Desenvolvimento de uma Rede de Sensores Sem Fio Utilizando *ZigBee* para Aplicações Diversas

Trabalho de Conclusão de Curso

Engenharia da Computação

Aluno: Leandro Honorato de Souza Silva

Orientador: Prof. Sergio Campello Oliveira

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

Leandro Honorato de Souza Silva

Desenvolvimento de uma Rede de Sensores Sem Fio Utilizando *ZigBee* para Aplicações Diversas

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, Junho de 2011.

Dedico aos meus pais e minha irmã.

Agradecimentos

Agradeço primeiramente a Deus, por todas as oportunidades a cada amanhecer. Agradeço também a toda minha família, que me deu o suporte necessário e imprescindível ao meu desenvolvimento pessoal e profissional, em especial, aos meus pais Antônio Honorato e Maria de Fátima, meu avô Joaquim Franklin, minha irmã Laize Andréa, meu padrinho Francisco Honorato, minha madrinha Rosa Maria e meu primo Cristiano André.

Agradeço aos professores que contribuíram com a minha formação, em especial, agradeço ao meu orientador, Sergio Campello, pela paciência despendida comigo, pela oportunidade de ter sido seu aluno de iniciação científica e pela sua enorme contribuição na minha vida acadêmica, profissional e também pessoal.

Agradeço aos meus amigos que me ajudaram muito, aos meus companheiros de iniciação científica, Ismael Mascarenhas e Diego Liberalquino, pela agradável convivência e por todos os amigos que conviveram comigo durante a graduação: André Hermenegildo (meu consultor de Word), Carlos Eduardo Buarque, Jefferson Amorim, Pedro Corrêa, Péricles Miranda, Marcos Oliveira, Matheus Torres e me desculpe se eu me esqueci de alguém tão importante quanto esses.

Agradeço aos meus companheiros da FITec que me ajudaram muito no desenvolvimento deste trabalho, em especial, Hilton Lima, Gabriel Liberalquino, Ricardo Lisboa, Felipe Lemos, Helmut Kemper e Gabor Sipkoi.

Por fim, agradeço a minha namorada/noiva/futura esposa, Rita Poliana, por ter me aturado durante o desenvolvimento deste trabalho e por sempre estar comigo nos momentos mais importantes.

Resumo

As redes de sensores sem fio são necessárias em diversas aplicações, facilitando segmentos da economia e diversas atividades do dia a dia. Essas aplicações vão desde monitoramento ambiental, monitoramento e controle de processos industriais, automação em transporte até aplicações em saúde. O *ZigBee* é um padrão que define um conjunto de protocolos de comunicação de baixa taxa de transmissão e curto alcance muito utilizado em redes de sensores sem fio. Buscando diminuir o tempo de desenvolvimento das aplicações que necessitem de monitoramento remoto, este trabalho desenvolveu um sistema completo, composto de plataformas de *hardware, firmware* e *software* para implantação de uma rede de sensores sem fio utilizando a tecnologia *ZigBee*. Do *hardware*, foi desenvolvido o esquema elétrico e *layout*. Nessas plataformas foram utilizados o microcontrolador PIC18F4550, o módulo *ZigBee Xbee-PRO ZNet 2.5 OEM* e a alimentação é realizada através de uma bateria que é recarregada por painel solar. O *firmware* foi desenvolvido tendo como alvo o microcontrolador referido, utilizando a linguagem C. O *software*, desenvolvido em Java, tem o objetivo de adquirir os dados coletados pelos sensores e armazenar em um banco de dados. Com a finalidade de obter os dados coletados pelos sensores também foi desenvolvido um protocolo de comunicação. O sistema obteve resultados satisfatórios, estando pronto para ser utilizado em aplicações que necessitem de monitoramento remoto, diminuindo o tempo de desenvolvimento, tendo em vista que elas precisam apenas selecionar os sensores e analisar os dados.

Abstract

Wireless sensor Networks are necessary in many applications, facilitating segments of the economy and various activities of everyday life. These applications range from environmental monitoring, monitoring and industrial process control, automation, transportation to health applications. ZigBee is a standard that defines a set of communication protocols for low data rate and short range widely used in wireless sensor networks. Aiming to reduce the development time for applications that require remote monitoring, this study developed a complete system, consisting of hardware platforms, firmware and software for deploying a wireless sensor network using ZigBee technology. The hardware platform, which schematic diagram and layout was completely developed in this work, uses a PIC18F4550 microcontroller, ZigBee Module XBee-PRO ZNet 2.5 OEM and power is held by a battery that is recharged by solar panel. The firmware was developed for the microcontroller above, using the language C. The software, developed in Java, has the purpose of acquiring the data collected by sensors and stored in a database. In order to obtain the data collected by the sensors was also developed a communication protocol. The system achieved satisfactory results, and is ready to be used in applications that require remote monitoring, reducing development time, considering that they only need to select the sensors and execute data analysis.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

ACK - *Acknowledgement*

ADC – *Analog-to-Digital Converter*

AODV - *Ad-Hoc On-demand Distance Vector Routing*

APS - *Application Support*

ARQ – *Automatic Repeat Request*

BD - Banco de Dados

BPSK - *Binary Phase Shift Keying*

CAD - *Computer-aided Design*

CI - Circuito Integrado

CRC - *Cyclic Redundancy Check*

CSMA/CA - *Carrier-Sense Multiple Access with Collision Avoidance*

DSSS - *Diret Sequence Spread Spectrum*

ED – *Event Detection*

FEC – *Forward Error Correction*

FFD – *Full Function DeviceI*

GPIO - *General Purpose Input/Output*

ISM – *Industrial, Scientific and Medical*

LDR - *Light Dependent Resistor*

LQI - *Link Quality Indication*

MAC – *Media Access Control*

MANET - *Mobile and Ad Hoc Network*

MEMS – *Micro-eletromechanical system technology*

MFR - MAC *Footer*

MHR - MAC *Header*

MIPS - *Million Instructions Per Second*

MPDU - MAC *Payload Data Unit*

MSDU - MAC *Service Data Unit*

NI - *Node Identifier*

O-QSKY - *Offset Quadrature Phase Shift Keying*

PAN ID - *Personal Area Netword Identification*

PCI - Placa de Circuito Impresso

PHR - *Physical Header*

PPDU - *Physical Protocol Data Unit*

PSDU - *Physical Service Data Unit*

RF - Radiofrequência

RFD – *Reduced Function Device*

SGBD - Sistema Gerenciador de Banco de Dados

SHR - *Synchonization Header*

SMD - *Surface Mounted Components*

SPE – *Spatial Process Estimation*

UART - *Universal Asynchronous Receiver/Transmitter*

WPAN - *Wireless Personal Area Network*

WSN - *Wireless Sensor Network*

ZDO - *ZigBee Device Objects*

Capítulo 1

Introdução

As redes de sensores sem fio (*Wireless Sensor Network* - WSN) têm sido tema de intensa pesquisa recentemente. As WSNs possuem o potencial de revolucionar vários segmentos da economia e atividades do dia a dia através de aplicações que vão desde monitoramento ambiental, controle de processos industriais, automação em transporte até aplicações em saúde. Trata-se de uma área que envolve processamento de sinal, redes e protocolos, sistemas embarcados, gerenciamento de informação e sistemas distribuídos [1].

O *ZigBee* é um padrão que define um conjunto de protocolos de comunicação para redes sem fio, que possuem baixa taxa de transmissão e curto alcance. Devido a essas características o *ZigBee* tem sido amplamente utilizado em várias aplicações que necessitam de redes de sensores sem fio.

Para essas aplicações, a finalidade é a monitoração remota dos sensores, sendo a tecnologia *ZigBee* o meio. A montagem e configuração da WSN utilizando a tecnologia *ZigBee*, apesar de toda a documentação e módulos que implementam a pilha de protocolos *ZigBee*, aumenta o tempo de desenvolvimento dessas aplicações. Este trabalho desenvolveu um sistema completo (plataformas de *hardware*, *software* e *firmware*) que permitem montar uma WSN utilizando a tecnologia *ZigBee*, com a finalidade de ser utilizado em uma ampla variedade de aplicações. Sendo assim, uma nova aplicação pode ser desenvolvida abstraindo a implementação da WSN, preocupando-se apenas com a seleção dos sensores e análise dos dados coletados.

1.1 Estrutura do Documento

Este trabalho está dividido em seis capítulos. O capítulo dois e três compõem a revisão bibliográfica, abordando as redes de sensores sem fio o e o padrão *ZigBee*, respectivamente. O capítulo 4 descreve o projeto e desenvolvimento da WSN proposta. O capítulo 5 apresenta os resultados obtidos e o sexto capítulo apresenta a conclusão e discute trabalhos futuros.

Capítulo 2

Redes de Sensores sem Fio

Os avanços nas áreas: de comunicação sem fio, de sistemas de tecnologia micro-eletromecânica (MEMS – *Micro-eletromechanical system technology*) e de sistemas embarcados têm proporcionado uma nova geração de redes de sensores [1]. Algumas aplicações possíveis são: monitoramento de pedestres ou tráfego veicular, monitoramento de animais silvestres e habitat para conservação ambiental, detecção de incêndios florestais [2] monitoramento de linhas de produção em fábricas entre outras [3].

Neste capítulo são abordados conceitos das redes de sensores sem fio a iniciar por uma definição de termos comuns na área (seção 2.1). Na seção 2.2 são descritos os fundamentos gerais das WSNs, incluindo restrições, vantagens e uma descrição da arquitetura em camadas. Logo em seguida, a seção 2.3 finaliza o capítulo apresentando uma classificação das aplicações das redes de sensores sem fio.

2.1 Definições de Termos

Para facilitar o entendimento dos tópicos seguintes, serão definidos alguns termos conforme encontrado em [3]:

- **Nó sensor:** este elemento é a unidade básica da rede de sensores. É normalmente composto de sensor(es), processador, moldem *wireless* (no caso das WSNs) e fonte de alimentação. Neste trabalho, este elemento também é chamado simplesmente de nó.
- **Serviços de um nó:** são as funcionalidades oferecidas pelo nó, como por exemplo, sincronização de tempo, localização do nó, requisição de leitura de sensor até funções de mobilidade.
- **Métricas de avaliação:** uma medida quantitativa que descreve o quão bem o sistema está executando suas funções. Exemplos de medidas são:

taxa de perda de pacotes, tempo de permanência da rede (i.e. o tempo que a rede permanece em funcionamento sem descontinuidade dos serviços prestados) e latência de processamento. Essas métricas podem ser utilizadas para comparar um sistema experimental com outros sistemas de *benchmark*.

- **Transmissão multi-saltos e transmissão em um único salto:** em uma rede sem fio, uma transmissão entre dois nós é em um único salto se não transitar por nenhum outro nó da rede, como na Figura 2.1 (a). Já uma transmissão multi-saltos é aquela que se utiliza de outros nós da rede para traçar uma rota entre o emissor e o receptor, como exemplificado na Figura 2.1 (b).

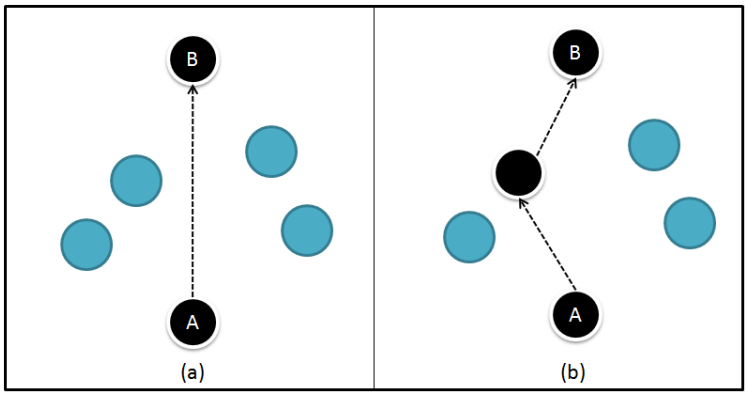


Figura 2.1 - Em (a) transmissão de A para B em um único salto e em (b) transmissão de A para B em dois saltos.

2.2 Fundamentos de WSNs

Uma WSN pode ser definida como um conjunto de nós capazes de comunicar a informação obtida da área monitorada (e.g. temperatura, umidade, vibração) através de um enlace sem fio. Essa informação é transmitida para um nó concentrador (também chamado de controlador, monitor ou em inglês *sink*) que pode localmente se utilizar das informações coletadas ou estar conectado a uma outra rede, em geral Internet ou Satélite, para oferecer acesso às informações coletadas para diversos usuários [3] [4].

A Figura 2.2 mostra a interação entre redes de sensores e a Internet. O elemento denominado *gateway* é utilizado para enviar comandos ou requisições

para um determinado nó da rede de sensores e receber as informações adquiridas em campo, assim como também dados de estado da rede. Elementos de armazenamento, opcionalmente, podem se encontrar junto ao *gateway* com a finalidade de guardar registro das leituras e operações realizadas. Os repositórios também servem como um intermediário entre os usuários e as redes de sensores, possibilitando persistência das informações coletadas. É interessante ressaltar que o *gateway* deve possuir duas interfaces de comunicação: uma para a rede de sensores sem fio e outra para a internet neste exemplo. Em algumas aplicações não existe a possibilidade de utilizar a Internet, sendo a comunicação via satélite a alternativa mais viável [5].

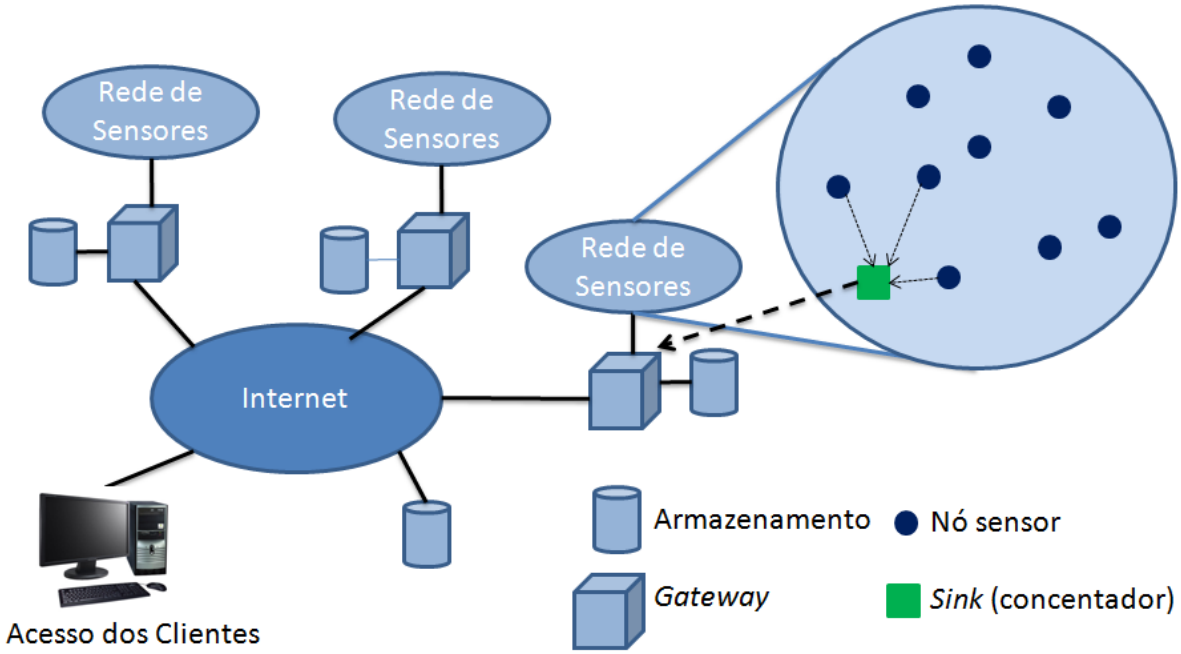


Figura 2.2 – Visão de como as WSNs podem interagir com outras redes [3][4].

Outro elemento presente na Figura 2.2 é o *sink* ou concentrador. Neste nó da rede a presença de dispositivos de sensoriamento é opcional. Ele tem o objetivo de transferir todas as informações coletadas para o *gateway* [4]. Na Figura 2.2 existe um problema de escalabilidade: aumentando o número de nós na WSN, o concentrador pode se tornar muito ineficiente em transferir as informações coletadas para o *gateway*. Além disso, por razões relacionadas ao acesso ao meio (MAC – *Media Access Control*) e aspectos de roteamento, a performance da rede não pode ser considerada independente do seu tamanho [4].

Uma boa solução para o problema de escalabilidade é utilizar vários concentradores dentro de uma WSN como exemplificado na Figura 2.3. Um número maior de concentradores diminuirá a probabilidade de isolamento de conjuntos de nós que não consigam transmitir suas informações coletadas devido a condições desfavoráveis de propagação. Entretanto, esta solução não é de trivial implementação, pois surge o problema de seleção do concentrador por parte do nó. Do ponto de vista do protocolo esta decisão será tomada principalmente com base nos seguintes critérios: atraso mínimo, vazão (*throughput*) máxima e número mínimo de saltos [4]. Desta forma, o uso de múltiplos concentradores melhora o desempenho da WSN em relação ao uso de um único concentrador, mas os protocolos para múltiplos concentradores são mais complexos e exigem que se considere esta característica desde o início do projeto.

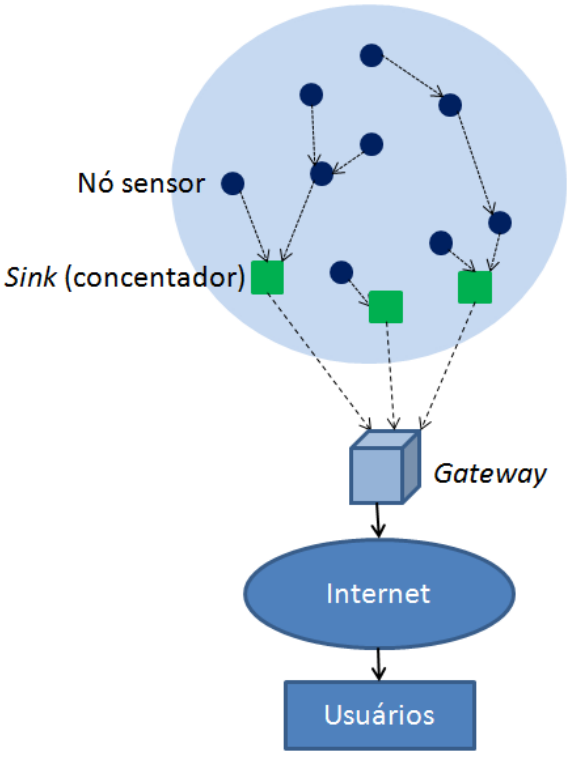


Figura 2.3 – WSN com múltiplos concentradores, extraída de [4].

2.2.1 Restrições das WSNs

As WSNs possuem problemas semelhantes às redes *ad hoc*, como por exemplo, acesso ao meio, criação e gerenciamento de rotas, mobilidade e segurança. Redes *ad hoc* são redes de propósito geral, nas quais não existe um nó central para onde todas as informações devem convergir (como um ponto de

acesso) e a topologia é dinâmica. Embora diversos protocolos tenham sido propostos para as redes sem fio *ad hoc*, estes algoritmos não se adéquam perfeitamente às necessidades e restrições das redes de sensores sem fio. Conforme encontrado em [1] segue uma lista das principais diferenças entre WSNs e redes *ad hoc*:

- O número de nós em uma WSN pode ser muito maior do que a quantidade de nós em uma rede *ad hoc* e dispostos de forma mais densa (maior quantidade de nós por unidade de área);
- Os nós da WSN são mais propensos a falhas, já que estão mais sujeitos a falta de alimentação e fatores ambientais;
- Os nós de uma WSN são limitados em relação ao consumo de energia, capacidade de processamento computacional e memória, enquanto que os nós das redes ad hoc normalmente são mais robustos (e.g. notebooks) [4].

As principais características das redes de sensores sem fio são: escalabilidade (em relação ao número de nós na rede), auto-organização (o que inclui a capacidade de se recuperar de perdas de nós), eficiência energética, conectividade suficiente entre os nós (objetivando que todos os nós da rede sejam capazes de transmitir suas informações para o concentrador), baixa complexidade, baixo custo e pequeno tamanho dos nós. Uma definição de um protocolo que atenda a todas essas necessidades não é uma tarefa simples, sendo assim, pesquisas ainda são necessárias [4]. A seguir serão abordados alguns fatores que devem ser levados em consideração durante o projeto de uma WSN.

- **Tolerância a falhas**

Um nó de uma rede de sensores sem fio pode apresentar alguma falha, não ter alimentação suficiente disponível ou ser danificado fisicamente devido a alguma interferência do ambiente [1]. A falha de nós isolados da rede não deve afetar as tarefas executadas pelos demais elementos da WSN. A tolerância a falhas é a capacidade de manter as funcionalidades da rede de sensores sem interrupções devidas a falhas nos nós da rede. A tolerância a falhas $R_k(t)$ de um sensor, a qual é

modelada como uma distribuição de Poisson para capturar a probabilidade de um sensor não apresentar falha no intervalo (0,t), é apresentada na equação abaixo [6]:

$$R_k(t) = \exp(-\lambda_k t), \quad (2.1)$$

onde λ_k é a taxa de falha (idealmente constante) e *t* é o período. Isso implica que a probabilidade de um nó apresentar falha cresce exponencialmente ao longo do tempo.

- **Escalabilidade**

Conforme já mencionado, algumas aplicações podem exigir uma grande quantidade de nós. Dependendo da aplicação, a quantidade de nós pode chegar a centenas ou milhares [1]. A densidade da rede pode ser calculada por:

$$\mu(R) = (N \,.\pi\,.\, R^2)/A, \quad (2.2)$$

onde *N* é a quantidade de nós dispersos na região de área *A* e *R* é o raio de alcance dos nós. Na Figura 2.4 pode-se visualizar esses parâmetros numa área com quatro nós ($N = 4$). O termo $\pi\,.\,R^2$, encontrado no numerador, indica a área de cobertura de um nó, que quando multiplicada por *N* representa a soma da área de cobertura dos *N* nós. Essa soma da área de cobertura pode ser maior que a área *A* (já que a soma não despreza as interseções entre a cobertura dos nós). Quanto maior for $\mu(R)$, maior a densidade da rede (existem mais interseções entre nós) e maiores são as preocupações com tarefas como acesso ao meio e gerenciamento de múltiplos concentradores.

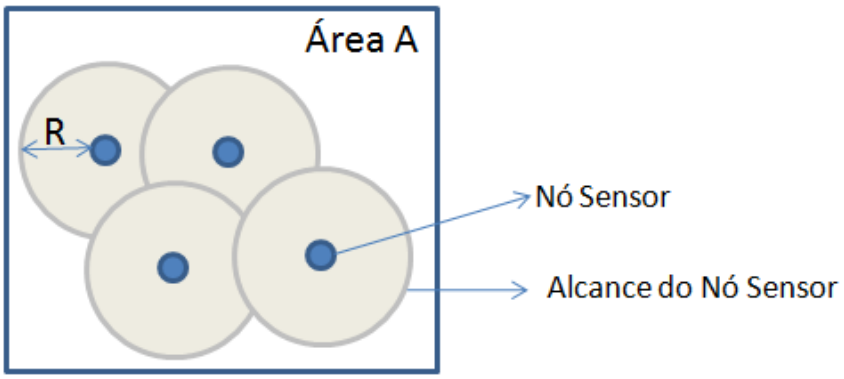


Figura 2.4 – Exemplo de densidade de uma WSN

- **Restrições de Hardware**

Os nós das WSNs são constituídos basicamente de quatro componentes: uma unidade de sensoriamento, uma unidade de processamento, um transmissor e uma unidade de alimentação, interligados como mostrado na Figura 2.5. A unidade de sensoriamento obtém uma medida analógica do fenômeno mensurado, a qual é convertida para informação digital pelo conversor analógico digital (ADC – A*nalog-to-Digital Converter*). O processador, normalmente acoplado a uma memória, tem a função de controlar a execução das tarefas de sensoriamento e processos colaborativos entre nós sensores [1]. A unidade de transmissão é responsável por conectar o nó à rede sem fio. A unidade de alimentação deve fornecer energia para o nó e pode se utilizar de elementos de geração de energia, como por exemplo, painéis solares.

Ainda existem dois elementos opcionais: o sistema de localização e o elemento de movimentação. Algumas técnicas de roteamento em redes de sensores e tarefas de sensoriamento exigem a informação de localização do nó com alta precisão [1]. Já o elemento de movimentação é necessário quando é preciso movimentar os nós em um ambiente para a realização de tarefas.

