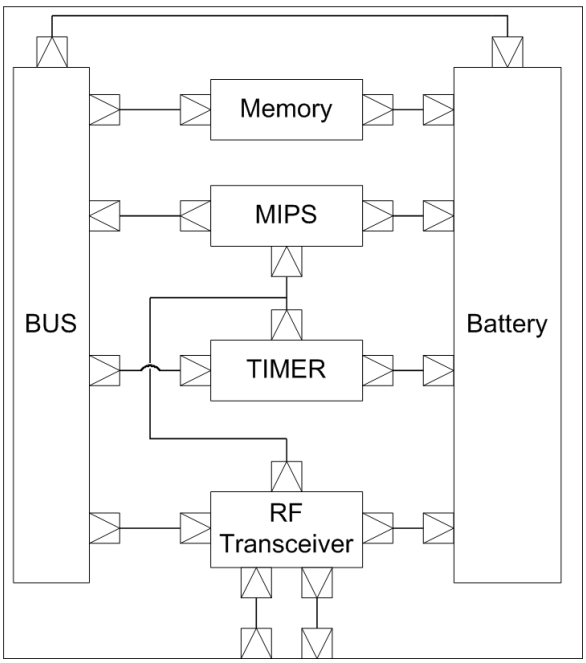

Figura 6.1: Modelo de sistema em chip em nível arquitetural implementado para este trabalho.

O conjunto formado por processador, barramento e memória forma o núcleo de processamento. Um temporizador foi modelado para permitir o tratamento de eventos periódicos e um transceptor RF foi adicionado para que possa haver comunicação entre vários SoCs formando uma RSSF. Um

módulo bateria foi adicionado para proporcionar estimativas de consumo de energia para o resto do SoC, além de desligá-lo na ausência de energia.

Neste SoC, o processador é o mestre do sistema, ou seja, toda a comunicação entre módulos através barramento é iniciada por ele. Alguns periféricos, como o temporizador e o transceptor RF, podem gerar interrupções no processador mas todo o tratamento das informações deve ser feito pelo processador. Para facilitar a comunicação entre os blocos conectados ao barramento, os módulos devem implementar registradores mapeados em memória.

6.2 MODELO DO PROCESSADOR

A modelagem do processador foi feita usando a ADL ArchC. Uma arquitetura RISC 32 bits com instruções baseadas no MIPS foi escolhida para obter baixo consumo e alta performance do hardware e usar as ferramentas de compilação e debug já disponíveis. Informações sobre a arquitetura foram apresentadas no Capítulo 2. O código gerado por ArchC foi modificado já que a implementação do mecanismo de interrupções exigiu a implementação de um banco adicional de registradores. Um esquema de temporização mais realista foi implementado assim como diferentes modos de consumo. Todo o sistema em chip foi modelado usando TLM 1.0 visando compatibilidade com os protocolos herdados das bibliotecas ArchC. Mais informações sobre a implementação do modelo do processador podem ser encontradas em [3].

A decisão de modelar o processador em nível de instruções foi tomada com objetivo de obter uma estimativa acurada do comportamento do processador, assim como do seu consumo de energia, e permitir a execução de software embarcado.

6.3 MODELO DE BARRAMENTO

O barramento foi modelado como uma estrutura de interconexão que (i)recebe todas as transações do processador, (ii)modifica o endereço conforme o mapa de memória e (iii)encaminha a transação ao periférico correto. As transações de resposta são enviadas ao processador. Essa forma de modelagem permite a adição de outros periféricos ao sistema em chip simplesmente adicionando uma nova porta ao barramento e uma faixa de endereços para o novo periférico.

O consumo de energia do barramento não foi levado em consideração neste trabalho devido à falta de informações. Em microcontroladores e SoC atuais, o barramento está integrado com o processador, memória e comumente alguns periféricos como ADC e temporizadores. Uma vez que o consumo de energia e tempo acesso é um parâmentro interno ao chip essa informação não é fornecida.

No caso de módulos IP, no entanto, toda a informação pode ser obtida por simulações com níveis de acurácia até nível de transistor. Frente a esta grande disponibilidade de dados, o uso de módulos IP é perfeitamente compatível com a metodologia adotada para estimativa de energia.

6.4 MODELO DE MEMÓRIA

A memória foi implementada como um vetor de ponteiros uma vez que ela apenas responde comandos de leitura e escrita vindos do processador, armazenando dados no caso de operações de escruita. A decisão de não modelar a memória como uma *thread* foi tomada com vista na performance de simulação uma vez que muitos acessos são feitos à este módulo, evitando trocas de contexto mais frequentes.

Como a arquitetura MIPS trabalha com dados organizados em bytes [28], a memória foi implementada com dados de 8 bits cujo tamanho é definido em tempo de compilação.

O uso de interfaces bloqueantes tem um papel importante para a memória, uma vez que é impossível chamar a função *wait()* em módulos SystemC que não implementam *threads*. A chamada de função *wait()* deveria ser feita no *transport()* e, desta forma, o tempo de acesso estaria modelado fora do módulo propriamente dito. No entanto, essa abordagem causa erro em tempo de execução mesmo sendo teoricamente legal e prevista na documentação do TLM. A causa do problema não foi encontrada mesmo após consulta a especialistas na área. A modelagem de consumo de energia deste módulo fixa o tempo de acesso à memória na bateria tal como mostrado na Tabela 7.3 e, portanto, não usa a função sc_time_stamp(). Isso não causa problemas para a estimativa de gasto de energia já que o tempo de acesso à memória é deterministico e bem documentado.

Esta abordagem de cálculo deverá ser usada sempre que o módulo não implementar uma *thread* já que é impossível chamar a função *wait()* para simular a passagem do tempo, neste caso.

6.5 MODELO DE TRANSCEPTOR

O transceptor RF é responsável por fazer a comunicação do SoC com a rede e foi modelado como um módulo que implementa registradores mapeados em memória que são usados para recebimento e envio de dados para a rede. Desta forma, para fins de comunicação, todo o hardware de RF foi abstraído e apenas o comportamento do circuito foi modelado.

Este bloco foi modelado como uma máquina de estados finitos tal como mostrado na Figura 6.2. Os estados correspondem aos modos desligado, recepção e transmissão, que por sua vez foi dividido em subestados correspondentes à implementação do procotolo de acesso ao meio descrito no padrão do ZigBee [32] descrito no padrão IEEE 802.15.4 [55] que está modelado como bloco de hardware neste trabalho. Esse protocolo foi usado pois é implementado em hardware no transceptor Texas Instruments CC2420 [2] que é o mais comum em aplicações de RSSF. Neste trabalho, parâmetros como potência, taxa, sensibilidade e consumo do CC2420 foram carregados no simulador.

Para controle do módulo, foram implementados 11 registradores mostrados na Tabela 6.1.

O registrador $tcv_config é usado para configurar o modo de operação do transceptor entre recepção (RX) e transmissão (TX). A Figura 6.3 mostra como é a seleção de modos neste registrador.

Para efeitos práticos neste trabalho esse registrador pode assumir três valores uma vez que neste trabalho parâmetros como potência de transmissão (0dBm), taxa (250kbps) e sensibilidade do trans-

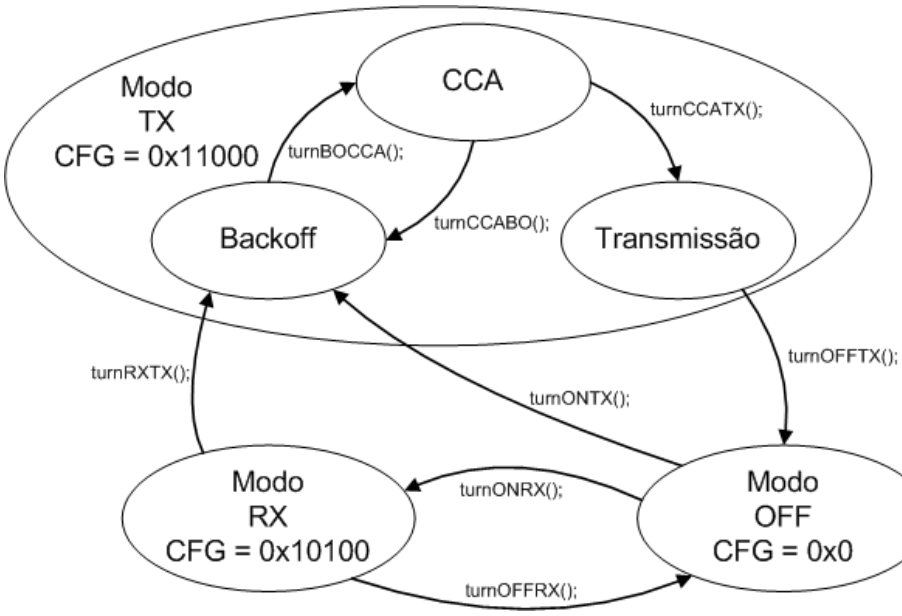

Figura 6.2: Diagrama de estados do transceptor.

Figura 6.3: Estrutura do Registrador $tcv_config.

ceptor (-95dBm) estão fixas:

- 0x0: transceptor desligado.
- 0x11000: transceptor habilitado em modo transmissão.
- 0x10100: transceptor habilitado em modo recepção.

O registrador $tcv_status é usado para acompanhar o andamento das operações do transceptor. O bit menos significativo se mantem em nível lógico alto caso exista uma transmissão em curso e o segundo bit menos significativo se mantem em nível lógico alto caso um pacote tenha sido recebido e não tratado pelo processador.

Os dados recebidos pelo transceptor são colocados nos registradores $tcv_receive_data. Neste trabalho, os usos dos registradores $tcv_receive_data estão comentados abaixo. A rigor, no entanto, qualquer dado poderia ter sido escrito nos registradores.

- $tcv_receive_data0: palavra de 32 bits que identifica o nó remetente.
- $tcv_receive_data1: palavra de 32 bits que identifica o destinatário.

Tabela 6.1: Mapa de registradores do transceptor modelado.

Nome do Registrador	Endereço	Uso
$tcv_config	0x0000 0000	Configuração do modo de operação
$tcv_status	0x0000 0001	Status
$tcv_send	0x0000 0002	Inicia transmissão
$tcv_send_data0	0x0000 0003	Dados a serem transmitidos
$tcv_send_data1	0x0000 0004	
$tcv_send_data2	0x0000 0005	
$tcv_send_data3	0x0000 0006	
$tcv_receive_data0	0x0000 0007	Dados recebidos
$tcv_receive_data1	0x0000 0008	
$tcv_receive_data2	0x0000 0009	
$tcv_receive_data3	0x0000 000A	

- $tcv_receive_data2: 32 bits de dados.
- $tcv_receive_data3: palavra de 32 bits para propósito geral.

Caso um dado SoC esteja dentro do alcance de outro que envia um pacote, a biblioteca SCNSL se encarrega de entregar esse pacote. No entanto, o pacote é recebido apenas se o transceptor estiver habilitado no modo recepção. Caso contrário, o pacote é simplesmente descartado. Na hipótese de recebimento bem sucedido, um evento é gerado que dispara uma *thread* que gera uma interrupção para o processador. Desta forma é possível tratar o recebimento de pacotes com eficiência.

Os dados a serem transmitidos devem ser colocados nos registradores $tcv_send_data via programa de aplicação. Uma vez que os dados estão colocados nos registradores e o transceptor habilitado em modo transmissão, uma escrita no registrador $tcv_send dispara a *thread* de transmissão, mostrada na Figura 6.4, enviando os dados para a rede. Neste trabalho, os usos dos registradores $tcv_send_data estão mostrados abaixo:

- $tcv_send_data0: palavra de 32 bits que identifica o nó remetente.
- $tcv_send_data1: palavra de 32 bits que identifica o destinatário.
- $tcv_send_data2: 32 bits de dados.
- $tcv_send_data3: palavra de 32 bits para propósito geral.

O protocolo de acesso ao meio (MAC) usado neste trabalho foi o descrito no padrão do ZigBee [32] descrito no padrão IEEE 802.15.4 [55]. Esse protocolo é implementado em hardware no transceptor Texas Instruments CC2420, comumente usado em aplicações de RSSF. A ideia é verificar a disponibilidade do canal antes de transmitir algum dado, evitando assim, colisões de pacotes.

Na Figura 6.4, o MAC está destacado. Ele consiste em aguardar um tempo aleatório e verificar a disponibilidade do canal. Caso o canal esteja ocupado, outro período de tempo é esperado. Caso o canal esteja livre, a transmissão é realizada. Note que o uso desse protoloco MAC não impede a colisão de pacotes. Coincidências de envio e problemas clássicos como terminal escondido [8] podem causar colisões.

