FAUSTO PEREZ RODRIGUES

SOLUÇÃO DE AUTOMAÇÃO REMOTA UTILIZANDO GNU/LINUX EMBARCADO E O PROTOCOLO ZIGBEE/IEEE802.15.4

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2012

Rodrigues, Fausto Perez
R696s Solução de automação remota utilizando GNU/LINUX embarcado e o protoloco ZIGBEE/IEEE 802.15.4 / Fausto Perez Rodrigues; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2012.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. Sistemas embarcados. 2. Protocolo ZIGBEE/IEEE 802.15.4. 3. GNU/LINUX embarcado. 4. Acesso remoto. 5. Rede de sensores. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fausto Perez Rodrigues

Título: "Solução de Automação Remota Utilizando GNU/Linux embarcado e o Protocolo ZIGBEE/IEEE802.15.4"

Trabalho de Conclusão de Curso defendido e aprovado em 03 / 12 / 2012,

com NOTA 9,0 (Nove, Zero), *pela Comissão Julgadora:*

Prof. Associado Evandro Luís Linhari Rodrigues (Orientador)
SEL/EESC/USP

Profa. Assistente Luiza Maria Romeiro Codá
SEL/EESC/USP

Prof. Dr. Dennis Brandão
SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Dedicatória

Aos meus pais, por todo apoio, amor e confiança.

Agradecimentos

Primeiramente à milha família,

que sempre me apoiou e permitiu meu crescimento como pessoa

além de ser meu exemplo máximo.

Aos meus amigos, que foram fundamentais nas diversas fases da graduação

Aos professores que encaram a missão de ensinar com seriedade.

Ao Professor Dr. Evandro Luís Linhari Rodrigues,

que além de acreditar neste projeto

Sempre manteve sua porta aberta

Ao desenvolvedor Attie Grande pela imensa paciência

e trabalho na biblioteca libxbee

Ao pessoal do laboratório

Ao Departamento da Engenharia Elétrica

À Escola de Engenharia e à USP

Sumário

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

Resumo

Neste trabalho é implementado um sistema de acesso e controle remoto a uma rede de sensores sem-fio, ondo tal rede de sensores utiliza o protocolo *wireless* IEEE 802.15.4/Zigbee. O sistema foi desenvolvido em um Módulo de Desenvolvimento *FriendyARM Tiny6410,* possuindo um processador ARM11 e o sistema operacional Xubuntu GNU/Linux. Buscou-se atender as necessidades apresentadas no paradigma das *WSN (Wireless Sensor Network)*, que são robstez, escalabilidade, modularidade e consumo de energia.

Palavras-chaves: Sistemas Embarcados, Protocolo Zigbee/IEEE 802.15.4, GNU/Linux Embarcado, Acesso Remoto, Rede de Sensores.

Abstract

This paper consists on the project development of a system that allows access and control of a wireless sensor network, where is used the IEEE 802.15.4/Zigbee as the communication protocol. This system was developted on the *FriendlyARM Tiny*6410 board, with a ARM11 proccessor and the Linux Xubuntu Distribution. As result, this project aims to fulfill the criteria of the WSN paradigm as reability, scalability, modularity and energy efficiency.

Keywords: Embedded Systems, Zigbee/IEEE 802.15.4 Protocol, Embedded GNU/Linux, Remote Access, Sensor Network.

1 Introdução

1.1 Apresentação

A área de sistemas embarcados em equipamentos portáteis apresentou e apresenta um crescimento acentuado, atraindo assim, grandes investimentos em desenvolvimento científico e técnico. Desta forma, a investigação de aplicações utilizando as diferentes arquiteturas representa um grande valor comercial e intelectual para o contínuo desenvolvimento na área.

Juntamente com a evolução dos sistemas embarcados (como em memórias, rendimento de baterias, peso, capacidade de processamento e arquiteturas) surge a nova geração dos serviços em *internet* (e.g. *Google Maps*), e a necessidade de uma infraestrutura de comunicação em grande escala, criando o paradigma da rede de sensores sem-fio (WSN - *Wireless Sensor Network*). A integração de módulos *wireless* não só gera um novo modo de comunicação, mas permite que os sistemas portáteis adquiram novas funções(LEADERS, 2007).

A evolução da *WSN* (*Wireless Sensor Network*) permitirá uma grande gama de novas aplicações e utilizações, tais como automação residencial (segurança, HVAC, controle de iluminação, entre outros), aparelhos eletrônicos (controles remotos), automação industrial (monitoramento de energia, controle de ambiente) e saúde pessoal(monitoramento cardíaco, de glicose). Dentre os novos desafios para a implementação das WSNs se encontram: eficiência energética, escalabilidade, mobilidade, robustez, segurança e estratégias de sincronização.

Existe uma grande variedade de protocolos de comunicação *wireless*, os principais apresentados na Figura 1 para as mais diversas aplicações (e.g. voz, vídeo, comunicação geral de dados), todas possuem um compromisso entre a taxa de transmissão (*bit rate*) e a cobertura de transmissão (distância). Segundo Cunha (2007) uma WSN não tem grandes restrições quanto a largura de banda de dados utilizada, porém possui a necessidade de um consumo reduzido de energia, robustez à interferências e segurança. Desta forma, considerando a figura 1, também é válido expressar que entre os protocolos apresentados apenas o *Zigbee* e o *Bluetooth* apresentam grande potencial para a utilização nas *WSNs*.

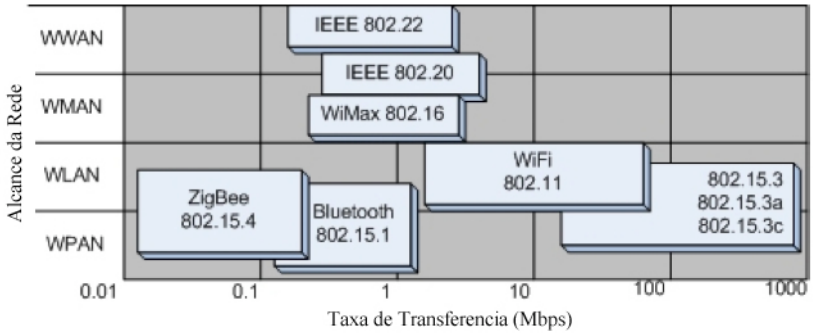

Figura 1 Panorama dos protocolos de comunicação wireless. Adaptado de: CUNHA (2007)

Atualmente, existe um grande número de soluções proprietárias para comunicações sem fio, inabilitando a interoperabilidade entre estes dispositivos de diferentes tecnologias. O esforço conjunto da IEEE 802.15.4 Task Group (IEEE, 2003) e a Aliança Zigbee (ZIGBEE ALIANCE, 2006) permitiu a criação e normalização de uma pilha de protocolo para comunicações sem fio de baixa banda(Zheng e Myung, 2004). A pilha IEEE 802.15.4/Zigbee é composta por quatro camadas onde: o protocolo IEEE 802.15.4 especifica as subcamadas de Enlace (Medium Access Control) e Física. A especificação Zigbee, que se basea na pilha IEEE802.15.4, constrói as camadas de aplicação e de rede da comunicação de dados, conforme mostrado na Figura 2.

A tecnologia Zigbee está aumentando o interesse da indústria e academia, sendo considerada uma solução de baixo custo e baixo consumo para sistemas de automação sem fio(ADAMS, 2005; CULTER, 2005). Os interesses abordados pela proposta da Aliança Zigbee são domótica (foco deste trabalho), sistemas de saúde, segurança e controle ambiental. Desta forma, o protocolo Zigbee apresenta-se como um protocolo para a solução de problemas em WSNs.

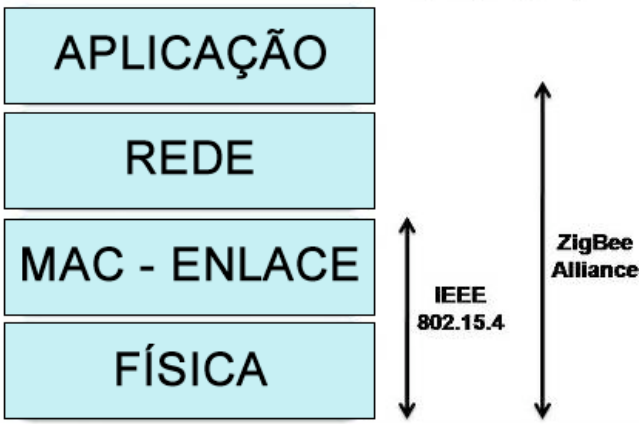

Figura 2 Relação entre as camadas de comunicação e o protocolo Zigbee/IEEE 802.15.4

1.2 Contextualização

Este trabalho foi desenvolvido dentro do contexto da pesquisa e desenvolvimento de sistemas embarcados como *gateway* para a exploração em diversos níveis de comunicações, focando na implementação do serviço de automação utilizando a pilha IEEE 802.15.4/Zigbee junto ao sistema embarcado operando um kernel Linux(BARR; MASSA, 2007)

A escolha de um sistema embarcado operando o sistema operacional GNU/Linux foi adotado para facilitar a implementação COTS (*Commercial-Off-The-Shelf*), onde uma solução pode ser apresentada de maneira mais rápida em sistemas comerciais. O gateway Zigbee permite a exploração da rede IEEE 802.15.4 por meio de um servidor *web*, sendo assim um gateway para a *internet*. Segundo Cunha (2007) os requerimentos que se pretende observar e alcançar com este trabalho são:

- Robustez
- Escalabilidade
- Mobilidade
- Eficiência energética

A hipotese de que este conjunto proposto possa suprir as necessidades da rede de sensores e desta forma a implementação da mesma será fundamental para compreender melhor os comportamentos do protocolo assim como uma abertura para trabalhos seguintes, como testes, avaliações e melhorias.

1.3 Objetivo

O objetivo deste trabalho é avaliar a capacidade do protocolo Zigbee/IEEE 802.15.4, sendo testado pela implementação de disposivos utilizando o protocolo IEEE802.15.4/Zigbee para uma rede de sensores e atuadores sem fio, sendo controlado por um sistema GNU/Linux embarcado em microcontroladores ARM, conforme apresentado na figura 3.

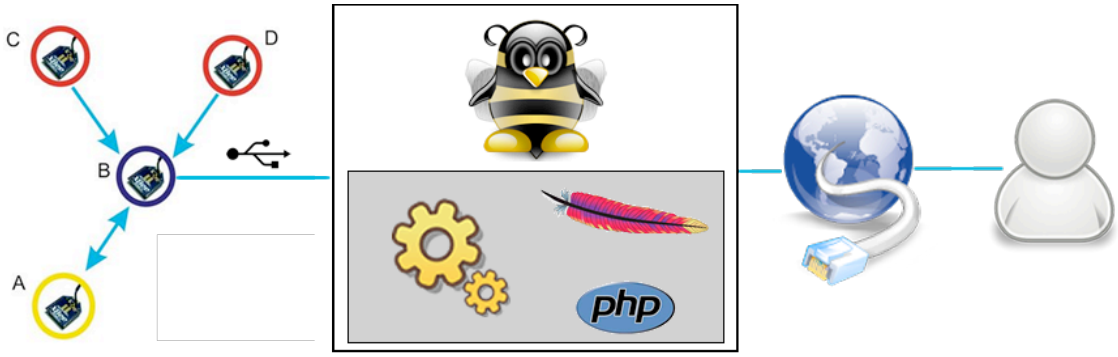

Figura 3 Diagrama da proposta de desenvolvimento do projeto

1.4 Organização

Neste capítulo foi realizada a introdução e justificativa do trabalho, assim como objetivo e contexto. O capítulo 2 contêm informações realcionadas ao embasamento teórico para a realização deste projeto, como um estudo do protocolo Zigbee/IEEE 802.14.5 e sistemas embarcados. Nos capítulo 3 e 4 são descritos os processos da metodologia utilizada, dividida em duas componentes: análise do *software* e configurações para testes. No capitulos 5 é apresentada a implementação, assim como os resultados do processo de implementação. No capitulo 6 são apresentados os testes realizados em relação à proposta do trabalho e os seus respectivos resultados. Por fim, no capítulos 7 são apresentadas discussões com base nos resultados obtidos, conlusões sobre o trabalho e propostas para trabalhos futuros.

2 Fundamento Teórico

Neste capitulo serão apresentados os principais conceitos que definem o fundamento teórico das diversas áreas abordadas neste trabalho.

2.1 Sistemas Embarcados

Um dos primeiros sistemas embarcados foi o computador de navegação *Apollo* (ACG), desenvolvido por Charles Stark Draper. Durante a criação do projeto, o sistema de navegação foi considerado o item com maior risco de falha dentro do projeto *Apollo* já que o sistema utilizado os mais novos componentes monolíticos com peso e tamanhos reduzidos. Porém que característica o definiu como um sistema embarcado?

Um sistema embarcado é uma combinação entre *hardware* e *software* – podendo incorporar outras partes, tanto mecânicas quanto elétricas – projetadas para realizar uma determinada função. Um simples exemplo é o forno micro-ondas, pois quase todas as casas possuem um e, apesar do desconhecimento por parte de seus usuários, possui um processador e um *software* envolvidos na preparação das refeições (GIRIO, 2010).

De acordo com Barr e Massa (2007) o *design* de um sistema embarcado para realizar uma função específica está em contraste direto com o de um computador pessoal. Este também é composto por *hardware*, *software* e partes mecânicas (unidades de disco, por exemplo). Entretanto, um computador pessoal não é designado para realizar uma função específica. Ao contrário, ele pode realizar muitas tarefas distintas e talvez desconhecidas para seus projetores. Muitas pessoas utilizam o termo computador de propósito geral para tornar clara a distinção. O usuário final tem a possibilidade de utilizar seu computador pessoal como um servidor de dados, editor de imagens ou para lazer.

É comum que um sistema embarcado seja componente de um sistema maior, composto por demais sistemas embarcados. Por exemplo, carros modernos possui uma rede de sistemas embarcados distribuídos. Um sistema para a frenagem, outro para interface com o motorista, e um terceiro para a injeção de combustível. Nestes casos, os sistemas embarcados em uma aplicação possuem ou estão inseridos em uma rede de comunicação de dados.

2.2 Módulo *FriendlyARM Tiny6410*

Para a implementação do projeto, optou-se por um módulo de desenvolvimento já testado, com a finalidade de minimizar os riscos envolvendo configuração em baixo nível e *hardware*. Para que os testes propostos atinjam resultados de forma independente da

performance do módulo de desenvolvimento, é necessário que o mesmo possua uma alta capacidade de processamento, memória e interfaces robustas já implementadas. Desta forma optou-se pelo módulo de Desenvolvimento *FriendlyARM Tiny6410*. O módulo de desenvolvimento Tiny6410, fabricado pela empresa chinesa *FriendlyARM* (FRIENDLYARM, 2010), utilizado para desenvolvimento da estação central de controle do projeto, conta com um processador Samsung S3C6410 ARM11 de 533MHz, 256MB de memória RAM e Flash de 2GB. O modulo de desenvolvimento Tiny6410 possui uma grande variedade de interfaces para periféricos, entre eles se destacam:

- 3 USB host;
- 4 portas serial DB9;
- entrada e saída de áudio P2;
- saída de TV;
- porta Ethernet;
- *display* 4,3"TouchScreen resistivo;
- *slot* de SD card.

Nas Figuras 4 e 5 é possível observar com mais detalhes o módulo de desenvolvimento e o núcleo de processamento.

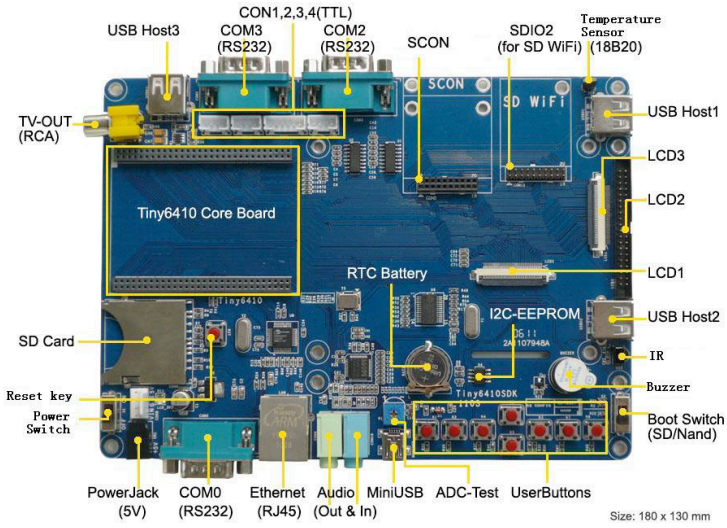

Figura 4 Visão geral dos periféricos do módulo de desenvolvimento *FriendlyARM Tiny*6410 (FRIENDLYARM, 2011)

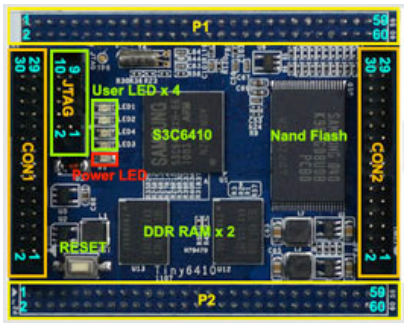

Figura 5 Visão geral da placa núcleo (*core*) (FRIENDLYARM, 2011)

O módulo é acompanhado por dois DVDs que possuem manuais sobre o *hardware* da placa de desenvolvimento assim como manuais sobre a instalação dos sistemas operacionais que também acompanham o módulo, que são: Windows CE, Android 2.1, Linux(Qtopia) e Xubuntu.