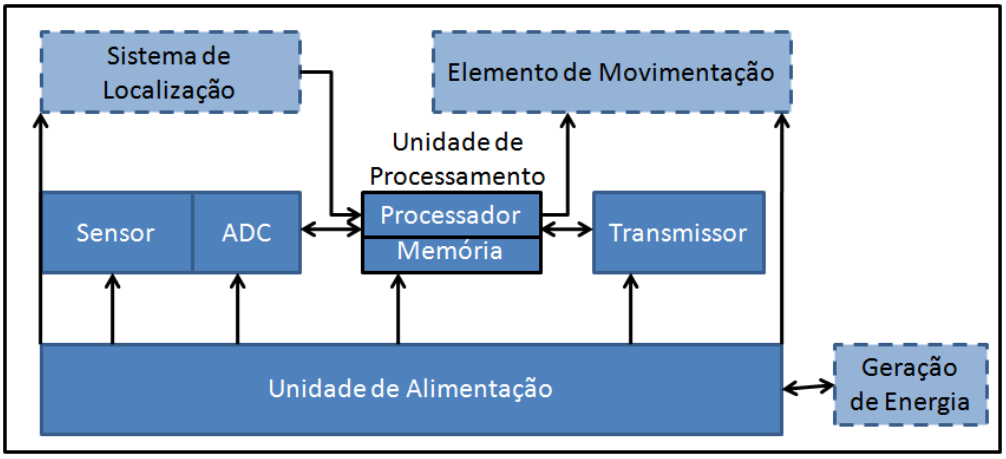


Figura 2.5 – Diagrama de blocos do nó sensor de uma WSN, extraída de [1].

- **Custo de Produção**

Tendo em vista que aplicações podem requerer uma grande quantidade de nós, é de extrema importância que o custo de cada nó seja o mais baixo possível a fim de minimizar o custo total da rede. Trata-se de uma restrição importante, já que,

caso o custo da rede seja mais elevado do que a implantação de sensores tradicionais, a rede não encontra seu custo justificado [1].

- **Manutenção da Topologia da Rede de Sensores**

Os nós de uma rede podem ser dispostos de diversas maneiras com densidades maiores do que 20 nós/m³ [1]. A implantação de tantos sensores exige uma preocupação com a manutenção da topologia. Os problemas relacionados à topologia são divididos em três: pré-implantação, pós-implantação e adição de novos nós.

Na fase de pré-implantação é levado em consideração como os nós serão inicialmente implantados, ou seja, se os nós serão colocados um a um (por um humano ou robô) ou simplesmente jogados em massa (jogados de um avião, por exemplo). Na fase de pós-implantação as preocupações são com as mudanças na topologia causadas por mudanças na posição dos nós (provocadas por mudanças no ambiente), falta de energia ou mau funcionamento de alguns nós. Já a fase de adição de novos nós prevê a possibilidade de inserção e substituição de nós com defeito.

- **Ambiente**

Os nós de uma WSN devem ser dispostos muito perto, ou dentro do fenômeno que se deseja observar. Os nós podem estar funcionando em ambientes hostis, como: áreas geográficas remotas, no interior de grandes máquinas, no fundo do oceano e ambientes contaminados quimicamente ou biologicamente, por exemplo.

- **Consumo de Energia**

A maior restrição das WSNs está associada à eficiência energética. Tipicamente os nós são alimentados através de baterias, as quais devem ser trocadas ou recarregadas (e.g. usando painéis solares) periodicamente. Em alguns casos não é possível realizar nem a substituição nem a recarga das baterias, como por exemplo, WSNs para monitorar movimentos glaciais ou campos de batalha. Nestes casos, deve-se maximizar a vida útil do sensor [4] [7].

Durante a transmissão ocorre um pico de consumo de energia, drenando mais carga da bateria do que quando apenas o processador está ativo. Protocolos que exijam muitas transmissões de dados não úteis como mensagens de controle devem ser evitados.

Também não é desejado que o nó sensor permaneça durante muito tempo no estado de recepção. Mesmo consumindo menos energia que nas transmissões ainda assim este é um estado de alto consumo [3] e a presença prolongada nesse estado, o que é chamado de *overhearing,* deve ser evitada.

2.2.2 Vantagens das WSNs

A maior vantagem das WSNs está relacionada à robustez e escalabilidade. Um sistema de sensoriamento descentralizado é inerentemente mais robusto do que um único nó sensor que concentra toda a comunicação, já que o sistema descentralizado possui redundâncias que o torna mais tolerante a falhas. Além disso, os algoritmos descentralizados são mais escaláveis para aplicações práticas, que exigem inserção de novos nós e podem ser a única solução que atenda a aplicações que necessitam de uma grande quantidade de nós [3].

As WSNs também são mais flexíveis que as redes de sensores que utilizam uma estrutura cabeada, já que é mais fácil inserir novos nós e não existe um custo tão alto para remover um determinado nó ou re-planejar a topologia da rede.

- **Eficiência Energética**

Devido às características únicas de atenuação dos sinais de radiofrequência (RF), uma rede que se utiliza deste canal e faz uso de múltiplos saltos proporciona uma economia de energia maior do que se fosse utilizado uma rede que toda comunicação é realizada em um único salto. Para comprovar esta afirmação, considere uma rede de *N-saltos*. Assumindo que a distância de transmissão global é *N.r,* onde r é a distância de transmissão em um único salto, a potência mínima que deve ser recebida em um nó para que seja possível entender a informação sem erro é $P_{receive}$, e a potência de transmissão de um nó é P_{send}. Então, o modelo RF de atenuação próximo à terra é dado por [3].

$$P_{receive} \propto \frac{P_{send}}{r^{\alpha}}. \quad (2.3)$$

Onde r é a distância da transmissão e α é o expoente de atenuação RF. Devido a interferências de várias rotas para atingir o receptor e outros fatores, o valor de α é tipicamente entre 2 e 5 [3]. De forma equivalente,

$$P_{send} \propto r^{\alpha}.P_{receive}. \quad (2.4)$$

Desta maneira, a vantagem em relação ao consumo de energia de uma rede de N-saltos sobre uma rede de um único salto é:

$$n_{rf} = \frac{P_{send(Nr)}}{N.P_{send(r)}} = \frac{(Nr)^{\alpha}P_{receive}}{N.r^{\alpha}.P_{receive}} = N^{\alpha-1}. \quad (2.5)$$

Esta vantagem pode ser visualizada na Figura 2.6, onde claramente pode ser observado que é mais eficiente energeticamente realizar a transmissão em vários saltos.

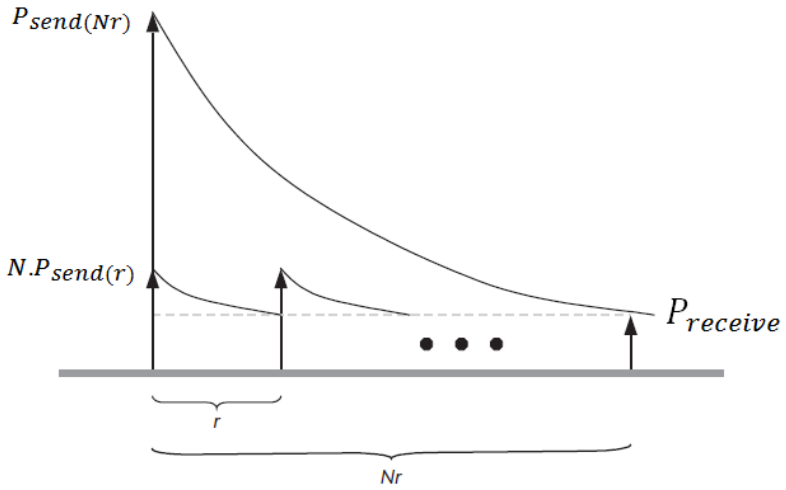


Figura 2.6 – Relação entre a potência de transmissão em uma rede N-saltos e em uma rede de único salto, extraída de [3].

Quanto maior o valor de N (quantidade de saltos) maior é a economia de energia, no entanto um N muito grande significa um maior custo com transmissores RF, além de também haver consumo de energia do próprio módulo e não apenas da transmissão como considerado em (2.5). Sendo assim, um projeto deve buscar uma boa eficiência energética com um custo acessível. Latência e robustez também

podem influenciar contra a decisão de ter um grande número de nós com um baixo alcance de transmissão [3].

2.2.3 Camadas das WSNs

A pilha de protocolos de um nó sensor é exibida na Figura 2.7. Nela além das camadas podem ser observados os planos de gerenciamento. Estes planos são interesses transversais, presentes em todas as camadas. Os planos de gerenciamento buscam ajudar a coordenação dos serviços dos nós e a diminuição do consumo de energia. A seguir, discutiremos brevemente algumas das camadas da pilha de protocolos das WSNs. Uma discussão mais aprofundada pode ser encontrada em [1].

A camada física possui a responsabilidade de realizar a seleção da freqüência de transmissão, a detecção de sinal, a geração de portadora de frequência e a modulação dos dados. Para a frequência, a faixa reservada para aplicações industriais, científicas e médicas (ISM – *Industrial, Scientific and Medical*) tem sido amplamente sugerida [1]. A faixa ISM é composta de por três outras faixas: 900 MHz, 2,4 GHz e 5,8 GHz. A faixa de 900 MHz não é utilizada na Europa, por ser utilizada em outra aplicação nessa região. A faixa de 2,4 GHz é a mais utilizada, sendo compartilhada por diversas aplicações como *WiFi*, *Bluetooth* e *ZigBee* (IEEE 802.15.4).

A camada de enlace é responsável principalmente pela detecção de quadros de dados, acesso ao meio e correção de erros. Ela permite a realização de comunicações ponto-a-ponto e ponto-a-multipontos na rede de sensores.

O controle de acesso ao meio (MAC) deve ter dois objetivos. O primeiro é a criação de uma infraestrura de rede, ou seja, estabelecer links de comunicação para transferências de dados entre os nós. O segundo objetivo é a divisão dos recursos de forma justa e eficiente entre os nós. Os modelos de MAC utilizados por outras redes sem fio são inadequados para as redes de sensores sem fio. Por exemplo, nas redes de celulares e MANETs (*Mobile and Ad Hoc Network*), o objetivo principal está relacionado com a qualidade do serviço e não existe uma restrição rígida em relação à economia de energia. MACs baseados em demanda podem ser

inadequados às WSNs, devido ao *overhead* de mensagens e atrasos no enlace. Alguns protocolos de MAC utilizados pelas WSNs podem ser encontrados em [1].

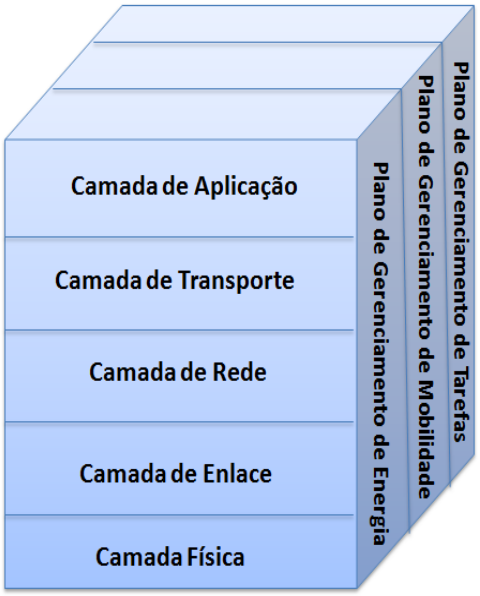


Figura 2.7 – Pilha de protocolos das redes de sensores sem fio.

Outra função importante da camada de enlace é o controle de erro. Existem duas abordagens principais: FEC (*Forward Error Correction*) e ARQ (*Automatic Repeat Request*). A desvantagem da ARQ é o custo com a retransmissão das informações. Já a abordagem FEC possui uma decodificação mais complexa das informações, já que a capacidade de correção de erro deve ser realizada no próprio nó destino.

Nas redes multi-salto a camada de rede é responsável pela localização da(s) rota(s) entre o nó emissor e o receptor. As rotas devem levar em consideração a potência disponível nos nós ou a energia necessária para transmissão nos enlaces ao longo da rota. Uma rota entre nós A e B pode ser escolhida por diversas abordagens, por exemplo: máximo de potência disponível nos nós, menor energia de transmissão, menor número de saltos, entre outras. Estas abordagens encontram-se mais detalhadas em [1].

Já a camada de transporte é necessária para que a WSN possa ser acessada por outras redes externas (e.g. a Internet). Por fim, a camada de aplicação está

diretamente relacionada à execução dos serviços da WSN, no que diz respeito ao gerenciamento das tarefas de sensoriamento.

2.3 Classificação das Aplicações

As diversas aplicações das WSNs podem ser classificadas de acordo com a informação que devem ser coletadas pela rede. Quase todas as aplicações podem ser classificadas dentro de uma ou duas categorias: Detecção de Evento (ED – *Event Detection*) ou Estimativa de Processo Espacial (SPE – *Spatial Process Estimation*).

As aplicações de detecção de eventos são aquelas cujo objetivo da coleta de informações pelos sensores é gerar uma saída para informar se um determinado evento está acontecendo, ou prestes a acontecer. Exemplos de aplicações de ED são detecção de incêndios, tremores de terra e vazamentos de gazes em um ambiente industrial. O processamento de sinal dentro dos nós da rede é bem simples, já que basicamente se trata de comparar um valor medido com um determinado limiar para ativar ou não o sinal de detecção. Nestas aplicações, a rede deve possuir densidade suficiente para realizar a detecção com o mínimo de erro.

Já as aplicações de estimativa de processo espacial, almejam estimar um fenômeno físico, por exemplo: pressão atmosférica em uma região, variações de temperatura em uma região vulcânica, entre outras. Estas aplicações podem ser modeladas como um processo aleatório bi-dimensional, normalmente não estacionário [4]. Em aplicações de SPE o desafio é obter estimativas do comportamento completo do fenômeno observado baseando-se apenas nas amostras coletadas pelos sensores, os quais são tipicamente dispostos em localização desconhecida. O processamento dos sinais nesta classe de aplicações é mais elaborado em relação às aplicações de ED.

Algumas aplicações podem se enquadrar dentro das duas classes descritas anteriormente. Como por exemplo, uma aplicação cuja meta é economia de energia em uma casa. Neste caso, a rede de sensores sem fio é disposta de forma espalhada em uma construção para gerenciar de maneira eficiente o consumo de energia realizado por todos os aparelhos elétricos. Para cumprir este objetivo, os nós

devem realizar medidas do consumo de cada aparelho elétrico da região monitorada; por outro lado, o sistema também pode detectar o evento de presença de pessoas para ligar/desligar alguns aparelhos elétricos.

Existem aplicações as quais requerem, com alta prioridade, que os nós sensores sejam pequenos, descartáveis, de baixo custo e possuam alta eficiência energética. Uma aplicação onde esses requisitos são muito importantes é o monitoramento de incêndios florestais [2], na qual os nós sensores são jogados de um helicóptero na região florestal a ser monitorada e precisam se auto-organizar para transmitir as informações coletadas.

Existem outros cenários onde os requisitos mais importantes são confiabilidade, possibilidade de monitoramento em tempo real, tempo de vida útil dos sensores e autonomia para formação da rede. Uma aplicação que se enquadra segundo cenário é o monitoramento de um processo industrial, ou monitoramento de tráfego [8], na qual não existe uma restrição rígida com relação ao tamanho do nó, eles não são descartáveis e muitas vezes monitoram equipamentos de alto valor, então, é tolerável ter um custo maior do que do que no primeiro cenário. A WSN desenvolvida nesse trabalho atende às aplicações deste último cenário.

Capítulo 3

O Padrão *ZigBee*

ZigBee é um padrão que define um conjunto de protocolos de comunicação de taxa de transmissão máxima de 250 Kbps e curto alcance para redes sem fio. O *ZigBee* padrão opera na frequência de 2,4 GHz e seu nome *ZigBee* vem do comportamento das abelhas que voam em *zig zag* trocando informações sobre distância, direção e localização de onde encontrar alimentos[1].

O padrão *ZigBee* foi desenvolvido pelo consórcio denominado *ZigBee Alliance* [9]. O propósito deste consórcio era descrever os protocolos de maneira que os fabricantes possam produzir dispositivos compatíveis entre si. Ou seja, produtos de fabricantes diferentes que implementem o padrão *ZigBee* são compatíveis.

O *ZigBee* é implementado sobre o padrão IEEE 802.15.4, o qual define as camadas física e de enlace. Uma visão mais detalhada da tecnologia IEEE 802.15.4 é dada na seção 3.1. Na seção 3.2 há uma descrição das camadas implementadas pelo *ZigBee*. Na seção 3.3 são discutidas possíveis topologias de rede *ZigBee* e na seção 3.4 são mostrados alguns módulos que implementam este padrão.

3.1 O IEEE 802.15.4

O IEEE 802.15.4 é uma tecnologia de comunicação sem fio de curto alcance, que tem o propósito de servir a aplicações que não possuem requisitos de alta taxa de transmissão e baixa latência [4].

As principais funcionalidades desta tecnologia são: baixa complexidade, baixo custo e baixo consumo de energia. Devido a estas características, a maior aplicação desta tecnologia são as WSNs [4]. A seguir será abordada a camada física (PHY) e de enlace (MAC) propostas pelo IEEE 802.15.4.

[1] Duas outras versões foram encontradas: uma informa que *ZigBee* é um acrônimo para: "*Zonal Intercommunication Global-standard, where Battery life was long, which was Economical to deploy, and wich exhibited Efficient use of resources*" [11]; a segunda versão diz que o nome veio de uma lenda Norueguesa [11]. Entretanto, a versão mais coerente parece ser a das abelhas.

3.1.1 Camada Física (PHY) IEEE 802.15.4

A camada física opera em três diferentes faixas de freqüência não-licenciadas de acordo com a localização geográfica. O padrão especifica 27 canais *half-duplex* para as três bandas, conforme pode ser visualizado na Tabela 3.1 [4] [10].

Tabela 3.1 – Alocação de canais e bandas nos diferentes locais.

Banda	Frequência (MHz)	Local Disponível	Largura de Banda	Quantidade de Canais
2,4 GHz (ISM)	2400 – 2483,5	Todo o mundo	250 Kbps	16
915 MHz (ISM)	902 – 928	EUA	40 Kbps	10
868 MHz	868 – 868,6	Europa	20 Kbps	1

A modulação, método de propagação e a distância entre os canais variam de acordo com a banda utilizada. A Tabela 3.2 resume estas informações para cada uma das bandas. As faixas de 868 MHz e 915 MHz implementam uma modulação BPSK (*Binary Phase Shift Keying*) e utilizam o método de propagação (*spreading method*) DSSS (*Diret Sequence Spread Spectrum*) [4]. Já a banda de 2,4 GHz utiliza modulação O-QSKY (*Offset Quadrature Phase Shift Keying*).

Tabela 3.2 – Alocação de canais e bandas nos diferentes locais.

Banda	Distância entre canais	Modulação	Método de Propagação
2,4 GHz (ISM)	5 MHz	O-QPSK	16-array orthogonal
915 MHz (ISM	2 MHz	BPSK	DSSS Binário
868 MHz	-	BPSK	DSSS Binário

Na Tabela 3.3 pode ser observada a potência mínima para recepção e o alcance máximo em cada uma das bandas. O alcance máximo é obtido em condições ideais, ou seja, sem considerar efeitos de reflexão, difração e espalhamento [4].

Tabela 3.3 – Potências Mínimas de recebimento e alcance máximo.

Banda	Sensibilidade de Recepção	Alcance máximo
2,4 GHz (ISM)	-85 dBm	200 m
915 MHz (ISM	-92 dBm	1 Km
868 MHz	-92 dBm	1 Km

Ainda existem especificações opcionais para as bandas de 815/915 MHz. Um resumo destas especificações pode ser encontrado em [4], assim como também maiores detalhes sobre a camada física do IEEE 802.15.4 em [4], [10], [12] e [11].

A preocupação com o consumo de energia também está presente na camada PHY. Os dispositivos que implementam o padrão IEEE 802.15.4 devem permanecer no estado ativo por curtos períodos de tempo (o padrão possibilita que dispositivos fiquem no estado de *sleep* por 99% do tempo [4]). Além disso, a taxa de transmissão instantânea é muito próxima da taxa máxima proporcionada pelo padrão.

De acordo com o padrão IEEE 802.15.4, a transmissão é organizada em quadros (*frames*) que diferem entre si de acordo com o propósito. A estrutura de um quadro geral é mostrada na Figura 3.1. Existem quatro tipos de quadros, que compõem o PPDU (*Physical Protocol Data Unit*): quadro de *beacon*, quadro de dados, quadro de ACK (*Acknowledgement*) e quadro de comando MAC. Todos os quadros são estruturados com um *Synchonization Header* (SHR), um *Physical Header* (PHR) uma *Physical Service Data Unit* (PSDU), este último é composto por uma MAC *Payload Data Unit* (MPDU), que por sua vez, é construído com um MAC *Header* (MHR), um MAC *Footer* (MFR) e um MAC *Service Data Unit* (MSDU). Ainda para assegurar que uma mensagem foi recebida corretamente é utilizado CRC (*Cyclic Redundancy Check*. O campo MSDU é preenchido de acordo com o tipo do quadro, exceto no quadro de ACK, onde este campo é ausente. Uma estrutura detalhada do MSDU para cada quadro pode ser encontrada em [4], [11] e [12].

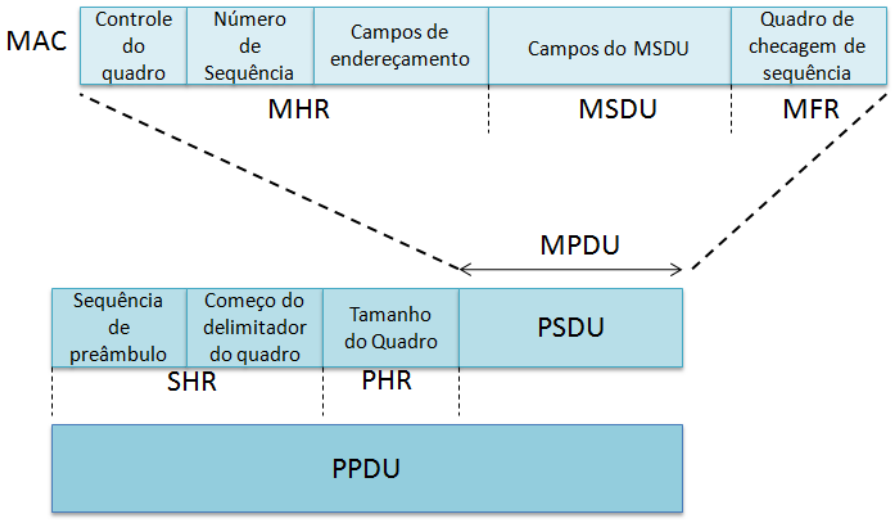


Figura 3.1 – Estrutura geral de um quadro IEEE 802.15.4 [4].

3.1.2 Topologias de Rede e Modos de Operação IEEE 802.15.4

Antes de descrever o MAC no IEEE 802.15.4 é necessário uma discussão das possíveis topologias e modos de operação.

A tecnologia IEEE 802.15.4 descreve dois tipos de dispositivos: dispositivos de função completa (FFD – *Full Function Device*) e dispositivos de função reduzida (RFD – *Reduced Function Device*). O dispositivo FFD possui o conjunto completo dos serviços MAC e pode operar como o coordenador de uma rede, ou como um simples dispositivo da rede. Já os dispositivos RFD possuem uma parte dos serviços MAC e operam apenas como um dispositivo da rede.

Existem duas topologias permitidas no padrão, que não são completamente descritas, já que a definição das camadas mais altas (i.e. rede e transporte) está fora do escopo do IEEE 802.15.4. As topologias permitidas são: estrela e *peer-to-peer*. Na topologia estrela, o nó coordenador (um FFD que inicia a rede) possui comunicação direta para os demais nós e estes apenas podem ter comunicação com o coordenador. Já na topologia *peer-to-peer* é permitido que os nós que não são o coordenador da rede se comuniquem entre si. A Figura 3.2 ilustra as duas topologias. A topologia estrela é preferível quando a área de cobertura da WSN é pequena e existe um requisito de baixa latência. A topologia *peer-to-peer* permite a formação de redes mais complexas, com transmissões multi-saltos, sendo mais adequadas para cobrir grandes áreas.

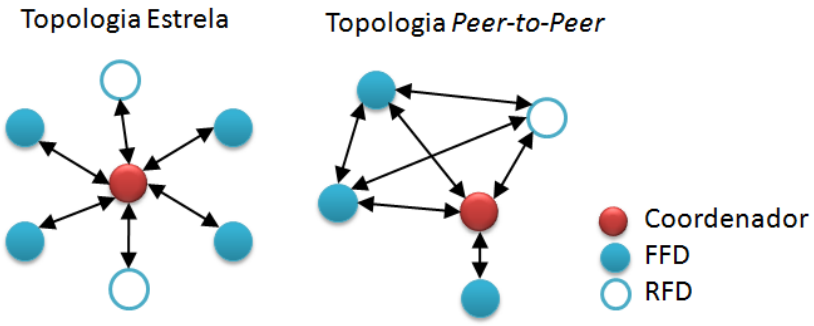


Figura 3.2 – Topologias permitidas no IEEE 802.15.4.

Toda a rede, independente da topologia, possui um PAN ID (*Personal Area Network Identification*), que é um endereço de 16 bits que a identifica. Cada dispositivo também possui um endereço de 16 bits que o identifica dentro de uma rede IEEE802.15.4.