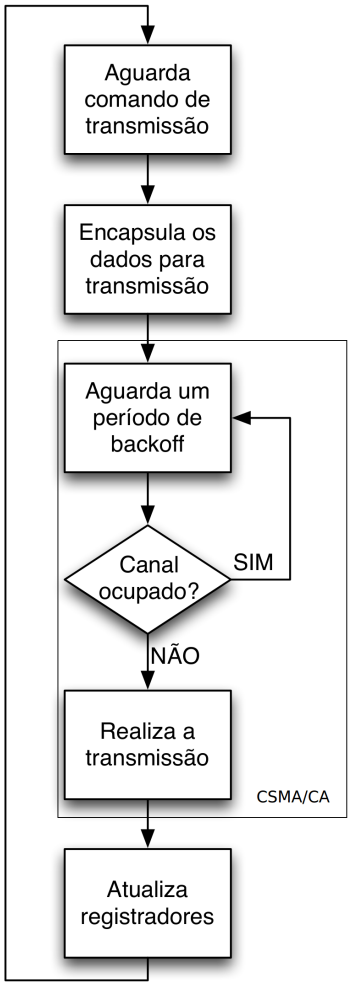

Figura 6.4: Fluxo de transmissão do transceptor.

6.6 MODELO DE TEMPORIZADOR

Um temporizador foi modelado para que o sistema em chip pudesse gerar eventos periódicos. Esse módulo é implementado por meio de uma *thread* que gera transações periódicas em sua porta de saída que está conectada à porta de interrupção do processador como mostrado na Figura 6.1. A periodicidade do temporizador é definida controlando os registradores via software de aplicação. Os registradores deste módulo estão mostrados na Tabela 6.2.

O registrador $tmr_start controla a habilitação do módulo. Caso seu valor seja diferente de zero, o temporizador estará habilitado. O período dos eventos gerados pelo módulo dependem do período

Tabela 6.2: Mapa de registradores do temporizador modelado.

Nome do Registrador	Endereço	Uso
$tmr_start	0x0	Inicia ou para o temporizador
$tmr_prescaler	0x1	Divisor de relógio
$tmr_comparator	0x2	Comparador

do relógio definido para o temporizador (62,5 ns neste trabalho) e dos valores dos registradores $tmr_prescaler e $tmr_comparator. Assumindo que o registrador $tmr_prescaler possua valor N e $tmr_comparator possua valor K e o período de relógio assumido para o temporizador seja t, a frequência dos eventos gerados será dada pela Equação 6.1.

$$f = \frac{1}{t \cdot N \cdot K} \tag{6.1}$$

7 RESULTADOS E DISCUSSÃO

Neste Capítulo, serão apresentadas as simulações usando o modelo de SoC descrito anteriormente. Inicia-se por uma simulação de um SoC isolado em que uma comparação entre a execução do algoritmo de criptografia AES [56], quer modelado como bloco de hardware quer executado como programa de aplicação. É feita comparação de consumo de energia e performance do hardware entre os dois casos. Em seguida, mostra-se a simulação de uma RSSF em estrela composta dos mesmos SoCs já descritos. Mostra-se o consumo de energia de cada nó da rede para a aplicação tal que os nós respondem requisições da estação de campo. Por fim, a simulação de uma RSSF com esquema de roteamento que usa *multihopping* é mostrada e os resultados analisados.

7.1 RESULTADOS DE SIMULAÇÕES DE SOC PARA RSSF

Como forma de avaliar o modelo de consumo de energia, um estudo de caso com um sistema em chip isolado foi feito. Neste estudo de caso, o algoritmo de criptografia Advanced Encryption Standard (AES) foi inicialmente executado num módulo SystemC, simulando um bloco de hardware. Em outra simulação, o algoritmo foi executado como software embarcado [57]. A Tabela 7.1 mostra os parâmetros de consumo carregados no simulador. O consumo do circuito AES foi obtido a partir de [58].

Tabela 7.1: Dados de consumo de energia de hardware usados para simulação do SoC.

	Consumo de Corrente [mA]	Tempo drenando carga da bateria
MIPS [59]	11,6	62,5ns
Memória [33]	33	85ns
Temporizador [35]	0,1	Sempre que ligado
AES [58]	61,1	110ns

A simulação executada consiste em executar 10 vezes o processo de cifragem baseado nas palavras abaixo. As palavras abaixo foram obtidas do texto do Instituto Norte-americano de Padrões (NIST) [56].

- Chave: 0x000102030405060708090A0B0C0D0E0F
- Palavra a ser cifrada: 0x00112233445566778899AABBCCDDEEFF
- Palavra cifrada: 0x69C4E0D86A7B0430D8CDB78070b4C55A

A bateria do SoC foi instanciada em simulação com uma carga inicial de 2500mAh, simulando uma pilha recarregável comum do mercado como a NH-AA-B4EN da Sony [60].

```
Total energy consumed by mips << 0.00143187 C (12.2124%)
Total energy consumed by bus << 0 C (0%)
Total energy consumed by AES << 0 C (0%)
Total energy consumed by timer << 0 C (0%)
Total energy consumed by memory << 0.0102929 C (87.7876%)
Total charge consumed << 0.0117248 C
Remaining charge in the battery << 8999.99 C

ArchC: Simulation statistics
    Times: 3.18 user, 0.01 system, 3.20 real
    Number of instructions executed: 1975000
    Simulation speed: 621.07 K instr/s
```

Trecho de Código 7.1: Saída do simulador relativa à consumo de energia.

Quando sendo executado como software embarcado, o algoritmo exigiu a execução de 1975000 instruções no processador para executar a tarefa de criptografar dez vezes a citada palavra. Essa simulação exigiu 3,2s numa máquina AMD Turion X2 rodando à 1,9GHz com 2GB de memória RAM em ambiente Linux Ubuntu 10.10. Foi observado nesse cenário que, devido ao grande número de acessos à memória, 87% da energia usada pelo sistema em chip foi consumida pelo bloco de memória. Como os módulos de hardware AES e temporizador não foram usados nesta simulação, o consumo de cada um foi nulo. Como pode ser observado no trecho acima, é possível visualizar o consumo de cada bloco de hardware isoladamente.

Quando o algoritmo foi simulado como bloco de hardware, o processador executou cerca de 1485 instruções para criptografar dez vezes a mesma palavra. Essa nova simulação exigiu 0,02s para ser feita na mesma máquina já citada. Além disso, o consumo de energia se mostrou cerca de 800 vezes menor para a execução da tarefa como um todo. Nota-se novamente que a memória foi a responsável pela maior parte do consumo de energia mas, neste caso, o consumo total foi 3 ordens de grandeza inferior. Nota-se também que os módulos AES e temporizador consumiram energia visto que foram ligados. Nesta simulação, o temporizador era responsável por gerar interrupções ao processador que ordenava uma nova operação de cifragem feita pelo módulo AES.

```
Total energy consumed by mips << 1.0759e-06 C (6.92139%)
Total energy consumed by bus << 0 C (0%)
Total energy consumed by AES << 6.721e-08 C (0.43237%)
Total energy consumed by timer << 9e-09 C (0.057898%)
Total energy consumed by memory << 1.43925e-05 C (92.5883%)
Total charge consumed << 1.55446e-05 C
Remaining charge in the battery << 9000 C

ArchC: Simulation statistics
    Times: 0.00 user, 0.00 system, 0.02 real
    Number of instructions executed: 1485
    Simulation speed: (too fast to be precise)
```

Trecho de Código 7.2: Saída do simulador relativa à consumo de energia.

A Tabela 7.2 mostra uma comparação entre as duas simulações.

Tabela 7.2: Comparação entre as simulações de SoC executando AES.

	AES em Software Carga consumida [μC]	%Total	AES em hardware Carga consumida [μC]	%Total
MIPS	1431,87	12,21	1,08	6,92
Memória	10292,9	87,79	14,4	92,59
Temporizador	0	0	0,009	0,06
AES	0	0	0,07	0,43
TOTAL	11724,8	100	15,54	100

Esse comportamento era esperado pois sabe-se que blocos de hardware dedicados são mais eficientes quando se trata de performance e consumo de energia.

Esta abordagem permite ao projetista ter informações sobre qual bloco de hardware é o maior responsável por consumo de energia dentro do SoC. De posse desta informação é possível canalizar os esforços para otimizar o bloco com maior consumo visando uma redução do consumo do SoC como um todo. Usando este estudo de caso e supondo, por exemplo, que o algoritmo AES devesse ser embarcado no processador, seria mais vantajoso do ponto de vista do consumo de energia do SoC como um todo otimizar a memória já que ela é responsável por cerca de 88% do consumo. E por otimizar pode-se entender reprojetar a memória de um ASIC, alterar o módulo IP a ser integrado no SoC ou mesmo alterar o chip de memória usado no caso de uma solução discreta.

7.2 RESULTADOS DE SIMULAÇÕES DE RSSF

Esta Seção apresesenta resultados de simulações de RSSF implementadas usando o modelo já descrito. Os dados de consumo para os componentes do hardware modelados foram obtidos a partir de documentação dos circuitos já mostrados na Tabela 7.1 e o transceptor foi baseado no CC2420 [2].

7.2.1 Resultados de Simulações de Rede em Estrela

Como um estudo de caso, foi simulada uma RSSF em topologia estrela, conforme mostrado na Figura 7.1, para avaliar o consumo de energia em cada nó da rede. Apesar do número de nós na rede ser arbitrário e facilmente modificável, uma rede pequena foi simulada, com 4 nós e uma estação de campo. Nessa rede, todos os nós foram modelados em nível arquitetural, como descrito no Capítulo 6, uma vez que o objetivo é estudar o consumo de energia em cada nó.

Nós descritos no nível de detalhe mostrado no Capítulo 6 apresentam grande peso computacional. Portanto, para simular uma rede com grande número de nós, a adição de nós modelados em nível funcional é recomendada, apesar de não permitirem a estimativa de consumo, gerando uma rede com hardware modelado em diferentes níveis de abstração. Nós modelados em nível arquitetu-

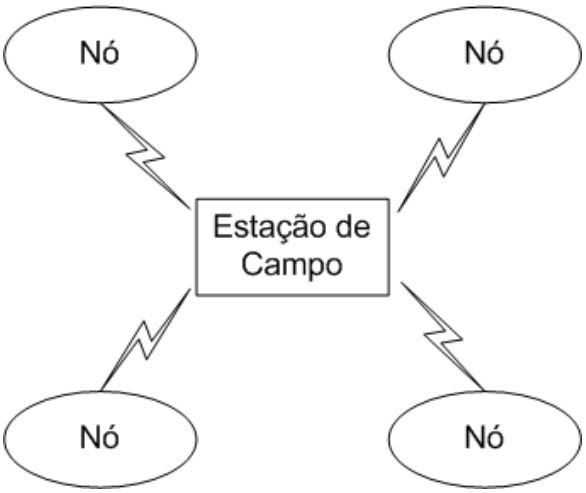

Figura 7.1: Topologia em estrela para uma RSSF.

Tabela 7.3: Dados de consumo de energia de hardware usados para todas as redes simuladas.

	Consumo de Corrente [mA]	Tempo drenando carga da bateria
MIPS [59]	11,6	62,5ns
Memória [33]	33	85ns
Temporizador [35]	0,1	Sempre que ligado
Transceptor [2]	17,4 em modo TX	Sempre que ligado
	19,7 em modo RX	

ral seriam colocados em pontos estratégicos ou representativos de forma que a análise de consumo proporcionada por eles pudesse valer para outros nós modelados em nível funcional.

Para a estimativa de consumo os dados da Tabela 7.3 foram carregados no modelo tal como já discutido.

A seguir está mostrado o comportamento da rede simulada. A rede foi contruída de forma que os nós respondam às requisições de dados vindos da estação de campo. A aplicação foi criada dessa forma por simplicidade. Manter o transceptor ligado é extremamente custoso do ponto de vista do consumo de energia e deve ser evitado para aumentar o tempo de vida da rede. Resultados mostrados em [61] com base em caracterizações elétricas de motes para RSSF, mostram que essa estratégia não leva a um aumento linear no tempo de vida do nó.

Dado o nível de detalhes modelados neste trabalho, é possível observar nas simulações que existe um número de nós na rede para o qual as respostas enviadas chegam mais rápido do que a estação de campo é capaz de tratá-las. Observa-se que este número é função do código de aplicação já que o número de instruções a serem executadas pelo processador para o tratamento de um pacote vindo da rede varia conforme esse código. Esta observação mostra a importância de se conhecer a capacidade do hardware usado para implementação de uma RSSF.

1. No início da simulação, a estação de campo e os nós são ligados e configurações iniciais, como

configurar o temporizador e montagem do pacote de dados, são feitas. Na estação de campo são efetuadas configurações do temporizador e montagem do pacote a ser enviado. Para os nós, a montagem dos pacotes é feita e todos são colocados em modo recepção aguardando que a estação de campo se manifeste.