2.3 Sistema operacional Linux

Neste trabalho foi utilizado o sistema operacional Xubuntu GNU/Linux embarcado no módulo de desenvolvimento *FriendlyARM tiny6410*, sendo assim os demais sistemas operacionais suportados pelo módulo não serão abordados.

2.3.1 Linux e Linux embarcado

Linux é o sistema operacional que surgiu em 1991, desenvolvido por Linus Torvalds (Linux, 2012). Pelo fato de possuir o código disponível para todos, o sistema operacional tornou-se popular rapidamente e criou-se uma grande quantidade de colaboradores. Com o aumento de desenvolvedores permitiu o avanço do sistema em velocidade e qualidade, pois de forma acelerada foram agregadas funcionalidades, recursos, documentação.

O Linux é referenciado como o *kernel* do sistema operacional, que é o núcleo do mesmo. Em dispositivos embarcados o *kernel* é o mesmo para diferentes plataformas, apenas sendo necessitado a sua compilação para a determinada arquitetura a ser implementada. Como foi definido anteriormente, o sistema embarcado é o conjunto de *software* e *hardware* com finalidade específica. Dada a natureza do Linux é possível modificar o sistema operacional para atender somente as necessidades que o sistema exige. Desta forma existe uma melhor alocação e aproveitamento dos recursos disponíveis, onde geralmente existem limitações em relação a consumo de energia permitido, memória e capacidade processamento (CUNHA, 2007).

Desta forma a versão do Linux desenvolvida para sistemas embarcados possui as características e qualidades do *kernel* mantido pela comunidade, tais como suportes a padrões e protocolos de comunicação, drivers e módulos a interfaces e periféricos consolidados.

Estas versões focadas em sistemas embarcados também possuem um gama de desenvolvedores trabalhando em conjunto para a criação de ferramentas específicas, permitindo assim um avanço em direções levemente diferentes em relação as distribuições voltadas para computadores pessoais e servidores. Assim, surgem comunidades de desenvolvimento em plataformas específicas, como *FriendlyARM* (FRIENDLYARM, 2012), possuindo diversos trabalhos em documentação e suporte em fóruns de usuários. Porém para algumas plataformas ainda jovens, o nível de documentação e suporte em comunidades é reduzido.

A implementação do Linux nas diversas famílias de arquitetura de sistemas embarcados que trabalham em 32-bits(ARM, MIPS, entre outras) foi principalmente motivada devido ao diferencial comercial do Linux, que sendo distribuído sobre a licença GNU-GPL, que permite qualquer usuário ou empresa utilizar, modificar e redistribuir o sistema isento de cobrança. Assim. existe um incentivo no sentido de diminuição do custo gerado pelo desenvolvimento do projeto, onde, geralmente, o custo é crucial para a realização de diversos projetos na área.

O sistema de GNU/Linux embarcado é composto de maneira básica por três componentes: *Bootloader*, *Kernel* e *Root Filesystem*. O *Bootloader* é responsável pela inicialização do sistema, e desta forma criar as condições necessárias para execução do sistema operacional. O *Kernel*, conforme foi discutido anteriormente, consiste no sistema operacional em si, gerenciando os recursos em relação as aplicações do sistema. O *Root Filesystem* (sistema de arquivos raiz) contém as aplicações, bibliotecas, configurações, dados gerais. A Figura 6 representa de maneira geral tais componentes do Linux embarcado. Nas próximas seções cada um destes componentes do sistema GNU/Linux serão abordadas especificamente.

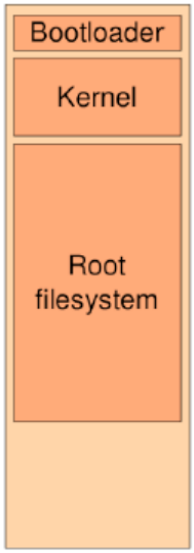

Figura 6 Estrutura simplificada do sistema GNU/Linux (ELECTRONS, 2012)

2.3.2 *Bootloader*

O *bootloader* é responsável pela inicialização dos principais componentes de *hardware* para a execução do *kernel* do sistema operacional, descrito na próxima seção, verificando o *hardware* disponível e mínimo para carregar o *kernel* na memória RAM do sistema. O *bootloader* é posicionado na memória do dispositivo no endereço inicial de leitura do processador, a fim de ser a primeira sequência de execução a ser processada. (Linux, 2012)

Apesar dos diversos *bootloaders* disponíveis para os diferentes *hardwares* e sistemas operacionais destacam-se o GRUB e o LILO para sistemas Linux em computadores pessoais. Para sistemas embarcados opta-se pelo U-boot(U-boot, 2012)um *bootloader* mais simples e ajustável aos diferentes *hardwares* utilizados em sistemas embarcados, além de funções poderosas como alteração do *kernel* do sistema embarcado remotamente.

2.3.3 *Kernel*

O Linux em si é o *kernel* (núcleo do sistema operacional) do sistema operacional, ou seja, ele não possui as ferramentas para interface com o usuário como ambiente gráfico, *softwares* gerais (estes sendo fornecidos pela distribuição completa) (Linux, 2012). A interação do *kernel* com as aplicações gerais, do ponto de vista de blocos de funções, é ilustrado na figura 7.

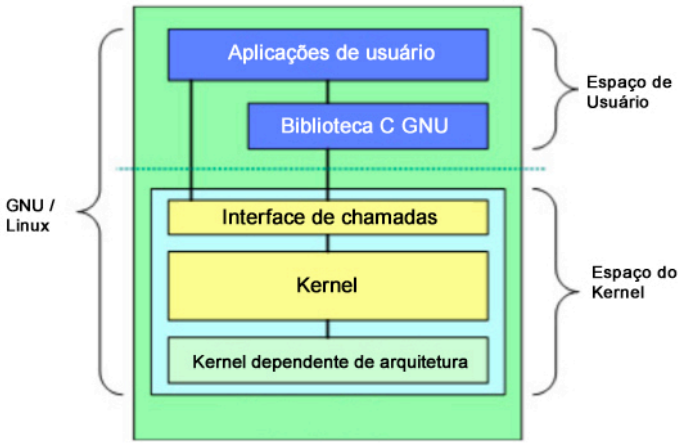

Figura 7 Blocos de funções do *Kernel* do GNU/Linux

Apesar do *kernel* ter sido desenvolvido por Linus Torvalds, hoje é mantido por diversos desenvolvedores. O projeto do Linux pode ser encontrado no site *www.kernel.org*. O *kernel* opera numa área da memória denominada *kernel space* (uma área de acesso restrito) apresentada na Figura 8(a figura foi mantida em inglês para manter a relação do nome com as siglas correspondentes) e pode ser segmentada em relação aos blocos que o compõe. Nesta divisão existem três camadas no *kernel* do Linux, onde a primeira é a camada de interface entre o *user space* e o *kernel space* nessa camada é necessário uma preocupação com a biblioteca *glibC*(principal biblioteca de funções utilizadas no sistema operacional) utilizada na Distribuição GNU utilizada.

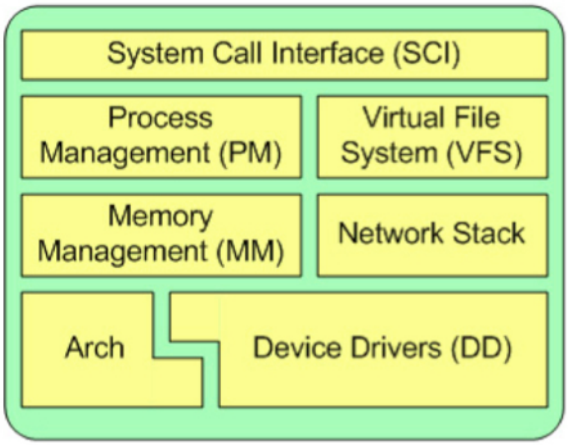

Figura 8 Blocos de funções do *kernel*

A segunda camada é comum a todas os sistemas que utilizam Linux, independente da arquitetura utilizada e contém as principais rotinas de gerenciamento de memória, controle do

sistema de arquivos e o gerenciador de processos, este último contém as rotinas como escalonador de tarefas, controles de sincronização e tratamento de informações.

A terceira camada é a interface entre o *hardware* e a tratamento de processos e arquivos do *kernel*, portanto é a camada que contém os *device drivers* e definições da arquitetura utilizada. Assim as mudanças necessárias para embarcar o sistema Linux para as diversas plataformas e arquiteturas são realizadas nesta camada.

2.3.4 Sistema de arquivos raiz (*Root filesystem*)

O sistema de arquivos raiz é responsável pela leitura e escrita e armazenamento persistente de dados, documentos, softwares e bibliotecas que são necessários para o sistema operacionais e aplicações gerais dos usuários. O sistema de arquivos realiza tal gerenciamento de acordo com o *hardware* de armazenamento de dados. De acordo com Curvello e Santos (2011), dentre os sistema de arquivo raiz existentes destacam-se os seguintes para o Linux:

• EXT3: *Third Extended File System*, em português: Terceiro Sistema de Arquivos Estendido. É amplamente usado em ambientes *desktop* com sistemas baseados em UNIX, tendo como principal característica possuir mecanismos cofiáveis e robustos para escrita de arquivos, prevenindo danos em ocasiões onde o sistema é indevidamente desligado. Não possui implementação de mecanismos adequados de gestão de células lógicas FLASH o que torna sua aplicação limitada em relação a sistemas embarcados em longo prazo.

• YAFFS2: *Yet Another Flash File System 2*, ou em português: "Ainda Outro Sistema de Arquivos para Flash 2". É amplamente usado por dispositivos que possuem sistema Android.

• JFFS2: Abreviação de *Journalling Flash File System 2*. Recentemente era um dos sistemas de arquivos mais usados em sistemas embarcados. Com o avanço das tecnologias de memória FLASH, tornando-se defasado para sistemas embarcados.

• UBI: Abrevição de *Unsorted Block Images*. É o sistema de arquivos que possui características para implementação em sistemas embarcados, porém é o mais complexo. Trabalha com o sistema de arquivos na forma de blocos, como sua distribuição não é linear e orientada a blocos é ideal para células lógicas *FLASH*.

Em sistemas embarcados o principal dispositivo para o armazenamento de dados de maneira persistente são células de memória *FLASH* (conforme citado anteriormente). A utilização de outros tipo de dispositivos, tais como disco-duros acarretariam grande consumo de energia e encarecimento do sistema de maneira considerável.

A memória *FLASH* apresenta um desgaste em relação ao tempo, e ao uso inadequado por meio do sistema ou usuário. Logo, é aconselhável a utilização de um sistema de arquivos que permita o gerenciamento das células danificadas, permitindo a otimização do sistema de arquivos e *hardware*.(GUIA DO HARDWARE, 2012)

2.3.5 Distribuição GNU Linux Xubuntu

Conforme citado anteriormente o sistema operacional Linux é em si o *kernel*, quando colocado em conjunto com aplicações e gerenciadores do sistema operacional são formadas as distribuições GNU/Linux (LINUX, 2012). Existem hoje mais de 300 distribuições sendo mantidas atualizadas, sendo muitas *forks* (projetos derivados) de outras distribuições, organizadas em diversas categorias.

Dentre a distribuições mais conhecidas destacam-se o Debian, Slackware, Fedora, OpenSuse, Archlinux e Ubuntu. Existem desde pequenas diferenças a grandes modificações e maneiras de se trabalhar entre as diversas distribuições. Dessa maneira o desenvolvedor ou usuário tem a opção de escolher a distribuição que melhor solucione as suas necessidades.

Para sistemas embarcados existe uma grande variação entre as diversas distribuições. Neste trabalho foi utilizada a distribuição Xubuntu GNU/Linux. O Xubuntu é uma distribuição baseada na distribuição Ubuntu (um *fork* da distribuição Debian) adaptada para aplicações em sistemas embarcados, onde existe uma maior sensibilidade ao gerenciamento de memória e arquitetura de processadores.

Uma grande vantagem do sistema Xubuntu para a implementação de soluções de rápido desenvolvimento para sistemas embarcados são ferramentas de alto nível para instalação de aplicações desenvolvidas pelo seu antecessor Debian. Dentre as ferramentas destaca-se o gerenciador de pacotes APT (Inglês) que permite a instalação de outras aplicações, tal como o servidor *web Apache*, e instalação de bibliotecas de linguagens de programação como PHP, Python entre outros.

2.4 Transmissão de dados

Esta seção aborda o modo de transmissão de dados na rede sem fio de sensores (WSN) com a utilização do protocolo Zigbee/IEEE 802.15.4. Também será abordado a comunicação serial que é necessária para a comunicação do módulo coordenador Xbee com o módulo de desenvolvimento por meio da interface USB.

2.4.1 Transmissão serial

Um dos métodos mais comuns de transmissão de dados entre dispositivos, é a transmissão serial de dados, onde cada bit de informação é transmitidos pelo mesmo canal e em ordem sequencial. A figura 9, abaixo, ilustra o funcionamento básico de transmissão de dados.

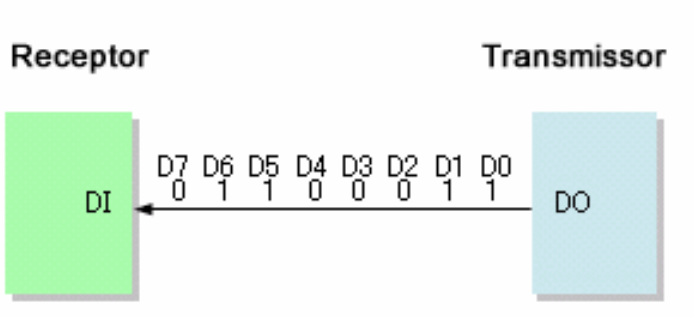

Figura 9 Modelo simples de transmissão de dados serial

Apesar da transmissão serial apresentar velocidade menor quando comparada a transmissão paralela de dados, a comunicação serial apresenta grande simplicidade na lógica de controle quanto na implementação física. A comunicação serial pode ser síncrona ou assíncrona. Na comunicação síncrona há a necessidade de um sinal de *clock* para determinar o controle de fluxo de dados entre transmissor e receptor. Na segunda forma, a transmissão assíncrona e forma de controle de fluxo de dados é realizada por meio de conjunto determinado de dados que são transmitidos no mesmo canal dos dados de informação.

Apesar da comunicação síncrona apresentar maior desempenho apresenta também um maior nível de complexidade e custo de *hardware*. Portanto para projetos simples e de custo reduzido a implementação de sistemas assíncronos é mais utilizada, onde incide menor custo em *hardware*, menor complexidade de projeto.

A comunicação serial assíncrona apresenta a inserção de bits de controle junto ao canal de dados. Logo a velocidade de transmissão de dados de informação torna-se ainda menor. Como utiliza-se como solução o protocolo Zigbee/IEEE 802.15.4 onde não ocorre a necessidade de altas taxas de transferência, este aspecto não deve gerar atrasos na comunicação sem fio.

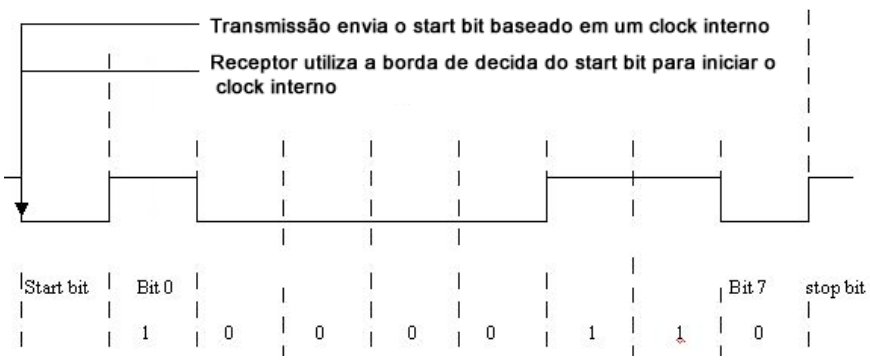

Figura 10 Modelo de transmissão serial assíncrona

Na linha de transmissão são enviados pacotes conforme a figura 10. Permanecendo em nível alto durante o repouso, quando ocorre a mudança de estado sinalizando o inicia da transmissão. Este primeiro bit recebe o nome de *Start Bit*, em seguida são transmitidos dados que podem ter tamanho de 5 a 8 bits. Ao final são adicionados 1 ou 2 bits de paridade (sendo configurado por *software*). A transferência de dados possui uma taxa base de transmissão denominada *Baud Rate* e, neste trabalho, a mesma será utilizada 9600bps.

2.4.2 Protocolo e dispositivos *Zigbee*/IEEE 802.15.4

Zigbee/IEEE 802.15.4 é o protocolo de comunicação que opera nas camadas de aplicação rede, enlace e física, segundo parâmetros das camadas ISO/OSI, conforme apresentado anteriormente na Figura 2. O protocolo *Zigbee* pode ser compreendido como uma solução em protocolo que, com suporte na norma IEEE 802.15.4, desenvolvido pelo consorcio *Zigbee Alliance* (ZIGBEE ALLIANCE, 2012), onde o objetivo é permitir a comunicação com baixas taxas de transmissão de dados, implicando em baixo consumo de energia e baixo custo operando na frequência de 2.4GHz. Neste trabalho o conjunto *Zigbee*/IEEE 802.15.4 será mencionado simplesmente como protocolo *Zigbee*.

O método implementado pelo protocolo *Zigbee* permite a criação de uma rede onde a transmissão de dados pode ocorre por roteamento de dados, ou seja, a comunicação entre dois dispositivos não necessita ser direta entre o dispositivo inicial e final da transmissão. As direções de roteamento são configuradas previamente, editando-se os registradores, podendo ser modificadas enquanto a rede está online por meio de comandos remotos caso seja utilizado o modo API (este modo de operação será abordado na seção intitulada *API mode*).