3.1.3 Camada de Enlace (MAC) IEEE 802.15.4

As funções executadas pela camada de enlace na tecnologia IEEE 802.15.4 são: associação e desassocição, controle de segurança, funções para redes na topologia estrela (gerenciamento de *time slot* para transmissão, geração de ACKs) e suporte à aplicação para as duas topologias suportadas.

A camada MAC usa um protocolo baseado no algoritmo CSMA/CA (*Carrier-Sense Multiple Access with Collision Avoidance*), o qual requer que se escute o canal antes de transmitir para que a probabilidade de colisões durante a transmissão seja diminuída [4]. Existem dois modos de operação, que correspondem a dois mecanismos diferentes de acesso ao meio: um modo chamado *beacon-enabled* e outro *non beacon-enable*. No modo *non beacon-enable* o nó procura um canal para realizar a transmissão, enquanto que no modo *beacon-enabled*, existe uma estrutura chamada *superframe* que provê uma espécie de sincronização para transmissão evitando colisões, ou seja, para que um nó tenha acesso ao canal, ele deve receber um quadro de *beacon*. Uma descrição detalhada do acesso ao meio no IEEE 802.15.4, incluindo fluxogramas dos algoritmos CSMA/CA *beacon-enabled* e *non beacon-enable* encontra-se disponível em [4].

3.1.4 Procedimento de Formação de uma rede IEEE 802.15.4

Na formação de uma rede, inicialmente o coordenador, para estabelecer uma WPAN (W*ireless Personal Area Network*), precisa achar um canal que seja livre de interferências, já que outras WPAN podem coexistir na mesma região. A seleção do canal é realizada através de uma varredura para detectar o nível de energia nos canais. Quando o coordenador selecionar o canal, estabelecendo assim a WPAN, outros dispositivos poderão se associar à rede.

A operação de associação à rede pode ser resumida da seguinte forma: o nó procura por WPANs disponíveis, seleciona uma WPAN para se associar e inicia o processo de associação com um dispositivo FFD que já faz parte da rede [13].

3.2 As camadas Implementadas pelo *ZigBee*

O *ZigBee* é um padrão que implementa camadas acima do IEEE 802.15.4. A arquitetura da pilha de protocolos *ZigBee* pode ser vista na Figura 3.3.

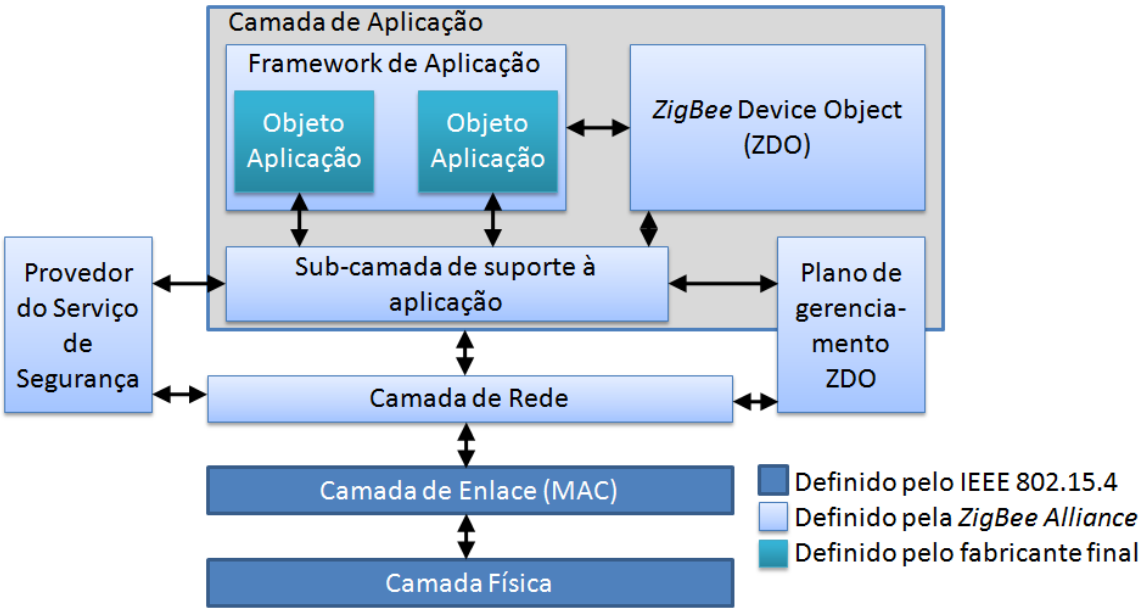


Figura 3.3 – Pilha de protocolos *ZigBee* [4].

O *ZigBee* define três tipos de dispositivos: coordenador, roteador e dispositivo final. O coordenador é o nó responsável por iniciar a WPAN, por aceitar os pedidos de associação à rede e pode dar assistência ao roteamento de dados numa rede *mesh*; existe apenas um coordenador por rede *ZigBee,* assim como nas redes IEEE 802.15.4. O nó roteador deve se associar a uma WPAN existente e pode permitir que outros dispositivos se associem à rede, enviar e receber dados e rotear dados através da rede. Já os dispositivos finais não podem permitir associações de novos dispositivos, não fazem roteamento de dados e passam a maior parte do tempo no estado *sleep*.

Uma informação importante é que *ZigBee* apenas funciona sobre a banda de 2,4 GHz do IEEE 802.15.4, já que as outras bandas não oferecem taxa de transmissão suficiente [11].

O padrão *ZigBee* suporta os seguintes serviços de segurança (são opcionais): criptografia dos dados, autenticação de dispositivos e dados e proteção contra quadros repetidos [12].

3.2.1 A Camada de Rede *ZigBee*

As responsabilidades da camada de rede *ZigBee* são: oferecer mecanismos para que um indivíduo participe/deixe uma rede, segurança do quadro, roteamento, descoberta de rotas, descoberta de vizinhos diretos (i.e. cuja distância é um salto) e armazenamento de informações dos vizinhos [12].

A camada de rede *ZigBee* atribui um endereço de rede de 16 bits para cada dispositivo da rede (semelhante ao endereço IP das redes de computadores). A seguir discutiremos como é realizada a transmissão *broadcast* e *unicast* nas redes *ZigBee*.

- **Transmissão *Broadcast***

A transmissão *broadcast* é aquela onde a mensagem transmitida deve ser propagada para todos os nós da rede. Para isso, todos os nós que recebem o *broadcast* retransmitirão o pacote três vezes. Cada nó que transmite o *broadcast* cria uma entrada em uma tabela local de transmissões *broadcast*. Esta entrada é utilizada para rastrear se o nó já retransmitiu um pacote *broadcast*. Cada entrada persiste por oito segundos e a tabela suporta até oito entradas. Nesta tabela, também é guardada a seção de dados do pacote *broadcast*, para caso seja necessário uma retransmissão.

- **Transmissão *Unicast***

As transmissões *unicast* são aquelas em que existe apenas um nó transmitindo para um nó destino. No *ZigBee* esta transmissão utiliza sempre o endereço de rede de 16 bits do nó destino. Cada nó também possui um endereço de 64 bits permanente, o qual é um número de série do módulo e não pode ser modificado. Este endereço de 64 bits pode ser utilizado para descobrir o endereço de rede do destino.

Para descobrir o endereço de rede de um dispositivo, a partir do seu endereço de 64 bits, o emissor envia um *broadcast* que contem o endereço de 64 bits do dispositivo que será o receptor. Dispositivos que recebem o *broadcast* checam se seu endereço de 64 bits é igual ao do *broadcast*. Se for igual, o dispositivo manda um pacote com o seu endereço de rede para o emissor do *broadcast*.

Descoberto o endereço de rede, outro componente necessário para a transmissão é uma rota. Dispositivos roteadores e o coordenador podem participar do estabelecimento de rotas em um processo chamado descoberta de rota. O processo é baseado no protocolo AODV (*Ad-Hoc On-demand Distance Vector Routing*).

O AODV é realizado usando tabelas de roteamento em cada nó. Estas tabelas indicam qual o próximo salto para um determinado destino. Quando uma rota não é conhecida o nó que deseja obter a rota envia um comando de requisição de rota em *broadcast*. Esta requisição contém o endereço de rede destino e fonte, e um campo para o custo da rota. A metodologia padrão para custo de rota *ZigBee* é,

$$C\{l\} = \min\{7, round(P_l^{-4})\}, \qquad (3.1)$$

onde P_l é a probabilidade de um pacote ser entregue em um enlace l, o que é obtido através do LQI (*Link Quality Indication*) fornecido pelas camadas PHY e MAC do IEEE 802.15.4. O *link* cujo custo é maior que três não é levado em consideração, é como se não houvesse conectividade entre os nós. O custo total da rota é composto pela soma do custo dos enlaces. Outras metodologias para custo de rota *ZigBee* podem ser encontradas em [12].

Na medida em que a requisição vai se propagando pela rede, os nós que a recebem re-transmitem a requisição, em *broadcast*, atualizando o campo de custo da rota e criam entradas temporárias em sua tabela de roteamento. Quando a requisição chega ao nó destino, este compara o campo de custo com o custo de rota já estabelecido anteriormente (se houver). Se o novo custo for menor que o anterior, o nó de destino transmite um pacote de resposta de rota para o nó que originalmente requereu a rota. O *ZigBee* ainda inclui pacotes ACK, tanto em nível de MAC (os ACKs do IEEE 802.15.4 estão habilitados) assim como também ACK em nível da camada de rede. A sequência de transmissão é encontrada na Figura 3.4, assim como também maiores detalhes sobre a camada de rede *ZigBee* em [12] [13].

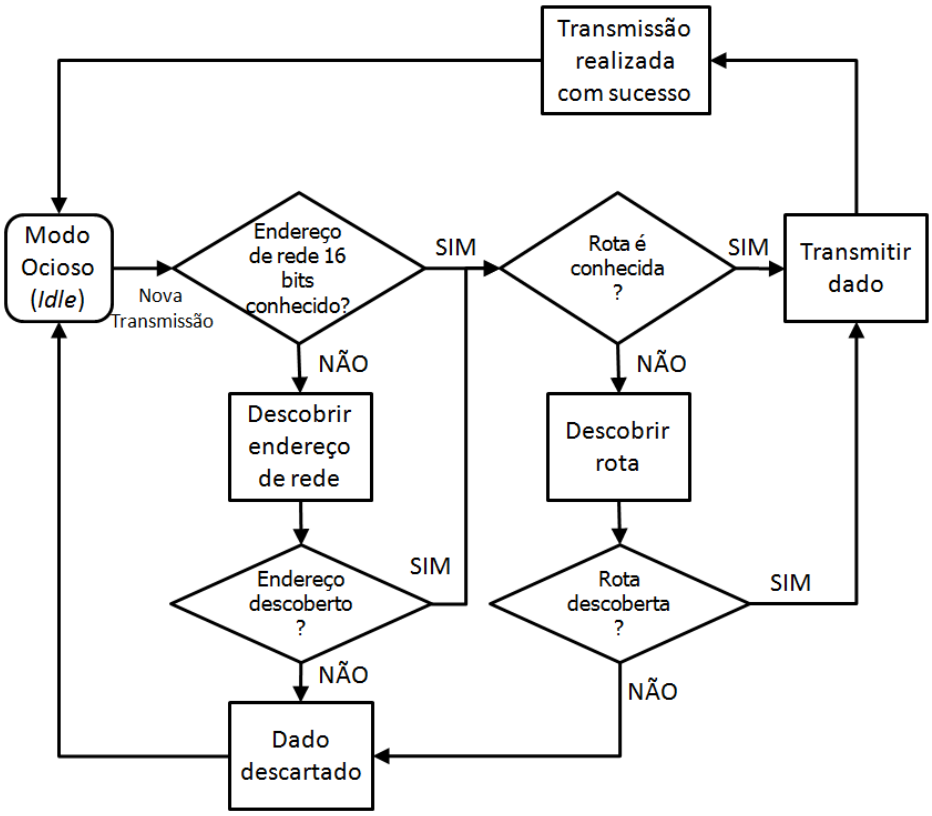


Figura 3.4 – Sequência de Transmissão [13].

3.2.2 A camada de Aplicação *ZigBee*

Esta camada é dividida em três partes: a sub-camada de suporte à aplicação (APS - *Application Support*), o *ZigBee Device Objects* (ZDO) e o *Framework* de Aplicação, as quais podem ser observadas na Figura 3.3.

A APS é uma interface entre os serviços providos pela camada de rede e a camada de aplicação, cuja função é: manter as tabelas de roteamento, encaminhar mensagens entre dispositivos, gerenciar endereços, mapear o endereço de 64 bits no endereço de rede (16 bits) e dar suporte a realização do transporte de dados. O *Framework* de Aplicação é o ambiente no qual as aplicações são hospedadas para controlar e gerenciar as camadas de protocolos no dispositivo *ZigBee*. As aplicações objetos são desenvolvidas pelos fabricantes (*end manufactures*) e é neste elemento que o dispositivo *ZigBee* é personalizado para várias aplicações. Já o ZDO utiliza os serviços da APS e da camada de rede para dar função ao dispositivo *ZigBee* (lembrando que os dispositivos podem ser coordenador, roteador e dispositivo final); além disso o ZDO também encontra dispositivos na rede e suas aplicações, e realiza

tarefas relacionadas à segurança. Uma especificação detalhada acerca de cada elemento da camada de aplicação e seus sub-elementos é encontrada em [12].

3.3 Topologias *ZigBee*

Existem três diferentes topologias suportadas pelo *ZigBee*: estrela, árvore e *mesh*. A topologia estrela é a mais simples, mas não é escalável, no que diz respeito à quantidade de nós na rede e área de cobertura. As topologias árvore e *mesh* são mais favoráveis quando existe uma grande área a ser coberta e muitos dispositivos fazem parte da rede.

A especificação *ZigBee* define uma topologia baseada em árvore (i.e. é uma topologia hierárquica), como um caso particular da rede *peer-to-peer* do IEEE 802.15.4 [4]. A vantagem desta topologia é que o roteamento é uma tarefa mais simples, já que existe apenas uma rota entre um par de nós, por outro lado, a não existência de rotas alternativas torna a topologia não robusta a falhas nos enlaces.

Na topologia *mesh* não existe a relação hierárquica inerente à árvore, os dispositivos podem se conectar com qualquer outro dispositivo, seja diretamente ou através de dispositivos roteadores. Nessa topologia, em geral, existe mais de uma rota entre dois nós, sendo tarefa do protocolo de roteamento selecionar a rota menos custosa.

3.4 Módulos *ZigBee*

Um módulo *ZigBee* é um *hardware* que está pronto para aplicação. Estes módulos são certificados por agências regulamentadoras, no caso do Brasil a Anatel. Os módulos normalmente podem ser alimentados por baterias e podem incluir pinos de entrada/saída GPIOs (*General Purpose Input/Output*) ou ADC. Alguns módulos *ZigBee* podem ser observados na Figura 3.5.

Um módulo bastante utilizado é o *XBee*™ da Digi® [14]. Existe uma grande variedade de módulos *XBee*™ e nem todos implementam a pilha de protocolos *ZigBee* (alguns implementam apenas o IEEE 802.15.4) [15]. Os módulos variam em potência de transmissão (e consequentemente alcance e gasto de energia), tipos de antenas e capacidade de *hardware* (i.e. o microcontrolador embarcado no módulo).

Na Figura 3.6 podem ser observados os módulos *XBee*™ *Serie 2*, que possuem um microcontrolador da Ember Networks e suportam a pilha de protocolos *ZigBee*, e os diferentes tipos de antenas. Em alguns dos módulos mostrados é possível utilizar antenas que aumentam a sensibilidade de recepção, aumentando o alcance dos nós.

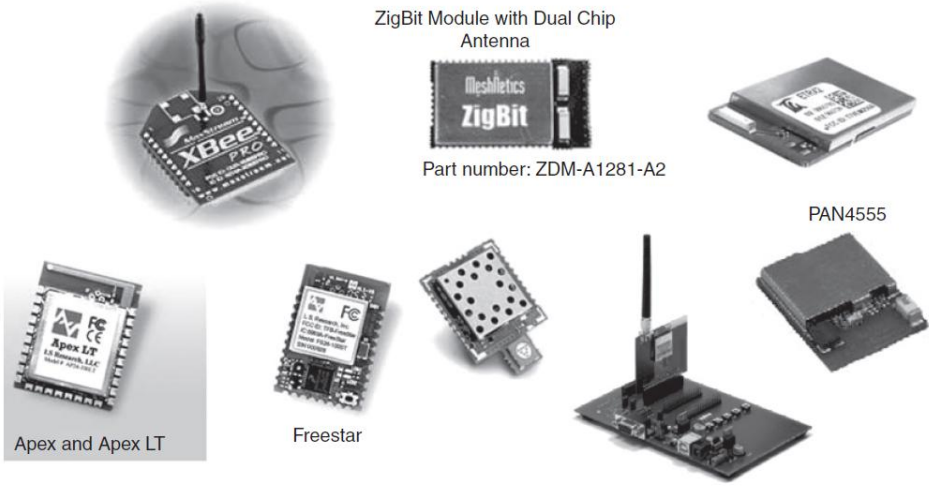


Figura 3.5 – Alguns módulos *ZigBee*, extraída de [11].

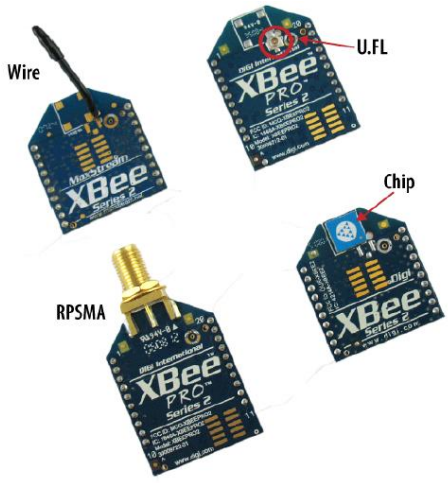


Figura 3.6 – Módulos *XBee Serie 2* e os tipos de antenas, extraída de [15].

Capítulo 4

WSN Desenvolvida

O objetivo deste trabalho é o desenvolvimento de uma WSN, para diversas aplicações, utilizando a tecnologia *ZigBee*. Este capítulo trata do projeto dessa rede de sensores sem fio, a qual é composta por projeto de *hardware*, *firmware* e *software*. Na seção 4.1 são apresentados os motivos da escolha pela tecnologia *ZigBee.* Na seção 4.2 é descrita uma visão geral do projeto desenvolvido, na seção 4.3 o projeto do *hardware* é detalhado, dividindo-o em blocos e descrevendo cada um deles. Na seção 4.4 é descrito o projeto de *firmware e software*, através de diagramas e a seção 4.5 trata do protocolo de comunicação desenvolvido para extrair as informações obtidas pela WSN e realizar funções de gerenciamento.

4.1 Escolha da tecnologia *ZigBee*

A especificação IEEE 802.15.4 é muito eficiente para WSNs. Entretanto, há outras tecnologias comerciais projetadas sobre este padrão e existem outros que servem ao mesmo propósito do IEEE 802.15.4 [11].

No entanto, o IEEE 802.15.4 possui uma excelente definição das camadas física e MAC para redes de sensores sem fio, o que o torna de fato um padrão para WSNs [4] [11]. A tecnologia *ZigBee* foi a primeira a propor camadas superiores para o IEEE 802.15.4 e assegurar um padrão e interoperabilidade, deixando o usuário livre para escolher entre diversos fabricantes. Sendo assim, além de todas as vantagens encontradas no Capítulo 2, a escolha da tecnologia *ZigBee* se deu devido ao fato de se tratar de um padrão bem estabelecido que garante a interoperabilidade inter fabricantes.

4.2 Visão Geral

A WSN desenvolvida é composta de quatro elementos: plataformas de *hardware*, módulo coordenador da rede, módulo concentrador e um computador com um SGBD (Sistema Gerenciador de Banco de Dados) instalado.

As plataformas de *hardware* são os nós da rede que de fato possuem capacidade de sensoriamento e podem exercer a função de roteador ou dispositivo final. Elas são sistemas microcontrolados que provêm suporte para o módulo *ZigBee* utilizado e são alimentados através de baterias com recarga por painel solar.

O módulo coordenador e o concentrador, na WSN desenvolvida, não possuem capacidade de sensoriamento. É desejável que o coordenador da rede possua redundância de alimentação, então a mesma plataforma para os nós sensores pode ser utilizada com esse fim. O concentrador está ligado a um computador através de um adaptador USB para módulos *XBee*, o qual pode ser visualizado na Figura 4.1. O módulo coordenador também poderia ser utilizado como concentrador, mas quando o número de nós de uma rede *ZigBee* é muito grande (o padrão *ZigBee* aceita até 65.535 nós, já que são 16 bits para endereço de rede) o dispositivo com a função de coordenador fica muito sobrecarregado com funções de gerenciamento da rede, não sendo eficiente utilizar esse nó também como concentrador, embora esta solução seja mais simples, já que o endereço de rede do coordenador é fixo (sempre zero) e o endereço dos demais nó é variável. Além do motivo citado, utilizando um nó que não é o coordenador como concentrador torna mais fácil estender a WSN para um cenário de múltiplos concentradores.

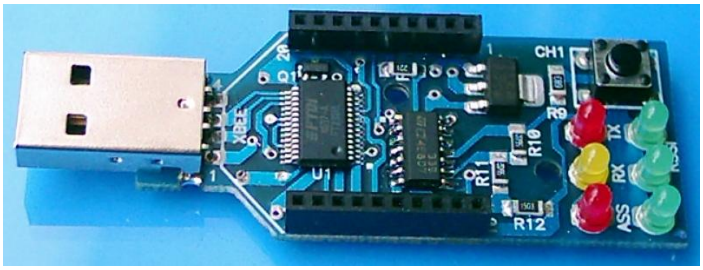


Figura 4.1 – Adaptador USB-*XBee*.

O último elemento é um computador com um SGBD instalado. No projeto desenvolvido foi utilizado o banco de dados *MySQL*. A função deste elemento é capturar os dados coletados pela WSN e armazenar no BD (Banco de Dados) para posterior análise. Com a finalidade de realizar esta tarefa, foi desenvolvido um protocolo para que seja possível capturar os dados e enviar comandos para os nós sensores. Os dados obtidos pela WSN também poderiam ser disponibilizados na Internet para múltiplos usuários. Um esquema geral da WSN desenvolvida é encontrado na Figura 4.2.

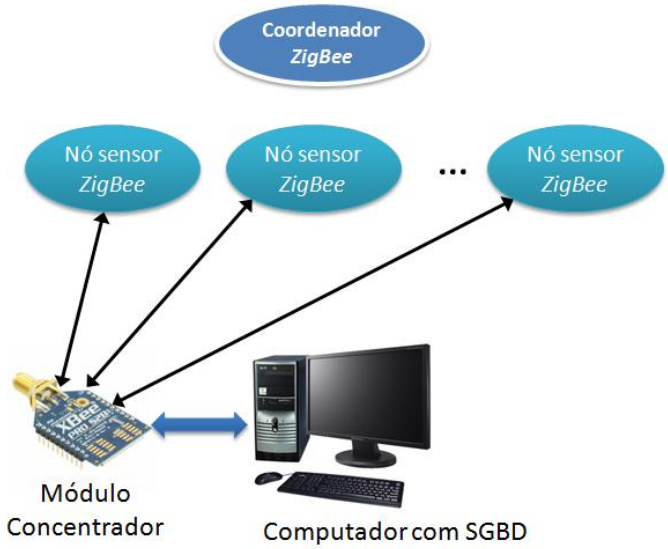


Figura 4.2 – Esquema geral da WSN desenvolvida.

4.3 Projeto do Hardware

O hardware desenvolvido contempla três blocos:

1. bloco do microcontrolador;
2. bloco do módulo *ZigBee*;
3. bloco de Alimentação.

O projeto de hardware foi desenvolvido na ferramenta CAD (*Computer-aided Design*) *Altium Designer* [16]. O diagrama esquemático e *layout* podem ser visualizados por completo no Apêndice A. Uma visão dos blocos e suas interconexões é exibida na Figura 4.3. Cada um desses blocos será explicado a seguir.