2. A cada 10 segundos, o circuito temporizador gera uma interrupção no microcontrolador que por sua vez ordena o envio do pacote de requisição de dados para os nós. Ao fim da transmissão, o transceptor da estação de campo é colocado em modo recepção para receber as respostas dos nós.

3. Ao receber o pacote de requisição vindo da estação de campo, todos os nós geram um dado aleatório, simulando a aquisição de informações do ambiente, mudam o modo do transceptor para transmissão, transmitem o dado e retornam para o modo recepção aguardando o próximo pedido de dados a vir da estação de campo.

4. Ao receber a resposta de todos os nós da rede, a estação de campo desliga o transceptor e aguarda a próxima interrupção do temporizador. Este processo se repete por um número arbitrário de vezes, neste caso 50.

A Figura 7.2 apresenta a evolução da carga restante nas baterias ao longo do tempo de vida da rede e associada à Figura 7.1, permite levantar o mapa de energia da rede.

Na Figura 7.2(a), a evolução da carga restante em todos os nós da rede. Observando essa Figura, pode-se notar dois comportamentos bem distintos na simulação: da estação de campo e dos nós. Observa-se que os resultados obtidos para a descarga da bateria foram lineares. Isso ocorre já que a equação que implementa o consumo é linear para cada módulo e o comportamento dos nós da rede não muda ao longo da simulação. Resultados não-lineares poderiam acontecer caso a o comportamento da rede fosse alterado por algum evento.

Os nós apresentam consumo bastante superior ao da estação base como esperado, o que pode ser notado pela inclinação das retas. Como os nós ficam a maior parte do tempo no modo recepção e o consumo de carga neste modo é constante obtem-se um comportamento linear para o consumo. No entanto este não é um resultado geral. A forma como a aplicação foi definida criou este cenário. Qualquer outra aplicação poderia ter sido simulada com o mesmo modelo de hardware com resultados diferentes. Observa-se também que os nós tem gasto de energia bastante similar. Esse comportamento é esperado uma vez que todos os nós se comportam exatamente da mesma forma, respondendo a requisições da estação de campo. A única diferença entre eles é o instante em que as ações são executadas como pode ser observado em 7.2(b). Ao fim da simulação, a carga restante nas baterias de cada nó era de cerca de 8990C. Estimando linearmente o tempo de vida dos nós, a rede estaria funcional por cerca de 122 horas. Portanto, neste cenário, para uma rede estrela com N nós, pode-se substituir N-1 nós descritos em nível arquitetural por modelos funcionais e generalizar o consumo de um nó para todos os outros visando ganhos em performance de simulação. Esse procedimento tem como objetivo reduzir o custo computacional da simulação. De fato, a Figura 7.3 mostra o tempo de simulação para uma rede estrela sob as mesmas condições com número variado de nós. Nota-se que o tempo necessário para a simulação da rede com todos os nós modelados em

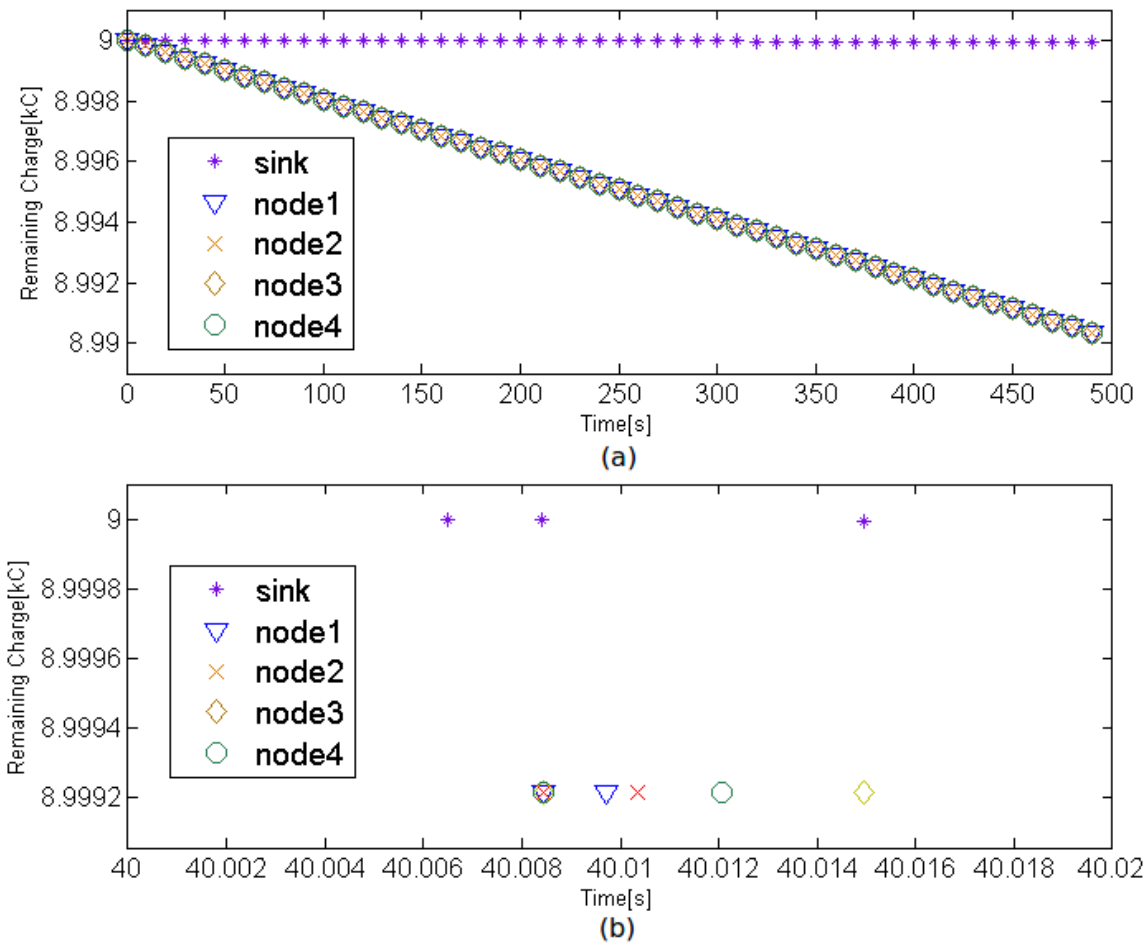

Figura 7.2: (a) Evolução da carga restante nos nós da rede ao longo de toda a simulação. (b) Detalhe da evolução da carga em torno de $40s_{sim}$.

nível arquitetural aumenta rapidamente. Frente ao resultado mostrado na Figura 7.3, é importante identificar as classes de nós da rede, como comentado no Capítulo 4, e substituir nós modelados em nível arquitetural por modelos funcionais.

O gasto de energia da estação de campo é muito menor do que o dos nós pelo fato deste hardware passar maior parte do tempo desligado. De fato, ao longo de toda a simulação a estação de campo manteve o transceptor ligado, seja em modo recepção ou transmissão, durante apenas $0,38s_{sim}$ enquanto os nós mantiveram o transceptor por toda o tempo, cerca de $490s_{sim}$. Ao fim da simulação, a bateria da estação de campo continha 8999,94C.

A Tabela 7.4 mostra o consumo de cada bloco de hardware modelado para a estação de campo e para um dos nós já que todos se comportam de maneira similar e a Tabela 7.5 mostra o tempo acumulado que os transceptores ficaram ligados durante a simulação.

Esta simulação levou 0,55 segundos em uma máquina AMD Turion X2 à 1,9GHz com 2GB de memória em ambiente Linux Ubuntu 10.10.

De posse dos dados da Tabela 7.4, é possível ao projetista da rede avaliar se o compromisso entre processamento local dos dados é proveitoso frente à transmissão dos dados coletados sem nenhum

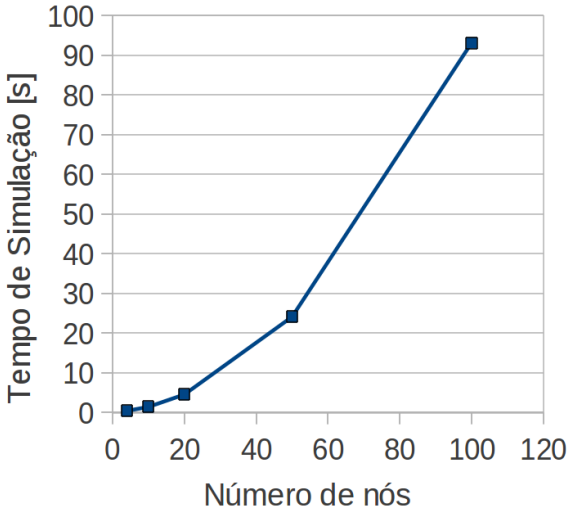

Figura 7.3: Dependência entre tempo necessário para a simulação da RSSF e número de nós arquiteturais.

Tabela 7.4: Consumo dos blocos de hardware dos nós da RSSF em estrela.

	Estação de Campo	Nó Sensor
MIPS	2.0e-05 C (0,036%)	1.06e-05 C (0.0001%)
Memória	0.0001 C (0,2%)	6.16e-05 C (0,0006%)
Temporizador	0.05 C (86,14%)	0
Transceptor	0.008 C (13,82%)	9.85 C (99,99%)

tratamento. Isso colabora com a ideia de RSSF mais complexas já que os compromissos de projeto podem ser mais bem trabalhados.

Além disso, a simulação fornece dados sobre o tempo de uso do transceptor, mostrados na Tabela 7.5, já que grande parte do consumo está associada a este circuito. Com estas informações é possível ainda otimizar o tempo em que o transceptor é mantido ligado permitindo avaliar, por exemplo, performance de um protocolo de comunicação qualquer a ser usado numa solução. Pela forma como a aplicação foi construída, nota-se que o transceptor da Estação de Campo está quase sempre desligado enquanto que o transceptor dos nós está ligado todo o tempo e em modo de recepção. Isso poderia apontar para uma necessidade de otimização na forma de comunicação entre os nós caso

Tabela 7.5: Tempo acumulado com o transceptor RF ligado em s_{sim}.

	Estação de Campo	Nó Sensor
Tempo com o Transceptor em modo TX	0,03s_{sim}	0,04s_{sim}
Tempo com o Transceptor em modo RX	0,32s_{sim}	489,9s_{sim}

esta fosse uma aplicação real.

7.2.2 Resultados de Simulações com Rede com Multihopping

Como um segundo estudo de caso com objetivo de obter resultados para redes com outras topologias, foi simulada uma RSSF como mostrada na Figura 7.4. Esta rede contém um nó roteador responsável por gerar um dado e repassar o dado de um nó isolado devido à distância.

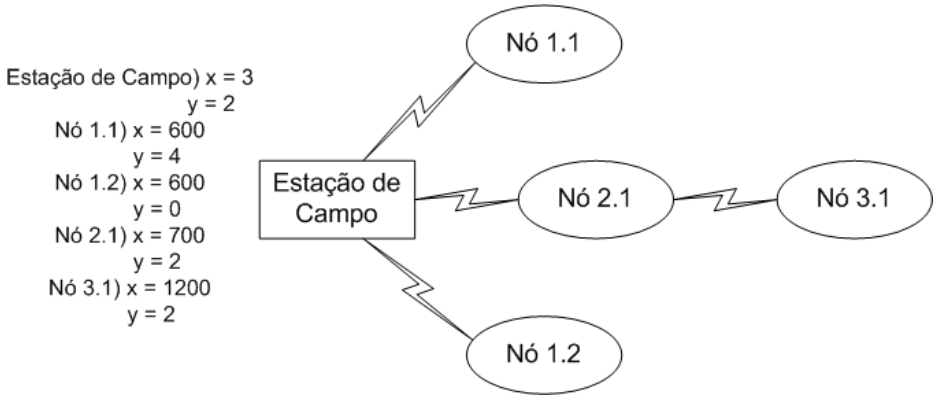

Figura 7.4: Topologia multihop para uma RSSF.

Nesta RSSF, identificam-se quatro classes de nós: (i) Nós 1.x , (ii) Nó 2.1, (iii) Nó 3.1 e a estação de campo. Cada classe de nós identificada possui um dado comportamento apesar de todos possuirem o mesmo hardware.

- Estação de Campo: responsável por receber todos os dados.
- Nós 1.x: possui comunicação direta com a estação de campo, se assemelhando portanto, à rede estrela.
- Nó 2.1: possui comunicação direta com a estação de campo mas é responsável por encaminhar os dados vindos do nó 3.1 para a estação de campo além de entregar seus próprios dados.
- Nó 3.1: dada à localização deste nó, não existe comunicação direta com a estação de campo devendo enviar seus dados ao nó 3.1.