Hoje, existem no mercado, diversas soluções em módulos utilizando o protocolo *Zigbee* dentro das características COTS(*Commercial Off-The-Shelf)*, ou seja, que são soluções prontas para implementação. Dentre estas, serão abordados os módulos Xbee da Digi

International, devido a sua presença no mercado brasileiro e custo dentro do limite para desenvolvimento do projeto. As principais características dos módulos Xbee serão apresentadas no decorrer deste trabalho (ROGERCOM,2012).

2.4.2.1 Topologia de Rede *Zigbee*

Para discutir as possíveis topologias de rede *Zigbee* é necessário compreender a distinção que pode haver em relação as funções de cada nó da rede. Existem duas categorias, em relações as funções habilitadas no dispositivo e funções de nós na rede. A classificação segundo as funções habilitadas por dispositivo são:

• **RFD - *Reduced Function Device*:** São dispositivos que possuem maior simplicidade quanto ao *hardware*, não permitindo a implementação completa da pilha de comunicação. Atuam somente como *End Devices* na rede e, portanto, só se comunicam com coordenadores e roteadores. Na prática, os RFDs são dispositivos com acesso a rede que constituídos por interruptores, sensores, controladores, relés entre outros.

• **FFD - *Full Function Device*:** São dispositivos que possuem a capacidade de operar como Coordenadores, Roteadores ou *End Devices*. Portanto, possuem *hardware* necessário para a implementação da pilha completa de comunicação, processamento e consumo de energia. Os FFDs podem se comunicar com quaisquer outro dispositivo da rede. Os módulos Digi Xbee estão dentro desta categoria de dispositivos.

A segunda maneira de classificar os dispositivos que operam a rede *Zigbee* é em relação a função desempenhada na rede na qual está incluído, podendo ser:

• ***Coordinator*/Coordenador:** O dispositivo com a função de coordenador é responsável pela inicialização da rede (controle da PAN), controle e enumeração (distribuição de endereços para os dispositivos) e reconhecimento dos dispositivos. Na rede *Zigbee* só existe um coordenador atuante, sendo possível que haja um segundo dispositivo com função de backup em caso de falha do primeiro coordenador. Somente implementável por um dispositivo FFD.

• ***Router*/Roteador:** A função de roteador também só podem ser executada por um dispositivo FFD, porém podem haver vários roteadores em uma mesma rede *Zigbee*. Os roteadores são responsáveis pela transmissão de dados entre dispositivos, permitem a entrada de dispositivos, entre outras funções.

• ***End Device*/Dispositivo Final:** Nesta função é possível utilizar dispositivos FFD e RFD, pois apenas comunicam dados para os roteadores e coordenadores. Os dispositivos finais, neste trabalho referidos como *End Devices*, consomem menos energia que os

coordenadores e roteadores, pois permitem a função *sleep* (tal operação será abordada posteriormente) do dispositivo e não permitida nos demais.

Considerando as possíveis funções desempenhadas pelos dispositivos *Zigbee*, as topologias de rede são: (a) *Star* (Estrela), (b) *Cluster Tree* (Árvore) e (c) *Mesh* (Malha), conforme apresentado na Figura 11.

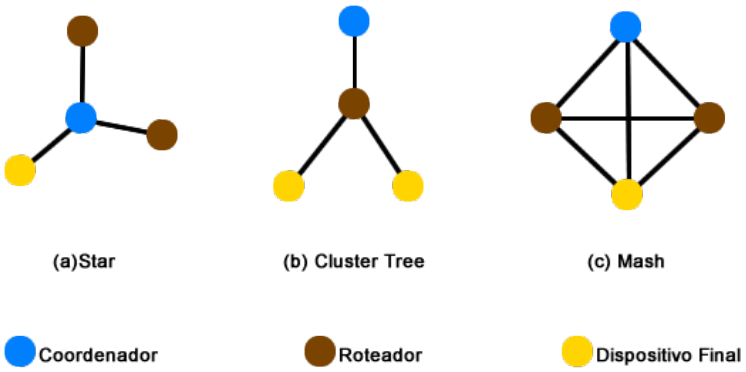

Figura 11 Topologias de rede para rede Zigbee.
(a)Estrela(*Star*)(b)Árvore(*Cluster Tree*)(c)Malha(*Mesh*)
Adaptada de: ROGERCOM (2012)

A topologia Estrela (figura 11(a)) é a topologia mais simples, onde o coordenador de rede comunica-se diretamente com os dispositivos de rede. Esta topologia é aplicada para sistemas simples, onde não existem muitos obstáculos para a transmissão de dados, assim como uma taxa de transmissão média baixa.

A topologia de Árvore (figura 11(b)) possui um nível de complexidade acima da topologia Estrela, pois apresenta de hierarquia, ou rota, definida para a transmissão de dados. Porém tal topologia apresenta uma inflexibilidade para o tráfego de dados.

Na topologia de Malha (figura 11(c)) ocorre o ajuste automático de roteamento de dados, assim como o monitoramento dos nós, este tipo de topologia aproveita as características diferencias do protocolo Zigbee/IEEE 802.15.4 para aplicações com diversos nós, longas distâncias e taxas de transmissão de dados variadas.

2.4.2.2 Inicialização da Rede Zigbee

A inicialização da rede Zigbee ocorre com a energização do dispositivo coordenador, onde o mesmo escolhe um PAN ID (*Personal Area Network Identifier*). Os dispositivos encontrados pelo dispositivo coordenador, e configurados apropriadamente (a configuração será abordada no Capítulo 3 - Metodologia), herdam o PAN ID do coordenador, passando a

fazer parte da rede. Ao “entrar” na rede do coordenador, o dispositivo recebe um endereço de rede (16bits). O processo de criação de rede ocorre em duas etapas:

1. Varredura de energia(*Energy Scan*): O coordenador varre os diversos canais utilizados no padrão IEEE 802.15.4, detectando níveis de energia, correspondentes a canais utilizados por outras redes, equipamentos, entre outros. Desta forma o coordenador elimina canais para utilização da rede.

2. Varredura de PAN(*PAN Scan*): Após a varredura de energia, o coordenador busca por PANs dentro dos possíveis canais de comunicação por meio de um sinal de *broadcast* (*Beacon Request*), mostrado na Figura 11. Os dispositivos que recebem o sinal de *beacon* respondem o coordenador, com informações relacionadas a permissões do dispositivo, a PAN em que o dispositivo está e sobre o próprio dispositivo.

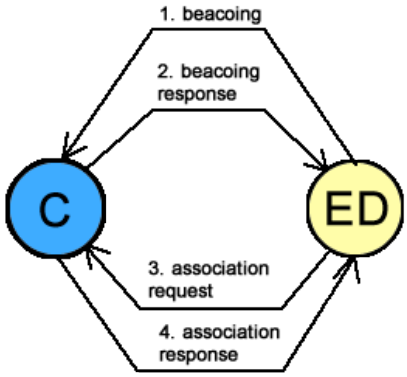

Figura 12 Processo de *beaconning* para inclusão de dispositivo a rede Zigbee. Adaptada de: DIGI INTERNATIONAL (2010)

Ao finalizar a varredura de PAN, o dispositivo coordenador analisa os *frames* recebidos e determina a nova PAN, PAN ID e canal, utilizada pelos dispositivos. Quando os demais dispositivos são inicializados, eles juntam-se a PAN disponível, este processo pode ocorrer por dois métodos:

• ***Beaconning:*** Os dispositivos roteadores permanecem ligados e emitem um sinal de sinalização(*beaconning*), descobrindo os demais dispositivos na rede. Enquanto os dispositivos finais(*End devices*) permanecem e *sleep* e tornam- se ativos por um processo de temporização a fim de observar o sinal de *beaconning*. Desta forma existe uma economia de energia pelo sistema.

• ***Non-Beaconning:*** Neste modo todos os dispositivos permanecem ativos, consumindo mais energia. Assim, não há necessidade de sinalização temporal entre os dispositivos.

Desta forma, o dispositivo encontra a rede do coordenador e é inserido, e se for permitida sua entrada, em uma PAN válida. Ao encontrar a PAN válida, o dispositivo envia um *Association Request* ao nó patriarca, conforme apresentado na Figura 11. A permissão de entrada de outros roteadores ou dispositivos na rede é determinada, conforme citado anteriormente, pelo próprio atributo de permissão de entrada e o número máximo de nós-filhos pré-determinado.

O atributo de permissão de entrada determina a possibilidade de associação de outros dispositivos a rede na qual o dispositivo coordenador ou roteador se encontra. Tal atributo pode permitir a entrada, permitir a entrada por um determinado período de tempo ou não permitir a entrada de mais dispositivos.

O número máximo de conexões de um dispositivo, coordenador ou roteador, é determinado segundo o número estimado de transmissão em relação ao número de dispositivos conectado ao roteador ou coordenador, ou seja, pelo trafego de dados considerado limite.

2.4.2.3 Endereçamento e envio de dados

Na rede Zigbee ocorre o endereçamento dos dispositivos em dois níveis da camada de comunicação ISO/OSI, endereçamento de dispositivos(podendo ser na camada de rede ou física) e o endereçamento na camada de aplicação.

O endereçamento de dispositivos podem ser feitos em duas camadas separadas. De uma forma, por um endereço de 64 bits que é permanente e imutável para cada radio Zigbee, estabelecido pelo fabricante. Este endereço se assemelha ao endereço MAC em dispositivos de rede Ethernet. A segunda forma é dada por um endereço de rede, de 16 bits, configurável e mutável de acordo com as necessidades da rede na qual o dispositivo se encontra.

O endereçamento na camada de aplicação considera o dispositivo final, de destino (*End Point*), e a identificação da informação enviada, *Frame ID*. O valor do *End Point*, semelhante a um socket TCP, representa uma tarefa ou aplicação no dispositivo de destino. O cluster ID define o comando, ou ação, propriamente dito.

Os pacotes de dados na rede Zigbee utilizam as duas formas de endereçamento, por dispositivo e por camada de aplicação, para transmitir os dados. Desta forma, a transmissão em si pode ser feita por *unicast* (direcional) ou por *broadcast*, e caso não haja conexão direta

entre os dispositivos inicial e final existe a transmissão por meio de outros dispositivos da rede(*hops*). Para que o envio encontre o endereço destino ocorre:

• **Descoberta de endereço:** Consiste em associar um endereço de rede mutável (16 bits) a um endereço de dispositivo imutável (64 bits). O emissor envia uma mensagem em *broadcast* informando o endereço 64 bits de destino, quando o dispositivo receptor compara os endereços de dispositivo de 64 bits e são idênticos, o mesmo responde confirmando seu endereço de rede (16 bits), desta forma o emissor inicial inicia a transmissão de dados de informação de aplicação.

• **Descoberta de rota:** A rota na rede Zigbee é determinada por um procedimento *Ad Hoc On-Demand Distance Vector*, onde a rota é determinada relacionando a distancia de acordo com a utilização dos nós da rede. Cada transmissão de *frames* de dados entre dispositivos de roteamento é considerado um *hop* e pode ser determinado um número máximo de retransmissões.

A transmissão de dados na rede Zigbee possui confirmação de sucesso na transmissão, conhecida como ACK, referente ao termo *acknowledge* do inglês (reconhecimento ou confirmação). A cada hop entre dispositivos é transmitido um ACK na direção do emissor, a fim de informar o sucesso da transmissão. Caso ocorra alguma falha, o radio Zigbee realiza uma tentativa de retransmissão do pacote (sendo, por padrão, no máximo duas tentativas) e no caso de falha nas retransmissões a rede percebe o erro ocorrido, permitindo ações de controle.

Na seção 2.4.2.5 será abordado novamente o endereçamento de dispositivos em relação ao modo de operação utilizado.

2.4.2.4 Módulos Digi Xbee

No mercado, existem diversas empresas que fabricam dispositivos Zigbee, neste trabalho serão utilizados dispositivos Xbee e Xbee PRO ZNET2.4, o dispositivo Xbee é apresentado na Figura 13. Vale ressaltar neste ponto que o dispositivo de qualquer empresa “X” que atue oficialmente dentro das normas do protocolo Zigbee/IEEE 802.15.4 se comunicara sem problemas com o dispositivo da empresa “Y” .

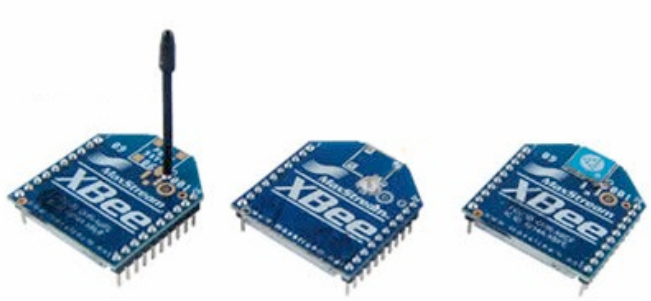

Figura 13 Módulos Xbee da Digi International com diferentes antenas.
Adaptado de: ROGERCOM (2012)

A versão Xbee PRO difere quanto ao desempenho em relação ao Xbee, desempenho aqui medido em distância de transmissão, principalmente. Porém tal desempenho acarreta um maior consumo de energia pelo dispositivo. A tabela 1 fornece as principais informações em relação aos dispositivos Xbee e Xbee PRO (DIGI INTERNATIONAL, 2010).

TABELA 1 Especificações gerias dos dipositios Xbee e Xbee PRO Series 2. (DIGI INTERNATIONAL, 2010)

Característica	Xbee	Xbee PRO
Potência de saída	1 mW (0 dBm)	60 mW (18 dBm)
Alcance em ambientes internos	30m	100m
Alcance de RF em linha visível	100m	1600m
Sensibilidade do receptor	-92 dBm	-100 dBm
Taxa de dados de RF	250.000 bps	250.000 bps
Tensão de alimentação	2.8-3.4V	2.8 à 3.4v
Corrente de transmissão (típico)	45 mA @ 3.3 V	215 mA @ 3.3 V
Corrente de Recepção (típico)	50 mA @ 3.3 V	55 mA @ 3.3 V
Dimensões	2.438cm x 2.761cm	2.438cm x 3.294cm
Tipo de espalhamento espectral	DSSS	DSSS
Criptografia 1	28-bit AES	128-bit AES

Os módulos Xbee e Xbee PRO também possuem diversos periféricos já implementados, tais como saídas e entradas digitais, conversores analógico-digital, saída por PWM, canal de comunicação UART (*Universal Asynchronous Receiver/Transmitter*). Desta forma é possível implementar soluções de maneira rápida e eficiente, com *hardware* já validado. O diagrama de pinos é apresentado conforme o Quadro 1

Neste trabalho ambos dispositivos serão tratados por dispositivo Xbee, e quando for necessário a diferença entre os dispositivos será adequadamente apresentada, com os respectivos resultados.

Pin #	Name	Direction	Default State	Description
1	VCC	-	-	Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / CONFIG	Input	Input	UART Data In
4	DIO12	Both	Disabled	Digital I/O 12
5	RESET	Both	Open-Collector with pull-up	Module Reset (reset pulse must be at least 200 ns)
6	RSSI PWM / DIO10	Both	Output	RX Signal Strength Indicator / Digital IO
7	DIO11	Both	Input	Digital I/O 11
8	[reserved]	-	Disabled	Do not connect
9	DTR / SLEEP_RQ/ DIO8	Both	Input	Pin Sleep Control Line or Digital IO 8
10	GND	-	-	Ground
11	DIO4	Both	Disabled	Digital I/O 4
12	CTS / DIO7	Both	Output	Clear-to-Send Flow Control or Digital I/O 7. CTS, if enabled, is an output.
13	ON / SLEEP	Output	Output	Module Status Indicator or Digital I/O 9
14	VREF	Input	-	Not used for EM250. Used for programmable secondary processor. For compatibility with other XBEE modules, we recommend connecting this pin voltage reference if Analog sampling is desired. Otherwise, connect to GND.
15	Associate / DIO5	Both	Output	Associated Indicator, Digital I/O 5
16	RTS / DIO6	Both	Input	Request-to-Send Flow Control, Digital I/O 6. RTS, if enabled, is an input.
17	AD3 / DIO3	Both	Disabled	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Both	Disabled	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Both	Disabled	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0 / Commissioning Button	Both	Disabled	Analog Input 0, Digital IO 0, or Commissioning Button

QUADRO 1 Diagrama e especificação da pinagem dos módulos Xbee e Xbee PRO. Fonte (DIGI INTERNATIONAL, 2010)

2.4.2.5 Comunicação entre dispositivos Digi Xbee

Utilizando a interface serial do módulo Xbee, pode-se trabalhar de duas maneiras: de modo transparente, utilizada entre dois dispositivos específicos, e modo API (*Application Programming Interface*), que fornece uma interface para outras aplicações em relação a rede Zigbee. As principais características de cada modo de operação é descrita a seguir. De acordo com o *datasheet* DIGI INTERNATIONAL (2010), estão implementados os seguintes modos de operação:

• **Modo Transparente:** É o modo de operação mais simples, onde é configurado por meio de registradores o endereço de destino (DL e DH), e ao se transmitir dados no pinto DIN (RX) do módulo Xbee a informação é transmitida de forma transparente para o pino DO (TX) do módulo Xbee de destino. O endereço de destino pode ser editado pelo canal serial também utilizando comandos AT, no *datasheet*, p.129, dos dispositivos Xbee da Digi International (2010) é possível encontrar a lista completa com descrição dos comandos AT.