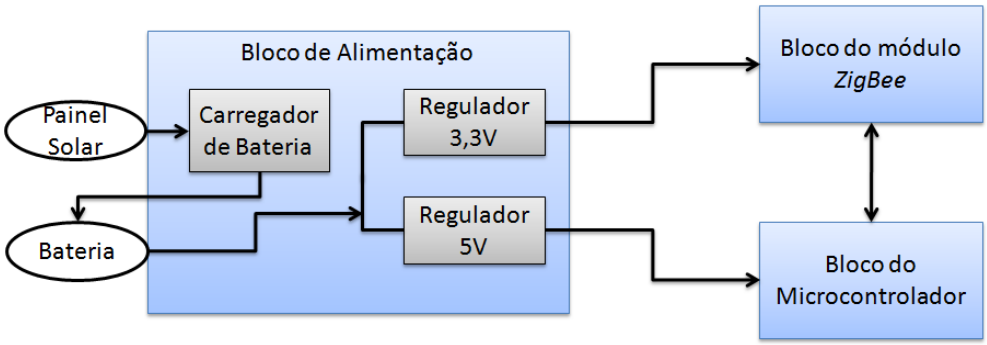


Figura 4.3 – Visão em blocos do *hardware* desenvolvido

4.3.1 Bloco do Microcontrolador

O microcontrolador utilizado no projeto de *hardware* é o PIC18F4550. Esse microcontrolador de oito bits, fabricado pela Microchip, chega a alcançar 12 MIPS (*Million Instructions Per Second*) e possui 32 KB de memória flash de programa e 2 KB de memória RAM. Em termos de periféricos, o PIC18F4550 possuí uma porta USB *Full Speed*, suportando velocidades de até 12 Mbps, uma UART, uma SPI/I2C, treze canais ADC de 10 bits, além de *timers*, pinos de GPIO e interfaces de captura, comparadores e PWM [17]. O microcontrolador se comunica com o módulo *ZigBee* por meio da interface UART (*Universal Asynchronous Receiver/Transmitter*).

Os circuitos básicos para o microcontrolador PIC18F4550 são: interface de gravação, oscilador e reset. Esses circuitos são exibidos na Figura 4.4. O circuito oscilador utiliza um cristal de 20 MHz e capacitores de 15 pF, como especificado no *datasheet* do microcontrolador. Já o circuito de gravação foi projetado tendo o vista o gravador e depurador da Microchip ICD3 [18]. Este gravador requer um conector RJ-11 e algumas precauções para que a função de *debug* não fique comprometida. O circuito de *reset* também está relacionado ao de gravação e segue as especificações do ICD3.

Também faz parte do bloco de *hardware* do microcontrolador um *display* LCD, de dezesseis colunas por duas linhas, para visualização de mensagens de estados da plataforma de *hardware* e do módulo *ZigBee*. O esquema do *display* LCD e os sinais que fazem a conexão deste elemento com o PIC18F4550 podem ser visualizados na Figura 4.5.

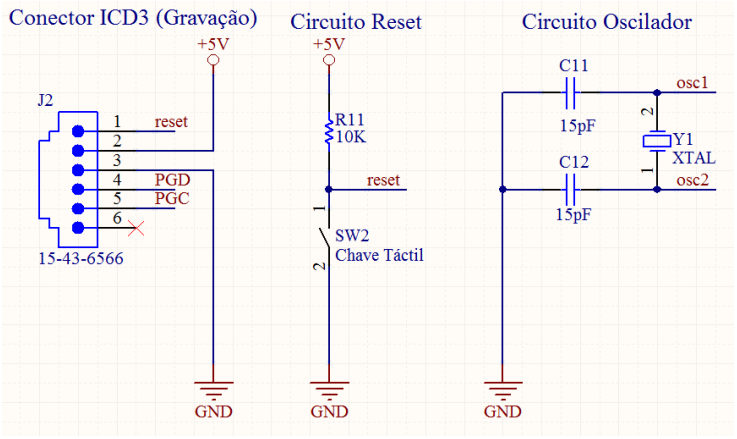


Figura 4.4 – Circuito de gravação, reset e oscilador.

Para teste da WSN desenvolvida, foi gerado um sinal analógico, utilizando um LDR (*Light Dependent Resistor*). O LDR é um elemento que varia a sua resistência de acordo com a luminosidade do ambiente (quanto maior a presença de luz, menor a resistência entre os terminais do LDR). Esse componente faz parte de um divisor de tensão resistivo, gerando assim um sinal analógico a ser lido pelo microcontrolador (utilizando um canal ADC) para transmissão através da WSN. O LDR e o divisor resistivo podem ser vistos na Figura 4.5. Além do LCD e o LDR, a Figura 4.5 ainda apresenta duas chaves táteis (botões) utilizados para testes durante o desenvolvimento.

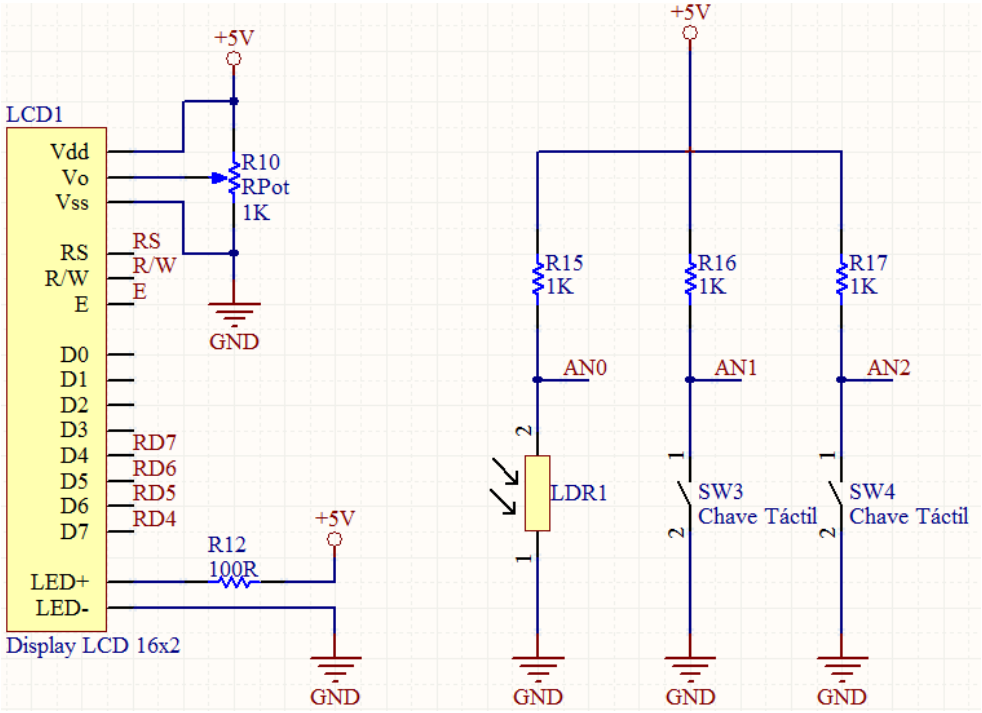


Figura 4.5 – Display LCD, entrada analógica LDR e botões.

Ainda encontram-se disponíveis oito canais ADC do microcontrolador no conector CN1, o qual pode ser observado na Figura 4.6. O objetivo desse conector é oferecer possibilidade de inserir novos sensores para que as informações coletadas por eles sejam transmitidas pela rede. Sendo assim, várias aplicações podem fazer uso desta plataforma, apenas selecionando os sensores necessários e conectando-os a interface ADC disponibilizada pelo conector CN1.

A Figura 4.6 também mostra o PIC18F4550. Os capacitores C13 e C14 são capacitores de desacoplamento, cuja função principal é evitar que ruídos de alta frequência interfiram na fonte de alimentação. Na Figura 4.6 também podem ser

visualizados todos os sinais que saem/chegam do microcontrolador, incluindo a interface UART, destacada pelas portas Tx PIC e Rx PIC. Existe um divisor resistivo no pino de transmissão UART do PIC18F4550 devido ao fato de este microcontrolador funcionar na tensão de 5 V e o módulo *ZigBee* na tensão de 3,3 V.

Por fim, o bloco de *hardware* do microcontrolador ainda oferece os elementos básicos para que se possa utilizar a USB do PIC18F4550. Todos os elementos podem ser observados no Apêndice A.

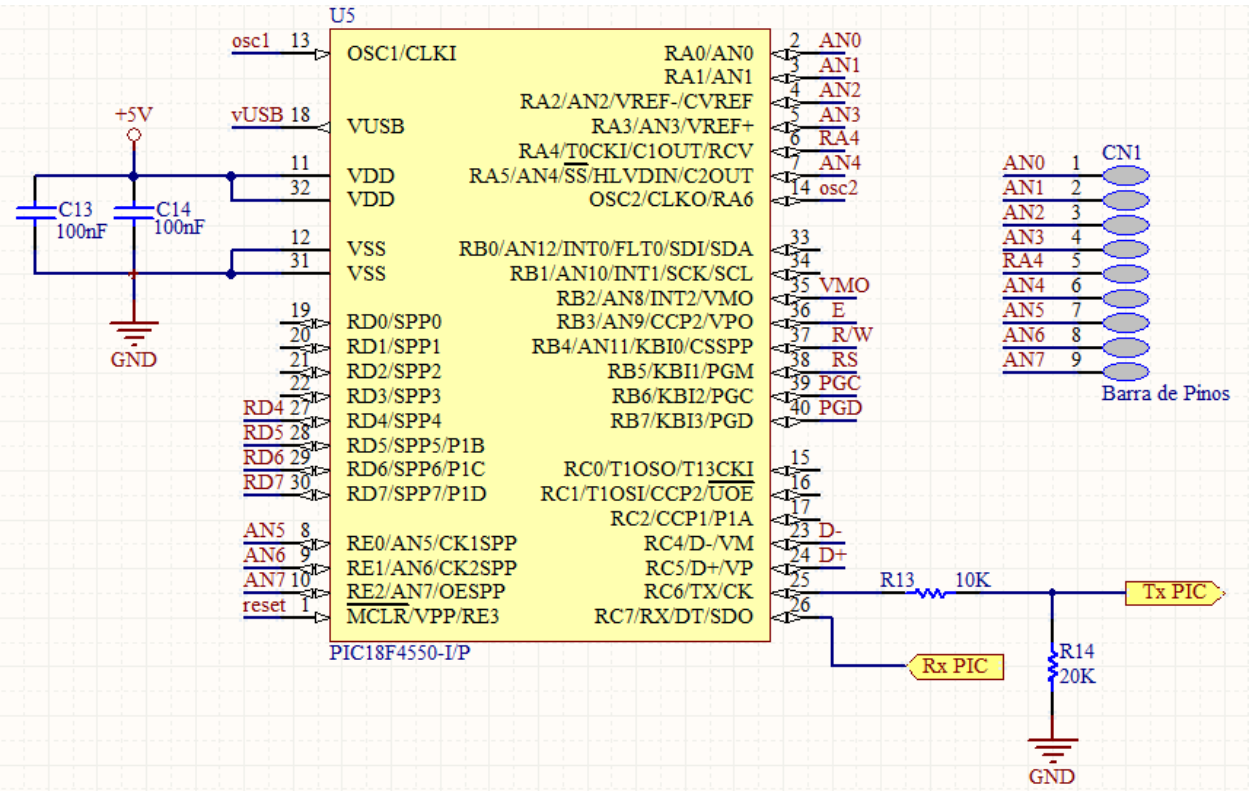


Figura 4.6 – PIC18F4550, canais ADCs disponíveis e interface UART para o módulo *ZigBee*.

4.3.2 Bloco do Módulo *ZigBee*

O módulo *ZigBee* utilizado no projeto é o *Xbee-PRO ZNet 2.5 OEM* (também conhecido como *Xbee Series 2*) fabricado pela Digi. As principais características deste módulo são encontradas na Tabela 4.1. Ainda existe uma variação deste módulo (chamado *Xbee ZNet 2.5*) que é mais econômico do ponto de vista energético e consequentemente possui alcances menores.

Os módulos *ZigBee* utilizados no projeto precisaram ser configurados quanto à função exercida na rede (i.e. se o módulo operará como coordenador, roteador ou dispositivo final). Essa configuração é realizada gravando o *firmware* correspondente à função no módulo *ZigBee*. A gravação/atualização *firmware* dos *Xbee* é realizada utilizando um *software* chamado X-CTU, disponibilizado pela Digi, e o adaptador USB-*XBee* mostrado na Figura 4.1. Esse software, além de gravação/atualização de *firmware*, permite que se estabeleça comunicação com o módulo, ou seja, envio e recebimento de comandos, e que se faça leitura/gravação de parâmetros.

Tabela 4.1 – Especificações do Módulo XBee-PRO ZNet 2.5 OEM

Alcance *indoor*	100 m
Alcance *Outdoor*	1,6 km
Potência de transmissão	63 mW (+ 18 dBm) 10 mW (+10 dBm) para variação internacional
Sensibilidade de Recepção	-102 dBm
Tensão de alimentação	3,0 – 3,4 V
Consumo de corrente na potência máxima de transmissão - I_{Tr}	295 mA (sendo alimentado por 3,3 V)
Consumo de corrente no modo de recepção - I_{Rc}	45 mA (sendo alimentado por 3,3 V)
Consumo de corrente no modo ocioso - I_{oc}	15 mA (sendo alimentado por 3,3 V)

O módulo da Digi apresenta dois modos de operação: o modo transparente e o modo API. No modo transparente, as configurações do módulo são realizadas através de comandos AT e o módulo está por padrão sempre pronto para transmissão/recepção. No modo API, existe uma maior interação com as capacidades da rede e toda a comunicação com o módulo é realizado através de *frames* e notificações de eventos. Como o modo transparente é mais simples e

oferece possibilidade de configuração das propriedades necessárias para o desenvolvimento da WSN, esse modo foi o escolhido.

O *hardware* projetado para o módulo *ZigBee* pode ser visto na Figura 4.7. O capacitor C9 serve para desacoplamento e o capacitor C8 serve para fornecer energia para possíveis picos de corrente quando o módulo estiver transmitindo. Também pode ser visto o circuito de *reset*, com uma chave para acionamento manual e dois *leds*: um que indica que o módulo está ligado (D8) e outro que indica se o módulo está associado a uma rede (D9). Caso a função do módulo seja a de coordenador, o *led* D9 piscará uma vez por segundo se a rede foi inicializada com sucesso; caso o módulo exerça a função de roteador/dispositivo final e tenha conseguido se associar a uma rede com sucesso esse *led* piscará duas vezes por segundo.

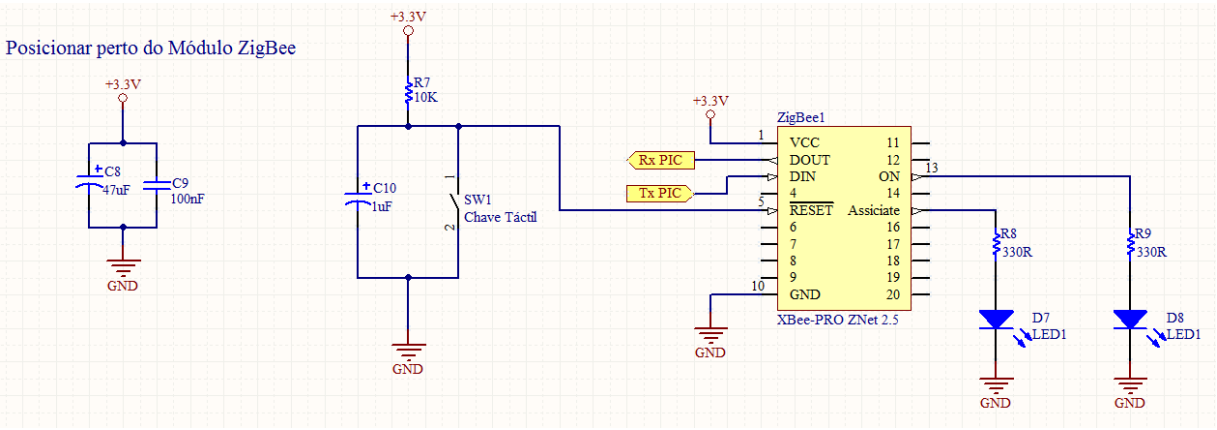


Figura 4.7 – Bloco de *hardware* do módulo *ZigBee*.

4.3.3 Bloco de Alimentação

A alimentação da plataforma é realizada por uma bateria que será recarregada por um painel solar. A plataforma também oferece outra opção de alimentação que é apenas uma fonte de tensão com saída entre 7 V e 25 V no lugar do painel solar. Essa última opção é interessante para testes da plataforma.

Para cálculo da potência do painel foi realizada uma estimativa de consumo da plataforma. O consumo de corrente do microcontrolador PIC18F4550 e do módulo *ZigBee* encontram-se nas Tabela 4.2 e Tabela 4.1, respectivamente.

Tabela 4.2 – Características de consumo do PIC18F4550

Corrente máxima entrando no pino VDD - $Ivdd_{MAX}$	250 mA
Corrente máxima fornecida/drenada por um pino de I/O	25 mA
Corrente máxima fornecida por todas as portas I/O – Ip_{MAX}	200 mA

A potência consumida apenas pelo PIC18F4550 é obtida através de

$$P_{PIC} = 5\,.\,(Ivdd_{MAX} - Ip_{MAX}) = 5\,.\,50.\,10^{-3} = 250\,mW, \qquad (4.1)$$

onde a constante 5 é o valor da tensão de alimentação do PIC18F4550.

A potência consumida pelo módulo *ZigBee* é mais difícil de estimar, tendo em vista que depende de quanto tempo o módulo permanece transmitindo e do período de permanência no estado de recepção. A estimativa do tempo de transmissão é feita com base no protocolo desenvolvido que é explicado nas seções seguintes. Já o tempo no estado de recepção depende de um parâmetro que pode ser configurado no módulo, o qual informa o período de tempo em que o nó deve permanecer no estado de *sleep*. Com base nessas informações, foi estimado que o módulo deve permanecer durante 20% do tempo transmitindo, 70% no estado de recepção e 10% no estado ocioso, a potência consumida pelo módulo *ZigBee*, com base nos valores de corrente da Tabela 4.1 é:

$$P_{ZigBee} = 3{,}3\,.\,(0{,}2.\,I_{Tr} + \,0{,}7.\,I_{Rc} + \,0{,}1.\,I_{oc}) = 303{,}6\,mW, \qquad (5.1)$$

onde a constante $3{,}3$ é o valor da tensão de alimentação do módulo *ZigBee*.

Uma característica importante é que apenas módulos configurados como dispositivos finais podem entrar no estado *sleep*, logo, roteadores e o coordenador estarão sempre no estado de recepção. No entanto, como foi considerado que os dispositivos finais permanecerão apenas 10% do tempo no modo ocioso (*sleep*), a diferença entre a potência estimada requerida por um dispositivo final e um roteador/coordenador não passa de 10 mW.

Sendo assim, a potência total ($P_{ZigBee} + P_{PIC}$) é de $553{,}6\,mW$. Utilizando um fator de 5 vezes a potência total, para cobrir os gastos com os reguladores de tensão e os outros elementos, a potência mínima do painel solar deve ser de aproximadamente 3 W (utilizando um arredondamento para cima).

Com base nessa estimativa, foi escolhido o painel da Figura 4.8 (a). Um requisito importante atendido por ele é que a tensão de saída é suficiente para carregar a bateria sem necessidade de nenhum circuito elevador de tensão. Esse painel possui as dimensões de 190 mm x 310 mm x 28 mm, tensão máxima de 17,6 V, corrente nominal de 0,29 A e potência nominal de 5 W [19].

Foi adotada uma bateria chumbo-ácida regulada por válvula, conhecida no mercado como bateria selada de 6 V 4,5 Ah [20], exibida na Figura 4.8 (b). Os motivos utilizados para escolha desta bateria foram: custo e presença no mercado local.

O primeiro circuito do bloco de alimentação, e o mais simples, é mostrado na Figura 4.9. Esse circuito é a entrada do painel solar ou fonte de alimentação. O conector utilizado é o P4, muito comum nas fontes de alimentação, seguido por uma ponte de diodos, cujo objetivo é não fixar as polaridades.

(a) (b)

Figura 4.8 – Painel solar e bateria.

Também estão presentes no bloco de alimentação dois reguladores de tensão (ambos recebem como entrada a alimentação que vem da bateria): um para 5 V (a tensão utilizada pelo microcontrolador e periféricos) e um para 3,3 V (para o módulo

ZigBee). Para o regulador de 5 V é utilizado o circuito integrado 7805CT. Para o regulador de 3,3 V é utilizado o regulador de tensão ajustável LM317T.

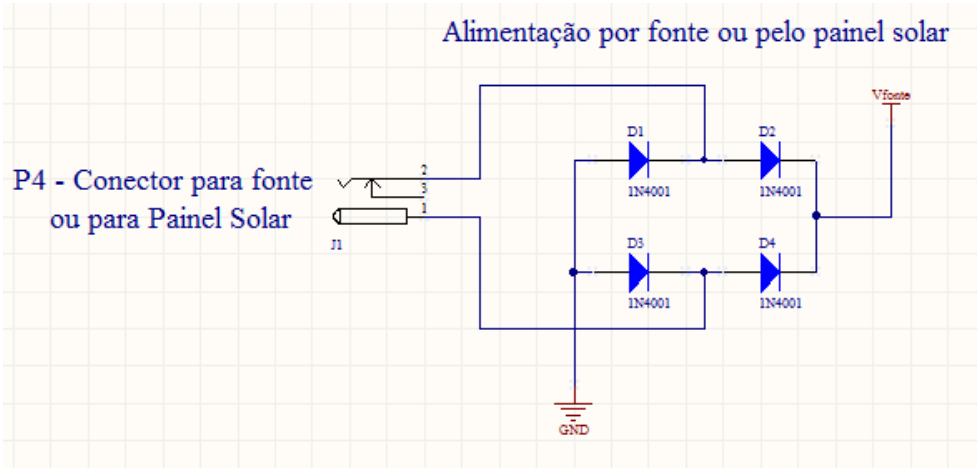


Figura 4.9 – Entrada do Painel Solar ou Fonte de alimentação.

Por fim, o bloco de alimentação também engloba um controlador de carga para bateria. A função desse elemento é limitar a corrente a ser drenada do painel solar, regular a tensão proveniente do painel para um nível aceitável pela bateria e impedir que cargas elétricas transitem da bateria para o painel solar (essa última função é executada pelo diodo D6). Na Figura 4.10 pode-se observar o esquema do controlador de carga. O primeiro CI (Circuito Integrado) LM317T exerce a função de regular a corrente máxima a ser drenada do painel (no caso, a corrente máxima é de aproximadamente 300 mA) enquanto que o segundo regula a tensão de carga da bateria. Os esquemas de limitador de corrente e regulagem de tensão com LM317T estão disponíveis em [21].

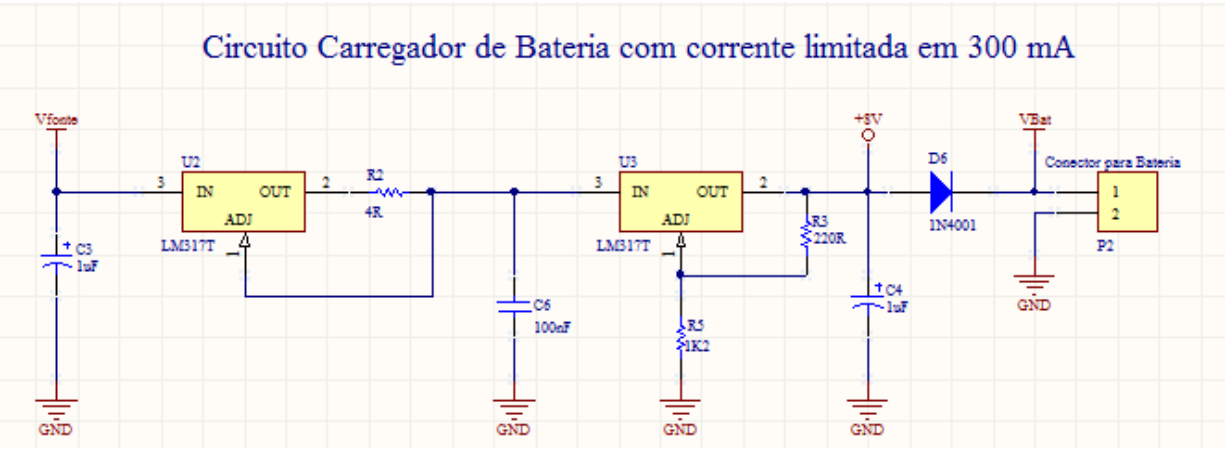


Figura 4.10 – Controlador de carga.

4.3.4 *Layout* Desenvolvido

O *layout* é a organização dos componentes no espaço físico de uma placa e os caminhos que os interligam (trilhas). A tarefa de *layout* basicamente se divide em três etapas: elaborar *footprint* dos componentes, dispor os componentes no espaço da placa (chamado de *placement*) e por fim rotear as ligações entre eles.

O *footprint* é a representação gráfica da forma real do componente com suas dimensões precisas. A ferramenta CAD disponibiliza bibliotecas nas quais se encontram *footprints* dos elementos mais utilizados. Entretanto, alguns componentes que não são comuns, como por exemplo, o módulo *XBee*, não são encontrados nessas bibliotecas, sendo necessário que se produza tanto a representação do elemento no diagrama esquemático quanto sua representação para o *layout* (*footprint*). Alguns *footprints* desenhados estão dispostos na Figura 4.11. Nela, as linhas em verde compõem a serigrafia da placa projetada. A serigrafia é a legenda dos componentes, a qual é visível na face superior da placa e é exibida por completo na Figura 4.12.