A simulação desta RSSF está baseada em eventos gerados pelos nós ao contrário da simulação da rede em estrela apresentada anteriormente, em que os eventos eram gerados na Estação de Campo.

1. No início da simulação, a Estação de Campo é ligada com transceptor em modo recepção e permanece desta forma até o fim da simulação. Os outros nós tem o circuito temporizador configurado para (i)10 segundos para os nós 1.x, (ii)10,5 segundos para o nó 2.1 e (iii)11 segundos para o nó 3.1. Isso tem o objetivo de evitar que todos os nós acessem o meio ao mesmo tempo.

2. Após 10 segundos, o circuito temporizador dos nós 1.x gera uma interrupção para o processador que ordena o envio de um pacote para a Estação de Campo. Após o envio, o nó é desligado visando economia de energia.

3. Após meio segundo, o circuito temporizador do nó 2.1 gera interrupção para o processador que comanda o envio de um pacote para a Estação de Campo. Após o envio, o nó liga o transceptor em modo recepção e aguarda o dado vindo do nó 3.1.

4. Após meio segundo, o circuito temporizador do nó 3.1 gera interrupção para o processador que comanda envio de um pacote para o nó 2.1. Após o envio, o nó é desligado.

5. Ao receber o pacote vindo do nó 3.1, o nó 2.1 o encaminha para a Estação de Campo e logo em seguida se desliga.

6. Nas interrupções subsequentes geradas pelos temporizadores, o período em que as interrupções devem ocorrer é reconfigurado para 10 segundos em todos os nós. Desta forma, o intervalo de meio segundo é mantido entre os eventos.

A Figura 7.5 mostra a evolução da carga restante nas baterias de cada nó com o tempo. Observando essa Figura, nota-se que existem três classes de nós quando trata-se de consumo de energia: (i)a estação de campo, (ii)o nó 2.1 e (iii)os nós 1.x e 3.1. Para melhorar a visualização, a Figura 7.6 mostra a evolução do consumo apenas para os nós 1.x, 3.1 e 2.1. Os consumo de energia dos nós foi obtido conforme esperado dado o comportamento da rede.

A estação de campo apresenta o maior consumo de energia já que se mantem ligada todo o tempo em modo recepção aguardando pacotes dos outros nós. Uma vez que sabe-se que este nó tem consumo superior para a aplicação descrita, uma possível solução seria a escolha de uma bateria com maior capacidade. O trecho de código 7.3 mostra a saída do simulador para a estação de campo. Observa-se que o grande responsável pelo consumo de energia neste nó é o transceptor que se mantém todo o tempo em modo recepção.

```
Sync

Total energy consumed by mips << 1.42173e-05 C (0.000144023%)
Total energy consumed by bus << 0 C (0%)
Total energy consumed by transceiver << 9.87138 C (99.999%)
Total energy consumed by timer << 0 C (0%)
Total energy consumed by memory << 8.45175e-05 C (0.000856178%)
Remaining charge in the battery << 8990.13 C
Total time using the transceiver << 501.086 s.
Total time in TX mode << 0 s.
Total time in RX mode << 501.086 s.
```

Trecho de Código 7.3: Saída do simulador relativa ao consumo de energia da estação de campo.

Os nós 1.x e 3.1 apresentam consumo semelhante. O comportamento destes nós se baseia em enviar dados para um outro nó, independente de qual seja, dentro dos mesmos intervalos de tempo de 10 segundos. Dessa forma, o comportamento do consumo de energia é esperado e, como eles

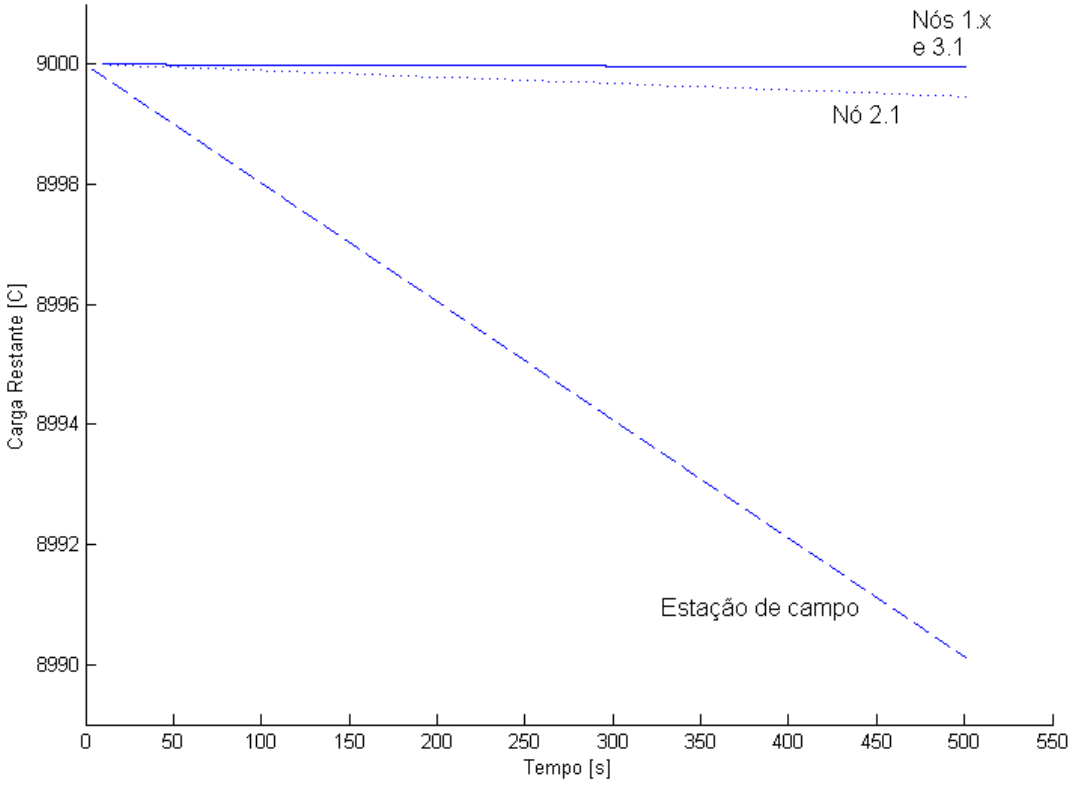

Figura 7.5: Evolução da carga restante das baterias ao longo da simulação.

se mantém desligados a maior parte do tempo, apresentam o menor consumo de energia como seria desejado para nós de RSSF. O trecho de código 7.4 mostra a saída do simulador para um destes nós.

```
node1_2

Total energy consumed by mips << 5.6405e-06 C (0.0111414%)
Total energy consumed by bus << 0 C (0%)
Total energy consumed by transceiver << 0.000584128 C (1.1538%)
Total energy consumed by timer << 0.0499984 C (98.7591%)
Total energy consumed by memory << 3.8465e-05 C (0.0759777%)
Remaining charge in the battery << 8999.95 C
Total time using the transceiver << 0.03264 s.
Total time in TX mode << 0.03264 s.
Total time in RX mode << 0 s.
```

Trecho de Código 7.4: Saída do simulador relativa ao consumo de energia dos nós 1.x e 3.1.

O nó 2.1 apresenta consumo de energia intermediário entre a estação de campo e os outros nós da RSSF já que acumula funções de enviar os dados para a estação de campo e encaminhar os dados vindos do nó 3.1. Observando o consumo deste nó e o comparando com os nós 1.x podemos inferir que o uso de comunicação direta com a estação de campo é preferível já que traz consumo de energia inferior. A adoção de comunicação com saltos ("multihopping") afeta, portanto, o mapa de energia

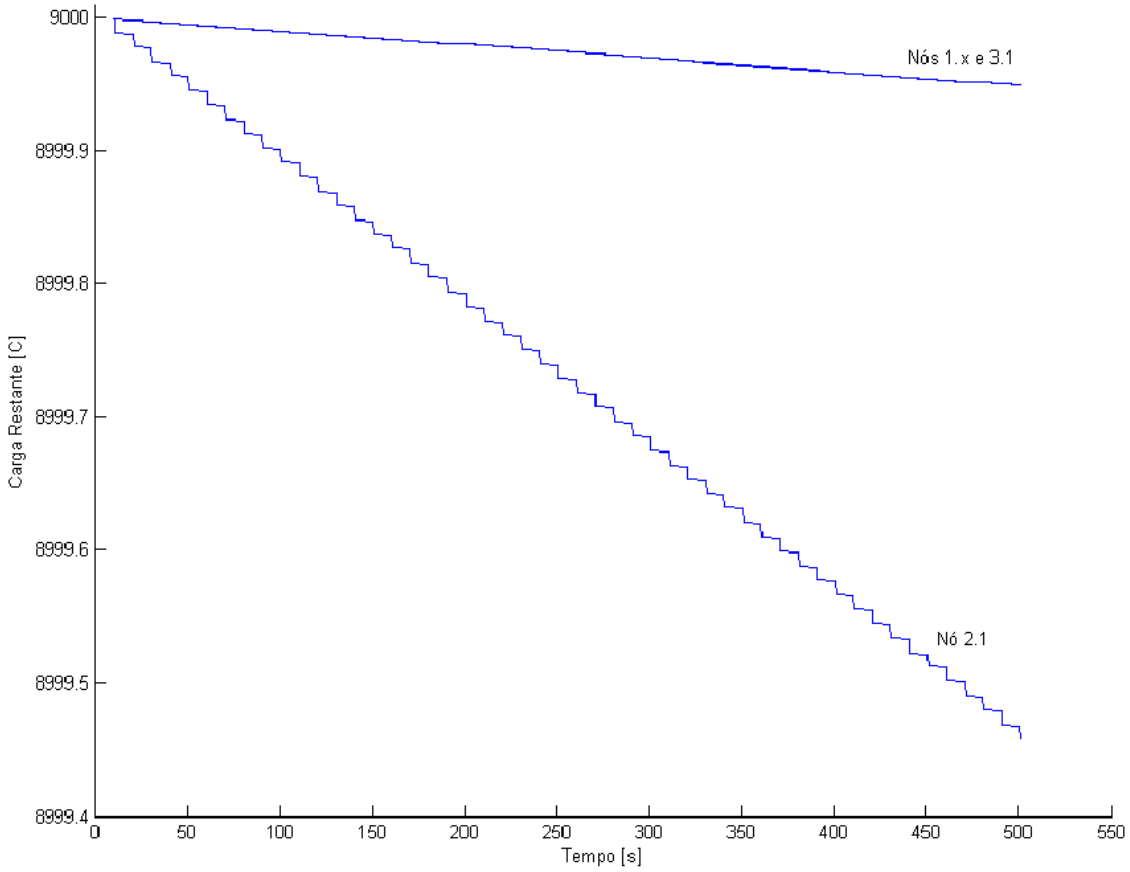

Figura 7.6: Evolução da carga restante das baterias dos nós 1.x, 3.1 e 2.1.

da rede em todos os nós que participam da rota. Outra razão para preferir comunicação direta à *multihopping* é o fato que a eficiência dos circuitos de RF do transceptor aumenta com a potência de saída. O trecho de código 7.5 mostra a saída do simulador para o nó 2.1.

```
node2_1

Total energy consumed by mips << 1.2702e-05 C (0.00234555%)
Total energy consumed by bus << 0 C (0%)
Total energy consumed by transceiver << 0.493857 C (91.1957%)
Total energy consumed by timer << 0.0475895 C (8.7879%)
Total energy consumed by memory << 7.59678e-05 C (0.0140282%)
Remaining charge in the battery << 8999.46 C
Total time using the transceiver << 25.0749 s.
Total time in TX mode << 0.064 s.
Total time in RX mode << 25.0109 s.
```

Trecho de Código 7.5: Saída do simulador relativa ao consumo de energia do nó 2.1.

A simulação desta RSSF levou 0,6 segundos em uma máquina AMD Turion X2 à 1,9GHz com 2GB de memória em ambiente Linux Ubuntu 10.10.

A Tabela 7.6 mostra uma comparação entre o consumo dos nós e a Tabela 7.7 mostra o tempo

acumulado em que o transceptor de cada módulo ficou ligado.

Tabela 7.6: Comparação entre o consumo dos nós da rede multihopping.

	Estação de Campo	Nó 1.x e 3.1	Nó 2.1
MIPS	1,42e-05 C (0%)	5,64e-06 C (0.01%)	1,27e-5 C (0,002%)
Memória	8,45e-5 C (0%)	3,85e-05 C (0,08%)	7,6e-5 C (0,014%)
Temporizador	0 C (0%)	0,05 C (98,76%)	0,048 C (8,79%)
Transceptor	9,87 C (99,99%)	0,0006 C (1,15%)	0,49 C (91,19%)

Tabela 7.7: Tempo acumulado com o transceptor RF ligado em s_{sim}.