• **Modo API:** Neste modo existe uma estrutura (*frame*) que são categorizados entre *frames* de requisito e de resposta (*Transmit Data Frames* e *Response Data Frames*). Este método permite a utilização máxima dos recursos da rede Zigbee, como transmissão

multiponto, configuração remota, entre outras aplicações. Além de maior controle dos dados, por introdução de dados de controle como controle de erro (*Checksum*) e endereço de origem.

Durante o processo de comunicação, o dispositivo Xbee possui uma série de estados em relação ao processo de comunicação. Tais estados são independentes do modo de operação utilizado e são os seguintes: *sleep*, repouso (*idle*), de transmissão, de recepção e de comandos AT.

• **Estado *sleep* :** Neste modo, permitido aos dispositivos que possuam função de dispositivo final, onde os mesmos permanecem a maior parte do tempo sem ou com o mínimo de *clock* de máquina, e com a grande maioria dos periféricos desligados. E assim, o consumo de energia é extremamente reduzido. Estes dispositivos podem sair deste estado por uma interrupção externa ou interna (e.g. *timer*).

• **Estado de repouso(*idle*):** Como os dispositivos coordenador e roteadores não podem entrar em *sleep*(desativação completa da rede) eles encontram-se em estado de repouso quando não estão transmitindo ou recebendo dados. Neste estado ocorre uma diminuição da capacidade de processamento (*clock*), e assim, ocorre uma economia de consumo de energia, porém não drástica como a função *sleep*.

• **Estado de transmissão:** Quando um dado é recebido por um pino de entrada de dados, o dispositivo entra em modo de transmissão. O dispositivo valida do endereçamento e rota da mensagem, caso não sejam conhecidos(endereço ou rota) é iniciado o processo de descobrimento de endereço ou rota, conforme apresentado na seção 2.4.2.3. O fluxograma apresentado na Figura 14 ilustra o processo para transmissão de dados pelo dispositivo Xbee.

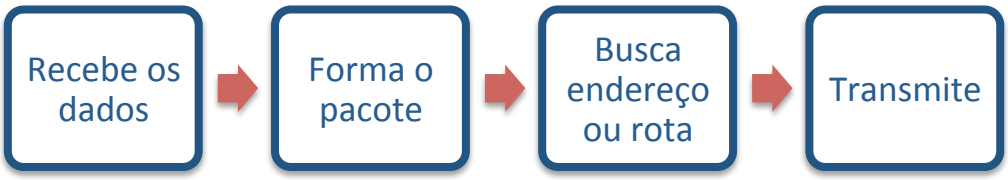

Figura 14 Processo para transmissão de dado

• **Estado de recepção:** O dispositivo entra no estado de recepção quando existe o sinal no canal utilizado pela PAN, verificando se o mesmo é o endereço de destino da informação. Caso seja o destino, envia o sinal de ACK para o dispositivo de origem do *hop* e guarda o dado recebido em um *buffer*. O processo é análogo porém invertido em relação à transmissão.

• **Estado de comandos AT:** O estado, ou modo, AT é um estado especial onde o dispositivo Xbee recebe comandos de configuração(leitura ou atualização de dados) pela serial UART.

Para ativar este modo enviam-se os caracteres "+++"(sem aspas) dentro de um segundo, o dispositivo responde "OK", confirmando que o dispositivo entrou em modo de comandos AT. A síntese dos comandos AT é ilustrada na Figura 15.

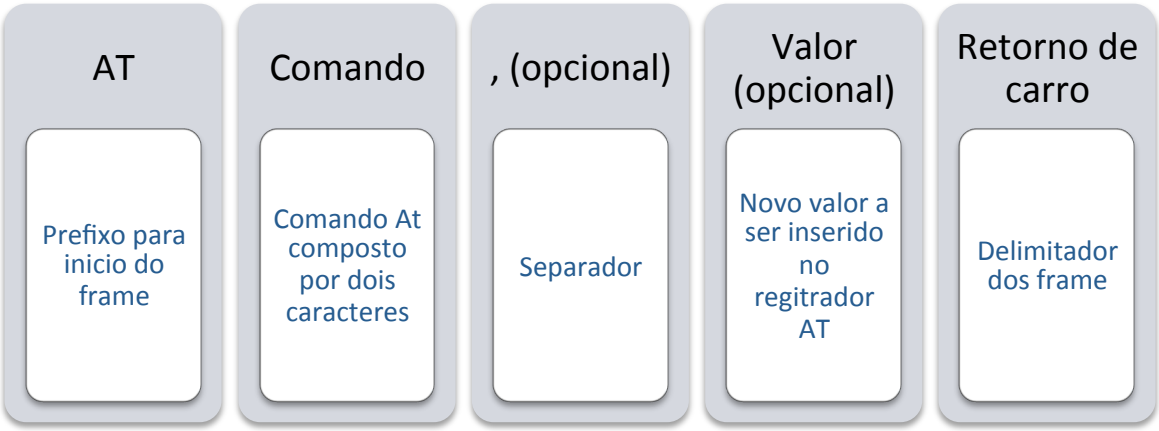

Figura 15 Estrutura do *frame* de transmissão do modo AT

Para encerrar o modo de comandos existe um *timeout* ou pelo comando “ATCN“.

Para garantir a comunicação entre os dispositivos é necessário realizar o endereçamento dos dispositivos, conforme apresentado na seção 2.4.2.3, de maneira coerente com o modo de operação utilizado. Desta forma tem-se para o modo transparente de operação as seguintes formas de endereçamento:

• **Endereço físico de 64 bits:** Por meio dos comandos AT, configura-se os campos de DH e DL (sendo respectivamente a parte mais e menos significativa do endereço de 64 bits) com o endereço do dispositivo de destino.

• **Parâmetro NI (*Name Indentifier*):** Por meio de comandos AT, utiliza- se o comando NI para editar e configurar o registrador NI, gerando um nome conveniente a aplicação. Também por meio de comandos AT configuram-se os registradores DH e DL para o atributo estabelecido como NI do dispositivo de destino.

Para o modo de operação API utiliza-se o *frame* padrão que já contém campos relacionados ao endereçamento por 64 bits, 16 bits ou NI. Tal *frame* será abordado na seção seguinte.

2.4.2.6 Modo de Operação API

Nesta seção será abordado o modo de operação API(*Application Programming Interface*) do dispositivo Digi Xbee. Este método difere do modo transparente pois a *frame* enviado ao pino de entrada de dados serial não será simplesmente direcionado ao pino de saída de dados do dispositivo de destino.

Ao invés, é utilizada uma estrutura de *frame* como interface para os diferentes comandos, A estrutura básica do *frame* API é ilustrada na Figura 16.

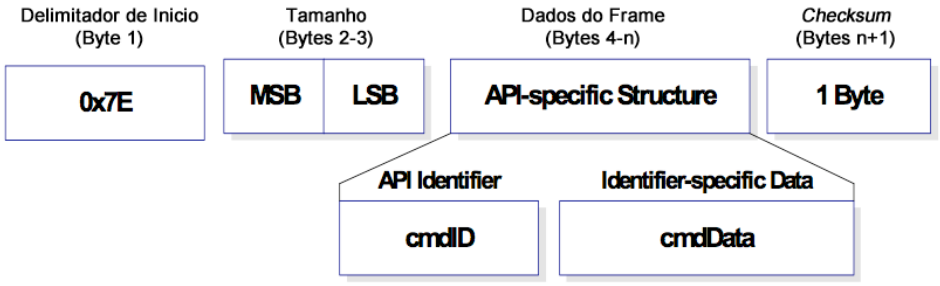

Figura 16 Estrutura do *frame* API (DIGI INTERNATIONAL,2010)

Se o *frame* não é recebido corretamente o módulo de destino responde com um *frame* de status indicando a natureza da falha. O *frame* possui os seguintes campos:

• **Delimitador do frame:** "0x7E" demarca o inicio do *frame*, qualquer dado recebido antes do delimitador não é considerado pelo dispositivo.

• **Tamanho:** indica o tamanho de dados do *frame*, este tamanho não inclui o campo de *checksum*.

• **Dados:** Variam de acordo com a mensagem, cada comando API possui um número e estrutura, a estrutura geral é demonstrada na Figura 15, e um detalhamento maior pode ser encontrado no datasheet, p-102. dos dispositivos Xbee da Digi International (2010) é encontrada uma descrição completa dos *frames*.O identificador do comando, cmdID, indica o comando do API que será executado, os comandos suportados são listados na Tabela 2

• **Checksum:** é um valor calculado para validar o *frame* recebido. É calculado somando-se os valores do *frame* de dados (excluindo delimitador e tamanho), portanto n-3 bytes, e o resultado dessa operação é subtraído do valor 0xFF. Na transmissão é realizado este algoritmo, e na recepção, o valor de *checksum* é comparado ao recebido no *frame* API.

TABELA 2 Comandos e identificadores do *frame* API (DIGI INTERNATIONAL, 2010)

API Frame Names	***API Identifier (ID)***
AT Command	0x08
AT Command - Queue Parameter Value	0x09
ZigBee Transmit Request	0x10
Explicit Addressing ZigBee Command *Frame*	0x11
Remote Command Request	0x17
Create Source Route	0x21
AT Command Response	0x88
Modem Status	0x8A
ZigBee Transmit Status	0x8B

Ao enviar um *frame* de *request*, o dispositivo que o recebe envia um *frame* de resposta de acordo com o comando API especificado. E mesmo quando a lina de comandos é remota, existe um *traceback* ad informação, conforme ilustrado na Figura 17.

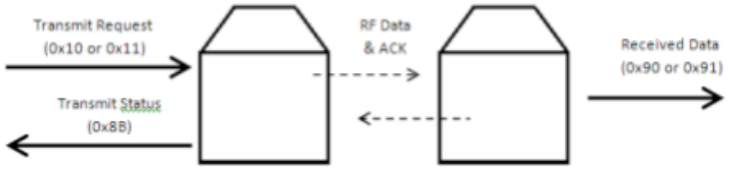

Figura 17 Ilustração da comunicação por meio de *frames* API.
Fonte: DIGI INTERNATIONAL (2010)

3 Metodologia

Neste capitulo serão apresentados os métodos para atingir o objetivo proposto. Para isso foi adotada uma abordagem sistemática para a elaboração do projeto baseada na informação apresentada anteriormente no Capítulo 2. A organização deste capitulo apresenta duas seções que apresentam de maneira linear a sequencia de atividades que antecedem a implementação do projeto.

A primeira parte aborda a funcionalidade do *software* compreendendo a análise de sistema, requisites, fluxo de dados e escolha de tecnologia. A segunda seção trata da modelagem do sistema físico mostrando as configurações, topologias e dispositivos utilizados na rede, configurações a serem feitas, juntamente metodologia dos testes para cada configuração.

3.1 Projeto de *software*

Nesta seção será abordado todo o processo de análise de sistemas realizados, análise de requisitos e detalhamento em relação ao processamento de dados realizados, seguindo o padrão de implementação em cascata. Para isso é necessário uma breve descrição geral, em termos da abordagem de engenharia de *software* utilizada. Os principais termos abordados são:

• **Usuário:** O usuário consiste no agente primário do sistema. Realiza os pedidos para o servidor *web* e é o observador primário de qualidade do sistema.

• **Coordenador:** O coordenador representa o módulo que desempenha o papel de *coordinator* na rede Zigbee. O módulo coordenador possui interface serial-USB com o módulo de desenvolvimento(base).

• **Entradas/Saídas Digitais:** O módulo de entradas, ou saídas, digitais possui a função de *end device* ou roteador dentro da rede Zigbee, sendo configurado o módulo Xbee para trabalhar diretamente com os comandos para acesso e controle dos pinos de entradas/saídas digitais.

• **Módulo de desenvolvimento/Base:** O módulo de desenvolvimento FriendlyARM será tratado, neste trabalho, também como Base, sendo responsável pelo processamento de dados, assim como o *host* do servidor *web* e controlador da interface serial-USB com o módulo coordenador da rede Zigbee

• ***Web-server*:** Principal interface com o usuário e o sistema de controle da rede Zigbee.

• ***Request*/*Response*:** Neste trabalho serão tratados como *requests* e *responses* quaisquer petições entre agentes ou módulos.

• **Xbee API:** Modo de operação do coordenador Zigbee, permitindo o controle por *frames* especificados na Norma Xbee Zigbee API.

3.1.1 Análise de Sistema

Nesta seção serão apresentados a análise de sistema do projeto e sua viabilidade de desenvolvimento e implementação. A discussão do cenário das redes sem fio de sensores, e sua necessidade já foi discutida previamente no Capítulo 1.

Assim vale destacar que este sistema primordialmente busca cumprir as necessidades propostas pelos novos desafios para a implementação das WSNs, onde se encontram eficiência energética, escalabilidade, mobilidade, robustez, segurança, estratégias de sincronização.

3.1.1.1 Análise de viabilidade técnica

Para que o sistema seja implementado, apenas considerando as necessidades anteriormente expostas (interface para o usuário interagir com uma rede sem fios de sensores), serão utilizadas tecnologias com alto grau de maturidade, ou seja, já aceitas e com grande suporte da comunidade desenvolvedora.

Para o sistema de *web-server* e *back-end* serão utilizados o pacote AMP, composto por MySQL, Apache 2 server e PHP5, sendo implementados sobre a plataforma GNU/Linux. Este conjunto de aplicações já possui grande suporte e difusão dentro da comunidade de desenvolvedores.

Como citado acima a plataforma operacional será o GNU/Linux que permite utilizar comandos do tipo *shell script* para a realização de tarefas, sendo possível a implementação modular de soluções para o sistema, assim como como funções *ad-hoc*. Desta forma, o projeto torna-se viável em termos técnicos de implementação, desenvolvimento e suporte.

3.1.1.2 Análise de viabilidade econômica

Dentre as soluções técnicas citadas acima, todas possuem licenças livres para a implementação em aplicações de terceiros. Logo, a viabilidade do sistema como todo depende de: valor de *hardware* (compra e configuração), inicialização (treinamento) e permanentes (salários, sistemas de apoio como *internet*, energia, aluguel). Para os fins deste documento somente serão abordados custos dos módulos de *hardware*, considerando que os demais são fornecidos por um terceiro agente (a universidade e o discente neste projeto).

O *hardware* proposto para a realização do sistema consiste em:

• **Módulo de desenvolvimento *FriendlyARM Tiny*6410:** responsável pela base do sistema principal de controle do sistema, possuindo o sistema operacional Linux embarcado. Contém as interfaces necessárias para a implementação do projeto (USB e conexão com a *internet*)

• **Módulo Xbee PRO Series 2:** módulo que será responsável pela comunicação wireless da rede sendo responsável pelas duas camadas inferiores segundo o modelo ISO/OSI (Física e Enlace).

• **Adaptador CON-USBEE Rogercom:** responsável pela comunicação física (conversão USB-UART) entre os módulos de comunicação wireless e a base do sistema de controle.

Como os componentes do circuito representam um modelo genérico correspondente aos sensores utilizados, não serão considerados. Já que a rede para os sensores não depende dos mesmos. E com isso, a tabela 3 mostra o custo de implementação, segundo uma pesquisa inicial .

TABELA 3 Cotação inicial de custo de aquisição de *hardware*

Qtdad	Descrição	Valor (US$)/Unidade
01	*FriendlyARM Tiny*6410	160.00
01	Rogercom CON-USBEE	50.00
03	Xbee PRO Series	225.00
	Total	275.00

3.1.1.3 Análise de viabilidade legal

Neste projeto, serão buscadas tecnologias que não possuem restrições legais, seja em relação ao uso e modificação de código. Assim o projeto não implica restrições à publicação de material desenvolvido.

3.1.2 Análise de Requisitos

Nesta seção serão apresentados os requisitos que o sistema deverá compreender em sua solução. Os requisitos são levantados de maneira a permitir uma modularidade no desenvolvimento no sistema. Os requisitos levantados são apresentados a seguir e descrevem o diagrama ilustrado na Figura 18.

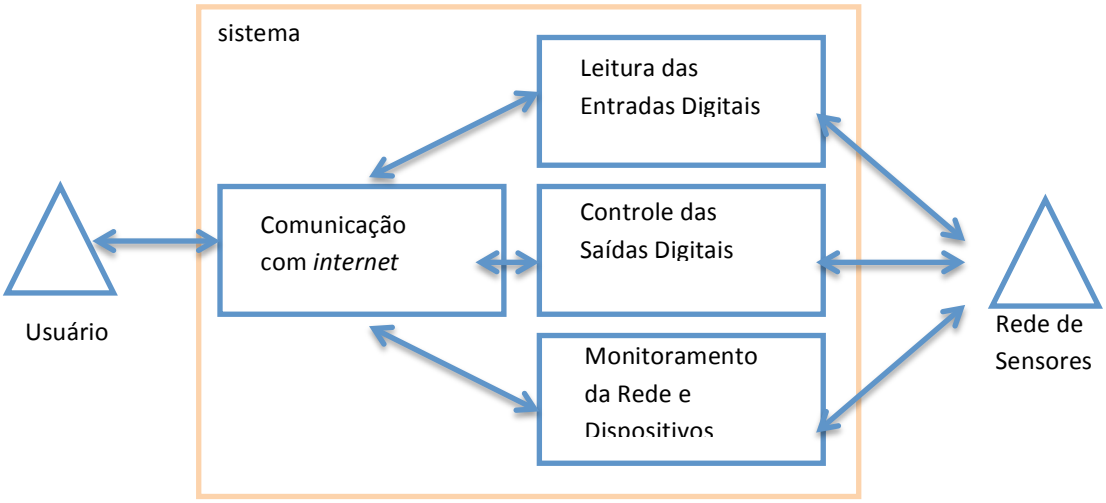

Figura 18 Diagrama de requisitos para o sistema Linux-Zigbee

3.1.2.1 Requisitos funcionais

Os requisitos funcionais correspondem as atividades que o sistema realiza de tal maneira a solucionar as necessidades propostas pelo cenário. Desta maneira, os seguintes requisitos funcionais foram avaliados:

A. Ler o valor das entradas digitais do módulo de entradas: Este requisito é formado pelo *request* do usuário/servidor para leitura de uma determinada entrada digital do módulo de entradas digitais disposto na rede de sensores.