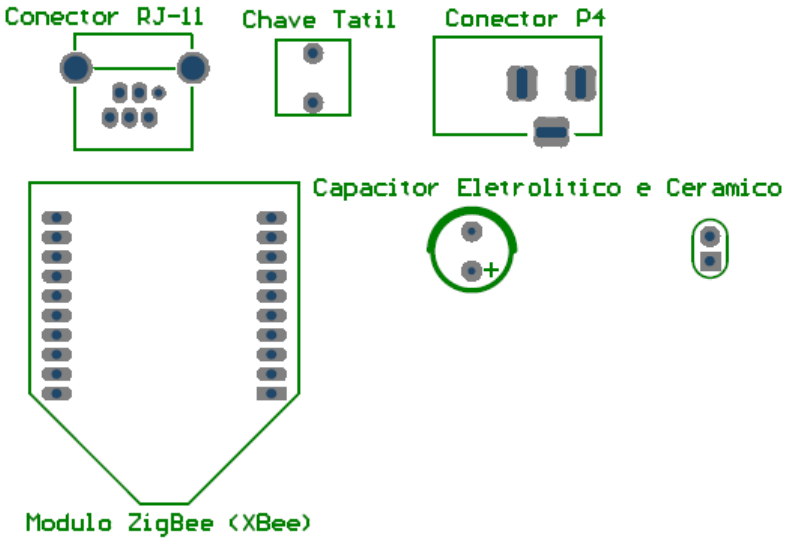


Figura 4.11 – Alguns *footprints* produzidos.

Para a etapa de roteamento, a ferramenta CAD oferece um editor de regas e restrições de projeto. Essa funcionalidade tem como objetivo realizar o gerenciamento das características do *layout* e gerar avisos caso alguma das regras seja violada. As regras básicas utilizadas no projeto foram:

1. distância mínima entre trilhas: 0,635 mm (25 mils);

2. largura mínima de trilha 0,5 mm;
3. largura máxima de trilha 5 mm;

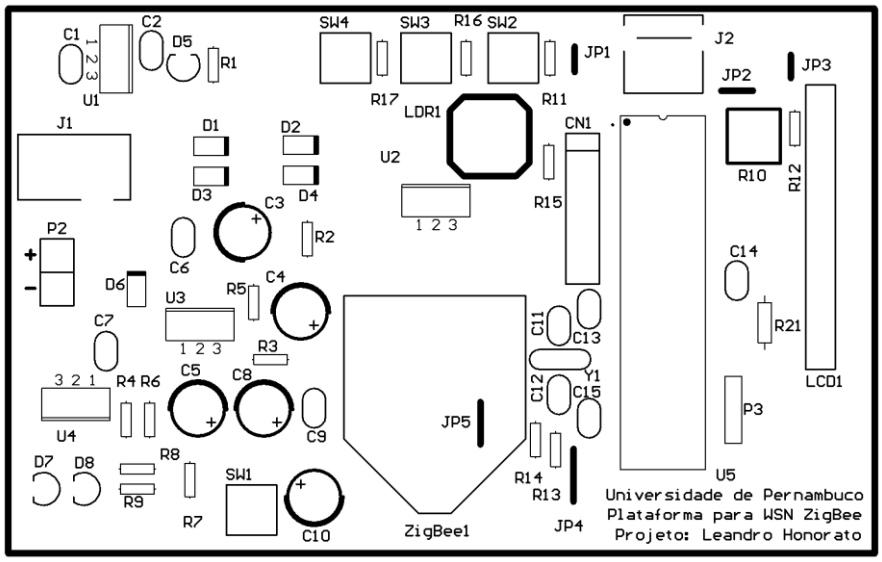


Figura 4.12 – Serigrafia da plataforma desenvolvida.

As regras adotadas têm em vista atender aos requisitos do processo de fabricação adotado. Na Figura 4.13 pode ser visualizado o resultado final da etapa de roteamento. O tamanho final da placa é de 131 mm x 82 mm. A ferramenta utilizada ainda oferece uma visão 3D da placa desenvolvida, com o objetivo de verificação visual do projeto. A visão 3D da plataforma desenvolvida pode ser observada na Figura 4.14.

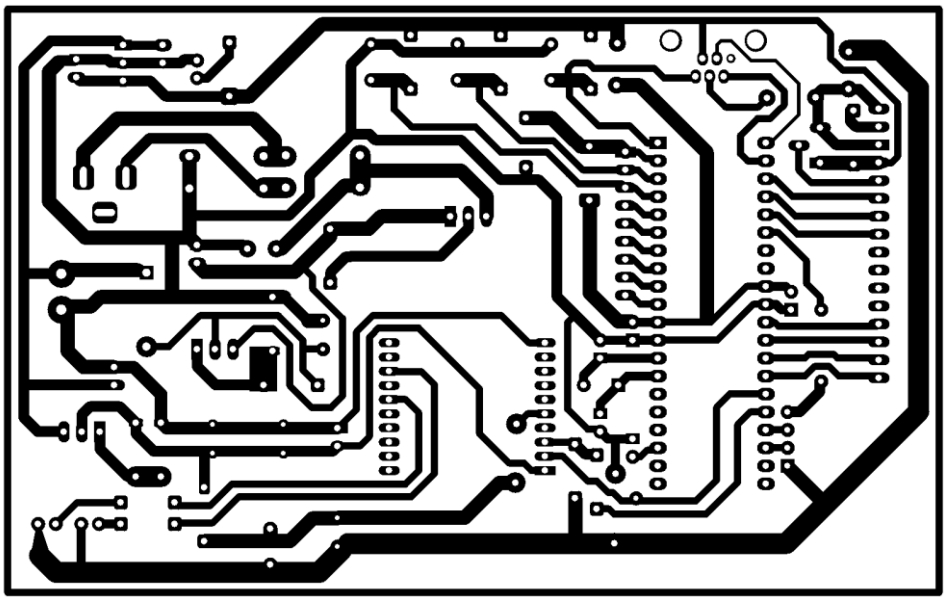


Figura 4.13 – *Layout* da plataforma desenvolvida.

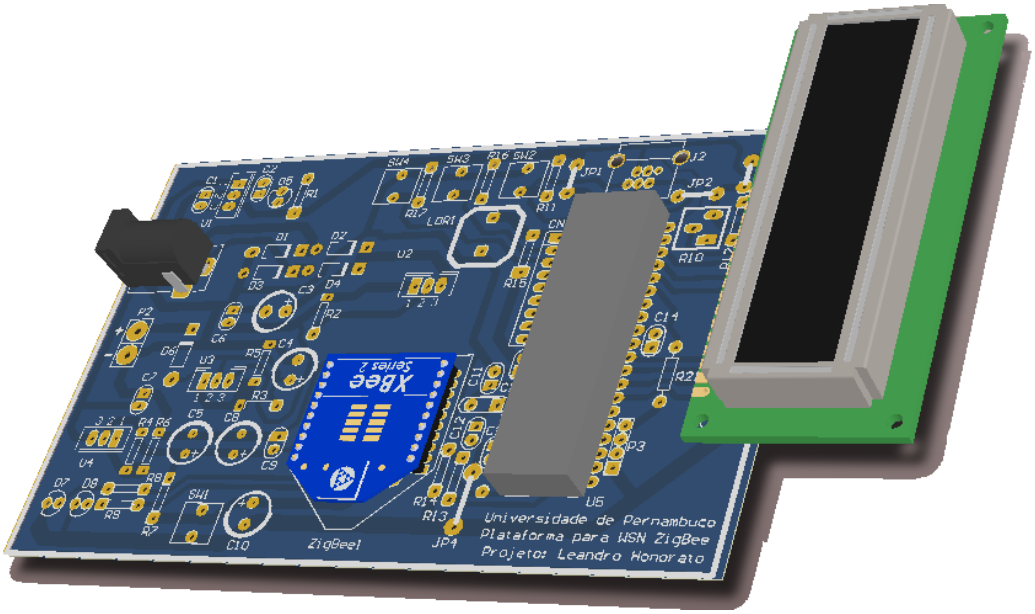


Figura 4.14 – Visão 3D da plataforma de *hardware*.

4.4 Projeto do *Firmware* e *Software*

Os objetivos do *firmware* desenvolvido para o microcontrolador PIC18F4550 são: prover acesso às leituras dos sensores e gerenciamento remoto das configurações do módulo *ZigBee*. Esse *firmware* foi desenvolvido na linguagem C e o compilador utilizado foi o CCS [22].

O objetivo do *software* é enviar requisições para todos os nós da rede, solicitando os valores lidos pelos seus sensores, e realizar o armazenamento dessas informações em um banco de dados. O *software* foi desenvolvido na linguagem Java e foi utilizando um banco de dados MySQL instalado localmente. O *firmware* e o *software* serão detalhados a seguir.

4.4.1 Projeto de *Firmware*

A Figura 4.15 mostra o fluxograma do *firmware* desenvolvido. Ele pode ser dividido em duas partes: a etapa de inicialização e o *loop* principal.

Na primeira etapa são inicializados: os buffers de recepção serial, os canais ADC, variáveis de controle e a interrupção da UART, a qual é ativada todas as vezes que chegar algum dado por essa interface. Realizada essa configuração, o *firmware* espera a inicialização do módulo *ZigBee*. O módulo *XBee Series 2* procura

automaticamente uma rede para se associar (caso seja um roteador/dispositivo final) ou inicia a criação de uma rede (caso seja coordenador). Após a espera pela inicialização do módulo, o *firmware* requisita o endereço físico (o endereço de 64 bits) do módulo e o armazena. Essa informação é importante para o protocolo definido (o qual será abordado na seção 4.5).

Ainda nessa etapa, é configurado o endereço do destino do módulo, o qual deve ser o concentrador. Os módulos *XBee* que suportam o padrão *ZigBee* possuem um parâmetro chamado NI (*Node Identifier*), o qual consiste em um nome do módulo. Os módulos utilizados também oferecem um serviço de descoberta do endereço de rede através do NI. O *firmware* desenvolvido utiliza esse serviço para obter o endereço de rede do concentrador, o qual possui o nome de "*SINK*", desta maneira, não existe problema em trocar o módulo concentrador (que consequentemente terá um endereço físico diferente do anterior) desde que o novo módulo concentrador mantenha o NI configurado como *"SINK"*. Essa tarefa de configurar o endereço de rede do concentrador é realizada periodicamente, então caso o concentrador apresente defeito pode ser substituído e os nós possuem autonomia suficiente para encontrar o novo módulo concentrador da rede.

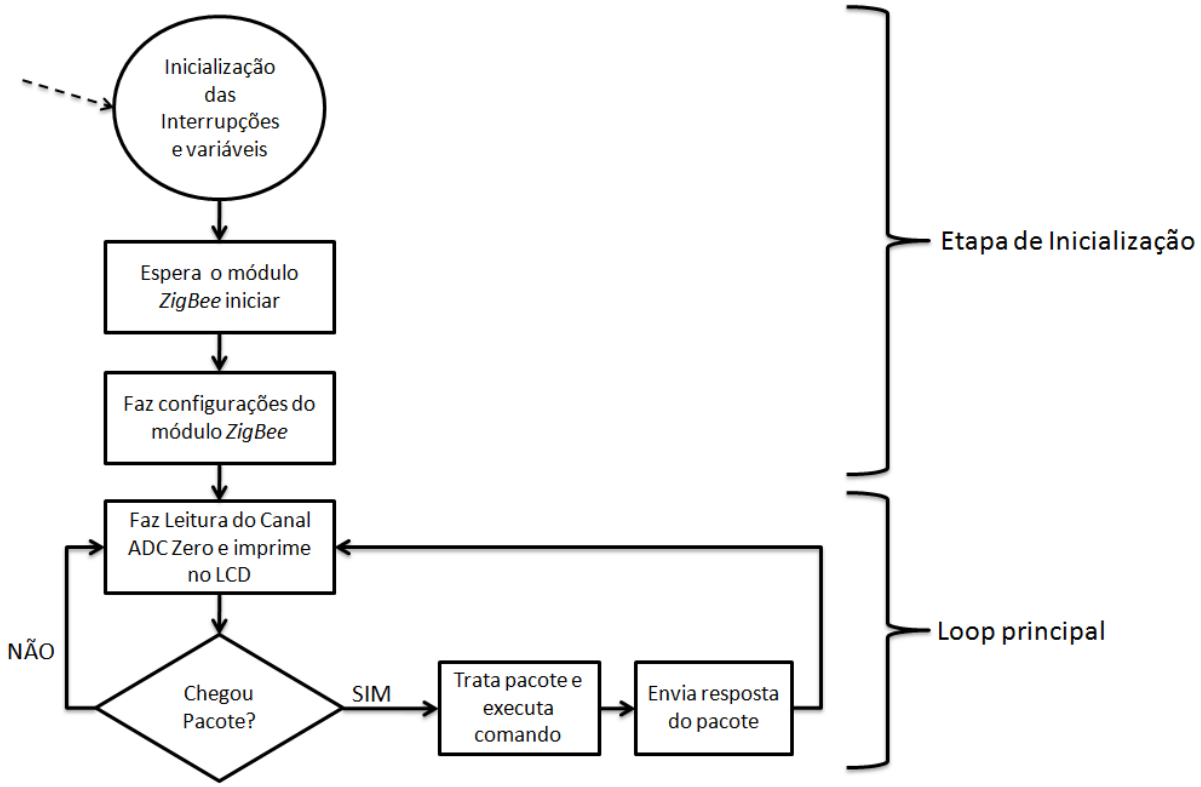


Figura 4.15 – Fluxograma do *firmware*.

Depois de realizada a etapa inicial, o *firmware* entra no *loop* principal, esperando receber um pacote para realizar a tarefa requisitada. No *loop* principal, também ocorre a impressão da leitura do canal ADC no *display* LCD que é atualizada uma vez a cada segundo. O fluxograma do tratamento de um pacote do tipo requisição de leitura pode ser observado na Figura 4.16. Onde x é uma constante inserida no *firmware* para que seja possível realizar a busca pelo nó coordenador periodicamente, o valor adotado para esta constante foi cem, ou seja, a cada cem pacotes enviados, a plataforma de hardware busca novamente o nó concentrador.

Na Figura 4.17 é encontrado o fluxograma do tratamento do pacote de comando remoto. Para enviar um comando para o módulo *XBee Series 2* é necessário inicialmente ir para o modo de comando. Depois de enviado o comando para o módulo *ZigBee* e a resposta é recebida, a plataforma volta para o modo transparente (i.e. o modo de transmissão e recepção de dados), envia a resposta do comando remoto e volta para o *loop* principal esperando por um novo pacote.

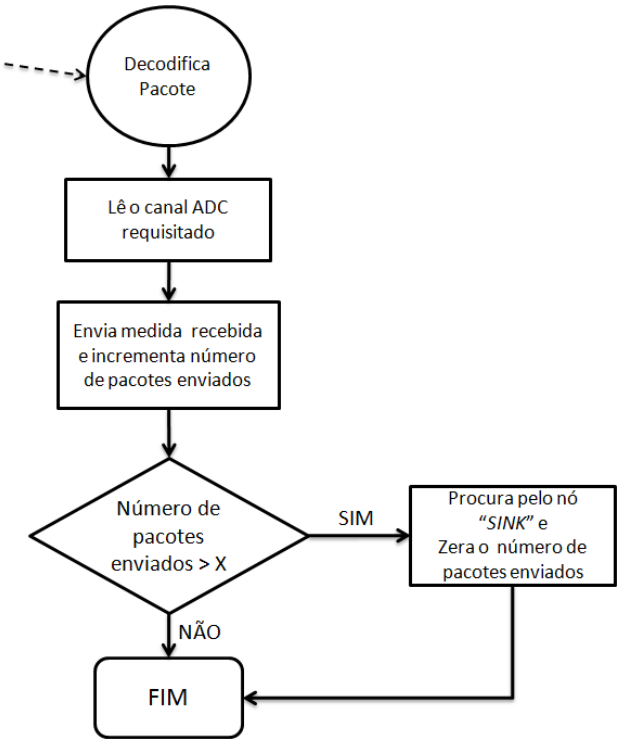


Figura 4.16 – Fluxograma do tratamento do pacote de requisição de leitura.

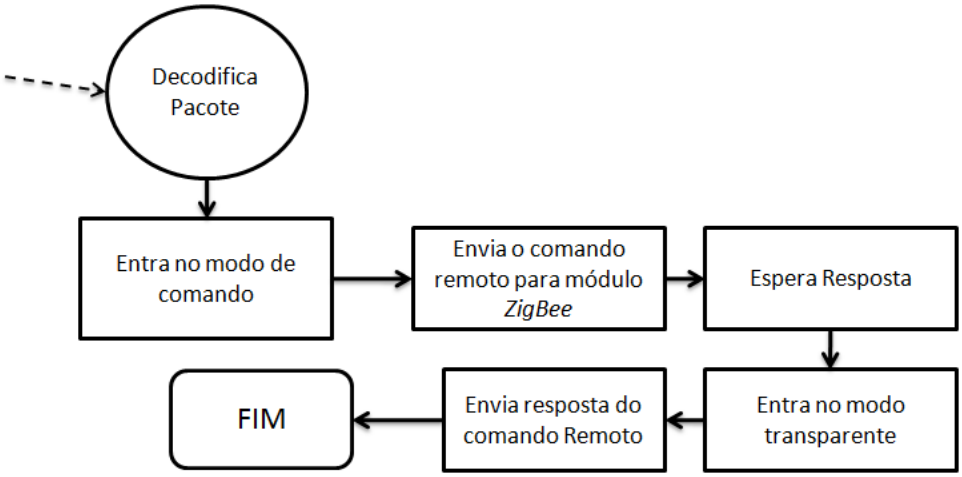


Figura 4.17 – Fluxograma do tratamento do pacote de comando remoto.

O *firmware* foi projetado de forma a prover uma biblioteca de funções para manipulação do módulo *Xbee*. A documentação das funções desenvolvidas está disponível no Apêndice B e o código do *firmware* pode ser visto no Apêndice C.

4.4.2 Projeto de *Software*

O *software* desenvolvido interage com um módulo *ZigBee,* conectado ao computador por meio de um adaptador USB, para obter as leituras dos sensores presentes nas plataformas de *hardware*. Para isso o banco de dados guarda as plataformas de *hardware* existentes e os sensores instalados em cada plataforma (cada sensor está relacionado unicamente a um canal ADC). O modelo lógico do banco de dados é exibido na Figura 4.18.

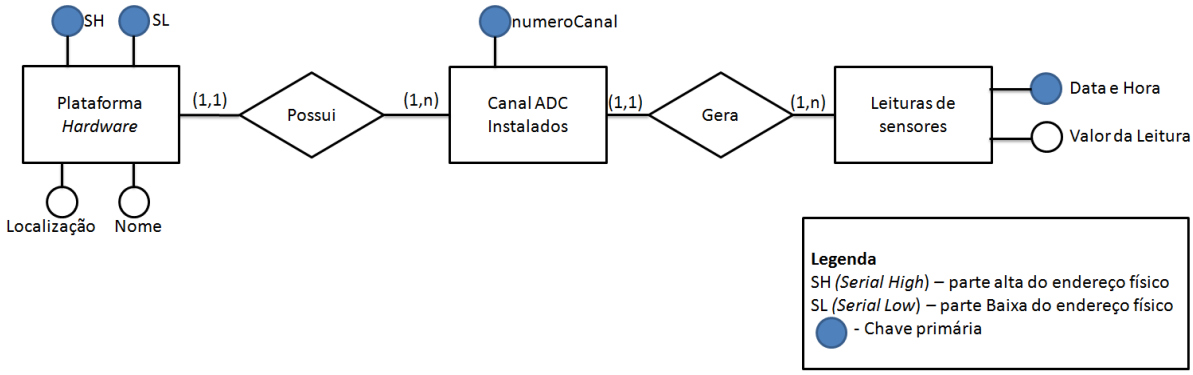


Figura 4.18 – Modelo lógico do banco de dados.

Com base nessas informações, o *software* envia pacotes de requisição de leitura para cada sensor cadastrado, obtém as respostas das requisições e as cadastra no banco de dados (na tabela Leituras de Sensores).

O *software* desenvolvido é composto de duas *threads*: uma de recepção e outra de envio. A *thread* de recepção fica apenas escutando a interface USB esperando por respostas das requisições de leitura. Quando um pacote desse tipo é recebido, ele é decodificado e uma nova leitura é inserida no banco de dados. Já a *thread* de envio tem o seu pseudocódigo apresentado na Figura 4.19.

No pseudocódigo existem três parâmetros: o tempo de espera entre o envio de requisições (x), o tempo de espera entre ciclos (y), onde um ciclo corresponde a uma rodada completa de envios, ou seja, compreende um envio de requisição para cada sensor e a última constante é a quantidade de ciclos necessários para obter uma nova lista de sensores instalados (z). Conforme informado na seção 4.3.3, os dois primeiros parâmetros estão relacionados com a frequência de transmissão das plataformas de *hardware*, já que quanto menor o intervalo entre as requisições, maior é o tempo de transmissão dos nós da rede, já que esses apenas transmitem respondendo requisições.

```
obtenha lista de todos os sensores instalados
Numero de ciclos = 0
repita sempre
   para cada sensor instalados
      envie requisição de leitura
      espere x segundos
   fim para
   espere y segundos
   numero de ciclos = numero de ciclos + 1
   se (numero de ciclos > z)
      obtenha lista de todos os canais instalados
      numero de ciclos = 0
   fim se
fim repita
```

Figura 4.19 – Pseudocódigo da *thread* de recepção.

4.5 O Protocolo Desenvolvido

Por não ter sido encontrado nenhum protocolo que atendesse às necessidades deste projeto, foi desenvolvido um protocolo simples que descreve quatro tipos de pacotes:

1. requisição de medida de sensor;
2. resposta de medida de sensor;
3. comando remoto para módulo *ZigBee*;
4. resposta de comando remoto para módulo *ZigBee*.

Cada pacote possui um identificador único que começa com o caractere '#', os campos são separados pelo caractere '*' e o término do pacote é indicado com um *carriage return* ('\r'). Os pacotes podem ser visualizados na Figura 4.20.

Para o pacote de requisição de leitura, exibido na Figura 4.20 (a), a sua informação é um número entre um e oito o qual indica o número do canal ADC a ser lido pelo microcontrolador. Quando a plataforma recebe um pacote desse tipo, é realizada a leitura requisitada e é enviada a resposta de requisição de leitura. O endereço físico do módulo que recebeu a requisição de leitura é dividido em duas partes (cada parte tem 32 bits) e o microcontrolador responde a requisição de leitura no formato exibido na Figura 4.20 (b).

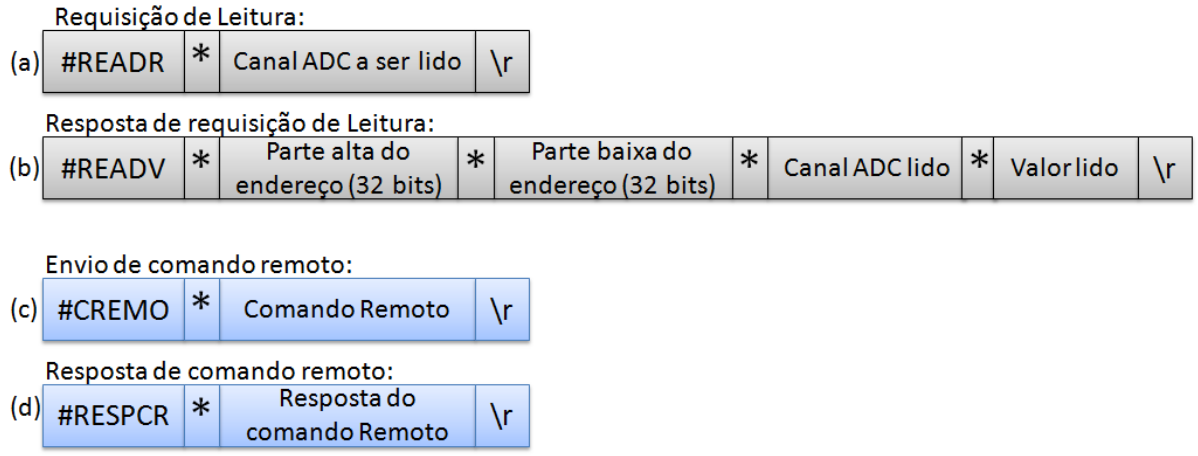


Figura 4.20 – Pacotes do protocolo desenvolvido

Quanto ao pacote de comando remoto, exibido na Figura 4.20 (c), a informação (campo Comando Remoto) é um comando AT a ser executado pelo

módulo destino do pacote. O pacote de resposta do comando remoto, mostrado na Figura 4.20 (c), traz a resposta do comando AT enviado remotamente. A Tabela 4.3 apresenta os principais comandos do módulo *ZigBee* utilizado e as respectivas respostas esperadas. Todos os comandos AT suportados pelo módulo *ZigBee* utilizado no projeto são encontrados em [13].

Tabela 4.3 – Principais comandos AT do módulo *Xbee-PRO ZNet 2.5 OEM*.