	Estação de Campo	Nó 1.x e 3.1	Nó 2.1
Tempo com o Transceptor em modo TX	0	0,033 s_{sim}	0,064 s_{sim}
Tempo com o Transceptor em modo RX	501,1 s_{sim}	0	25, 01 s_{sim}

A simulação de uma rede em que um nó encaminha dados de outro mostra que o simulador implementado é capaz de tratar redes mais complexas do que as apresentadas na seção anterior. Simulações de protocolos de comunicação específicos, técnicas de localização de rotas de comunicação entre outros problemas de RSSF não estão no escopo deste trabalho. No entanto, o uso dos modelos apresentados permite essas avaliações. De posse de um código de aplicação completo, a abordagem mostrada pode ser usada para simular a rede com todas as vantagens comentadas neste texto.

7.3 DISCUSSÃO

Apesar das RSSF simuladas serem mais simples que as redes encontradas na literatura, o objetivo deste texto é demonstrar a estratégia de estimação de consumo de energia. Usando o SoC descrito, qualquer RSSF pode ser simulada seguindo os passos a seguir:

1. Criação do número de nós desejados no ambiente de simulação;
2. Conexão dos nós com os *proxies* da biblioteca SCNSL seguido do posicionamento desejado para cada nó;
3. Recompilação do simulador;
4. Existência do código de aplicação à ser embarcado nos nós da rede que represente a aplicação desejada.

Sendo feitas as alterações no simulador descritas acima de acordo com a aplicação desejada, dispondo do programa de aplicação para cada nó, basta compilar os códigos de aplicação para o processador usado e executar a aplicação. Desta forma a abordagem de estimativa de consumo de energia é aplicável para qualquer RSSF.

Uma vez apresentados os resultados e capacidades do modelo descrito neste trabalho, uma breve comparação entre resultados dos trabalhos [49], [50], [51] e este será feita nesta seção. Esses trabalhos foram escolhidos por apresentar metodologia de modelagem semelhante à escolhida para o modelo apresentado, usando ferramentas semelhantes assim como as formas de descrição dos blocos de hardware.

O trabalho *Modeling Energy Consumption of Wireless Sensor Networks by SystemC* [49] usa a biblioteca SCNSL para modelar uma RSSF. Neste trabalho o hardware dos nós foi modelado como máquinas de estados finitos sendo que as transições entre os estados estão associadas a um consumo de corrente constante. Entre os resultados apresentados está o consumo de energia de cada bloco de hardware e o consumo por pacote. Este trabalho não deixa claro quais os estados ou tarefas foram modelados para o processador, levando à dúvidas na interpretação dos resultados.

Em *High level energy consumption estimation and profiling for optimizing Wireless Sensor Networks* [50], o processador implementa o conjunto de instruções, evitando as dúvidas do trabalho anterior, enquanto modela os outros circuitos, como o transceptor, como máquinas de estado. Uma aplicação fictícia é implementada e são apresentados resultados de consumo de energia por modo de operação do transceptor. Apesar de dispor de um modelo acurado para o processador, os resultados de consumo do processador não são apresentados e nem suas potencialidades, comentadas.

Dentre todos os analisados, este trabalho se assemelha ao *Flexible Energy-Aware Simulation of Heterogeneous Wireless Sensor Networks* [51] por permitir a simulação de nós de RSSF descritos na forma funcional e na forma arquitetural no mesmo ambiente de simulação. Resultados de consumo de energia do processador para diferentes comportamentos estão mostrados enquanto resultados de consumo para o transceptor são omitidos. A simulação apresentada em [51] permite avaliação de consumo de energia de forma bastante semelhante à descrita neste texto. No entanto, a adição de um módulo *bateria* capaz de atuar sobre os nós da rede desligando-os na falta de energia permite lidar com a saída de nós da rede devido à falta de energia. Esta funcionalidade não está implementada em nenhum dos três trabalhos.

Uma proposta de metodologia que usa modelagem de sistemas em alto nível está apresentada no Anexo IV. Essa metodologia tem como objetivo resolver os problemas identificados no Capítulo 4, servindo de base para o projeto de RSSF genéricas.

Este trabalho se concentrou em demostrar uma estratégia de uso de modelagem em alto nível para simulação do consumo de energia de RSSF genéricas. Trabalhos futuros involvem o desenvolvimento de uma biblioteca de modelos que permitirá fazer comparações de performance da RSSF comparando diferentes plataformas de hardware. Outra evolução deste trabalho é a simulação de redes mais próximas às construídas na realidade, com protocolos de comunicação mais realistas e maior número de nós.

8 CONCLUSÕES

Neste trabalho foi implementada uma metodologia de modelagem de consumo de energia capaz de estimar consumo de todos os componentes de hardware de um nó sensor simulado em RSSF associado a uma aplicação. Os principais objetivos deste trabalho eram:

- Desenvolver modelos de SoC capazes de permitir estimativas de consumo de energia;
- Validar os modelos por meio de estudos de caso.

O primeiro estudo de caso mostra um SoC isolado executando o algoritmo de criptografia AES em duas situações diferentes: embarcado no processador como código de aplicação e em hardware simulando um módulo IP ou ASIC. Os resultados da simulação mostram que o consumo de energia é cerca de 800 vezes menor quando o algoritmo é executado em hardware.

Uma primeira RSSF em estrela foi simulada. Essa rede é composta por quatro nós e uma estação de campo que recebe os dados de todos os nós. Foram mostrados gráficos com a descarga das baterias de todos os entes da rede e dados de consumo de energia de cada componente dos nós e da estação de campo. De posse dessas informações é possível otimizar o consumo de energia: (i)do hardware dos nós para uma dada aplicação e (ii)da rede.

Outra RSSF simulada apresenta um estágio com *hopping* de forma a demostrar a capacidade do modelo em lidar com topologias de rede mais complexas. Foram mostrados os gráficos de descarga das baterias para todos os nós da rede mostrando, conforme esperado, que o nó responsável por encaminhar pacotes apresenta consumo superior aos que entregam seus dados diretamente para a estação de campo, para o cenário estudado. Dados de consumo de energia de cada componente dos nós foi apresentado permintindo direcionar esforços de projeto de hardware e da rede.

Observando os resultados mostrados ao longo do texto, pode-se concluir que os objetivos foram atingidos. O modelo de consumo de energia está descrito no Capítulo 5 e a implementação do SoC contendo uma bateria está comentada no Capítulo 6. Os resultados de simulações de um SoC isolado, de uma RSSF em estrela e de uma outra RSSF com *multihopping* estão mostrados e comentados no Capítulo 7. Simulações de RSSF mais próximas às reais, com protolocos de comunicação típicos e maior número de nós, não foram mostradas devido à grande complexidade na criação de cenários e códigos de aplicação, que fogem ao escopo deste trabalho. Apesar dessas rede mais complexas não terem sido apresentadas, a metodologia de estimação de consumo de energia é geral e pode ser empregada na simulação de qualquer RSSF.

A abordagem usada neste trabalho apresenta as seguintes vantagens:

- Proporciona uma estimativa de consumo de energia para cada nó sensor;
- Mostra, dentro de um dado nó sensor, quais blocos são responsáveis por maior ou menor consumo;

- Permite levantar informações sobre o mapa de energia da rede ao longo do tempo;
- Melhora o fluxo de projeto de RSSF ao incorporar ferramentas de simulação mais adequadas ao projeto de sistemas;
- Oferece a capacidade de execução de software embarcado em etapas iniciais do projeto;
- Permite a simulação um SoC em ambiente de RSSF;
- Adota ferramentas de código aberto, com todas as vantagens inerentes a esta prática.

A estratégia de modelagem de energia difere das demais encontradas em trabalhos correlatos, apresentados no Capítulo 3 e apresenta funcionalidades capazes de auxiliar o projeto de RSSF. De fato, a existência de um módulo *bateria* permite: (i)agregar toda a informação de consumo de energia em um módulo criado para este fim, evitanto, portanto, alterações em outros módulos e (ii)desligar o nó quando não existe mais energia disponível para alimentá-lo, sendo possível estudar como a rede se comporta frente à saída de um ou mais nós.

Identificam-se como trabalhos futuros os itens a seguir:

- A criação de interface gráfica para visualização e reconfiguração do comportamento da rede;
- O desenvolvimento de uma biblioteca de modelos capazes de permitir simulação de RSSF com nós sensores diferentes;
- Refinamentos na comunicação, que atualmente se baseia em equações simples e não realistas em diversas situações;
- Validação do modelo apresentado por meio de comparação com caracterizações elétricas de hardware;
- Refinamento e implementação da metodologia e da ferramenta de auxílio a projeto descrita no Anexo IV.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] KARL, H.; WILLIG, A. *Protocols and Architectures For Wireless Sensor Networks*. [S.l.]: Wiley-Interscience, 2007. ISBN 0470519231.

[2] TEXAS INSTRUMENTS. CC2420: 2.4 GHz ieee 802.15. 4/zigbee-ready RF transceiver. *Available at Available at http://www. ti. com/lit/gpn/cc2420*, 2006.

[3] MEDEIROS, J. E. G. de. *Modelagem em SystemC de um SoC em Ambiente de Redes de Sensores Sem Fio*. Dissertação (Mestrado) - Universidade de Brasília, 2010.

[4] BLACK, D. et al. *SystemC: From The Ground Up*. [S.l.]: Springer Verlag, 2008. ISBN 0387699570.

[5] FUMMI, F.; QUAGLIA, D.; STEFANNI, F. A systemc-based framework for modeling and simulation of networked embedded systems. *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on*, p. 49 –54, sep. 2008.

[6] GHENASSIA, F. *Transaction-Level Modeling With Systemc: TLM Concepts and Applications For Embedded Systems*. [S.l.]: Springer Verlag, 2005. ISBN 0387262326.

[7] ENERGIZER. *Datasheet Bateria NH15-2500*. [Online; acessado em junho de 2011].

[8] LOUREIRO, A. et al. Redes de sensores sem fio. In: *Simpósio Brasileiro de Redes de Computadores (SBRC)*. [S.l.: s.n.], 2003. p. 179–226.

[9] MCCANNE, S.; FLOYD, S. *NS Network Simulator - Version 2*. Acessado em Novembro/2011. Disponível em: <http://nsnam.isi.edu/nsnam/index.php/Main_Page>.

[10] VARGA, A.; PONGOR, G. *OMNeT++*. Acessado em Novembro/2011. Disponível em: <http://www.omnetpp.org/>.

[11] DRAGO, N.; FUMMI, F.; PONCINO, M. Modeling Network Embedded Systems With NS-2 and SystemC. In: IEEE. *Circuits and Systems for Communications, 2002. Proceedings. ICCSC'02. 1st IEEE International Conference on*. [S.l.], 2002. p. 240–245. ISBN 5742202601.

[12] WANG, L. et al. A wireless sensor system for biopotential recording in the treatment of sleep apnea disorder. In: IEEE. *Networking, Sensing and Control, 2006. ICNSC'06. Proceedings of the 2006 IEEE International Conference on*. [S.l.], 2006. p. 404–409.

[13] LLOSA, J. et al. Remote, a wireless sensor network based system to monitor rowing performance. *Sensors*, Molecular Diversity Preservation International, v. 9, n. 9, p. 7069–7082, 2009.

[14] ASSOUS, N. et al. Wireless sensors for instrumented machines: Propagation study for stationary industrial environments. In: IEEE. *Computer Aided Modeling and Design of Communication Links and Networks, 2009. CAMAD'09. IEEE 14th International Workshop on*. [S.l.], 2009. p. 1–5.

[15] LU, B. et al. Energy evaluation goes wireless. *Industry Applications Magazine, IEEE*, IEEE, v. 13, n. 2, p. 17–23, 2007.

[16] LLORET, J. et al. A wireless sensor network deployment for rural and forest fire detection and verification. *Sensors*, Molecular Diversity Preservation International, v. 9, n. 11, p. 8722–8747, 2009.

[17] VERMA, S.; CHUG, N.; GADRE, D. Wireless sensor network for crop field monitoring. In: IEEE. *2010 International Conference on Recent Trends in Information, Telecommunication and Computing*. [S.l.], 2010. p. 207–211.

[18] MAINWARING, A. et al. Wireless sensor networks for habitat monitoring. In: ACM. *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. [S.l.], 2002. p. 88–97.

[19] WERNER-ALLEN, G. et al. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, Published by the IEEE Computer Society, p. 18–25, 2006.

[20] HILL, J.; CULLER, D. Mica: A wireless platform for deeply embedded networks. *Micro, IEEE*, IEEE, v. 22, n. 6, p. 12–24, 2002.