• **Entradas:** HTTP *request* do usuário ou chamada temporizada do servidor;

• **Processamento:** Verificação do *request*, identificação do comando e transmissão para o módulo de transmissão wireless, espera pela resposta e envio de resposta para o usuário;

• **Saídas:** Valor da entrada digital.

B. Controlar o valor das saídas digitais do módulo de saída: Este requisito é formado pela petição do usuário para mudança de uma determinada saída digital do módulo de saídas digitais disposto na rede de sensores.

• **Entradas:** HTTP *request* do usuário;

• **Processamento:** Verificação do *request*, identificação do comando, transmissão para o módulo de transmissão wireless, validação da ação e envio de resposta para o usuário;

• **Saídas:** Valor da saída digital e confirmação de execução.

C. Monitorar a rede de sensores: Este requisito é formado pela petição do usuário ou pelo próprio sistema para avaliar o estado e que a rede está, com detalhes sobre os nós.

• **Entradas:** HTTP *request* do usuário ou temporizado pelo sistema;

• **Processamento:** Verificação do *request*, identificação do comando e transmissão para o módulo de transmissão wireless, espera e validação da resposta da rede wireless;

• **Saídas:** *Array* da estrutura de pontos de rede Zigbee.

3.1.2.2 Requisitos não-funcionais

Os requisitos não funcionais serão compostos pelas determinações de eficiência, robustez, segurança e atributos adicionais do sistema. Estes requisitos determinam diretamente fatores de qualidade do sistema. Os requisitos não funcionais são descritos abaixo.

• **Eficiência:** O sistema deve apresentar uma relação de sucesso/erro na execução dos *requests* do usuário/servidor de 99%

• **Tempo de resposta:** O sistema deve responder em no máximo 1s depois do recebimento da petição do usuário.

• **Robustez:** O sistema deve ser capaz de identificar erros e informa-los sem comprometer a execução contínua do serviço.

• **Segurança:** O sistema não apresentará validação de segurança de usuário em sua primeira instância, versão alpha. Sua implementação sendo arbitrária na fase beta.

No Apêndice A - Fluxogramas e diagramas referentes a análise de sistemas são apresentados os fluxogramas orientados a fluxo de dados, assim como o processo final realizado pelo sistema.

3.2 Redes de sensores e atuadores

Com o *software* inicialmente proposto, são necessários modelos de redes de sensores para teste do sistema proposto. Nos modelos serão utilizados quatro dispositivos: um dispositivo contendo as entradas digitais, um contendo as saídas digitais, um coordenador(base) e um roteador intermediário sem saídas ou entradas.

Desta forma, as topologias de rede utilizadas foram em estrela, em árvore e mesh, conforme apresentada na figura 11. A configuração a ser implementada em cada elemento da

rede, em relação aos registradores AT dos dispositivos Xbee, é apresentada nos Quadros 2, 3, 4 e 5.

TABELA 4 Configuração dos registradores AT do dispositivo Xbee coordenador

Registrador AT	Valor
ID	1234
SH	0013A200
SL	405CC166
NI	COORDINATOR
DH	0
DL	FFFF
AP	1
BD	3
D7	1
D6	1

TABELA 5 Configuração dos registradores AT do dispositivo Xbee roteador

Registrador AT	Valor
ID	1234
SH	0013A200
SL	405CC150
NI	ROUTER
DH	0
DL	0

TABELA 6 Configuração dos registradores AT do dispositivo Xbee de entradas digitais

Registrador AT	Valor
ID	1234
SH	0013A200
SL	405CA290
NI	DIGITAL_INPUT
DH	0
DL	0
D0	3
D1	3
D2	3
D3	3

TABELA 7 Configuração dos registradores AT do dispositivo Xbee de saídas digitais

Registrador AT	Valor
ID	1234
SH	0013A200
SL	405CA265
NI	DIGITAL_OUTPUT
DH	0
DL	0
D0	5
D1	5

D2	5
D3	5

A descrição completa dos registradores AT pode ser encontrada no Anexo A – Lista de comandos AT.

3.3 Metodologia de testes

Os testes realizados tem por objetivo satisfazer a proposta de solução para automação, por meio da análise do sistema em relação: ao consumo de energia; à modularidade e à escalabilidade; à integridade e robustez da comunicação e à distância entre dispositivos.

3.3.1 Teste de consumo de energia

Para a análise do consumo de energia será utilizada a medição de corrente do modulo Xbee para os seguintes estados, *sleep*(para os dispositivos finais), transmissão e recepção.

Para tal medição será utilizado um resistor em série com o pino de alimentação com o valor de *0,75Ω* e a medição de queda tensão sobre o mesmo permitirá inferir o valor de corrente. Com o valor de resistência baixo não é gerado um erro na medição da corrente, sendo a topologia utilizada representada na Figura 19.

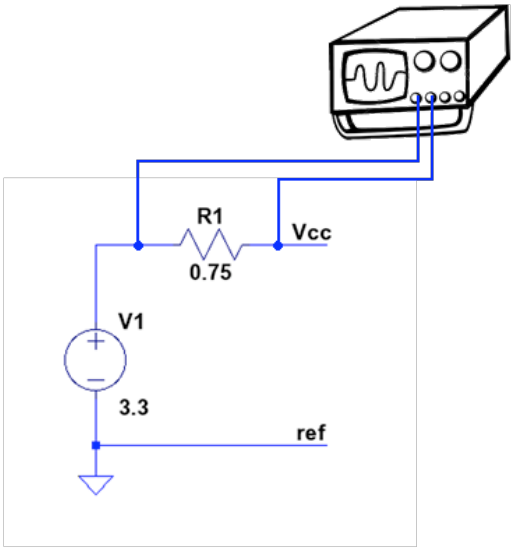

Figura 19 Topologia para medição do consumo de energia dos módulos Xbee

3.3.2 Teste de modularidade e escalabilidade

Definindo modularidade como a capacidade de trabalho em cada dispositivo de maneira particular de acordo, mantendo as características gerais aos dispositivos e

escalabilidade como a capacidade de adição e remoção de dispositivos da rede, sem comprometer a eficiência da mesma.

A protocolo Zigbee tem a capacidade de agregar novos dispositivos à rede, desde que estejam configurados para buscar a rede especifica. Portanto, para a realização desse teste, retira-se e se insere os dispositivos roteador e *end devices* e verifica-se a reinserção do dispositivo a rede.

3.3.3 Teste de integridade e robustez da comunicação

Um dos principais requisitos do sistema é a robustez em relação a comunicação de dados, para que o sistema não tome decisões incoerentes errôneas.

Para realizar esta medição cria-se uma rotina no servidor, que envia mensagens temporizadas a um dos dispositivos na rede, e é verificada a integridade da resposta, o tempo de envio e resposta.

3.3.4 Teste de distância entre módulos

Segundo a Digi International (2010), os módulos Xbee e Xbee PRO tem capacidade de alcance de 100 e 1600 metros, respectivamente, em linha visível das antenas.

Para a medição do nível de potência recebido e enviado seria necessário equipamento mais sofisticado com a finalidade de validar as medições. Desta forma serão realizados testes de comunicação e validação por inspeção visual, ou seja, uma rotina de mudança de estado em uma saída digital e confirmação visual associada a um ponto de distância.

Também será feita a medição do nível de potencia recebida pelo dispositivo remoto, enquanto o mesmo recebe o sinal. Para esta medida será utilizado o terreno adjacente ao departamento de engenharia elétrica, ilustrado na figura 20

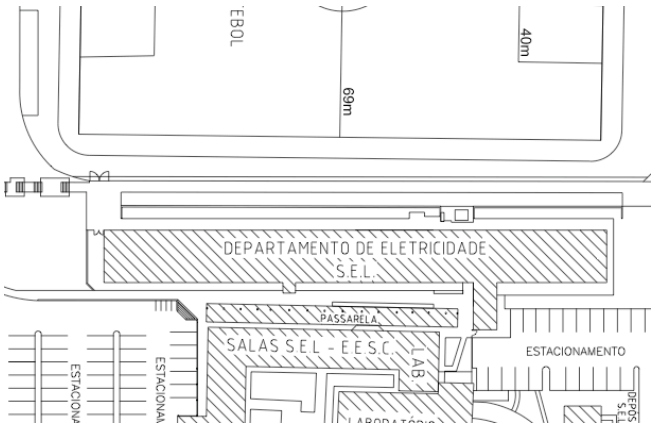

Figura 20 Planta das adjacencias do Departamente de Engenharia Elétrica

4 Implementação

Neste capítulo será abordada a implementação do sistema, assim como os resultados pertencentes à implementação em si. Este capitulo aborda: configuração do sistema GNU/Linux embarcado; a configuração e instalação da biblioteca *libxbee*, desenvolvida e mantida por Attie Grande (2012); o *software* do sistema desenvolvido em C e em linguagens *web* (HTML, CSS, JAVASCRIPT e PHP) e os circuitos utilizados para os módulos.

4.1 Circuitos dos módulos de teste

Para os testes foi desenvolvido um circuito *macro* para os *End devices*, apresentado na Figura 19. Tais circuitos possuem simplicidade suficiente para implementação em *protoboard*, não sendo necessária a impressão do circuito em placa. O resultado de tal implementação é apresentado nas figuras 21 e 22.

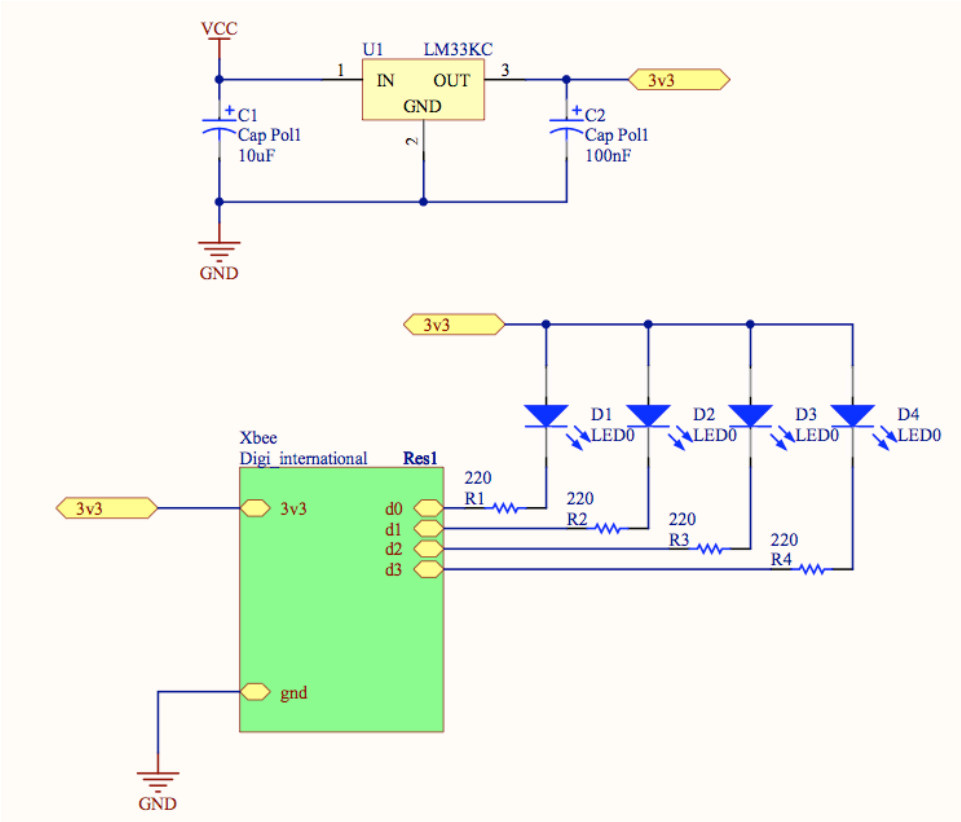

Figura 21 Circuito modelo para teste dos *End devices*

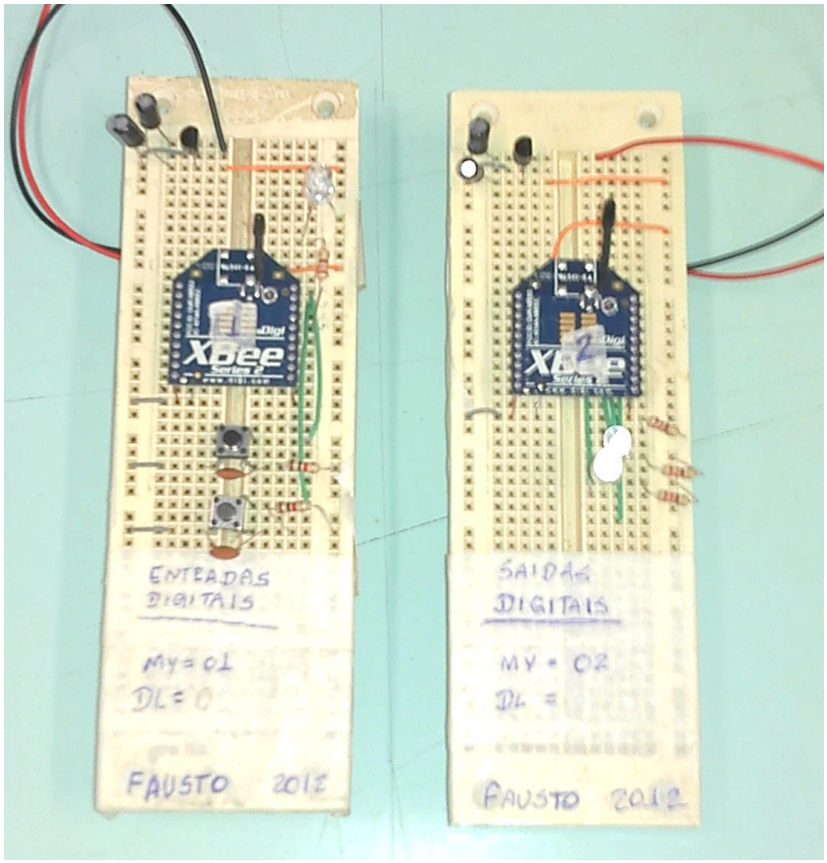

Figura 22 Montagem dos dispositivos finais em *protoboard*

Os demais módulos, roteador e coordenador, não possuem necessidade de montagem de um circuito particular, apenas a alimentação adequada. Tal alimentação é realizada pelo adaptador USB utilizado.

4.2 Comunicação serial com o modulo coordenador (RCOM CON-USBBEE)

A comunicação entre a módulo e o dispositivo coordenador da rede Zigbee é realizada por meio do adaptador ROGERCOM CON-USBBEE, mostrado na Figura 23. Para o funcionamento correto basta certificar a instalação do driver dos dispositivos FTDI, componente contido no adaptador.

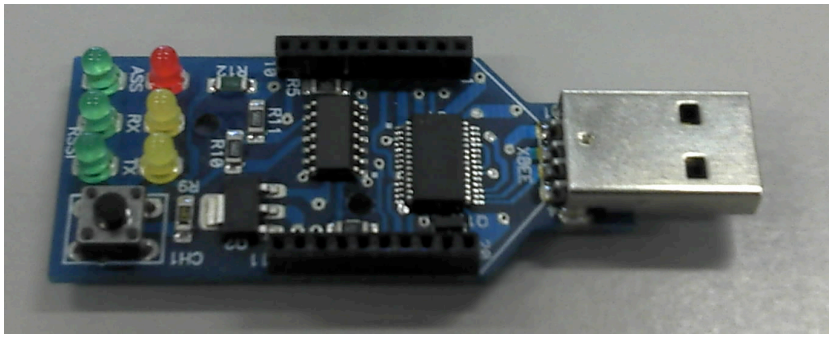

Figura 23 Adaptador Rogercom Con-USBBEE

O driver permite a criação de uma serial virtual relacionada à interface USB, logo a comunicação utilizada é, no fundo, a UART. No teste realizado para verificar o funcionamento é utilizado um terminal serial do tipo minicom, kermit, e são enviados comandos AT para o dispositivo.

A montagem no módulo de desenvolvimento *FriendlyARM tiny*6410 é mostrado na Figura 24.

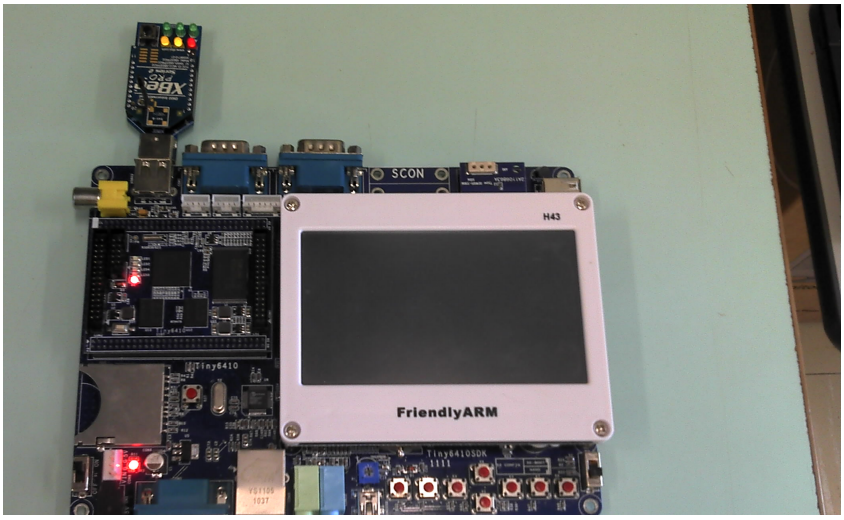

Figura 24 Dispositivo coordenador conectado no módulo de desenvolvimento *FriendlyARM Tiny6410*

O resultado é mostrado na Figura 25, mostrando a identificação do dispositivo pelo sistema operacional e validando a comunicação serial.