Comando AT	Descrição do comando	Parâmetros	Resposta Esperada
AT WR	Grava as configurações em memória não volátil	-	OK
AT RE	Restaura configurações para o padrão	-	OK
AT FW	*Reset* do *firmware* do módulo	-	OK
AT NR	*Reset* dos parâmetros da camada de rede	0 – Reseta os parâmetros apenas do nó local 1 – Envia um broadcast para a rede toda resetar	OK
AT SH	Comando para ler a parte alta do endereço físico	-	Parte alta do endereço físico (32 bits)
AT SL	Comando para ler a parte baixa do endereço físico	-	Parte baixa do endereço físico (32 bits)
AT MY	Comando para ler o endereço de rede do módulo	-	Endereço de rede (16 bits)
AT DH	Lê/escreve a parte alta do endereço físico destino	Parte alta do endereço físico (para escrita) ou nada (para leitura)	OK (para escrita) ou parte alta do endereço (para leitura)
AT DL	Lê/escreve a parte baixa do endereço físico destino	Parte baixa do endereço físico (para escrita) ou nada (para leitura)	OK (para escrita) ou parte baixa do endereço (para leitura)
AT NI	Lê/escreve o nome do módulo	Novo nome do módulo (para escrita) ou nada (para leitura)	OK (para escrita) ou o nome do módulo (para leitura)
AT DN	Descobre o endereço de um nó a partir de um nome (NI)	Nome do nó destino	ERROR (caso o nó destino não tenha sido

			encontrado) ou OK
AT CH	Lê o canal utilizado para transmissão e recepção	-	Número do canal utilizado
AT OP	Lê o endereço da WPAN cujo módulo está associado	-	Endereço da rede (16 bits)
AT EE	Configura/Lê o estado da criptografia	0 – Criptografia desabilitada 1 – Criptografia habilitada	OK (para configuração) ou 0-1 (para leitura)
AT KY	Configura a chave de criptografia 128-bit AES	Chave de criptografia de 128 bits	OK

Lembrando que para os comandos AT DH e AT DL ocorre o processo de busca do endereço de rede através do endereço físico. E todos os nomes são sequências de caracteres ASCII imprimíveis.

Capítulo 5

Resultados Obtidos

Neste capítulo são apresentados os resultados obtidos a partir do projeto descrito no capítulo anterior. Na seção 5.1 é apresentado o sistema embarcado produzido como um todo, ou seja, a plataforma de *hardware* juntamente com o *firmware* desenvolvido e na seção 5.2 são exibidos os resultados alcançados pelo *software* desenvolvido.

5.1 O Sistema Embarcado Produzido

As plataformas de *hardware* foram confeccionadas pelo processo fotolitográfico. Nesse processo a PCI (Placa de Circuito Impresso) virgem é coberta por um fotolito, o *layout* invertido (as trilhas ficam brancas e o restante fica preto) é impresso em uma transparência, colocado sobre a placa e exposto à luz. O resultado é que apenas as partes não protegidas (as ilhas e trilhas), ficam marcadas na placa. Depois dessa etapa, a placa passa pelo processo de corrosão através de solução de percloreto de ferro, no qual a camada de cobre que não se encontrar protegido é removida.

As placas confeccionadas também apresentam máscara de solda, a qual protege o cobre exposto contra oxidação e ajuda no processo de soldagem. Por fim, é realizada a serigrafia, legenda dos componentes, utilizando o processo serigráfico. Essa etapa de confecção das plataformas de *hardware* foi terceirizada. A Figura 5.1 mostra a face superior da placa e a Figura 5.2 mostra a face inferior. Nessa última figura, pode ser observada a máscara de solda (a camada verde).

Já a montagem dos componentes não foi terceirizada e foi realizada manualmente. O resultado final da plataforma de *hardware* é exibido na Figura 5.3, na qual também se encontram as indicações dos principais componentes. Um detalhe interessante é que foi constatado, em testes preliminares, que o CI responsável por limitar a corrente fornecida pelo painel solar necessitaria de um dissipador devido à alta temperatura atingida por esse elemento.

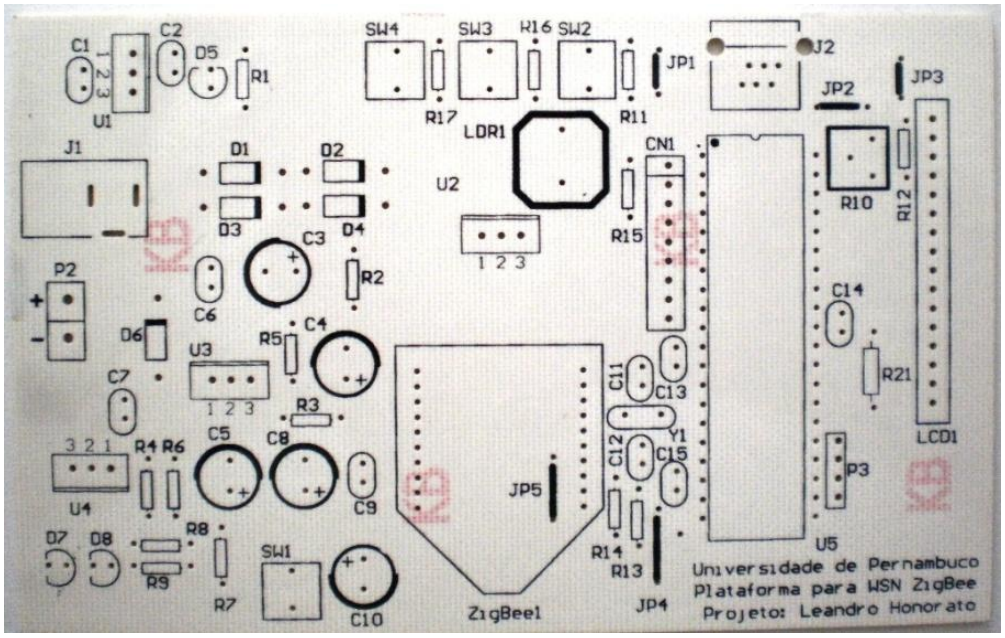


Figura 5.1 – Visão superior da PCI confeccionada.

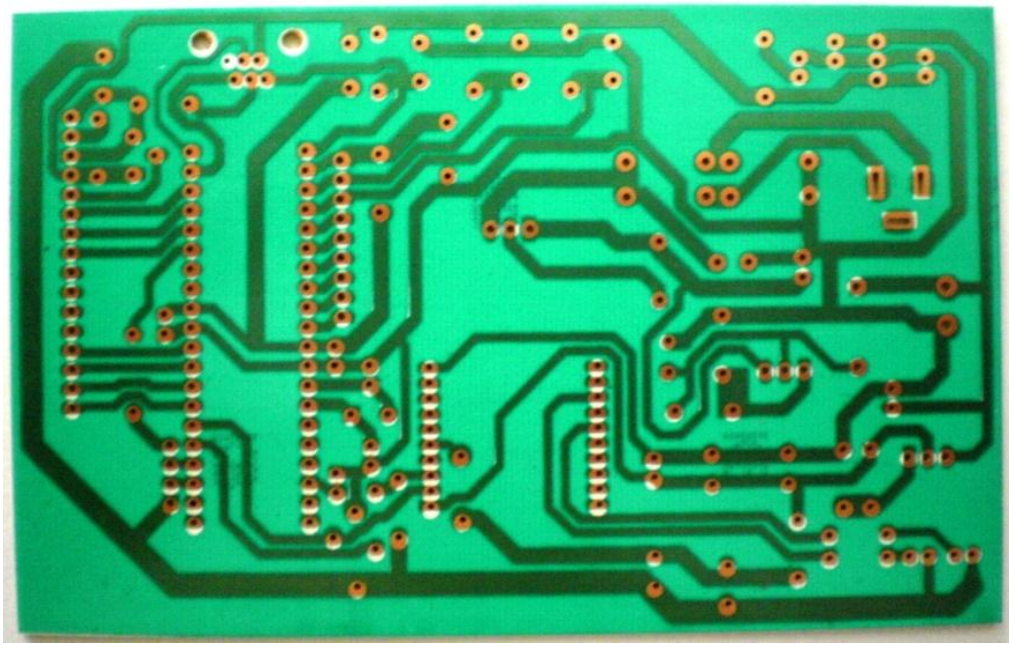


Figura 5.2 – Visão inferior da PCI confeccionada.

Confeccionadas as plataformas de hardware foram realizados testes do firmware desenvolvido. O primeiro teste foi a leitura do canal ADC no qual se encontra o sinal gerado pelo LDR. Esse teste consistiu em realizar a leitura do conversor analógico digital e imprimir o valor lido no display LCD. A Figura 5.4 exibe o display LCD com dois valores de leitura: o primeiro (832) é o valor proveniente do divisor de tensão com o LDR – esse valor varia de acordo com a luminosidade do

ambiente; o segundo valor (1023) é proveniente de do sinal AN1 (Figura 4.5) o qual assume os valores de zero ou cinco volts. Quando o sinal AN1 é zero volt a leitura do canal ADC correspondente também é zero, já quando o sinal AN1 é cinco volts a leitura do canal ADC é 1023 (capacidade máxima do conversor ADC, já que a resolução é de 10 bits). A Tabela 5.1 apresenta as mensagens que são impressas no display LCD durante o funcionamento da plataforma de *hardware* e os seus respectivos significados.

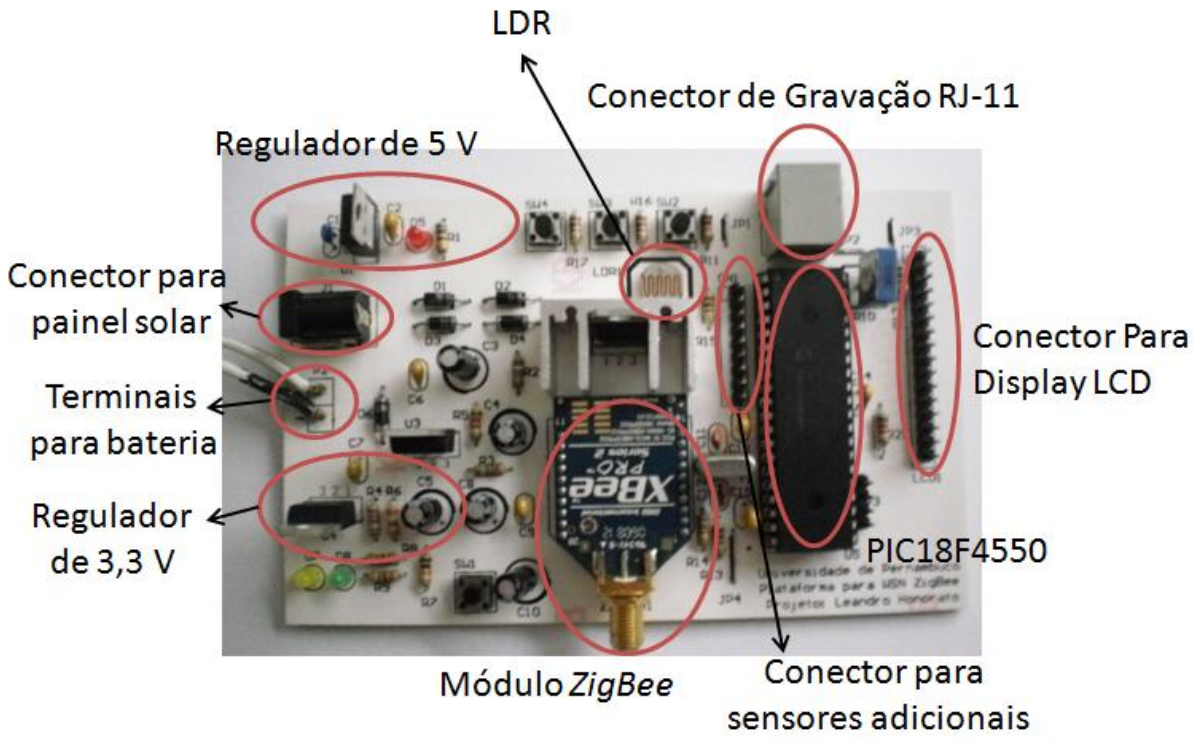


Figura 5.3 – Plataforma de *hardware* montada

Figura 5.4 – Display LCD com valor de leitura do canal ADC.

Tabela 5.1 – Mensagens exibidas pelo display LCD para visualização de estado

Mensagem	Significado
Inicializando*.*	O firmware está na Etapa de Inicialização
Modo Comando...	O módulo *ZigBee* será configurado para entrar no Modo Comando
Resp Recebida *xxxx*	*xxxx* é resposta obtida após o envio de um comando para o módulo *ZigBee*
Modo transparente...	O módulo *ZigBee* será configurado para entrar no Modo Transparente
Setando Destino...	O módulo *ZigBee* vai buscar o nó concentrador e configurar o endereço destino da transmissão como sendo esse nó
Destino Setado...	Informa que o nó concentrador já foi buscado e a transmissão está endereçada a ele
VOU ENVIAR *x*...	Informa que recebeu um pacote de requisição de leitura e vai realizar o envio do valor lido pelo do canal ADC *x*.
ENVIEI...	Informa que já realizou o envio do pacote de resposta da requisição de leitura

Foram realizados testes de alimentação. Na Figura 5.5 é exibida a plataforma completa de *hardware* sendo alimentada através de bateria, a qual é recarregada através do painel solar. Utilizando um multímetro foi constatado que a corrente máxima fornecida pelo painel solar é de 300 mA, o que garante uma das condições para que o painel opere no melhor ponto de potência (a outra condição é tensão de saída de 17,6 V o que depende da intensidade do sol).

Também foi realizado um teste de consumo, no qual a plataforma foi alimentada apenas com a bateria, cuja tensão nominal é de 6 V, e foi medido com multímetro qual a corrente drenada da bateria para funcionamento da plataforma. Foi constatado que o consumo do *hardware* desenvolvido oscila entre 70 mA e 80 mA. Utilizando o valor de corrente máximo medido, temos que a potência consumida é

$$P_{consumida} = 6 \,.\, 80 \,.\, 10^{-3} = 480\, mW, \qquad (6.1)$$

a qual é ainda menor do que a estimada na seção 4.3.3. Entretanto, como o multimetro não apresenta taxa de atualização suficiente para detectar picos de corrente, os quais ocorrem durante a transmissão, essa medida não possui uma grande precisão e serve apenas como uma indicativa preliminar da potência consumida.

Figura 5.5 – Plataforma completa de *hardware* sendo alimentada com bateria recarregada com painel solar.

Dado que o bloco de alimentação e a leitura dos canais ADC apresentaram resultados esperados, o passo seguinte foi o teste da implementação do protocolo desenvolvido. Para isso, foi estabelecido um ambiente de teste que é composto de dois nós ligados a adaptadores USB-*XBee*, exibidos na Figura 5.6, um com a função de coordenador e outro sendo o concentrador da rede, e a plataforma de *hardware* alimentada conforme mostrado na Figura 5.5. O *software* X-CTU, disponibilizado pela Digi, foi utilizado para enviar requisições e receber as respostas. Quando um adaptador USB-*XBee* é conectado à USB de um computador, é criada uma porta serial virtual (COM) através da qual é possível enviar comandos para o módulo *ZigBee* e estabelecer comunicação com outros dispositivos *ZigBee*. Na Figura 5.7 é possível observar a aba Terminal do X-CTU, na qual ocorre a interação direta com o módulo *XBee*. Nessa figura, os textos em azul são as requisições enviadas para a plataforma construída e os textos em vermelho são as respostas das requisições. Podem-se observar duas requisições de leitura de sensor (o sensor disposto no canal um e dois, respectivamente) e alguns comandos remotos (comandos remotos

para ler o nome do nó, ler o canal no qual o módulo está operando, obter o endereço de rede e mudar o parâmetro NI do nó) e as respectivas respostas dessas requisições.

Figura 5.6 – Módulos *XBee* conectados ao adaptador USB-*XBee*.

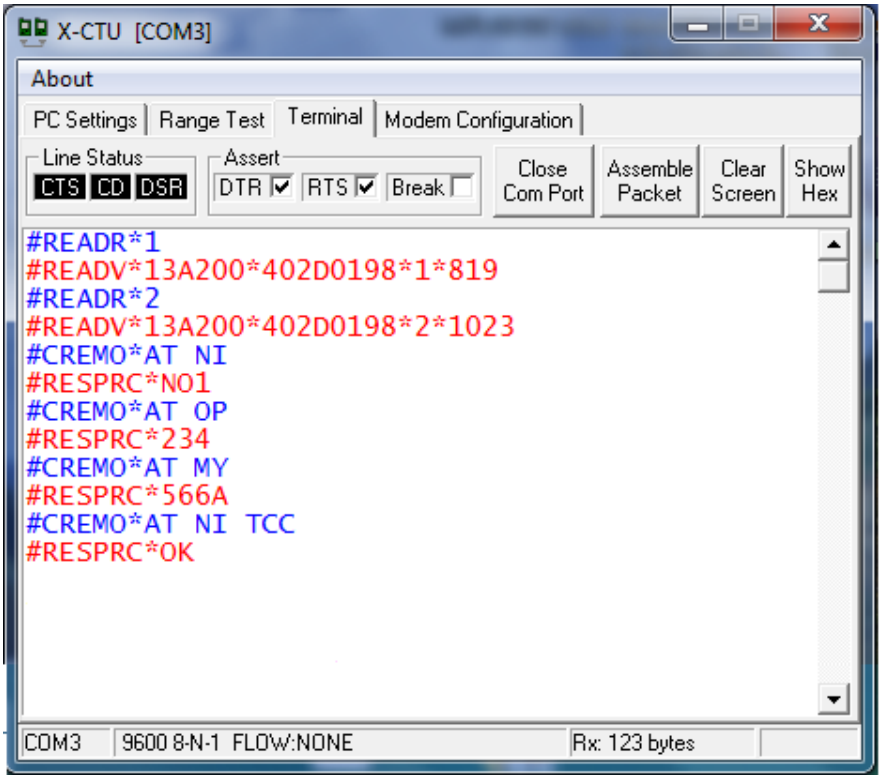


Figura 5.7 – *Software* X-CTU utilizado para testar a implementação do protocolo no *firmware*.

5.2 *Software* Desenvolvido

O *software* foi desenvolvido conforme o projeto desenvolvido no capítulo anterior. A primeira informação requerida é a COM na qual está conectado o

adaptador USB-*XBee*. A interface gráfica que solicita essa informação é exibida na Figura 5.8. Essa interface lista todas as COMs disponíveis no computador.

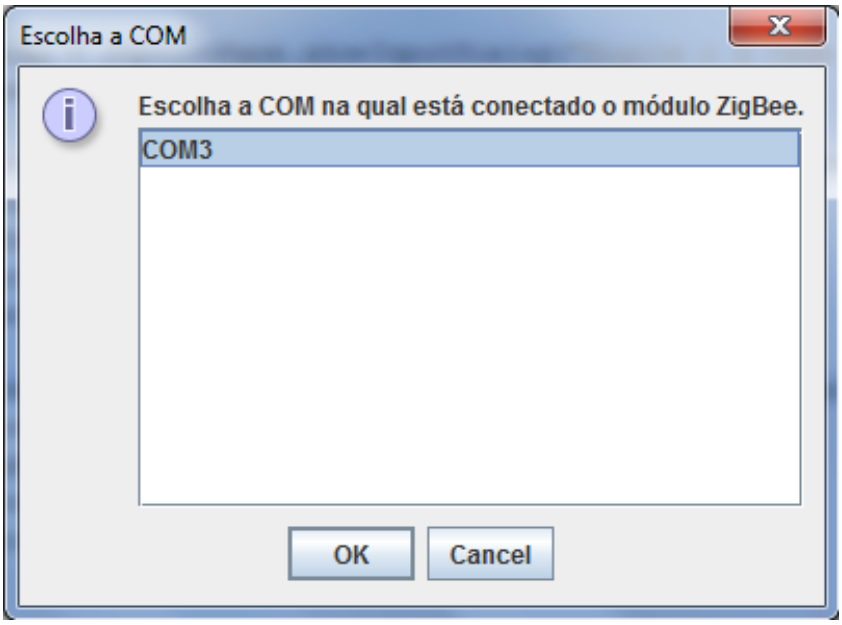
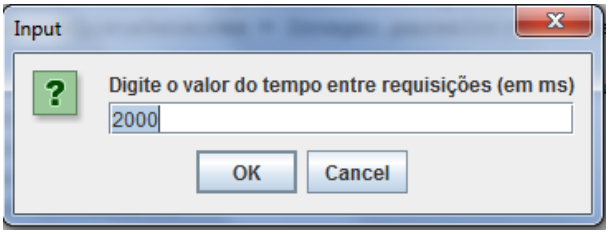
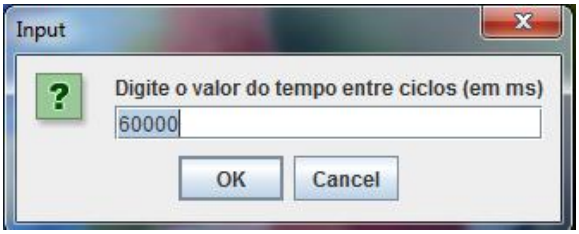


Figura 5.8 – Interface gráfica para escolha da COM.

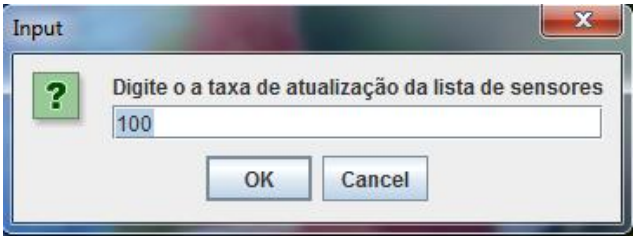
Após a escolha da COM, os três parâmetros descritos na seção 4.4.2 são requisitados através das janelas de entrada exibidas na Figura 5.9. Para cada parâmetro solicitado é sugerido um valor padrão, o qual também é apresentado nessa figura.



(a)



(b)



(c)

Figura 5.9 – Janelas de entrada de parâmetros.Em (a) o parâmetro intervalo entre requisições, em (b) o intervalo entre ciclos e em (c) a taxa de atualização da lista de sensores.

Realizadas as configurações descritas anteriormente, o sistema começa a execução da *thread* de envio de requisições e da *thread* de recepção das respostas, que operam conforme o descrito na seção 4.4.2. Na Figura 5.10 pode ser observada a tabela do banco de dados referente às leituras após a execução do *software*. Na tabela de leitura dos sensores mostrados nessa figura, estão presentes o endereço do nó que enviou a leitura (dividido em duas partes – fk_SH e fk_SL), o valor da leitura, a data e hora que a requisição foi recebida e o canal ADC lido (o valor 1 é a medida obtida pelo LDR e o valor 2 é o sinal AN1).

Por fim, foi desenvolvida uma interface gráfica de demonstração, a qual é exibida na Figura 5.11. Essa interface é atualizada sempre que é recebida uma resposta à requisição de leitura do sinal do LDR e o nó que enviou a resposta possui o endereço igual ao presente na tabela da Figura 5.10. É importante ressaltar que a taxa de atualização da interface de demonstração depende dos parâmetros de intervalo entre requisições e intervalo entre ciclos.

fk_SH	fk_SL	valorLeituraS1	dataHora	chanel_ADC
13A200	402D0198	823	2011-05-07 19:03:20	0000000001
13A200	402D0198	1023	2011-05-07 19:03:24	0000000002
13A200	402D0198	831	2011-05-07 19:04:18	0000000001
13A200	402D0198	1023	2011-05-07 19:04:21	0000000002
13A200	402D0198	828	2011-05-07 19:05:14	0000000001
13A200	402D0198	1023	2011-05-07 19:05:17	0000000002
13A200	402D0198	838	2011-05-07 19:06:11	0000000001
13A200	402D0198	1023	2011-05-07 19:06:15	0000000002
13A200	402D0198	901	2011-05-07 19:07:07	0000000001
13A200	402D0198	1023	2011-05-07 19:07:11	0000000002
13A200	402D0198	899	2011-05-07 19:08:06	0000000001
13A200	402D0198	1023	2011-05-07 19:08:10	0000000002
13A200	402D0198	900	2011-05-07 19:09:02	0000000001
13A200	402D0198	1023	2011-05-07 19:09:06	0000000002
13A200	402D0198	898	2011-05-07 19:09:58	0000000001
13A200	402D0198	1023	2011-05-07 19:10:02	0000000002
13A200	402D0198	898	2011-05-07 19:10:56	0000000001
13A200	402D0198	1023	2011-05-07 19:10:59	0000000002

Figura 5.10 – Tabela de leituras do BD com entradas inseridas pelo *software* desenvolvido.