[21] POLASTRE, J.; SZEWCZYK, R.; CULLER, D. Telos: Enabling ultra-low power wireless research. In: IEEE PRESS. *Proceedings of the 4th international symposium on Information processing in sensor networks*. [S.l.], 2005. p. 48–es.

[22] TEXAS INSTRUMENTS. Msp430 ultra-low-power microcontrollers. *http://focus.ti.com/"http://focus.ti.com*, 2009.

[23] CROSSBOW TECHNOLOGY. MICAz Datasheet. *San Jose, California*, 2006.

[24] ATMEL. Atmega 128l microprocessor datasheet. *Atmel Corporation*.

[25] DIGI International. *Xbee PRO Datasheet*. Acessado em julho/2011.

[26] BUSCHMANN, C.; PFISTERER, D. isense: A modular hardware and software platform for wireless sensor networks. *6. Fachgesprach Sensornetzwerke*, Citeseer, p. 15.

[27] COHN, G. et al. Snupi: Sensor nodes utilizing powerline infrastructure. In: ACM. *Proceedings of the 12th ACM iInternational Conference on Ubiquitous computing*. [S.l.], 2010. p. 159–168.

[28] MIPS Technologies Inc. *MIPS32 Architecture For Programmers Volume I: Introduction to the MIPS32 Architecture*. [S.l.]: MIPS Technologies, Inc., 2008.

[29] MIPS Technologies Inc. *MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set*. [S.l.]: MIPS Technologies, Inc., 2009.

[30] MIPS Technologies, Inc. *MIPS32 Architecture For Programmers Volume III: The MIPS32 Privileged Resource Architecture*. [S.l.]: MIPS Technologies, Inc., 2009.

[31] DANDAMUDI, S. P. *Guide to RISC Processors: for Programmers and Engineers*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 0387210172.

[32] ZIGBEE ALLIANCE. Zigbee specification 2006. *ZigBee Document, 053474r17*, 2008.

[33] HITACHI. *Datasheet HM62256A-8*. Acessado em julho/2011.

[34] ST Microelectronics. M25p80 datasheet. *Low Voltage, Serial Flash Memory With*, v. 40.

[35] NATIONAL SEMICONDUTORS. *Datasheet LMC 555*. Acessado em julho/2011.

[36] BESERRA, G. et al. System-level modeling of a mixed-signal system on chip for wireless sensor networks. In: IEEE. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. [S.l.]. p. 1–4.

[37] GROTKER, T. et al. *System Design With SystemC*. [S.l.]: Springer Netherlands, 2002. ISBN 1402070721.

[38] OSCI INITIATIVE. IEEE Standard SystemC Language Reference Manual. *IEEE Computer Society*, p. 1666–2005, 2006.

[39] AZEVEDO, R. et al. The archc architecture description language and tools. *International Journal of Parallel Programming*, Springer, v. 33, n. 5, p. 453–484, 2005.

[40] FUJIMOTO, K. S. P. R. M.; RILEY, G. F. *Network Simulation. Synthesis Lectures on Communication Networks*. [S.l.]: Morgan & Claypool Publishers, 2007.

[41] CORKE, P. et al. Environmental wireless sensor networks. *Proceedings of the IEEE*, IEEE, v. 98, n. 11, p. 1903–1917, 2010.

[42] PENOLAZZI, S.; HEMANI, A.; BOLOGNINO, L. A general approach to high-level energy and performance estimation in socs. In: IEEE. *VLSI Design, 2009 22nd International Conference on*. [S.l.], 2009. p. 200–205.

[43] DARGIE, W.; CHAO, X.; DENKO, M. Modelling the energy cost of a fully operational wireless sensor network. *Telecommunication Systems*, Springer, v. 44, n. 1, p. 3–15, 2010.

[44] MAK, N.; SEAH, W. How long is the lifetime of a wireless sensor network? In: IEEE. *2009 International Conference on Advanced Information Networking and Applications*. [S.l.], 2009. p. 763–770.

[45] SOMOV, A. et al. A methodology for power consumption evaluation of wireless sensor networks. In: IEEE. *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. [S.l.], 2009. p. 1–8.

[46] SANGIOVANNI-VINCENTELLI, A. Defining platform-based design. *EEDesign of EETimes*, 2002.

[47] COURTAY, A. et al. Wireless sensor network node global energy consumption modeling. In: *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*. [S.l.: s.n.], 2010. p. 54–61.

[48] DU, W.; MIEYEVILLE, F.; NAVARRO, D. Idea1: A systemc-based system-level simulator for wireless sensor networks. In: IEEE. *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*. [S.l.], 2010. p. 618–622.

[49] DU, W.; MIEYEVILLE, F.; NAVARRO, D. Modeling energy consumption of wireless sensor networks by systemc. In: IEEE. *Systems and Networks Communications (ICSNC), 2010 Fifth International Conference on*. [S.l.], 2010. p. 94–98.

[50] HAASE, J.; MOLINA, J. M.; GRIMM, C. High level energy consumption estimation and profiling for optimizing wireless sensor networks. In: IEEE. *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. [S.l.], 2010. p. 537–542.

[51] FUMMI, F. et al. Flexible energy-aware simulation of heterogeneous wireless sensor networks. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. [S.l.: s.n.], 2009. p. 1638–1643.

[52] RAZAVI, B. *Design of Analog CMOS Integrated Circuits*. [S.l.]: McGraw-Hill Series in Electrical and Computer Engineering. New York, USA: McGraw-Hill, 2001.

[53] RINCON-MORA, G.; ALLEN, P. Study and design of low drop-out regulators. *IEEE Journal of Solid-State Circuits*, v. 33, n. 1, 1998.

[54] ANTONOPOULOS, C. et al. Experimental evaluation of a wsn platform power consumption. In: IEEE. *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. [S.l.], 2009. p. 1–8.

[55] IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Std 802.15.4-2003*, p. 1 -670, 2003.

[56] FIPS, N. 197: Announcing the advanced encryption standard (aes). *Information Technology Laboratory, National Institute of Standards and Technology, Nov*, 2001.

[57] MADUREIRA, H. M. G. et al. Power consumption system-level modeling of a system on chip. *Proceedings of SForum 2011, Chip On The Cliffs*, 2011.

[58] LIU, L.; LUKE, D. Implementation of aes as a cmos core. In: IEEE. *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*. [S.l.]. v. 1, p. 53–56.

[59] MIPS Technologies Inc. *MIPS 32 4k*. Acessado em julho/2011.

[60] SONY ASIA PACIFIC. *NH-AA-B4EN: Rechargeable Batteries*. Acessado em Novembro/2011.

[61] NGUYEN, H. et al. Sensor node lifetime: An experimental study. In: IEEE. *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. [S.l.], 2011. p. 202–207.

[62] BEUTEL, J. Fast-prototyping using the btnode platform. In: EUROPEAN DESIGN AND AUTOMATION ASSOCIATION. *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*. [S.l.], 2006. p. 977–982.

ANEXOS

I. RELATO DOS DEPOIMENTOS DE PESQUISADORES CONSULTADOS

Neste anexo, relatos de conversas com pesquisadores serão relatadas. O objetivo é embasar algumas afirmações com opiniões de outros pequisadores que não tiveram contato com este trabalho.

A Prof. Dra. Katia Obraczka da UC Santa Cruz, California, EUA, fez uma visita ao Departamento de Engenharia Elétrica da Universidade de Brasília entre 18 e 19 de agosto de 2011. Nesses dois dias, ela fez duas palestras em que abordou, entre outros temas, o andamento da pesquisa em RSSF em sua Universidade.

Foram apresentadas, por exemplo, algumas aplicações visando monitoramento de animais da região e monitoramento dos ônibus do campus da UCSC. Em ambos os casos, o hardware para a implementação da RSSF foi construído pela equipe da Prof. Katia. No entanto, nenhuma etapa de avaliação de consumo foi feita. Simuladores de rede foram usados para projetar os protocolos e simular a RSSF de forma comportamental e, em seguida, partiu-se para o projeto do hardware e do software a ser embarcado que implementaria o comportamento desejado para a rede.

O Prof. Dr. Magdy A. Bayoumi da Universidade da Louisiana, EUA, esteve em visita à Universidade de Brasília inaugurando as atividades do Capítulo IEEE CAS na Seção Brasília em 5 de setembro de 2011. Sua palestra entitulada "*Wireless Sensors Networks: Current and Future Challenges*"apresentou desafios para pesquisa e implementação de RSSF além de dados sobre o mercado na área.

O Prof. Magdy trabalhou com implementação de RSSF em plataformas de petróleo para otimizar o andamento das atividades corriqueiras como catalogar informações da extração. Para esta aplicação, o hardware também foi projetado especificamente para monitoramento da plataforma. Não foram usados microcontroladores, como é usual em grande parte das RSSF, mas circuitos capazes de processamento digital de sinais (DSP) devido ao peso computacional da aplicação. Apesar da escolha do hardware não ter sido usual, nenhuma etapa de avaliação de performance ou consumo foi previamente executada.

A Prof[a]. Dr[a]. Linnyer Beatrys da Universidade de Maringá descreveu uma situação em que a aplicação era monitoramento do canto de pássaros. Foi identificado que o hardware usado não era capaz de adquirir o dado (no caso o canto do pássaro, que está na faixa de audio) devido à taxa de amostragem do conversor analógico-digital disponível. Outra limitação da plataforma de hardware é a largura de banda disponível. Caso fosse possível adquirir os dados, a taxa de bits necessária para o envio dos dados era superior à taxa disponível pelo circuito integrado de comunicação RF usado.

II. CARACTERIZAÇÃO DO KIT PARA RSSF DA NATIONAL INSTRUMENTS

Uma forma bastante precisa para aquisição de dados de hardware é a caracterização em laboratório. De posse das informações coletadas é possível estimar o comportamento do sistema quando operacional. Como um estudo de caso de caracterização de nós sensores para RSSF usou-se o kit de desenvolvimento de RSSF da National Instruments que dispõe de:

- Um nó sensor para aquisição de temperaturas por meio de termopares tipo J modelo NI WSN-3212;
- Um nó sensor para aquisição de tensões analógicas arbitrárias modelo NI WSN-3202;
- Uma estação base que deve ser conectada a um computador com LabView modelo NI WSN-9791.

Para a caracterização do consumo de energia, o esquema mostrado na Figura II.1 foi montado com o uso dos seguintes equipamentos:

- Kit de desenvolvimento de RSSF da National Instruments;
- Dois computadores (o primeiro com LabView 6 e outro com LabView 2009 versão de teste por motivos de compatibilidade com o kit);
- Uma fonte de tensão Agilent E3647A;
- Um osciloscópio HP 54600A;
- Um resistor de 15Ω.

Em uma primeira montagem, construiu-se uma RSSF em estrela, ou seja, todos os nós entregam dados diretamente para a estação base. A Figura II.2 mostra a evolução da corrente de alimentação ao longo do tempo para o nó sensor NI WSN-3212. Podemos observar a existência de três regimes para o consumo: (i)consumo baixo entre 0 e 1ms, (ii)consumo intermediário em torno de 1,5ms e (iii)consumo alto em torno de 2ms.

Apesar de podermos identificar esses regimes de consumo é impossível saber ao certo as operações executadas pelo nó sensor já que todo o controle da rede é feito automaticamente pelo LabView. Cabe ao usuário determinar apenas a periodicidade da leitura do sensor e configuração de rede em estrela ou *mesh*. Podemos inferir que o regime (iii) corresponde à comunicação uma vez que a corrente drenada é bastante alta.

Os resultados para o nó sensor NI WSN 3202 foram bastante semelhantes e serão omitidos.

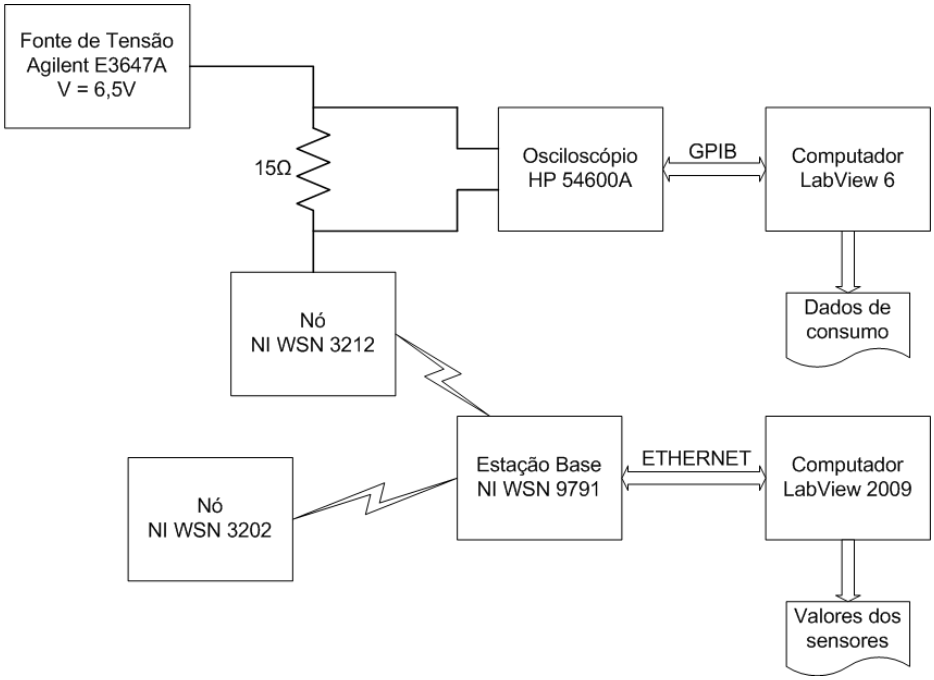

Figura II.1: Esquema de medida usado para a caracterização do hardware da NI.