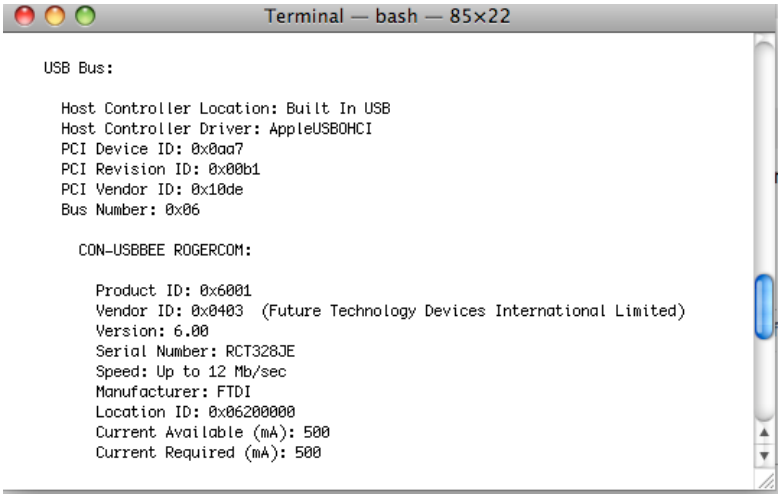

Figura 25 Resultado de conexão USB com o adaptador ROGERCOM CON-USBBEE

4.3 Compilação e validação da biblioteca *libxbee*

Para a compilação, tanto da biblioteca quanto dos códigos desenvolvidos foi utilizado o *gcc,* um compilador extremamente robusto e eficiente disponível no sistema utilizado.

Para a utilização dos modo API suportado pelos dispositivos Xbee, foi uti- lizada uma biblioteca livre, *libxbee* (ATTIE, 2012). A biblioteca gera as estruturas relacionadas ao controle, monitoramento e modelos da rede formada pelos dispositivos Xbee.

Assim, inicia-se a implementação do *software* ao compilar e validar a biblioteca *libxbee*, como a biblioteca é desenhada para ser utilizada em uma arquitetura x386 ou x64 é necessário realizar configurações de *cross-compile* para a compilação da biblioteca para a arquitetura ARM.

Dentro dos arquivos de configuração, o arquivo configure.mk controlam o grupo de *flags* relacionados a arquitetura do sistema. Edita-se a seguinte *flag*:

CROSS_COMPILE?= arm-linux-

Ao configura-los adequadamente o processo de compilação foi realizado pelos comandos:

make configure
make
make install (neste ponto é necessário ser *superuser* ou root)

Desta maneira são instalados: a biblioteca estática(libxbee.a) , uma biblioteca compartilhada(libxbee.so) e a documentação *man* para a comunicação utilizando o modo API para programas desenvolvidos em C/C++, com o compilador *gcc*.(ATTIE, 2012)

Para executar a compilação é necessário “linkar” a biblioteca ao processo, com apresentado no exemplo abaixo:

gcc my_code.c -lxbee -lpthread -lrt -o my_executable

A implementação também terá caráter sistemático, a fim de minimizar os erros e tornar o processo o mais linear. Inicialmente é validada a biblioteca libxbee, posteriormente o serviço *web* e suas configurações e por fim a conexão entre o sistema e o serviço *web*.

4.3.1 Aplicação de teste *simple−at*

Para validar a biblioteca *libxbee* é testado um *software* que permite verificar o funcionamento das estruturas principais. O programa implementado é chamado *simple−at.c*, sendo uma modificação do teste padrão proposto pelo autor da biblioteca para o teste inicial.

O *software* deve realizar uma petição ao dispositivo local, realizando um comando AT dentro do *frame* API, o comando API realizado é o 0x08 - AT Command (DIGI INTERNATIONAL, 2010). O código completo encontra-se no Apêndice B - Arquivos fonte utilizados no sistema e nos testes.

Na Figura 26 é ilustrado o processo para a utilização da biblioteca de forma adequada e na figura Figura 27 são mostrados os resultados retornados pelo *software*, impressos no terminal via SSH.

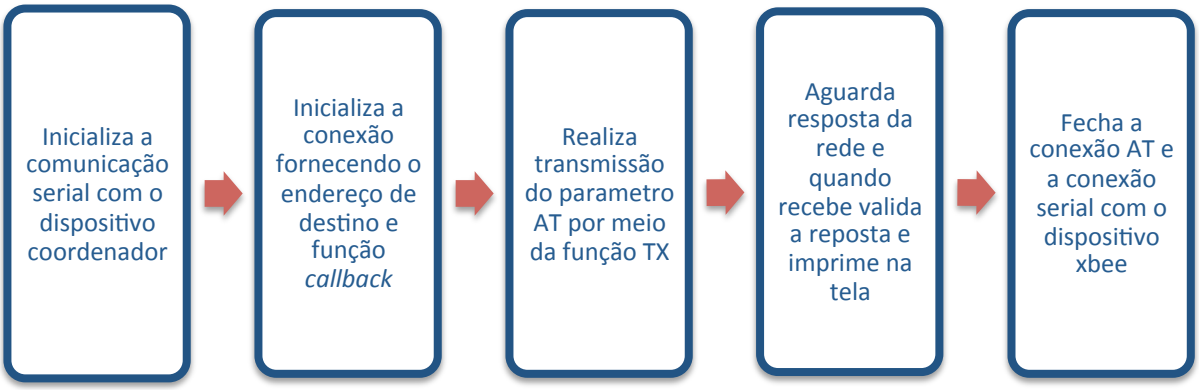

Figura 26 Processos simplificados de utilização da biblioteca *libxbee*

Desta forma, percebe-se que a biblioteca está instalada e funcionando de acordo com o proposto. Para as demais ações do sistema, serão necessárias outras funções da biblioteca e serão discutidas nos resultados.

```
root@FriendlyARM:~/fausto_xbee/Codigo# ./t_simple_at
Envia comando para teste no nome do dispositivo:
TX: [ATNI <CR>]

erro na transmissao: 0 (No error)
rx: [ COORDINATOR]
root@FriendlyARM:~/fausto_xbee/Codigo#
```

Figura 27 Resultado do código *simple-at.c*

4.4 Sistema desenvolvido em C do sistema de controle da rede Zigbee

Nesta seção o código implementado tem por objetivo saciar os requisitos levantados no Capítulo 3 – Metodologia. Para isso são abordadas algumas etapas desenvolvidas e funcionamento geral, porém o código fonte completo pode ser encontrado no Apêndice B - Arquivos fonte utilizados no sistema e nos testes.

4.4.1 Inicialização

Inicialmente, são geradas as variáveis globais e locais da função *main.c.* Também no processo de inicialização são levantados os endereços dos dispositivos da rede, e é aberta a conexão xbee com o módulo coordenador. Também são criadas as conexões do tipo "I/O" com os módulos de entradas e saídas digitais, estas conexões permitem "setar" um *callback* para tratar o recebimento de mudança de estado nas saídas digitais. A figura 28 mostra os trechos do código que inicializam as conexões xbee e "I/O" assim como o link com a função callback para as função *myCB_simpleIO()*.

```
// Opens the conection to the usb
if ((ret = xbee_setup(&xbee, "xbee2", "/dev/ttyUSB0", 9600)) != XBEE_ENONE) {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
}

// ****************************Create conections to the Xbee device
// ****************************IO conection to the end-devices
// Address 1 - Digital Input
if ((ret = xbee_conNew(xbee, &con_IO_1, "I/O", &address1)) != XBEE_ENONE) {
    xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
    return ret;
}
if ((ret = xbee_conCallbackSet(con_IO_1, myCB_simpleIO, NULL)) != XBEE_ENONE) {
    xbee_log(xbee, -1, "xbee_conCallbackSet() returned: %d", ret);
    return ret;
}
```

Figura 28 Trecho do código de inicialização implementado em C

4.4.2 Varredura inicial

A varredura é realizada por duas funções implementadas de forma que são criadas as conexões AT para cada *request.* O tráfego de informação entre as funções é realizado pela utilização de variáveis globais, assim o uso de memória torna-se mais rígido, porém mais simples de se desenvolver.

A varredura inicial tem por objetivo verificar quais dos dispositivos pré-relacionados, já estão conectados à rede assim como está o estado das saídas e entradas de cada dispositivo. Na chamada das funções é enviado o índice relacionado ao dispositivo xbee de destino, este índice está relacionado à estrutura de dados criada *my_xbee*.

Na figura 29 é apresentado o código relacionado a chamada de tais funções.

```
//***********Remote AT to the end-devices
for (j=0;j<3;j++) {
  send_at_command(j,"NI");
  for (i=0;i<20;i++) {
    my_xbee[j].NI[i] = aux_array[i];
  }
}

//***********Read the digital status
for (j=0;j<2;j++) {
  read_digital_port(j);
  sleep(2);
  printf("my_xbee[%d].D:%s\n",j,my_xbee[j].D);
}
```

Figura 29 Código que realiza a chamada das funções para a varredura inicial

4.4.3 *Loop* principal

No *loop* principal é composto pela temporização para o monitoramento dos dispositivos da rede. Também são realizadas rotinas para a verificação se existe algum *request* do usuário ou do servidor para a atuação em alguma saída de um dispositivo da rede.

As rotinas que realizam a leitura se existe um *request* do usuário e criam as estruturas para disponibilizar a informação para o servidor serão abordadas na seção 4.6 Integração entre os sistemas.

Na figura 30 são apresentados trechos do código implementado dentro do loop principal, sem abordar as rotinas de integração com o servidor *web*.

```
// Iniciate Loop
while (1)  {
    sleep(10);
    //sleep(60);
      (...)
    //***********Remote AT to the end-devices
    if (count >= 6) {
      count = 0;
      for (j=0;j<3;j++) {
        my_xbee[j].status = 0;
        send_at_command(j,"NI");
        for (i=0;i<20;i++) {
          my_xbee[j].NI[i] = aux_array[i];
        }
      }
    } else {
      count++;
    }

    // Checks for changes in the outputs
    need_output_update = 0;
    for (i=0;i<4;i++) {
      output_mask[i]=0;
      if (temp_output[i]-4 != my_xbee[1].D[i]-0x30) {
        need_output_update = 1;
        output_mask[i]=1;
      }
    }
```

Figura 30 Trechos do código do loop principal em C

4.4.4 Funções de chamadas e *callbacks*

Para a leitura das portas e envio de comandos AT, que são as principais estruturas API utilizadas, foram criados procedimentos que realizam o processo estabelecido na Figura 20, onde ocorre a abertura de conexão, realização da comunicação e fechamento da conexão.

A implementação destas funções está disponível no Apêndice B. Conforme citado anteriormente, o trafego entre os dados recebidos e retornados por estas funções é feito por meio de variáveis globais, e utilizado um *buffer* de bytes "aux_array".

As funções de *callback* são inseridas quando existe uma conexão definida, e espera-se uma resposta da rede Zigbee. Portanto é estimado um tempo de resposta máximo, e se a função *callback* não for chamada dentro deste período um erro é sinalizado.

4.5 *Software web*

O *software web* proposto tem por objetivo interagir com o sistema implementado em C que efetua o controle sobre a rede Zigbee. Neste ponto uma das páginas desenvolvidas serve a este objetivo sendo a seção "Rede" no site.

As demais seções tem por objetivo servir como um *hot-site* para o projeto, permitindo ao usuário entender o contexto do projeto, assim como ter acesso facilitado aos arquivos e documentação gerada. A implementação, dessa forma, transcende o objetivo proposto inicialmente, mas não o modifica quanto a premissa inicial de integração do serviço *web* para o controle e monitoramento de uma rede de sensores sem fio.

Os códigos completos estão disponíveis no Apêndice C – Código-fonte implementado no servidor *web – Front-end*.

4.5.1 *Back-end*: estrutura PHP e AJAX

Conforme citado anteriormente, o projeto utilizará a tecnologia PHP para o desenvolvimento do *back-end* da aplicação *web*. O projeto foi realizado utilizando a linguagem PHP sem nenhum tipo de *framework* de trabalho, porém seguindo alguns dos conceitos propostos numa aplicação MVC (*Model-View-Controller)*.

Também foi utilizada a tecnologia de chamadas Ajax, que permitem chamadas assíncronas ao servidor, sem que ocorra a atualização total da página no cliente (*web browser*). Desta forma, a experiência do usuário é favorecida por uma navegação mais suave.

Foi implementada a estrutura do dispositivo Xbee em PHP, assim independentemente do meio de integração entre os sistemas é possível trabalhar com uma estrutura própria.

Assim, foi gerada a classe Xbee, com os principais registradores AT utilizados, e por meio dos seguintes comandos são criadas as classes e objetos no arquivo PHP:

```
$xbee = Xbee::factory('order','status', 'SH', 'SL', 'NI','D');
$my_xbee[$i] = call_user_func_array(array($xbee, 'create'),$data);
```

Sendo que a variável $data contêm a informação fornecida pelo sistema de controle da rede e a variável $my_xbee é uma *array* que guardará todos os objetos Xbee da rede.

Para imprimir os valores obtidos no arquivo HTML, é utilizada a função *echo* ou pelos delimitadores de impressão "<?=" quando somente a informação dinâmica é levantada. A figura 31 mostra um trecho de código implementado no arquivo rede.php que utiliza as duas formas de impressão de informação para editar o HTML final.

```
<div class="xbee_data" id="router">
    <img class="xbee_status" id="router" src=
    <?php
        $my_xbee[2]->status ? $status = "on" : $status = "off";
        echo "images/network/status_".$status.".png"
    ?>
    ></img>
    <div class="xbee_at">
        STATUS: <?= $status ?><br/>
        NI: <?= $my_xbee[2]->NI ?><br/>
        SH: <?= $my_xbee[2]->SH ?><br/>
        SL: <?= $my_xbee[2]->SL ?><br/>
    </div>
</div>
```

Figura 31 Trecho do código PHP para tornar dinâmica a geração do HTML final

O a recepção das chamadas AJAX são semelhantes em relação ao processamento, porém diferem quanto a informação enviada para o cliente. Ao invés de enviar a página HTML resultante é enviado uma estrutura JSON, tal estrutura é, em aplicação, um objeto javascript formatado como string, contendo dados no formato: { *parameter : value;*}. O código abaixo contém um trecho do código utilizado para responder uma chamada Ajax.

```
echo json_encode(Array("id" => $output_id, "value" => $data[$output_id]));
```

Desta forma é possível gerar um serviço não limitado a página *web*, permitindo a utilização dos dados da rede por sites, clientes e aplicações diversas. A organização das chamadas é controlada pelo servidor *web* Apache, não sendo abordada nesse trabalho.

4.5.2 *Front-end*: HTML, CSS e JavaScript

Nesta seção é mostrada a implementação *front-end* do projeto. Conforme citado anteriormente, foi criado um hot-site para o projeto. Assim, foram estipuladas quatro seções: Inicio, Projeto, Rede e Contato.

Na seção de Inicio é contextualizado o projeto, o porquê está sendo realizado o projeto, por quem e outras informações gerais. Na seção projeto são encontrados links relacionados ao projeto, assim como os arquivos fonte e este documento. A figura 32 mostra o resultado obtido na página de projeto.

Figura 32 Seção de projeto do hotsite

A seção de rede contêm, propriamente, o sistema de interação e monitoramento da rede de sensores com protocolo Zigbee, são exibidos os módulos, seus status, seus principais dados e botões para atuação das saídas do respectivo dispositivo. A figura 33 mostra o resultado desta implementação.

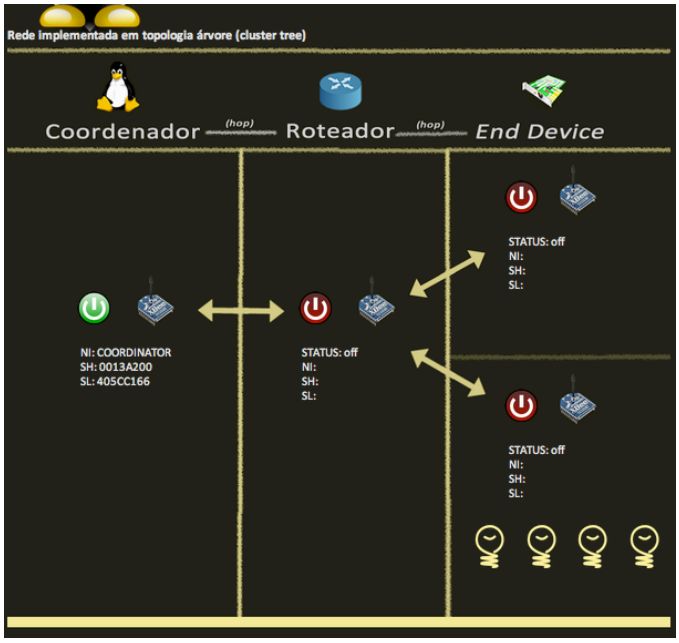

Figura 33 Seção de rede com implementação de acesso remoto a rede Zigbee

Conforme citado anteriormente, foi implementada uma chamada Ajax ao servidor para a atualização dos dados da rede para o usuário. Para esta chamada foi utilizada a biblioteca *JQuery* (JQUERY, 2012), extremamente utilizada na *internet* para implementação em *Javascript*. O código-fonte implementado no *front-end* está disponível no Apêndice C – Código-fonte implementado no servidor *web – Front-end.*

A seção Contato fornece dados e um formulário para o contato de pessoas interessadas sobre o projeto.