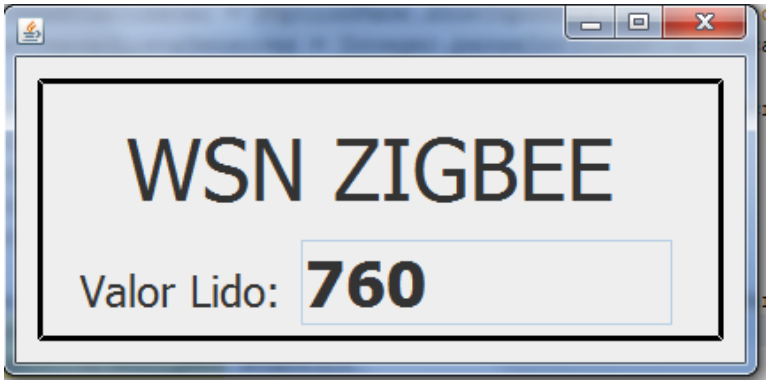


Figura 5.11 – Interface gráfica para demonstração.

Capítulo 6

Conclusão e Trabalhos Futuros

Esse trabalho propôs uma WSN utilizando a tecnologia *ZigBee* para aplicações diversas, a qual foi desenvolvida com sucesso. As redes de sensores sem fio podem ser utilizadas em numerosas aplicações (e.g. medicina, automação residencial, indústria, agricultura) e devido a natureza dessas aplicações, pode-se prever que em breve as WSN serão uma parte integral das nossas vidas. Já o *ZigBee* é um padrão amplamente utilizado nas redes de sensores já que proporciona baixo consumo, topologias dinâmicas e garante interoperabilidade.

O projeto concebeu três itens: plataforma de *hardware*, *firmware* e *software*. Das plataformas de *hardware* foi desenvolvido o esquema elétrico e *layout*. Nessas plataformas foram utilizados o microcontrolador PIC18F4550, o módulo *ZigBee Xbee-PRO ZNet 2.5 OEM* (também conhecido como *Xbee Series 2*) e a alimentação é realizada através de uma bateria que é recarregada por painel solar. O *firmware* foi desenvolvido tendo como alvo o microcontrolador referido, utilizando a linguagem C. O *software*, desenvolvido em Java, tem o objetivo de adquirir os dados coletados pelos sensores e armazená-los em um banco de dados. Com a finalidade de obter os dados coletados pelos sensores também foi desenvolvido um protocolo de comunicação.

O sistema desenvolvido obteve resultados satisfatórios. As plataformas foram confeccionadas com sucesso, a potência dimensionada para o painel solar foi adequada e suficiente para carregar a bateria e a aplicação de requisição dos dados coletados pelos sensores funcionou como esperado.

A WSN desenvolvida está pronta para ser utilizada em aplicações que necessitem de monitoramento remoto, diminuindo o tempo de desenvolvimento, tendo em vista que elas precisam apenas selecionar os sensores e analisar os dados.

6.1 Trabalhos Futuros

O primeiro trabalho sugerido é a obtenção de medidas de avaliação. Podem ser medidas taxa de perda de pacotes, tempo de permanência da rede e latência de processamento. Essas métricas podem ser utilizadas para comparar o sistema desenvolvido com outros sistemas de *benchmark*. Outra medida importante é o consumo da plataforma, obtido através de instrumentos mais precisos, inclusive, substituindo o módulo *ZigBee* por outros de menor potência.

Também pode ser inserido o controle de atuadores, remotamente, para que o sistema também seja capaz de tomar medidas caso alguma situação requeira alguma ação. Isso implica em adição de tipos de pacotes no protocolo desenvolvido.

Outra melhoria é a diminuição do tamanho do *hardware* desenvolvido. A plataforma de *hardware* apresentada foi desenvolvida tendo como restrições os componentes obtidos e os limites do processo de fabricação utilizado. Fazendo uso de componentes SMD (*Surface Mounted Components*) e um processo de fabricação que permita placas dupla face e trilhas mais finas, o tamanho da plataforma pode ser diminuído consideravelmente.

Por fim, a WSN desenvolvida pode ser aplicada para transmissão de dados coletados pela plataforma de monitoramento de corrente de fuga nas linhas de transmissão de alta tensão, descrito em [5], principalmente no ambiente das subestações.

Bibliografia

[1] Akyildiz, I.F.; Weilian Su; Sankarasubramaniam, Y.; Cayirci, E.; "***A* survey on sensor networks**," *Communications Magazine, IEEE* , vol.40, no.8, pp. 102- 114, Agosto 2002

[2] Guozhu Wang; Junguo Zhang; Wenbin Li; Dongxu Cui; Ye Jing; "**A forest fire monitoring system based on GPRS and ZigBee wireless sensor network**," *Industrial Electronics and Applications (ICIEA), 2010 the 5th IEEE Conference on* , vol., no., pp.1859-1862, 15-17 June 2010

[3] ZHAO, Feng; GUIBAS, Leonidas. **Wireless Sensor Networks: An Information Processing Approach.** Morgan Kaufmann, 2004. 376 p.

[4] BURATTI, Chiara et al. **Sensor Networks with IEEE 802.15.4:** Systems Distributed Processing, MAC, and Connectivity. Springer, 2011 269 p.

[5] de Lima, R.A.; Fontana, E.; Martins-Filho, J.F.; Prata, T.L.; Cavalcanti, G.O.; Lima, R.B.; Oliveira, S.C.; Cavalcanti, F.J.M.M. **Satellite telemetry system for pollution detection on insulator strings of high-voltage transmission lines.** *IMOC, 2009 SBMO/IEEE MTT-S International* , vol., no., pp.574-577, 3-6 Nov. 2009

[6] Hoblos, G.; Staroswiecki, M.; Aitouche. "**Optimal design of fault tolerant sensor networks**," *Control Applications, 2000. Proceedings of the 2000 IEEE International Conference on* , vol., no., pp.467-472, 2000.

[7] DARGIE, Waltenegus; POELLABAUER, Christian. **Fundamentals of Wereless Sesnor Networks:** Therory and Pratice. Wiley, 2010.

[8] GAVRILOVSKA, Liljana et al. **Application and Multidisciplinary Aspects of Wireless Sensor Networks:** Concepts, Integration, and Case Studies. London: Springer, 2011. 293 p.

[9] **Homepage ZigBee Alliance.** Disponível em: <http://www.zigbee.org/>. Acesso em: 26 maio 2011.

[10] LABIOD, Houda; AFIFI, Hossam; SANTIS, Costantino De. **WiFi Bluetooth ZigBee and WiMax.** Paris: Springer, 2007. 327 p.

[11] GISLASON, Drew et al. **ZigBee Wireless Networking.** London: Newnes, 2008. 427 p.

[12] FARAHANI, Shahin et al. **ZigBee Wireless Networks and Tranceiver.** London: Newnes, 2008. 364 p.

[13] **MANUAL do XBee / Xbee PRO ZNet 2.5.** Disponível em: < http://ftp1.digi.com/support/documentation/90000866_A.pdf>. Acesso em 26 maio 2011.

[14] **Digi International - Making Wireless M2M Easy.** Disponível em: <http://www.digi.com/>. Acesso em: 26 maio 2011.

[15]FALUDI, Robert. **Building Wireless Sensor Networks Download: A pratical Guide to the ZigBee Mesh Networking Protocol**. O'Reilly, 2010. 321 p.

[16] **ALTIUM - Next generation electronics design** Disponível em: <http://www.altium.com/>. Acesso em: 26 maio 2011.

[17] **DATASHEET do PIC18F4550 Disponível em**: <ww1.microchip.com/downloads/en/devicedoc/39632d.pdf>. Acesso em: 26 maio 2011.

[18] **MPLAB ICD 3 In-Circuit Debugger** Disponível em: <www.microchip.com/icd3/>. Acesso em: 26 maio 2011.

[19] **MANUAL do painel solar Silício Cristalino Sunlabs 5W** Disponível em: <http://www.sunlab.com.br/manuais/Datasheet_Painel_S-5.pdf>. Acesso em: 26 maio 2011.

[20] **DATASHEET da bateria Unipower UP645 Disponível em**: <http://www.unicoba.com.br/uploads/Baterias/Datasheet/6V/DATASHEET-UP645_300709.pdf>. Acesso em: 26 maio 2011.

[21]**DATASHEET do CI LM317 Disponível em**: <http://www.national.com/ds/LM/LM117.pdf>. Acesso em: 26 maio 2011.

[22]**CCS - Custom Compuer Services: C Compiler**. Disponível em: <http://www.ccsinfo.com/content.php?page=compilers>. Acesso em: 26 maio 2011.

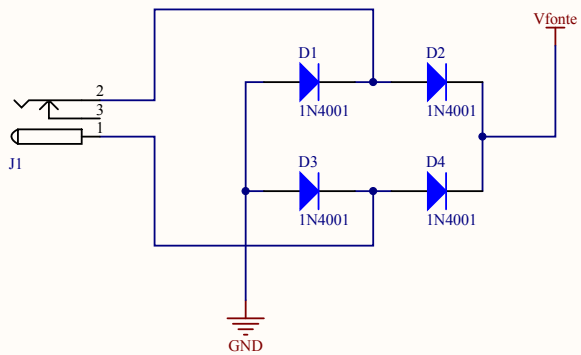
Apêndice A

Projeto de Hardware

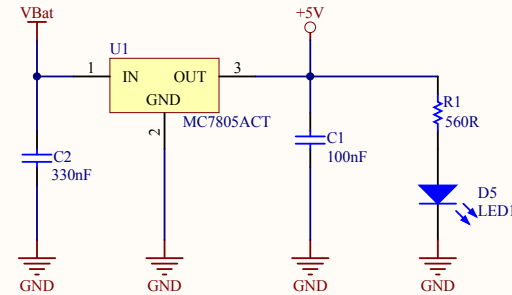
Diagrama esquemático e *layout* nas páginas seguintes.

Alimentação por fonte ou pelo painel solar

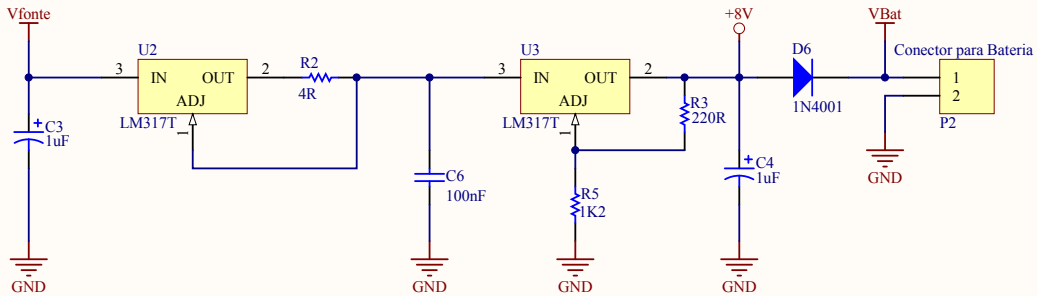
P4 - Conector para fonte ou para Painel Solar



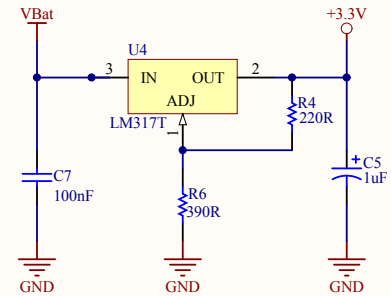
Regulador de Tensão para +5V



Circuito Carregador de Bateria com corrente limitada em 300 mA

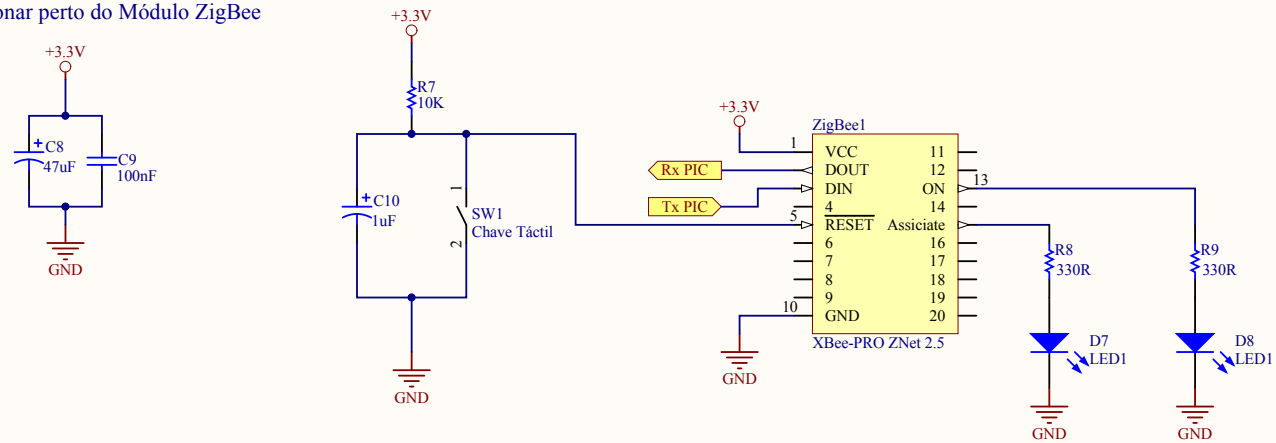


Regulador de Tensão para +3,3V

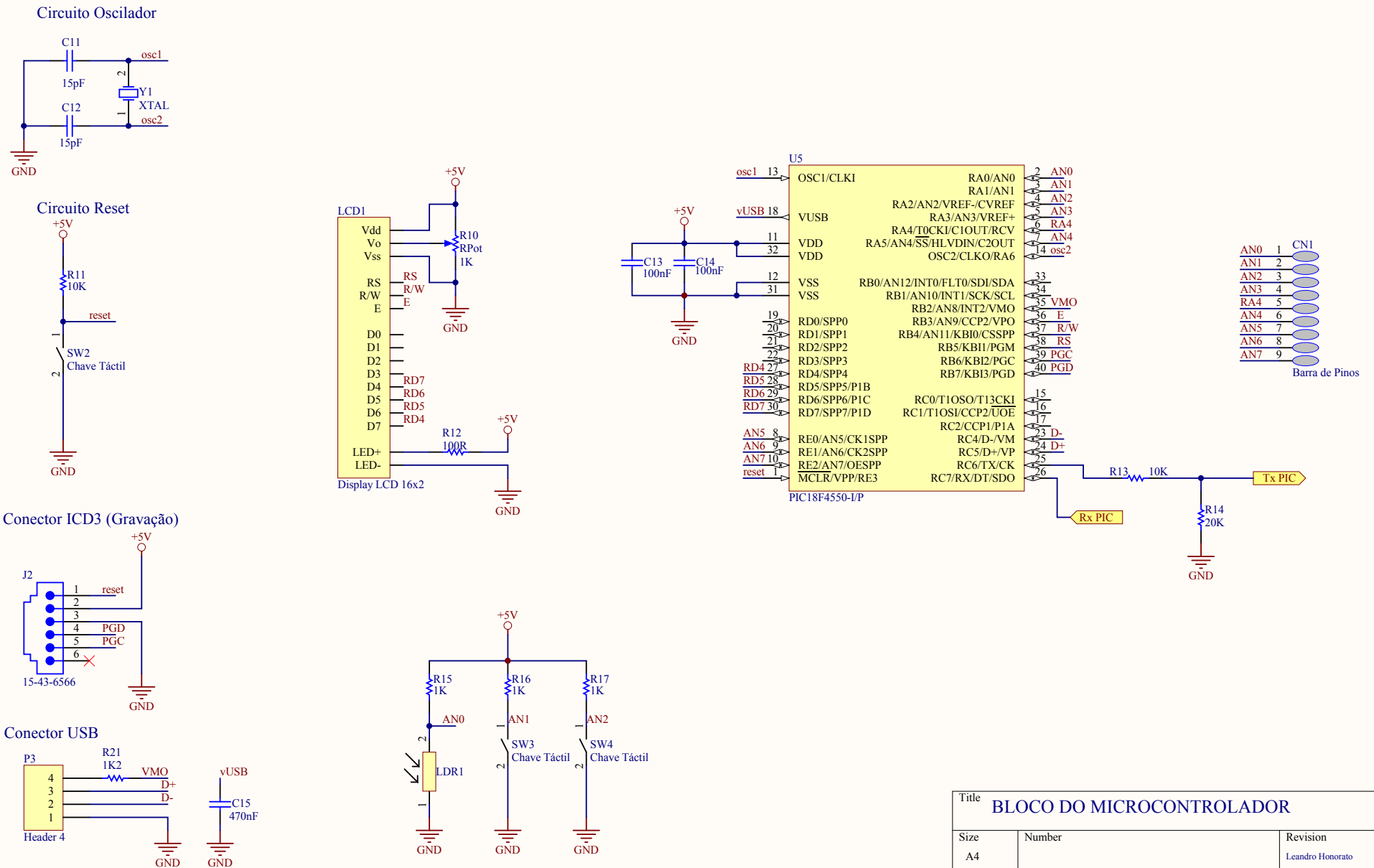


Title: BLOCO DE ALIMENTAÇÃO		
Size: A4	Number:	Revision: Leandro Honorato
Date: 6/5/2011		Sheet 1 of 4
File: C:\Users\..\Alimentação.SchDoc		Drawn By: Leandro Honorato

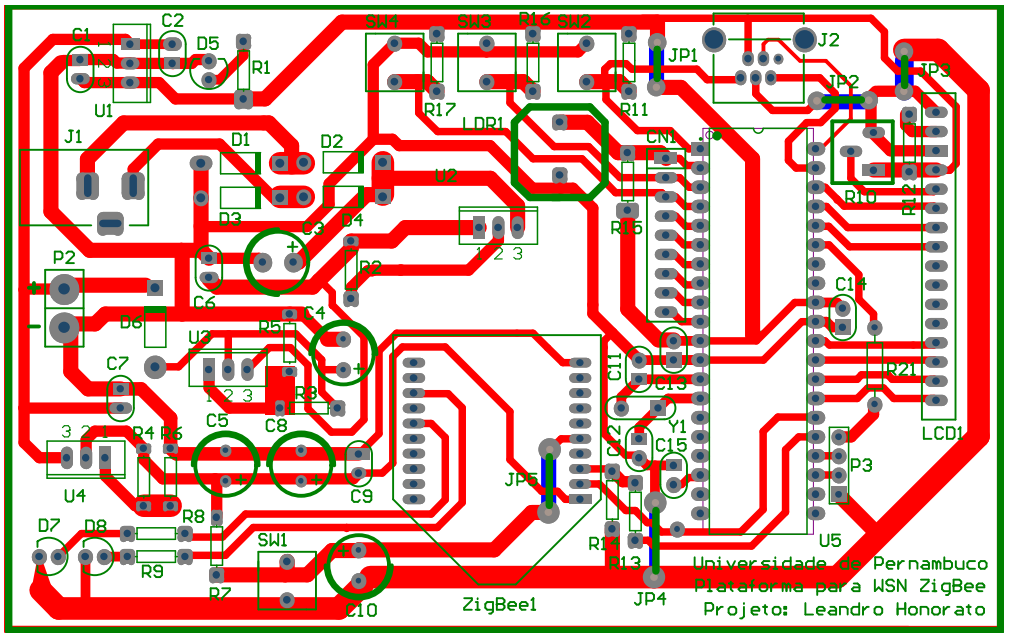
Posicionar perto do Módulo ZigBee



Title	BLOCO DO MÓDULO ZIGBEE	
Size: A4	Number:	Revision: Leandro Honorato
Date: 6/5/2011		Sheet 2 of 4
File: C:\Users\..\MóduloZigBee.SchDoc		Drawn By: Leandro Honorato



Title	BLOCO DO MICROCONTROLADOR	
Size: A4	Number:	Revision: Leandro Honorato
Date: 6/5/2011		Sheet 3 of 4
File: C:\Users\..\PlataformaPIC.SchDoc		Drawn By: Leandro Honorato


C1
C2
D5
R1
U1
J1
D1
D2
D3
D4
C3
R2
P2
C6
D6
C4
U3
R5
C7
R3
C5
C8
R4
R6
U4
C9
R8
D7
D8
R9
R7
SW1
C10
ZigBee1
JP5
JP4
R13
R14
SW4
SW3
R16
SW2
R17
LDR1
U2
R15
JP1
R11
CN1
J2
JP2
JP3
R10
C14
R21
C11
C13
C12
C15
Y1
P3
U5
LCD1
Universidade de Pernambuco
Plataforma para WSN ZigBee
Projeto: Leandro Honorato

Apêndice B

Documentação do *Firmware*

Funções para ADC:

- void init_ADC( ) – inicializa o conversor analógico digital do microcontrolador
- void ler_canal_AD_para_puint8 (int8 canal, int8 * p_valorChar) – lê o canal analógico passado no parâmetro ‘canal’ e retorna o valor da leitura no formato de string em ‘p_valorChar’, o qual deve estar alocado.
- void ler_canal_AD_para_int32 (int8 canal, int32 * p_valor) – semelhante a anterior, mas o valor lido é retornado numericamente em p_valor.

Funções para LCD:

- void lcd_init( ) – executa toda a sequência de passos necessários para configurar o inicializar o display LCD.
- void lcd_gotoxy(BYTE x, BYTE y) – desloca o cursos para a posição (x,y) passada nos argumentos.
- void lcd_putc(char letter) - escreve o caractere ‘letter’ na posição tual do cursor.
- void clearLine1( ) – limpa a primeira linha do display LCD.
- void clearLine2( ) – limpa a segunda linha do display LCD.
- void clearLCD( ) – limpa todo o display LCD.

As funções para manipulação do módulo *ZigBee* encontram-se documentadas no próprio código encontrado no Apêndice C.