Reconfigurando a rede para que o nó sensor NI WSN 3212 encaminhe os dados vindos do outro nó e repetindo o experimento obtem-se a Figura II.3. Podemos observar que a corrente drenada permanece alta por mais tempo indicando que o tempo de transmissão de dados é maior.

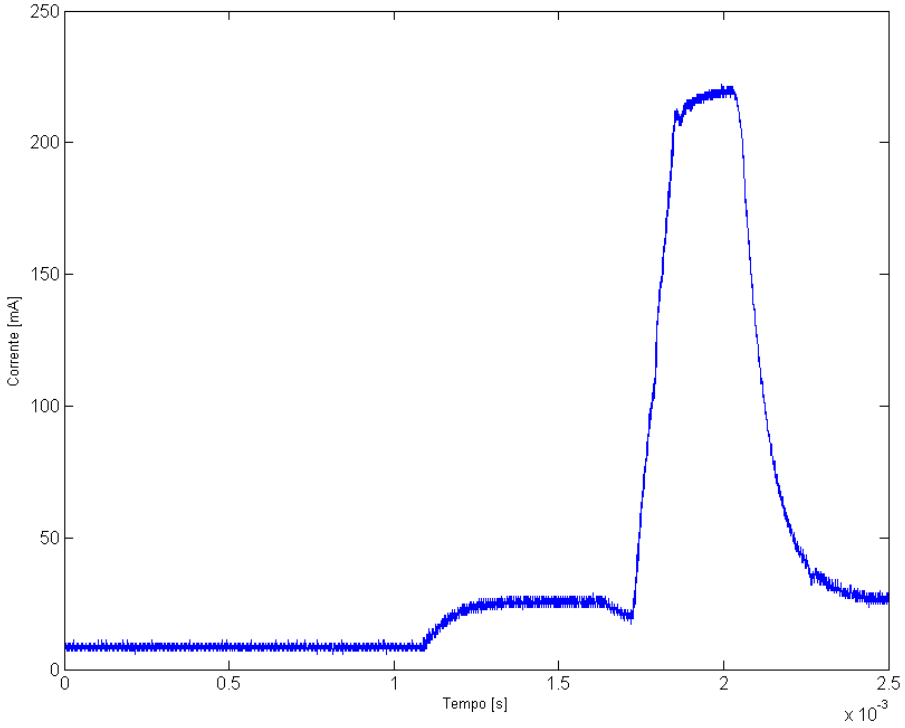

Figura II.2: Consumo de corrente para o hardware NI WSN 3212.

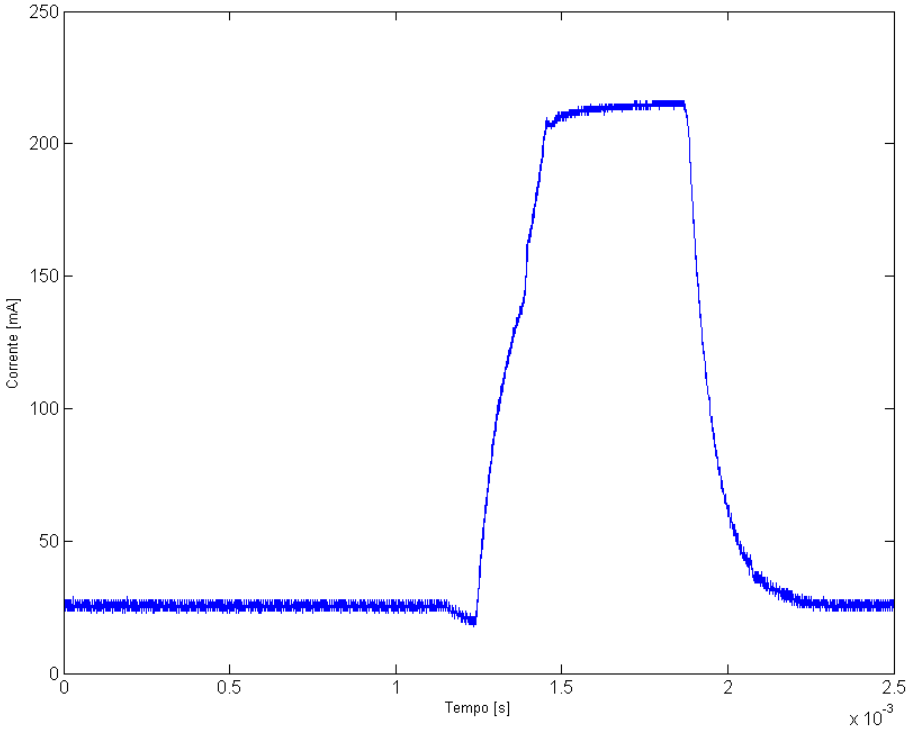

Figura II.3: Consumo de corrente para o hardware NI WSN 3212 quando ligado como roteador.

III. HARDWARE PARA RSSF

Neste anexo está mostrada uma Tabela que apresenta os componentes dos nós sensores comerciais.

Tabela III.1: Componentes dos motes comerciais.

Nome	Microcontrolador	Transceptor RF	Memória (Programa + Dados)	Programação
BTnode [62]	ATmega 128L	CC1000	64kBytes + 180kBytes	C
Mica	ATmega 103	RFM TR1000	128kBytes + 4kBytes	nesC
Mica2	ATmega 128L	CC1000	4kBytes	C
MicaZ [23]	ATmega 128L	CC2420	4kBytes	nesC
TelosB [21]	MSP430	CC2420	10kBytes	C
T-Mote Sky	MSP430	CC2420	10kBytes	C

No caso dos microcontroladores, nota-se uma predominância do ATmega 128L em nós sensores mais antigos e MSP430 nos mais atuais. Para os transceptores RF, nota-se uma predominância do CC1000 em nós sensores antigos e do CC2420 nos mais atuais.

IV. DESCRIÇÃO DO AMBIENTE DE PROJETO IMAGINADO

Neste anexo está descrito um ambiente de auxílio a projeto de RSSF baseado na metodologia apresentada no Capítulo 4. A ferramenta proposta aqui não foi implementada e carece de maturidade. A metodologia que foi proposta usa modelagem de sistemas em alto nível para resolver problemas identificados na forma de projeto de RSSF e tem por objetivo estabelecer um fluxo de projeto coeso para projetos de RSSF genéricas.

O ambiente a ser implementado deve seguir a metodologia de projeto apresentada na Seção 4.3. Portanto deve dispor de um simulador de rede, um simulador de hardware como o kernel do SystemC, uma interface de programação de aplicações embarcadas e uma biblioteca de modelos para construção de nós sensores. O ambiente será apresentado usando a RSSF hipotética mostrada na Figura IV.1.

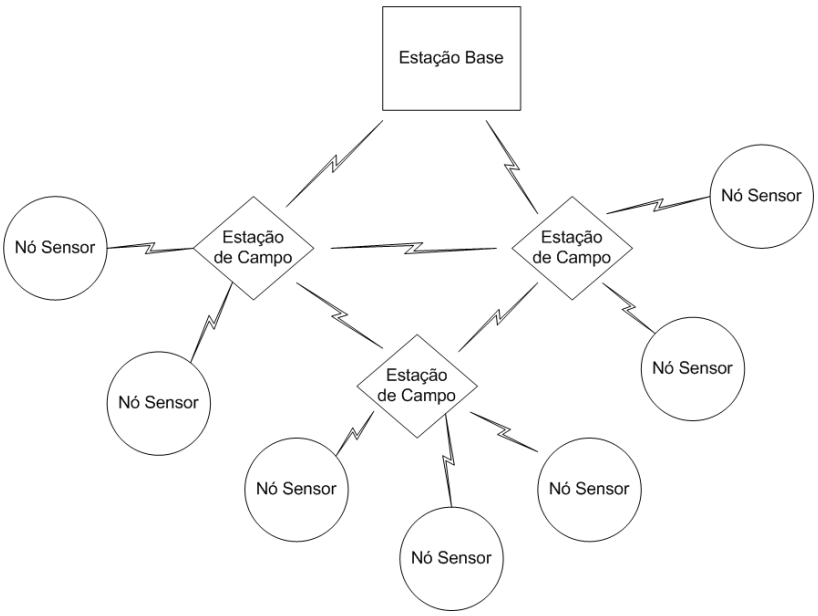

Figura IV.1: RSSF usada como exemplo.

A Figura IV.2 mostra a metodologia de projeto já proposta no Capítulo 4 e como um ambiente de projeto pode ser criado para implementar o fluxo apresentado. O Ambiente é composto de um *Simulador de Rede*, uma *Interface de Projeto de Protocolos*, um *Simulador de Hardware*, uma *Interface de Programação* e um *Simulador Misto*, todos construídos com interfaces gráficas para facilitar a configuração e as análises das simulações.

A etapa de Simulação de Rede com Nós Funcionais, mostrada na Figura IV.2 é tratada no Ambiente proposto pelo *Simulador de Rede*. Este simulador deve ser bastante semelhante a outros simuladores de rede como o NS-2. Ele deve, portanto, tratar da comunicação entre os nós da RSSF, desde o acesso ao meio, até protocolos de comunicação e formato dos pacotes de dados. Dessa

forma, é possível estimar o fluxo de dados na rede com precisão. Na RSSF hipotética da Figura IV.1, a topologia da rede e o posicionamento dos nós são dados e verifica-se a existência de três classes de nós: (i)Estação Base, (ii)Estações de Campo e (iii) nós sensores. Essa classificação é feita baseada apenas no comportamento dos nós. Por exemplo, a classe *nó sensor* é responsável, nesta rede hipotética, por adquirir os dados e transmiti-los para *Estações de Campo*. Caso existissem nós sensores com diferentes comportamentos, outra classe seria identificada.

A *simulação de rede* desta RSSF deve ocupar-se em estimar a qualidade dos canais de comunicação, quantidade de dados fluindo na rede, protocolos de roteamento, identificação de líder e qualquer outra característica demandada pela aplicação que possa ser tratada apenas no nível de rede sendo possível estimar a forma de organização e latência da rede.

A programação dos comportamentos dos nós pode ser feita com o uso da *Interface de Projeto de Protocolos*. Esta interface é composta por tantas linhas do tempo quanto forem as classes de nós e o comportamento de cada classe descrito por meio de ações no tempo, como transmitir pacotes, adquirir dados, esperar resposta, etc. A Figura IV.3 mostra a possível aparência desta interface para a RSSF tratada aqui.

Pode-se observar que os *nós sensores* são responsáveis por perceber os eventos de interesse, executar algum tipo de processamento sobre os dados e enviá-los para a *Estação de Campo* correspondente. Pode-se observar que o transceptor RF dos *nós sensores* deve ser mantido ligado apenas em torno de t_2 para envio. A *Estação de Campo* por sua vez deve manter o seu transceptor RF ligado para que os diversos nós possam enviar os dados entre t_2 e t_4. Um outro tratamento, como agregação de dados, é feito entre t_4 e t_5 e os dados são repassados para a *Estação Base*.

A *simulação de rede* deve trazer os primeiros requisitos para cada classe de nós, como taxa de comunicação necessária para cada ente da RSSF e ciclo de trabalho do transceptor RF além de requisitos de processamento necessários para a implementação dos protocolos desejados, como comunicação, roteamento, eleição de líderes, etc. Esses requisitos, juntamente com o comportamento de cada ente da rede, alimentam o *Simulador de Hardware* e a *Interface de Programação* com informações para que o hardware possa ser escolhido de acordo com a aplicação e o software a ser embarcado reflita o comportamento do nó.