4.6 Integração entre os sistemas

Neste ponto existem dois sistemas operando de maneira independente. Para realizar a integração entre os mesmos foram utilizados arquivos, devido a simplicidade do trabalho e geração de uma estrutura aplicável a base de dados mais complexas.

Ao ter-se dois sistemas diferentes operando sobre o mesmo arquivo notou-se problemas quanto as permissões assim como a integridade dos dados. Dessa forma optou-se pela estrutura apresentada na figura 34, onde somente um sistema tem permissão de escrita sobre um determinado arquivo, mas é permitida a leitura pelos demais.

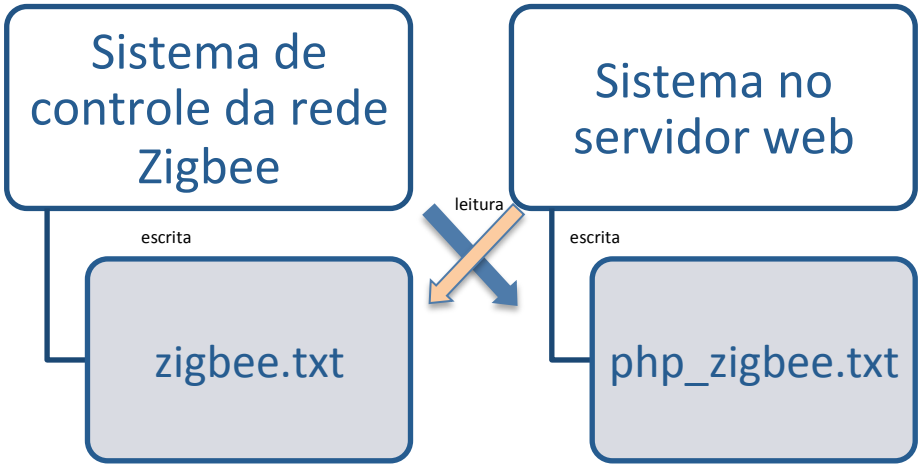

Figura 34 Relação entre os arquivos e os sistemas

O primeiro arquivo, *zigbee.txt,* contém informações que o sistema implementado em C, de controle da rede Zigbee. Desta forma o único que tem permissão de escrita neste arquivo é o sistema em C e o servidor *web* somente realiza a leitura do arquivo.

O segundo arquivo, *php_zigbee.txt,* permite a escrita pelo servidor *web* e tem por função informar ao sistema que controla a rede Zigbee de eventuais mudanças, tais como mudança de uma saída digital, ou atualização por comando AT.

Desta forma foram implementadas rotinas para trabalhar com os arquivos nos dois sistemas. As Figuras 35 e 36 contêm tais rotinas no sistema em C e no sistema do servidor we PHP, respectivamente.

```
if (save_on_file) {
  // Saves on file
  fp = fopen ("zigbee.txt","w");
  if (fp) {
    fprintf(fp,"1.%d.%X%X%X%X.%X%X%X%X.%s.%s\n",my_xbee[0].status,
            my_xbee[0].SH[0],my_xbee[0].SH[1],my_xbee[0].SH[2],my_xbee[0].SH[3],
            my_xbee[0].SL[0],my_xbee[0].SL[1],my_xbee[0].SL[2],my_xbee[0].SL[3],
            my_xbee[0].NI, my_xbee[0].D);
    fprintf(fp,"2.%d.%X%X%X%X.%X%X%X%X.%s.%s\n",my_xbee[1].status,
            my_xbee[1].SH[0],my_xbee[1].SH[1],my_xbee[1].SH[2],my_xbee[1].SH[3],
            my_xbee[1].SL[0],my_xbee[1].SL[1],my_xbee[1].SL[2],my_xbee[1].SL[3],
            my_xbee[1].NI,my_xbee[1].D);
    fprintf(fp,"3.%d.%X%X%X%X.%X%X%X%X.%s.%s\n",my_xbee[2].status,
            my_xbee[2].SH[0],my_xbee[2].SH[1],my_xbee[2].SH[2],my_xbee[2].SH[3],
            my_xbee[2].SL[0],my_xbee[2].SL[1],my_xbee[2].SL[2],my_xbee[2].SL[3],
            my_xbee[2].NI,my_xbee[2].D);
    fclose(fp);

  }
  save_on_file = 0;
}
```

Figura 35 Trecho do código do sistema de controle da rede para escrita no arquivo *zigbee.txt*

```php
//Opens the file php_zigbee.txt to read the information
while(!is_readable($root_file_path.'php_zigbee.txt')) ;
try {
    $handle = fopen($root_file_path.'php_zigbee.txt','r');
    if ($handle) {
        $full_data  = fgets($handle);
        fclose($handle);
        $data = explode(" ", $full_data);
        //var_dump($full_data);
        foreach ($data as $key => $element) $data[$key] = trim($element);
        $data[$output_id] == "4" ? $data[$output_id] = "5" : $data[$output_id] = "4";
        $full_data = implode(" ",$data);
        //var_dump($full_data);
        while(!is_writable($root_file_path.'php_zigbee.txt')) ;
        $handle = fopen($root_file_path.'php_zigbee.txt','w');
        fprintf($handle,"%s",$full_data);
        fclose($handle);
    }
} catch (Exception $e) {
    echo 'Exception:',  $e->getMessage(), "\n";
}
```

Figura 36 Trecho do código do sistema *web* da rede para escrita no arquivo *php_zigbee.txt*

5 Testes e resultados

Neste capítulo são apresentados os resultados dos testes realizados para validar a proposta deste trabalho. Os resultados foram divididos segundo as propostas de testes do Capítulo 3 – Metodologia.

5.1 Consumo de energia

Este teste foi realizado pela medição de corrente consumida por um módulo Xbee. Conforme observado na figura 37, a medição pelo resistor em serie não foi um método eficiente para a observação do consumo de corrente no módulo nos diferentes modos de operação que ele efetua durante a recepção e transmissão.

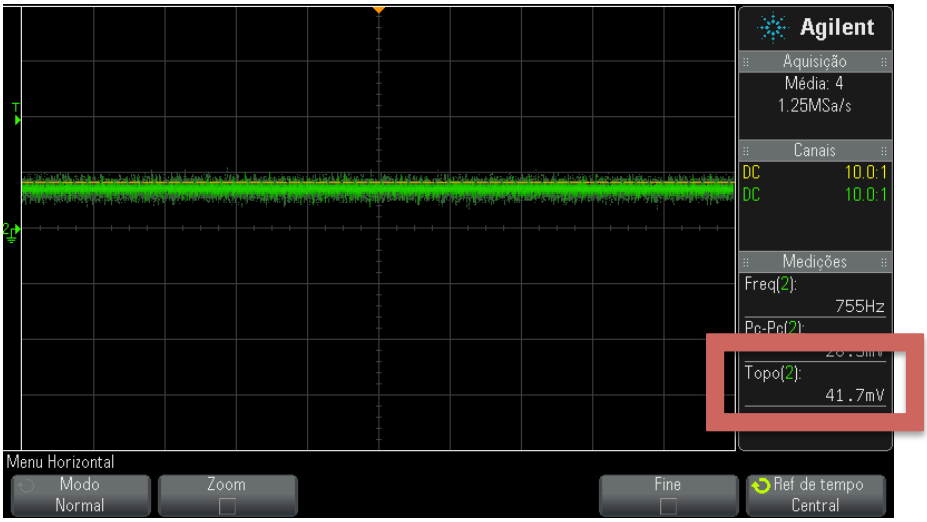

Figura 37 Medição do consumo de corrente médio para o dispositivo Xbee como roteador

Porém, foi possível determinar a corrente média do dispositivo. Realizando uma transmissão a cada 5 minutos, o valor de corrente da transmissão é diluído dentro do consumo base de energia, sendo expressa segundo a tabela 8.

TABELA 8 Resultado do consumo médio de corrente por dispositivo Zigbee

Dispositivo	Consumo Médio
Xbee (*End device*)	2mA
Xbee (Roteador)	54.6mA
Xbee PRO (roteador e coordenador)	150mA

Tais valores se apresentam dentro do esperado, para cada tipo de dispositivo (Xbee ou Xbee PRO) e a função desempenhada pelo mesmo, segundo o *datasheet*.

5.2 Integridade e robustez da comunicação

O teste de integridade de comunicação foi realizado pelo envio temporizado do comando de mudança da saída digital e confirmação da recepção e efetuação da mudança de estado.

Ao receber os dados o pacote, o dispositivo de destino envia a resposta por um *frame* API, porém isto é suficiente para a confirmação de execução do comando. Portanto, é criado o *callback* na conexão do tipo "I/O" tornando o processo de resposta redundante, permitindo a confirmação de mudança de estado.

O teste foi realizado utilizando janelas de envio de 100 pacotes, ou seja, gerando um total de 100 comandos de variação de estado em uma saída digital. O resultado obtido foi positivo. Dentro da janela amostral máxima, determinada pela velocidade serial o índice de erro foi nulo.

5.3 Modularidade e Escalabilidade

O teste de modularidade foi realizado pela retirada e reinserção de um dispositivo na rede Zigbee. O protocolo realiza o controle dos dispositivos na rede, sem a interferência do sistema de controle. Desta forma não foi possível realizar a medição precisa dos processos de *beaconning* e *association request*, e sim, somente o resultado final de reinserção a rede.

A rede foi capaz de realizar tal processo, porém o tempo total para este processo apresentou grande variação. Isto pode dever-se ao tempo necessário para envio do sinal de *beacon*. Tal intervalo de tempo, apesar de grande variação durante os testes não ultrapassou um minuto.

A escalabilidade mostrou-se satisfatória dentro do protótipo implementado, não sendo válidada para redes com mais dispositivos.

5.4 Distância máxima de comunicação entre dispositivos

Por meio do comando AT "DB" pode-se ver o nível de potencia recebido no ultimo *hop* recebido, esta informação não identifica toda a potência total utilizada na comunicação porém serve de medida quando utilizada na comunicação ponto-a-ponto.

Desta forma, foram escolhidos pontos conhecidos segundo a planta apresentada na seção de Metodologia de Testes, e os ensaios foram realizados para os dispositivos Xbee PRO e Xbee. Os resultados obtidos são mostrados na tabela 9, e uma comparação é feita no gráfico apresentado na figura 38.

TABELA 9 Potência do sinal recebido pelo comando AT DB para os diferentes dispositivos

DISTÂNCIA (m)	XBEE	XBEE S/ OBST	XBEE PRO	XBEE PRO S/ OBST
1	-53	-26	-17	-15
3	-59	-37	-37	-19
6	-65	-46	-49	-25
10	-72	-52	-52	-32
15	-74	-58	-57	-39
20	-82	-62	-62	-47
30	--	-70	-73	-56
40	--	-77	-78	-58
50	--	-81	--	-62
60	--	--	--	-65
70	--	--	--	-67
80	--	--	--	-70
90	--	--	--	-74
100	--	--	--	-79

* dados em dBm.

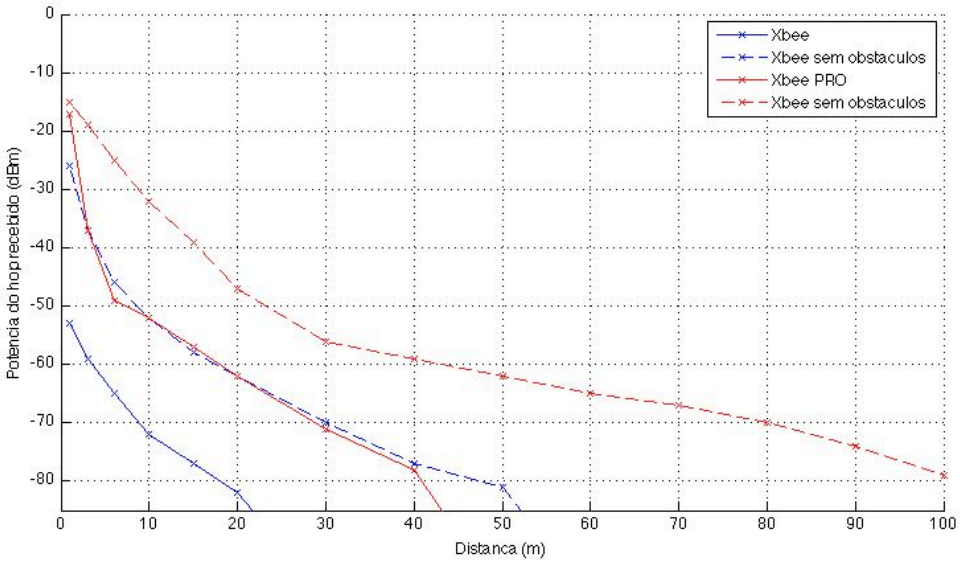

Figura 38 Resultado comparativos entre os dispositivos Xbee e Xbee PRO no teste de distância

6 Discussão e Conclusões

Neste trabalho foi possível implementar um sistema para o acesso remoto a uma rede de sensores sem fio, utilizando o protocolo Zigbee e o modo de operação API para os dispositivos Xbee da Digi International. O sistema final possui robustez (integridade na comunicação dos dados) e aplicabilidade no contexto das *WSNs*, considerando os critérios de consumo de energia, escalabilidade (dentro da limitação do protótipo implementado), modularidade, distância de alcance de transmissão e robustez do sistema.

Apesar da metodologia adotada não ter sido eficiente para a medição da energia consumida nas diferentes etapas, pode se utilizar os valores médios encontrados para uma estimação do consumo em produção do sistema de automação. O valor de consumo de 2mA para um dispositivo final é um valor significativo para o critério de baixo consumo de energia numa rede de sensores, sendo aplicável a diversas áreas.

Mesmo dentro das limitações impostas pela medição da distância máxima, o sistema representa uma boa solução para automação sem fio quando a distância entre os pontos não é muito elevada. E caso, as distâncias sejam muito grandes outras soluções de antena e radiação podem ser abordadas.

Dentro do desenvolvimento de *software*, o projeto e a implementação do sistema tiveram dois âmbitos distintos, a criação do *software* de controle da rede e a interface com o usuário. Dentro deste paradigma o desenvolvimento foi feito em paralelo transformando-se em duas aplicações separadas com definições especificas, e o trabalho de integração foi responsável pelo funcionamento do sistema como um todo.

Em relação a parte física utilizada o módulo de desenvolvimento *FriendlyARM Tiny*6410 apresentou falhas em relação a utilização constante da memória Flash(conforme citado na seção 5.4), correlacionando com outros projetos que também utilizam este módulo de desenvolvimento e apresentaram resultados semelhantes indicando um problema no *hardware*. Portanto, recomenda-se a substituição do módulo de desenvolvimento por outro.

Este projeto permite a implementação e criação de novos projetos com a rede Zigbee implementada, entre os possíveis projetos futuros destacam-se:

• **Testes de segurança:** Realização de testes de ataque a rede, por meio da frequência de transmissão utilizada pela rede, assim como pela *internet* ao sistema.

• **Implementação de Banco de Dados integrado:** Permitir que a base de dados seja única para o sistema de controle da rede, assim como para o sistema de serviço *web*, tornando a aplicação mais genérica e robusta

• **Integração com outros sistemas:** Integração com outros protocolos de comunicação como *Wi-fi, Bluetooth*, *CAN* entre outros.

• **Implementação de um controle móvel:** Utilização de um módulo que possua interface com o usuário por meio de um *touchscreen* e permita o usuário realizar comandos diretamente dentro da rede Zigbee.

7 Referências Bibliográficas

ADAMS, J. Building low power into wireless sensor networks using Zigbee technology. **Industrial Embedded Systems Resource Guide, Networking: Technology**, pp. 26-30, 2005.

ATTIE. *Attie.co.uk,* 2012. Disponível em: <http://attie.co.uk/libxbee>

BARR, M.; MASSA, A. J. **"Programming Embedded Systems: with C and GNU development"**, O'Reilly Media, Pequim, 2007.

CUNHA, A. R. On the use of IEEE 802.15.4/Zigbee as federating communication protocols for Wireless Sensor Networks, **Doctorate Thesis**, FEUP-UP, 2007

CULTER T. Deploying Zigbee in existing industrial automation networks. **Industrial Embedded System Resource Guide, Networking: Technology**, pp. 34-36, 2005.

CURVELLO, A. M. L; SANTOS, F. P. **ARM web-tv**, Monografia, Aplicações de Microprocessadores II, SEL-EESC-USP, 2011

ELECTRONS, F. *Free Electrons – Get the best of your hardware, 2012*. Disponível em: <http://free-electrons.com/>

_____. FRIENDLYARM. ***FriendlyARM Forum***, 2012. Disponível em: <http://www.friendlyarm.net/forum>

_____. FRIENDLYARM. **Tiny6410 User manual**, v1.0, 2010.

GIRIO, M. G. Utilização do sistema operacional tempo real MQX embarcado para aplicações de telemetria. **Trabalho de Conclusão de Curso**, EESC-USP, 2010. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-18112011-110444/ >

GUIA DO HARDWARE. *SLC, MLC e TLC: Por que as memórias Flash estão ficando piores, 2012*. Disponível em: <http://www.hardware.com.br/tutoriais/slc-mlc-tlc/>

_____. IEEE-TG15.4, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs**). IEEE standard for Information Technology**, 2003.