Apêndice C

Código do *Firmware*

- Biblioteca de manipulação do módulo *ZigBee*:

```
/*
*  Biblioteca de Funções para o Módulo ZigBee XBee
*/
#include <string.h>
#include <stdlib.h>

#define MAX_BUFFER 50 //Tamanho do Buffer de recepção Serial
#define MAX_PACOTE 30 //Tamanho do Array onde as respostas são armazenadas

int8 buffer_resposta_zigbee [MAX_BUFFER]; //Buffer que recebe os dados e comandos pela serial
int pacoteRecebido [MAX_PACOTE]; //array para armazenar os comandos recebidos na medida que
forem retirados do buffer

int contador_escrita; //Variável para saber onde escrever no Buffer de recepção Serial
(resposta_zigbee)
int contador_leitura; //Variável para saber onde ler no Buffer de recepção Serial
(resposta_zigbee)

char comandoREADR[7];
char comandoSET_NAME[7];
char comandoREMOTO[7];

//enum com os possíveis pacotes
typedef enum _TipoPacote
{
   READR,
   SET_NAME,
   COMANDO_REMOTO,
   PACOTE_NAO_IDENTIFICADO
}TipoPacote;

//Estrutura que define um pacote
typedef struct _pacote
{
   TipoPacote tipoPacote;
   char informacao[15];
} Pacote;

/**
 * Esta função deve ser chamada sempre que chegar dados pela serial
 * Armazena os dados que estão chegando no array resposta_zigbee
 */
void interrupcao_serial()
{
   int8 temp;
   while( kbhit() )
   {
         temp = getc();
         buffer_resposta_zigbee[contador_escrita] = temp;
         contador_escrita = (contador_escrita+1)%MAX_BUFFER; //Incremento circular: quando
chegar no final do array, volta a escrever no início
   }
}

/**
 * Esta função reseta o buffer pacoteRecebido
 */
void zerarBufferPacoteRecebido()
{
   int contador;
   for(contador = 0; contador < MAX_PACOTE; contador++)
   {
```

```
      pacoteRecebido[contador] = 0;
   }
}

/**
 * Esta função verifica se chegou pacote
 * se chegou pacote o retorno é 1 caso contrário, o retorno é 0
 */
int1 chegouPacote()
{
   int8 i;
   int8 temp_pacote[MAX_PACOTE];
   int8 contadorPacote = 0;


   for(i = contador_leitura; buffer_resposta_zigbee[i] != 0 ; i=(i+1)%MAX_BUFFER)
   {
      if( buffer_resposta_zigbee[i] == '\r' )
      {
         buffer_resposta_zigbee[i] = 0; //Finaliza a string
         temp_pacote[contadorPacote] = 0;

         contador_leitura = (i + 1)%MAX_BUFFER;

         zerarBufferPacoteRecebido();

         strcopy(pacoteRecebido, temp_pacote);


         return 1;
      }
      temp_pacote[contadorPacote] = buffer_resposta_zigbee[i];
      contadorPacote++;
   }
   return 0;
}

/**
 * Função para inicializar as variáveis de controle globais.
 *
 */
void inicializar_variaveis(){
   contador_leitura = 0;
   contador_escrita = 0;

   sprintf(comandoREADR, "#READR");
   sprintf(comandoSET_NAME, "#SETNA");
   sprintf(comandoREMOTO, "#CREMO");
}

/**
 * Função para limpar o buffer de recepção serial (resposta_zigbee)
 *
 */
void limpar_buffer(){
   int contador;
   for(contador = 0; contador < MAX_BUFFER; contador++)
   {
      buffer_resposta_zigbee[contador] = 0;
   }
   contador_leitura = 0;
   contador_escrita = 0;
}

/**
 * Função Restore Defaults
 * Restaura os parâmetros do módulo para as cofigurações de fábrica. Não reseta o parâmetro
ID.
 *
 */
void zigbee_restoreDefault()
{
   printf("AT RE"); //Envia o comando de reset
   putc(0x0d); //CR
}
```

```
/**
 * Função Software Reset
 * Reseta o módulo. O módulo responde com um "OK" e realiza o reset por volta de 2 segundos
depois. A camada de rede também pode ser resetada em alguns casos.
 *
 */
void zigbee_softwareReset()
{
   printf("AT FR");
   putc(0x0d);
}

/**
 * Função Network Reset
 * Reseta a camada de rede de um ou mais nós dentro de uma PAN (Personal Area Network). O
módulo responde imediantamente com um "OK" ou "ERROR".
 * Com o reset da camada de rede todos as configurações de rede e informações de roteamento
são perdidas.
 * Se parametro = "0" - O reset na camada de rede acontece apenas no módulo que está
executando a função
 *    parametro = "1" - Envia uma transmissão broadcast para resetar os parametros da camada
de rede em todos os nós da PAN.
 */
void zigbee_NetworkReset(char *parametro)
{
   printf("AT NR %s", parametro);
   putc(0x0d);
}

/**
 * Função Write
 * Escreve os parametros em uma memória não volátil (depois de um reset os parametros
continuam os memsmos).
 * Este comando responde com um "OK\r"
 *
 */
void zigbee_Write()
{
   printf("AT WR");
   putc(0x0d);
}

/**
 * Função para ler os 32 bits mais altos do endereço de destino
 * Este comando responde com os 32 bits mais altos do endereço de destino
 *
 */
void zigbee_getDestinationAddressHigh()
{
   printf("AT DH");
   putc(0x0d);
}

/**
 * Função para setar os 32 bits mais altos do endereço de destino
 * Este comando responde com "OK" ou "ERROR".
 *
 */
void zigbee_setDestinationAddressHigh(char* enderecoAlto)
{
   printf("AT DH %s", enderecoAlto);
   putc(0x0d);
}

/**
 * Função para ler os 32 bits mais baixos do endereço de destino
 * Este comando responde com os 32 bits mais baixos do endereço de destino
 *
 */
void zigbee_getDestinationAddressLow()
{
   printf("AT DL");
   putc(0x0d);
}
```

```
/**
 * Função para setar os 32 bits mais baixos do endereço de destino
 * Este comando responde com "OK" ou "ERROR".
 *
 */
void zigbee_setDestinationAddressLow(char* enderecoBaixo)
{
   printf("AT DL %s", enderecoBaixo);
   putc(0x0d);
}

/**
 * Função ler o endereço de rede de 16-bits
 * Este comando responde com o endereço do módulo
 */
void zigbee_MY()
{
   printf("AT MY");
   putc(0x0d);
}

/**
 * Função que retorna o endereço de rede de 16-bits do pai do nó.
 * Este comando responde com o endereço do módulo pai
 */
void zigbee_MP()
{
   printf("AT MP");
   putc(0x0d);
}

/**
 * Função que retorna a parte alta (32bits) do identificador único do módulo
 *
 */
void zigbee_SerialNumberHigh(){
   printf("AT SH");
   putc(0x0d);
}

/**
 * Função que retorna a parte baixa (32bits) do identificador único do módulo
 *
 */
void zigbee_SerialNumberLow(){
   printf("AT SL");
   putc(0x0d);
}

/**
 * Função que retorna a parte baixa (32bits) do identificador único do módulo
 *
 */
void zigbee_EnableEncryption(){
   printf("AT EE 1");
   putc(0x0d);
}

/**
 * Função que retorna a parte baixa (32bits) do identificador único do módulo
 *
 */
void zigbee_setKY(int8 *key){
   printf("AT KY %s", key);
   putc(0x0d);
}

/**
 * Função para armazenar uma string identificadora do nó. O nome apenas pode conter
caracteres ASCII imprimíveis.
 * O tamanho máximo do nome é de 20Bytes.
 *
 */
void zigbee_NodeIdentifier(char *nome){
```

```
   printf("AT NI %s", nome);
   putc(0x0d); //CR
}

/**
 * Função para ler o canal que o módulo está operando. Se o módulo não estiver associado em
nenhuma rede o retorno desta função é "0" (zero).
 *
 */
void zigbee_operatingChanel(){
   printf("AT CH");
   putc(0x0d); //CR
}

/**
 * Função para setar o endereço PAN. Quando o módulo iniciar ele vai procurar uma PAN com o
endereço setado para se associar.
 * Se pan = 0xFFFF -> o módulo entra em qualquer PAN.
 * Mudanças nesse parâmetro devem ser salvas através da função Write.
 *
 */
void zigbee_setPANID(char *pan){
   printf("AT ID %s", pan);
   putc(0x0d); //CR
}

/**
 * Função para ler o endereço PAN. Quando o módulo iniciar ele vai procurar uma PAN com o
endereço setado para se associar.
 * Se pan = 0xFFFF -> o módulo entra em qualquer PAN.
 * Mudanças nesse parâmetro devem ser salvas através da função Write.
 *
 */
void zigbee_getPANID(){
   printf("AT ID");
   putc(0x0d); //CR
}

/**
 * Função para setar o Node Discovery Timeout.
 * Seta a quantidade de tempo que o nó vai gastar descobrindo outros nós quando o comando DN
(ver mais a frente).
 * time = de 0x20 - 0xFF [x 100msec]
 */
void zigbee_setNT(char *time){
   printf("AT NT %s", time);
   putc(0x0d); //CR
}

/**
 * Função para ler o Node Discovery Timeout.
 * Seta a quantidade de tempo que o nó vai gastar descobrindo outros nós quando o comando DN
(ver mais a frente).
 */
void zigbee_getNT(){
   printf("AT NT");
   putc(0x0d); //CR
}

/**
 * Função Destination Node - Resolve um nome em um endereço físico.
 * No modo de comandos AT, quando o comando é realizado, os parâmetros de Destino (tanto a
parte alta quanto a parte baixa - SH e SL)
 * recebem o endereço do nó cujo nome é igual ao parametro nomeDestino.
 */
void zigbee_destinationNode(char *nomeDestino){
   printf("AT DN %s", nomeDestino);
   putc(0x0d); //CR
}


/**
 * Função para enviar uma resposta de requisição de leitura
 * O módulo deve estar no modo transparente
 */
```

```
void enviar_pacote_Leitura(int8* enderecoH, int8* enderecoL, int8 canalLido, int8*
valorLido){ //TESTADO
   printf( "#READV*%s*%s*%d*%s\r",enderecoH,enderecoL,(canalLido+1), valorLido );
   //putc(0x0d); //CR
}

/**
 * Função para deixar o módulo no modo de comando
 */
void irParaModoComando()
{
   delay_ms(1000);
   printf( "+++" );
   delay_ms(1000);
}

/**
 * Função para deixar o módulo no modo transparente
 */
void irParaModoTransparente()
{
   printf("ATCN\r");
   //delay_ms(100);
}

/**
 * Função para ficar esperando um pacote
 * essa função termina quando chegar um pacote ou ocorrer um timeout.
 */
void esperaPacote()
{
   int8 i = 0;
   while(chegouPacote() != 1 && i < 200)
   {
      ;
      delay_ms(100);
      i++;
   }
}

/**
 * Função para decodificar um pacote recebido
 * essa função preenche a estrutura Pacote passada por argumento
 */
void tratarPacote(Pacote *pacoteRecebidoStruct)
{
     char pacoteRecebidoAux[10];
     int i;
     int indexInformacao = 0;

     for(i = 0; i < 10 ; i++)
     {
         pacoteRecebidoAux[i] = 0;
     }


     for(i = 0; i < 10 ; i++)
     {
         if (pacoteRecebido[i] == 0)
      {
            pacoteRecebidoStruct->tipoPacote = PACOTE_NAO_IDENTIFICADO;
            return;
         }
         else if (pacoteRecebido[i] == '*')
         {
            pacoteRecebidoAux[i] = '\0';
            i++;
            break;
         }else
         {
            pacoteRecebidoAux[i] = pacoteRecebido[i];
         }
     }

     if(strcmp(pacoteRecebidoAux, comandoREADR) == 0)
```

```
     {
         pacoteRecebidoStruct->tipoPacote = READR;
     }else if(strcmp(pacoteRecebidoAux, comandoSET_NAME) == 0)
     {
         pacoteRecebidoStruct->tipoPacote = SET_NAME;
     }else if(strcmp(pacoteRecebidoAux, comandoREMOTO) == 0)
     {
         pacoteRecebidoStruct->tipoPacote = COMANDO_REMOTO;
     }else
     {
         pacoteRecebidoStruct->tipoPacote = PACOTE_NAO_IDENTIFICADO;
         return;
     }


     for(; i < MAX_PACOTE; i++)
     {
         if (pacoteRecebido[i] == '\r' || pacoteRecebido[i] == 0)
         {
            return;
         }else
         {
            pacoteRecebidoStruct->informacao[indexInformacao] =  pacoteRecebido[i];
            indexInformacao++;
         }
     }
}
```

- Biblioteca do Display LCD:

```
//Rotinas LCD

#define enable    PIN_B3
#define rs        PIN_B5
#define rw        PIN_B4

void lcd_putc( char c);
int inverte_bits(int numero);
void lcd_gotoxy( BYTE x, BYTE y);

#define lcd_line_two 0x40

void lcd_send_nibble( BYTE n ) {


      n = inverte_bits(n);
      output_d(n);

      delay_ms(1);

      output_high(enable);
      delay_cycles(1);


      output_low(enable);
      delay_cycles(1);
}


void lcd_send_byte( BYTE address, BYTE n ) {


      output_low(rs);

      if ( address == 1 )
      {
         output_high(rs);
      }
      delay_cycles(1);

      output_low(enable);
      delay_cycles(1);
```

```
      output_low(enable);
      lcd_send_nibble(n >> 4);
      lcd_send_nibble(n & 0x0F);
}


void lcd_init() {

  output_low(rs);
  output_low(rw);
  output_d(0x00);
  delay_ms(200);


   output_d(0x00);
   output_d(0x00);

   output_high(enable);

   delay_ms(10);
   output_low(enable);
   delay_ms(10);
   output_d(0b01000000);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0b01000000);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0b00010000);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0x00);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0b11110000);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0x00);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0b10000000);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0x00);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
   delay_ms(10);

   output_d(0b01100000);
   output_high(enable);
   delay_ms(10);
   output_low(enable);
```

```
   delay_ms(10);

   lcd_gotoxy(1,1);

}


void lcd_gotoxy( BYTE x, BYTE y) {
   BYTE address;

   if(y!=1)
     address=lcd_line_two;
   else
     address=0;
   address+=x-1;
   lcd_send_byte(0,0x80|address);
}

void lcd_putc( char c) {
   switch (c) {
     case '\f'   : lcd_send_byte(0,1);
                   delay_ms(2);
                                          break;
     case '\n'   : lcd_gotoxy(1,2);       break;
     case '\b'   : lcd_send_byte(0,0x10); break;
     default     : lcd_send_byte(1,c);    break;
   }
}



int inverte_bits(int numero){
   int byte_invertido;
   int count;
   int temp = 8;
   for(count = 0; count <8; count++){
   temp--;
   if(bit_test(numero,count))
   {
      bit_set(byte_invertido,temp);
   }else
      bit_clear(byte_invertido,temp);
   }

   return byte_invertido;
}

void clearLine1()
{
   lcd_gotoxy(1,1);
   printf(lcd_putc,"                ");
}

void clearLine2()
{
   lcd_gotoxy(1,2);
   printf(lcd_putc,"                ");
}

void clearLCD()
{
   clearLine1();
   clearLine2();
}
```

- Biblioteca do Conversor ADC do PIC18F4550:

```
#include <stdlib.h>

void ler_canal_AD_para_int32 (int8 canal, int32 * p_valor)
{
   set_adc_channel(canal);  //escolhe canal
   delay_ms(50);             //espera um tempo para a tensão se estabilizar
   *p_valor = read_adc();      //inicia a conversão espera ela ser concluida e retorna o
valor convertido
}

void ler_canal_AD_para_puint8 (int8 canal, int8 * p_valorChar)
{

   int32 valorLeitura;
   set_adc_channel(canal);  //escolhe canal
   delay_ms(50);  //espera um tempo para a tensão se estabilizar
   valorLeitura = read_adc();      //inicia a conversão espera ela ser concluida e retorna o
valor convertido
   itoa(valorLeitura,10,p_valorChar);
}

void init_ADC()
{
   setup_adc(ADC_CLOCK_DIV_32);   //configura conversor AD
   setup_adc_ports(AN0_TO_AN10);   //escolhe pinos de entradas analógicas
}
```

- Arquivo principal (main):

```
//FIRMWARE TCC WSN ZIGBEE - LEANDRO HONORATO
/*
*
*
*/

#include <18F4550.h>  // inclui arquivo de bibliotecas do dispositivo
#device adc = 10      // essa diretiva precisa vir imediatamente abaixo do include do arquivo
do processador
#fuses HS, NOWDT, PUT, NOLVP, DEBUG, NOPROTECT, NOBROWNOUT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, PARITY = N, ERRORS, bits = 8, stop = 1)
//Configurações da Serial

#include "Rotinas_LCD.c"
#include "ADC_Read.c"
#include "BibliotecaZigBee.c"
#include <string.h>
#include <stdlib.h>

#define TAXA_ATUALIZACAO_DESTINO 100

////////////////////////////////////////////////////////////////////
//VARIÁVEIS GLOBAIS//
////////////////////////////////////////////////////////////////////
int8 enderecoH[10];
int8 enderecoL[10];
int8 stringOK[3];
int8 stringDestinationNodeName[5];
int8 keyPassword[10];
int8 quantidadePacotesEnviados;
Pacote pacoteStruct;
int32 valorLeituraAD;
int8 valorStringLeituraAD[10];
////////////////////////////////////////////////////////////////////
//FIM DAS VARIÁVEIS GLOBAIS//
////////////////////////////////////////////////////////////////////
//--------------------------------------------------------------------
////////////////////////////////////////////////////////////////////
//CÓDIGO DE INTERRUPÇÃO SERIAL//
```

```
////////////////////////////////////////////////////////////////////
#int_rda
void reception ()
{
   interrupcao_serial(); //Na interrupção da serial basta chamar a função da Biblioteca
ZigBee
}
////////////////////////////////////////////////////////////////////
//FIM DO CÓDIGO DE INTERRUPÇÃO SERIAL//
////////////////////////////////////////////////////////////////////
//--------------------------------------------------------------------------
////////////////////////////////////////////////////////////////////
//FUNÇÕES UTILIZADAS PELO MAIN//
////////////////////////////////////////////////////////////////////
void modoComando()
{
   do
   {
      clearLCD();
      lcd_gotoxy(1,1);
      printf(lcd_putc,"Modo comando...");
      delay_ms(1000);

      irParaModoComando();
      esperaPacote();

      clearLCD();
      lcd_gotoxy(1,1);
      printf(lcd_putc,"Resp Recebida");
      lcd_gotoxy(1,2);
      printf(lcd_putc,"%s", pacoteRecebido);
      delay_ms(1000);
   }while ( strcmp( pacoteRecebido, stringOK) != 0 );
}

void modoTransparente()
{
   clearLCD();
   lcd_gotoxy(1,1);
   printf(lcd_putc,"Modo transparente...");
   delay_ms(1000);

   irParaModoTransparente();
   esperaPacote();

   clearLCD();
   lcd_gotoxy(1,1);
   printf(lcd_putc,"Resp Recebida");
   lcd_gotoxy(1,2);
   printf(lcd_putc,"%s", pacoteRecebido);
   delay_ms(1000);
   if ( strcmp( pacoteRecebido, stringOK) != 0 )
   {
      delay_ms(2000); //se nao recebeu resposta vai para modo transparente por delay
   }
}
void setarDesstinoConcentrador()
{
  modoComando();

  do
  {
     clearLCD();
     lcd_gotoxy(1,1);
     printf(lcd_putc,"Setando Destino...");
     delay_ms(1000);

     zigbee_destinationNode(stringDestinationNodeName);
     esperaPacote();

     clearLCD();
     lcd_gotoxy(1,1);
     printf(lcd_putc,"Destino Setado...");
     lcd_gotoxy(1,2);
     printf(lcd_putc,"%s", pacoteRecebido);
```

```
      delay_ms(1000);

  }while ( strcmp( pacoteRecebido, stringOK) != 0 );

}

void enviarLeitura()
{
   int8 canal;
   canal = atoi(pacoteStruct.informacao);
   canal = canal-1;
   ler_canal_AD_para_puint8( canal, valorStringLeituraAD );
   clearLCD();
   lcd_gotoxy(1,1);
   printf(lcd_putc,"VOU ENVIAR %d...", canal);
   delay_ms(1000);

   enviar_pacote_Leitura(enderecoH, enderecoL, canal, valorStringLeituraAD);

   clearLCD();
   lcd_gotoxy(1,1);
   printf(lcd_putc,"ENVIEI...");
   delay_ms(1000);

   quantidadePacotesEnviados = (quantidadePacotesEnviados+1);
   if(quantidadePacotesEnviados == TAXA_ATUALIZACAO_DESTINO)
   {
      quantidadePacotesEnviados = 0;
      setarDesstinoConcentrador();
   }
}
void setarNomeDevice()
{
   modoComando();
   do
   {
     clearLCD();
     lcd_gotoxy(1,1);
     printf(lcd_putc,"Setando Nome...");
     delay_ms(1000);

     zigbee_NodeIdentifier(pacoteStruct.informacao);
     esperaPacote();

     clearLCD();
     lcd_gotoxy(1,1);
     printf(lcd_putc,"Nome Setado...");
     lcd_gotoxy(1,2);
     printf(lcd_putc,"%s", pacoteRecebido);
     delay_ms(1000);

  }while ( strcmp( pacoteRecebido, stringOK) != 0 );

  do
     {
        clearLCD();
        lcd_gotoxy(1,1);
        printf(lcd_putc,"Gravando dados...");
        delay_ms(1000);

        zigbee_Write();
        esperaPacote();

        clearLCD();
        lcd_gotoxy(1,1);
        printf(lcd_putc,"Resp Recebida");
        lcd_gotoxy(1,2);
        printf(lcd_putc,"%s", pacoteRecebido);
        delay_ms(1000);
     }while ( strcmp( pacoteRecebido, stringOK) != 0 );

  delay_ms(2000); // para ir para modo transparente
}
void enviarComandoRemoto()
{
```

```
   char respostaComando[15];
   modoComando();
   printf("%s",pacoteStruct.informacao);
   putc(0x0d);//envia o '/r'
   esperaPacote();
   strcpy(respostaComando, pacoteRecebido);
   modoTransparente();
   printf("#RESPRC*%s",respostaComando);
}

void imprimirLeiturasLCD()
{
   clearLCD();
   lcd_gotoxy(1,1);
   printf(lcd_putc,"LEITURAs");
}

////////////////////////////////////////////////////////////////////
//FIM DAS FUNÇÕES UTILIZADAS PELO MAIN//
////////////////////////////////////////////////////////////////////
//--------------------------------------------------------------------
////////////////////////////////////////////////////////////////////
//MAIN//
////////////////////////////////////////////////////////////////////
void main(void){ //while(1) ;}
  //--------------INICIALIZAÇÃO-------------------//
  quantidadePacotesEnviados = 0;

  enable_interrupts(GLOBAL); //Habilita interrupção da serial
  enable_interrupts(int_rda);

  inicializar_variaveis(); //Inicializa variáveis de controle da recepção serial
  limpar_buffer(); //Limpa o Buffer de recepção serial

  lcd_init();
  clearLCD();
  lcd_gotoxy(1,1);
  printf(lcd_putc,"Inicializando*.*");
  delay_ms(5000); //espera a inicialização do modulo
  lcd_gotoxy(1,1);

  sprintf(stringOK, "OK");
  sprintf(stringDestinationNodeName, "SINK");
  sprintf(keyPassword, "senha");
  //--------------FIM DA INICIALIZAÇÃO-------------------//

  modoComando();

  //limpar_buffer();
  zigbee_SerialNumberHigh();
  delay_ms(500);
  esperaPacote();
  strcopy( enderecoH, pacoteRecebido );

  //limpar_buffer();
  zigbee_SerialNumberLow();
  delay_ms(500);
  esperaPacote();
  strcopy( enderecoL, pacoteRecebido );

  clearLCD();
  lcd_gotoxy(1,1);
  printf(lcd_putc,"%s", enderecoH);
  lcd_gotoxy(1,2);
  printf(lcd_putc,"%s", enderecoL);
  delay_ms(1000);

  modoTransparente( );

  setarDesstinoConcentrador();

  init_ADC();

  clearLCD();
  lcd_gotoxy(1,1);
```

```
  printf(lcd_putc,"LEITURAs");
  delay_ms(1000);

  while(1)
  {
      clearLine2();
      ler_canal_AD_para_puint8 (0, valorStringLeituraAD);
      lcd_gotoxy(1,2);
      printf(lcd_putc,"%s -", valorStringLeituraAD);
      ler_canal_AD_para_int32 (1, &valorLeituraAD);
      printf(lcd_putc,"- %Lu", valorLeituraAD);

      if (chegouPacote() == 1)
      {
         tratarPacote(&pacoteStruct);
         switch (pacoteStruct.tipoPacote)
         {
            case READR:
               enviarLeitura();
               imprimirLeiturasLCD();
               break;
            case SET_NAME:
               setarNomeDevice();
               imprimirLeiturasLCD();
               break;
            case COMANDO_REMOTO:
               enviarComandoRemoto();
               imprimirLeiturasLCD();
               break;
            default:
               //faz nada
         }
      }
      if(valorLeituraAD < 500)
      {
         enviarLeitura();
         imprimirLeiturasLCD();
      }

      delay_ms(1000);

  }
}
```

Apêndice D

Código do Software

Devido ao tamanho do código, aqui serão exibidos apenas as rotinas das *threads*. Todo o código desenvolvido pode ser obtido em <http://code.google.com/p/tccwsnzigbee/source/browse/>

- *Thread* de envio das requisições:

```
public void run() {
        int contador = 0;
        List<CanalInstalado> canaisInstalados = null;
        Iterator<CanalInstalado> iteratorCanaisInstalados;
        while(true)
        {
            try {
                if(contador == 0)
                {
                    canaisInstalados = canaisInstaladosDAO.getLista();
                }
                iteratorCanaisInstalados = canaisInstalados.iterator();
                while(iteratorCanaisInstalados.hasNext())
                {
                    enviarComandoLeitura(iteratorCanaisInstalados.next());
                    Thread.sleep(intervaloEntrePoolsPorSensor);//Aguarda 2s antes de fazer a
próxima rodada de leitura
                }
                contador = (contador+1)%taxaAtualizacaoListaSensores;
                Thread.sleep(intervaloEntrePools);//Aguarda 50s antes de fazer a próxima
rodada de leitura
            } catch (InterruptedException ex) {
                Logger.getLogger(ComunicacaoZigBee.class.getName()).log(Level.SEVERE, null,
ex);
            } catch (SQLException ex) {
                System.out.println("Mais Um erro para LOG");
                Logger.getLogger(ComunicacaoZigBee.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }

    }

 private boolean enviarComandoLeitura(CanalInstalado canalInstalado) throws SQLException {
     portaSerial.HabilitarEscrita();
     portaSerial.EnviarUmaString("+++");
     boolean chegouOK = esperaComandoOK();
     if(chegouOK == false)
     {
         return false;
     }
     portaSerial.EnviarUmaString("AT DH " + canalInstalado.getSerialNumberHigh() + "\r");
     chegouOK = esperaComandoOK();
     if(chegouOK == false)
     {
         return false;
     }
     portaSerial.EnviarUmaString("AT DL " + canalInstalado.getSerialNumberLow() + "\r");
     chegouOK = esperaComandoOK();
     if(chegouOK == false)
     {
         return false;
     }
     portaSerial.EnviarUmaString("ATCN\r");
```

```
      chegouOK = esperaComandoOK();
      if(chegouOK == false)
      {
          return false;
      }
      portaSerial.EnviarUmaString("#READR*"+ canalInstalado.getCanalLeitura() + "\r");
      return true;

         }
```

- *Thread* de Recepção:

```
public void serialEvent(SerialPortEvent ev){

        int novoDado = 0;

        switch (ev.getEventType()) {
            case SerialPortEvent.BI:
            case SerialPortEvent.OE:
            case SerialPortEvent.FE:
            case SerialPortEvent.PE:
            case SerialPortEvent.CD:
            case SerialPortEvent.CTS:
            case SerialPortEvent.DSR:
            case SerialPortEvent.RI:
            case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
            case SerialPortEvent.DATA_AVAILABLE:
                //Novo algoritmo de leitura.
                while(novoDado != -1){
                    try{
                        novoDado = entrada.read();
                        if(novoDado == -1){
                            break;
                        }
                        if('\r' == (char)novoDado){
                            ComunicacaoZigBee.tratarPacote(dadoRecebido);

                            dadoRecebido = "";
                        }else{
                            dadoRecebido = dadoRecebido + (char)novoDado;
                        }
                     }catch(IOException ioe){
                    System.out.println("Erro de leitura serial: " + ioe);
                     }
                }
            break;
        }

           }
```