Especificações de hardware são passadas para o *Simulador de Hardware* cujo objetivo é levantar um modelo arquitetural para cada classe de nó, com um processador, transceptor e interfaces de aquisição de dados adequadas à aplicação. Nesta fase do projeto, a disponibilidade de modelos de simulação para diversos componentes de hardware é essencial já que permite comparações e escolha do módulo mais adequado. É no *Simulador de Hardware* onde o processamento sobre os dados será mais bem detalhado. De posse de um modelo capaz de executar código embarcado, é possível simular o poder computacional necessário para a implementação de algoritmos como compactação de dados. A criação de códigos de aplicação seria feita na *Interface de Programação* e o uso de cross-compiladores específicos para cada processador permite a execução de software embarcado no hardware virtual, como apresentado no Capítulo 6.

Para o caso da RSSF usada como exemplo, seriam necessários três projetos de hardware, cada um dependente das condições vindas do *Simulador de Rede*. Esses hardwares para as diferentes

classes de nós podem ser totalmente diferentes: basta que atendam as especificações de hardware e que implementem, por meio do software embarcado, a camada de comunicação especificada anteriormente.

O *Simulador Misto* deve recuperar a topologia da rede assim como todo o comportamento de todos os nós criados no *Simulador de Rede*. Substitui-se alguns nós funcionais da *simulação de rede* por nós arquiteturais, criados no *Simulador de Hardware*, que são capazes de executar software embarcado, gerados na *Interface de Programação*. Essa substituição de modelos tem como objetivo verificar se o comportamento da rede como um todo é alterado além de permitir estimativas de consumo de energia para cada classe de nós. Como a RSSF simulada com sucesso no *Simulador de Rede* atende à aplicação, não deseja-se alterações nas características. Esse comportamento pode ser usado para fazer ajustes nos modelos de hardware e códigos de aplicação gerados.

Quando alcança-se o comportamento da rede desejado no *Simulador Misto*, pode-se afirmar que a comunicação entre os nós é adequada para a aplicação, assim como o hardware e o software embarcado podem tratar os dados satisfatóriamente dando fim ao projeto da RSSF: (i)existirá uma lista de componentes de hardware para cada nó, (ii)dados de latência, mapa de energia e qualidade de serviço estarão disponíveis, (iii)o software a ser embarcado estará em etapa avançada de projeto, (iv)todas as especificações, desde a comunicação até o tratamento de dados, foram satisfeitas conforme demandas da aplicação. Pode-se partir, portanto, para a fabricação dos nós e lançamento da RSSF.

A implementação dos simuladores do ambiente, tanto de rede quanto de hardware podem ser feitos em SystemC com auxílio de uma biblioteca como a SCNSL. O SystemC permite a simulação de eventos simultâneos e, portanto, simulações de rede e de hardware. O uso da biblioteca SCNSL implementa a infraestrutura de simulação necessária para a rede facilitando o projeto. Não há limitações de ferramentas para o desenvolvimento dos cross-compiladores e, dessa forma, pode-se visar alta performance.

O ambiente de projeto auxiliado por computador proposto aqui cobre a lacuna identificada durante o trabalho e descrita no Capítulo 4. O desenvolvimento desse ambiente de projeto busca concentrar todo o projeto de uma RSSF em um ambiente único e homogêneo e permite o tratamento de problemas específicos da área de RSSF por uma ferramenta desenvolvida especificamente com esse fim.

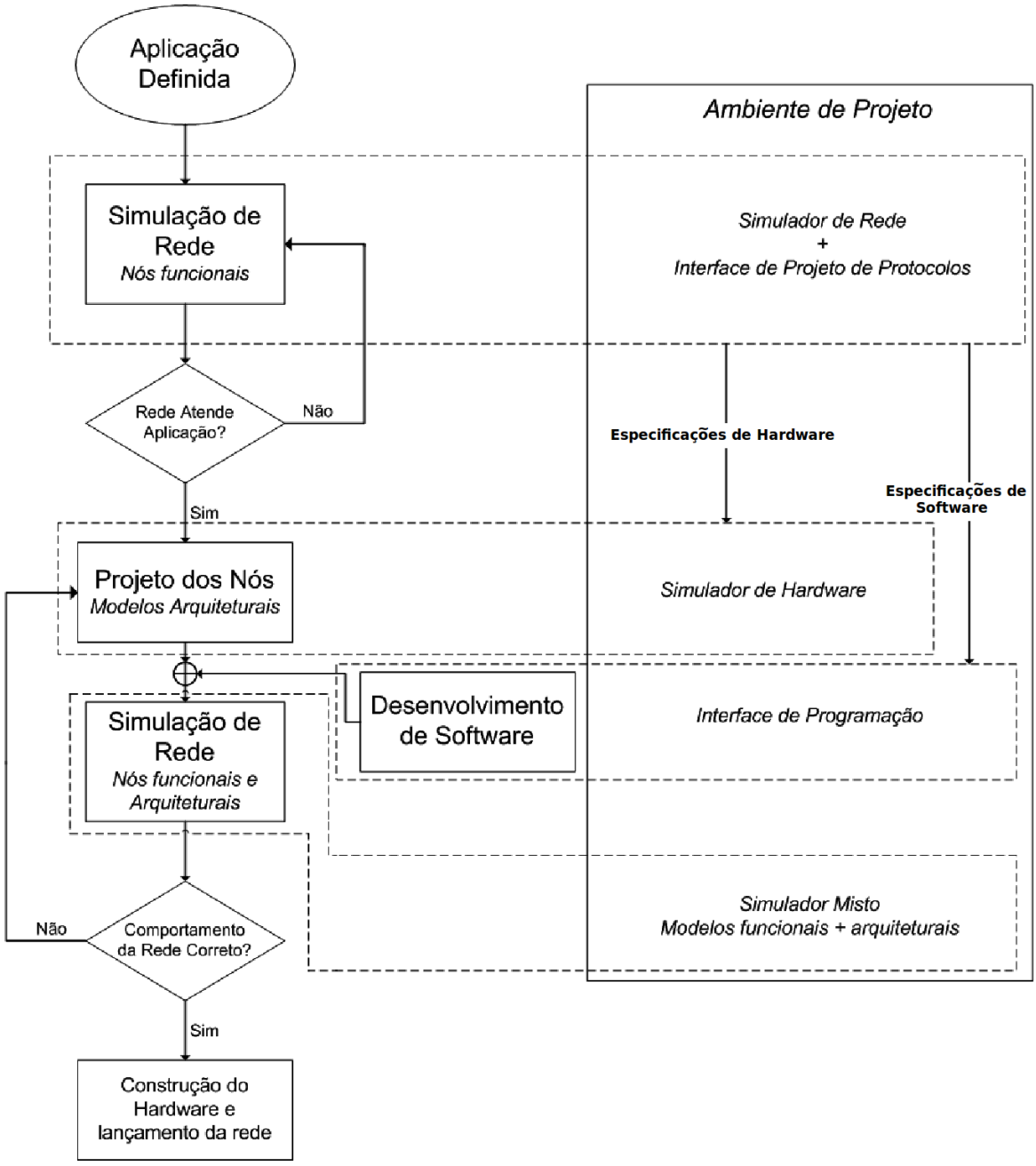

Figura IV.2: Fluxo de projeto implementado em Ambiente de Projeto.

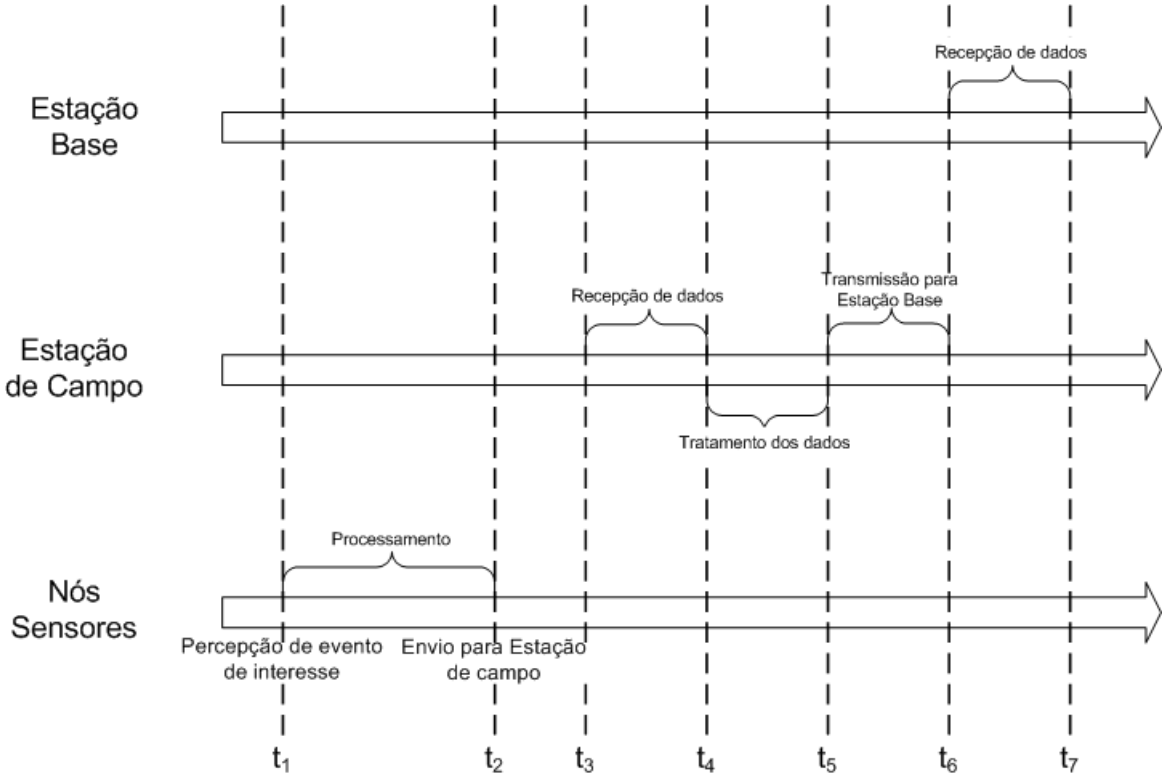

Figura IV.3: Linhas do tempo para cada classe de nó.

V. INFORMAÇÕES DO CC2420

Neste anexo estão disponíveis informações sobre o transceptor RF da Texas Instruments CC2420.

CC2420 Status: ACTIVE

Single-Chip 2.4 GHz IEEE 802.15.4 Compliant and ZigBee™ Ready RF Transceiver

Unitrode Products
from Texas Instruments

Datasheet

2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B) (PDF 883 KB)
20 Mar 2007
CC2420 Errata Notes (PDF 252 KB)
31 Jan 2006

View all technical documents (45)

Diagrams

Functional Diagram

Description

The CC2420 is a true single-chip 2.4 GHz IEEE 802.15.4 compliant RF transceiver designed for low-power and low-voltage wireless applications. CC2420 includes a digital direct sequence spread spectrum baseband modem providing a spreading gain of 9 dB and an effective data rate of 250 kbps.

The CC2420 is a low-cost, highly integrated solution for robust wireless communication in the 2.4 GHz unlicensed ISM band. It complies with worldwide regulations covered by ETSI EN 300 328 and EN 300 440 class 2 (Europe), FCC CFR47 Part 15 (US) and ARIB STD-T66 (Japan).

The CC2420 provides extensive hardware support for packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment, link quality indication and packet timing information.

View full Description in Datasheet

Features

- True single-chip 2.4 GHz IEEE 802.15.4 compliant RF transceiver with baseband modem and MAC support
- DSSS baseband modem with 2 MChips/s and 250 kbps effective data rate
- Suitable for both RFD and FFD operation
- Low current consumption (RX: 18.8 mA, TX: 17.4 mA)
- Low supply voltage (2.1 – 3.6 V) with integrated voltage regulator
- Low supply voltage (1.6 – 2.0 V) with external voltage regulator
- Programmable output power
- No external RF switch / filter needed
- I/Q low-IF receiver
- I/Q direct upconversion transmitter
- Very few external components
- 128(RX) + 128(TX) byte data buffering
- Digital RSSI / LQI support
- Hardware MAC encryption (AES-128)
- Battery monitor
- QLP-48 package, 7×7 mm
- Complies with ETSI EN 300 328, EN 300 440 class 2, FCC CFR-47 part 15 and ARIB STD-T66
- Powerful and flexible development tools available
- applications
  - 2.4 GHz IEEE 802.15.4 systems

View All Features in Datasheet

Parametrics

	CC2420
Frequency (Min) (MHz)	2400
Frequency (Max) (MHz)	2483.5
Device Type	Transceiver
Data Rate (Max) (kbps)	250
Operating Voltage (Min) (V)	2.1
Operating Voltage (Max) (V)	3.6
Programmable Output Power Ranging From (dBm)	-25 to 0

Samples

Inventory