JQUERY. *jQuery: The Write Less, Do More, JavaScript Library*, 2012. Disponivel em: < http://www.jquery.com/>

_____. LEADERS.When everything connects: Information technology has nothing to lose but its cables. **The Economist**. Abril, 2007. Disponível em: < http://www.economist.com/node/9080024>

LINUX, *Viva o*. Disponível em: <http://www.vivaolinux.com.br/linux/>

ROGERCOM. *Rogercom - O Maior conteúdo brasileiro sobre Porta Paralela*, 2012. Disponível em: < http://www.rogercom.com/ZigBee/ZigBee.htm>

U-BOOT. *Das U-Boot – the Universal Boot Loader,* 2012. Disponível em: <http://www.denx.de/wiki/U-Boot/WebHome>

ZHENG, J.; MYUNG, J. L. Will IEEE 802.15.4 Make Ubiquitous Networking a Reality? A Discussion on a Potential Low Power, Low Bit Rate Standard, **IEEE Communications Magazine**, vol. 42, No. 6, pp. 140- 146, 2004.

ZIGBEE ALIANCE, **Zigbee Specification,** 2006. Disponivel em: <http://www.zigbee.org/>

8 Apêndice A – Fluxogramas e diagramas referentes a análise de sistemas

Neste Apêndice são encontrado os principais diagramas de representação dos requisitos funcionais do sistema. Os requisitos funcionais determinados no Capítulo 3 foram:

1. Ler o valor das entradas digitais do módulo de entradas: Este requisito é formado pelo *request* do usuário/servidor para leitura de uma determinada entrada digital do módulo de entradas digitais disposto na rede de sensores.

2. Controlar o valor das saídas digitais do módulo de saída: Este requisito é formado pela petição do usuário para mudança de uma determinada saída digital do módulo de saídas digitais disposto na rede de sensores.

3. Monitorar a rede de sensores: Este requisito é formado pela petição do usuário ou pelo próprio sistema para avaliar o estado e que a rede está, com detalhes sobre os nós.

Assim, a Figura 38 representa os diagrama de fluxo de dados para os requisitos enumerados. Apenas uma estrutura macro de fluxo de dados é necessária para os três requisitos pois existe um padrão de funcionamento dentro da operação API.

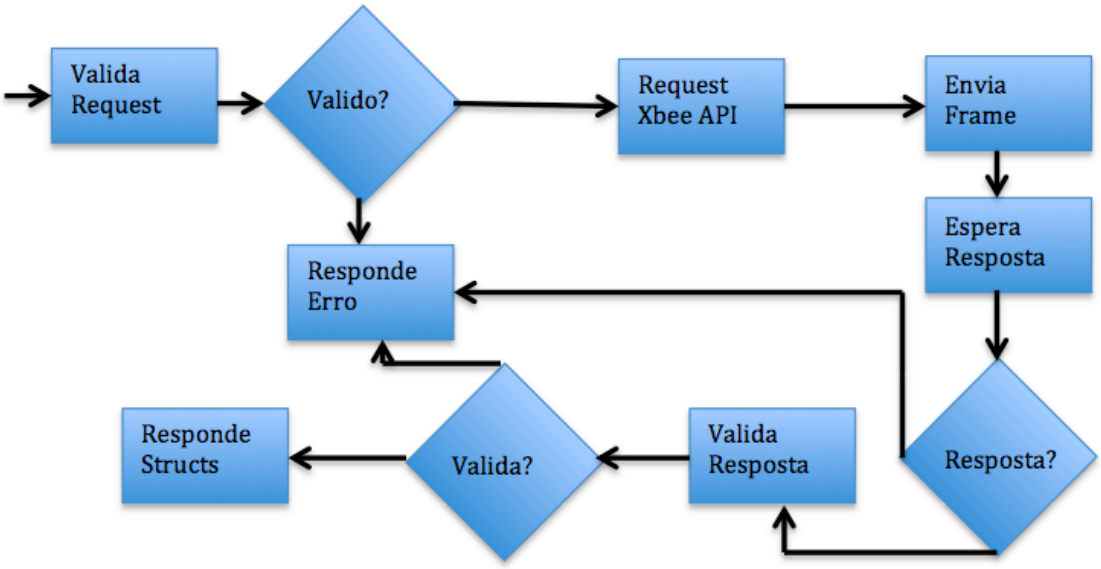

Figura 39 Diagrama de fluxo de dados para a aplicação de comando API

A resposta pode portanto ser constituída de uma mensagem de erro ou de um estrutura contendo: valores, confirmação de ação ou estrutura dos dispositivos da rede de acordo com o processo realizado segundo os acima enumerados, respectivamente.

É importante também ressaltar o funcionamento do recebimento do *request* pelo sistema do servidor *web*, o processo macro de fluxo de dados é exibido na Figura 39.

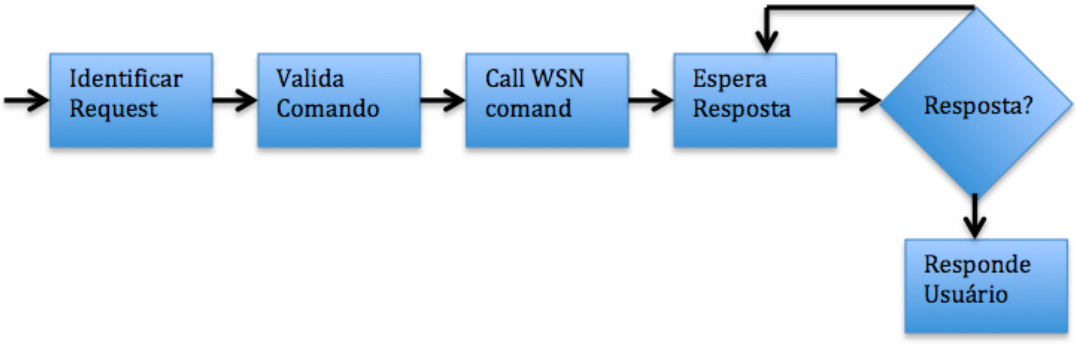

Figura 40 Diagrama de fluxo de dados para processamento do *request* do usuário

9 Apêndice B – Código-fonte utilizado no sistema e nos testes

Neste apêndice são apresentados os códigos-fontes utilizados para a implementação, testes e o arquivo final do sistema de controle da rede Zigbee.

ARQUIVO: INDENTIFY.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <xbee.h>

void myCB(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt
**pkt, void **data) {
    xbee_err ret;
    char *ni;
    struct xbee_conAddress *addr;

    printf("An XBee joined the network!\n");

    if ((ret = xbee_pktDataGet(*pkt, "NI", 0, 0, (void**)&ni)) ==
XBEE_ENONE && ni != NULL) {
        printf("  It is called: [%s]\n", ni);
    } else {
        printf("  Error while retrieving its NI - %d (%s)\n", ret,
xbee_errorToStr(ret));
    }

    /* you could also use 'Address (16-bit)' or 'Address (64-bit)' to get
the raw byte arrays */
    if ((ret = xbee_pktDataGet(*pkt, "Address", 0, 0, (void**)&addr))
== XBEE_ENONE && addr != NULL) {
        printf("  It's address is:\n");
        if (addr->addr16_enabled) {
            printf("      16-bit address:    0x%02X%02X\n", addr-
>addr16[0], addr->addr16[1]);
        } else {
            printf("   16-bit address: --\n");
        }
        if (addr->addr64_enabled) {
            printf("      64-bit address:    0x%02X%02X%02X%02X
0x%02X%02X%02X%02X\n",



              addr->addr64[0], addr->addr64[1], addr->addr64[2], addr-
>addr64[3],



              addr->addr64[4], addr->addr64[5], addr->addr64[6], addr-
>addr64[7]);
        } else {
            printf("   64-bit address: --\n");
        }
    } else {
        printf("  Error while retrieving its Address - %d (%s)\n", ret,
xbee_errorToStr(ret));
    }
}

int main(void) {
    void *d;
    struct xbee *xbee;
    struct xbee_con *con;
    char txRet;
    xbee_err ret;

    if ((ret = xbee_setup(&xbee, "xbee2", "/dev/ttyUSB0", 9600)) !=
XBEE_ENONE) {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
    }
    if ((ret = xbee_conNew(xbee, &con, "Identify", NULL)) !=
XBEE_ENONE) {
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret,
xbee_errorToStr(ret));
        return ret;
    }
    if ((ret = xbee_conCallbackSet(con, myCB, NULL)) !=
XBEE_ENONE) {
        xbee_log(xbee, -1, "xbee_conCallbackSet() returned: %d", ret);
        return ret;
    }

    sleep(120);

    if ((ret = xbee_conEnd(con)) != XBEE_ENONE) {
        xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
        return ret;
    }

    xbee_shutdown(xbee);

    return 0;
}
```

ARQUIVO: REMOTE_AT_D3.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <xbee.h>

void myCB(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt
**pkt, void **data) {
    xbee_err ret;
    char *ni;
    struct xbee_conAddress *addr;

    printf("An XBee joined the network!\n");

    if ((ret = xbee_pktDataGet(*pkt, "NI", 0, 0, (void**)&ni)) ==
XBEE_ENONE && ni != NULL) {
        printf("  It is called: [%s]\n", ni);
    } else {
        printf("  Error while retrieving its NI - %d (%s)\n", ret,
xbee_errorToStr(ret));
    }

    /* you could also use 'Address (16-bit)' or 'Address (64-bit)' to get
the raw byte arrays */
    if ((ret = xbee_pktDataGet(*pkt, "Address", 0, 0, (void**)&addr))
== XBEE_ENONE && addr != NULL) {
        printf("  It's address is:\n");
        if (addr->addr16_enabled) {
            printf("      16-bit address:    0x%02X%02X\n", addr-
>addr16[0], addr->addr16[1]);
        } else {
            printf("   16-bit address: --\n");
        }
        if (addr->addr64_enabled) {
            printf("      64-bit address:    0x%02X%02X%02X%02X
0x%02X%02X%02X%02X\n",



              addr->addr64[0], addr->addr64[1], addr->addr64[2], addr-
>addr64[3],



              addr->addr64[4], addr->addr64[5], addr->addr64[6], addr-
>addr64[7]);
        } else {
            printf("   64-bit address: --\n");
        }
    } else {
        printf("  Error while retrieving its Address - %d (%s)\n", ret,
xbee_errorToStr(ret));
    }
}

int main(void) {
    void *d;
```

```
	struct xbee *xbee;
	struct xbee_con *con;
	char txRet;
	xbee_err ret;

	if ((ret = xbee_setup(&xbee, "xbee2", "/dev/ttyUSB0", 9600)) !=
XBEE_ENONE) {
		printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
		return ret;
	}
	if ((ret = xbee_conNew(xbee, &con, "Identify", NULL)) !=
XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret,
xbee_errorToStr(ret));
		return ret;
	}

	if ((ret = xbee_conCallbackSet(con, myCB, NULL)) !=
XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conCallbackSet() returned: %d", ret);
		return ret;
	}

	sleep(120);

	if ((ret = xbee_conEnd(con)) != XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
		return ret;
	}

	xbee_shutdown(xbee);

	return 0;
}
```

ARQUIVO: FORCE_IO.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <xbee.h>

int main(void) {
	void *d;
	struct xbee *xbee;
	struct xbee_con *con;
	struct xbee_pkt *pkt;
	struct xbee_conAddress address;
	char txRet;
	int i;
	xbee_err ret;

	if ((ret = xbee_setup(&xbee, "xbee2", "/dev/ttyUSB0", 9600)) !=
XBEE_ENONE) {
		printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
		return ret;
	}

	memset(&address, 0, sizeof(address));
	address.addr64_enabled = 1;
	address.addr64[0] = 0x00;
	address.addr64[1] = 0x13;
	address.addr64[2] = 0xA2;
	address.addr64[3] = 0x00;
	address.addr64[4] = 0x40;
	address.addr64[5] = 0x5C;
	address.addr64[6] = 0xC2;
	address.addr64[7] = 0x65;
	if ((ret = xbee_conNew(xbee, &con, "Remote AT", &address)) !=
XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret,
xbee_errorToStr(ret));
		return ret;
	}


	for (i = 0; i < 60*4; i++) {
		unsigned char value;
		if ((ret = xbee_conTx(con, NULL, "D35")) != XBEE_ENONE)
;//break;

                                        if ((ret = xbee_conRx(con, &pkt,
NULL)) != XBEE_ENONE) ;//break;

		if ((ret = xbee_pktDigitalGet(pkt, 3, 0, &value)) !=
XBEE_ENONE) {
			printf("xbee_pktDigitalGet(channel=3): ret %d\n", ret);
		} else {
			printf("D3: %d\n", value);
		}

		xbee_pktFree(pkt);
		usleep(250000);
	}

	if (ret != XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conTx() or xbee_conRx() returned:
%d", ret);
		return ret;
	}

	if ((ret = xbee_conEnd(con)) != XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
		return ret;
	}

	xbee_shutdown(xbee);

	return 0;
}
```

ARQUIVO: LINUXZIGBEE.C

```
/*
	libxbee - a C library to aid the use of Digi's XBee wireless modules
		running in API mode.

	Copyright (C) 2009 onwards  Attie Grande (attie@attie.co.uk)

	libxbee is free *software*: you can redistribute it and/or modify it
	under the terms of the GNU Lesser General Public License as
published by
	the Free Software Foundation, either version 3 of the License, or
	(at your option) any later version.

	libxbee is distributed in the hope that it will be useful,
	but WITHOUT ANY WARRANTY; without even the implied
warranty of
	MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.  See the
	GNU Lesser General Public License for more details.

	You should have received a copy of the GNU Lesser General Public
License
	along with libxbee. If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Attie's library
#include <xbee.h>

// FLAGS
int save_on_file = 1;
int file_is_writable = 1;
int need_output_update = 1;

// ****** GLOBAL VARIABLES
struct my_xbee_struct {
		int status;
		int SH[4];
		int SL[4];
		char NI[50];
		char D[4];
} my_xbee[10];

char aux_dig_array[4];
char aux_array[50];
int temp_output[4];

void *d;
struct xbee_pkt *pktt;
struct xbee *xbee;
struct xbee_con *con;
struct xbee_con *con_IO_1;
struct xbee_con *con_IO_2;
struct xbee_con *con_remoteAT_1;
struct xbee_con *con_remoteAT_2;
struct xbee_con *con_remoteAT_3;
```

```
struct xbee_conAddress address1;  // Digital Input
struct xbee_conAddress address2;      // Digital Output
struct xbee_conAddress address3;      // Router

// ****** CALLBACK FUNCTIONS
// My callback to send AT commands
void myCB_remoteAT(struct xbee *xbee, struct xbee_con *con, struct
xbee_pkt **pkt, void **data) {
                                                int i = 0;

                                                printf("Callback remote
AT\n");

                                                if ((*pkt)->dataLen <=
1) {
        printf("too short...\n");
        return;
    }
    printf("rx: [%s]\n", (*pkt)->data);
                                                for       (i=0;i<=(*pkt)-
>dataLen;i++)           aux_array[i] = (*pkt)->data[i];
                                                return;
}

// ******
// My callback to read the digital port
void myCB_simpleIO(struct xbee *xbee, struct xbee_con *con, struct
xbee_pkt **pkt, void **data) {
    xbee_err ret;
    int value;
                                                int i = 0;

        printf("Callback remote IO\n");
   /* if ((*pkt)->dataLen < 2) {
        printf("too short...\n");
        return;
    }*/

                                                for (i=0;i<4;i++) {
            if ((ret = xbee_pktDigitalGet(*pkt, i, 0, &value)) !=
XBEE_ENONE) {
                printf("xbee_pktDigitalGet(channel=1): ret %d\n",
ret);
            } else {
                printf("D%d: %d\n",i, value);

                                                aux_dig_array[i]      =
value+0x30;
            }
                                                }

                                                save_on_file = 1;

                                                return;
}


// ****** CONNECTION FUNCTIONS
// Read Digital port
void read_digital_port(unsigned int xbee_id) {
        char txRet;
  xbee_err ret;
        int i = 0, j=0;
        my_xbee[xbee_id].status = 0;

        switch(xbee_id) {
                case 0:
                                if ((ret = xbee_conNew(xbee, &con,
"Remote AT", &address1)) != XBEE_ENONE)
                xbee_log(xbee,  -1,  "xbee_conNew()  returned:
%d (%s)", ret, xbee_errorToStr(ret));
                                break;
                case 1:
                                if ((ret = xbee_conNew(xbee, &con,
"Remote AT", &address2)) != XBEE_ENONE)
                xbee_log(xbee,  -1,  "xbee_conNew()  returned:
%d (%s)", ret, xbee_errorToStr(ret));
                                break;
                }

        //Reads once the inputs
        for (j = 0; j < 3; j++) {
                unsigned int value;

                if ((ret = xbee_conTx(con, NULL, "IS")) !=
XBEE_ENONE);
                if ((ret = xbee_conRx(con, &pktt, NULL)) !=
XBEE_ENONE);

                for (i=0;i<4;i++) {
                        if ((ret = xbee_pktDigitalGet(pktt, i,
0, &value)) != XBEE_ENONE) {

        my_xbee[xbee_id].status = 0;

        printf("xbee_pktDigitalGet(): ret %d\n", ret);
                        } else {

        my_xbee[xbee_id].status = 1;
                                printf("D%d:  %d  ",i,
value);
                                my_xbee[xbee_id].D[i]
= value+0x30;
                        }
                }
                printf("\n");
                xbee_pktFree(pktt);
                sleep(1);
  }

        if ((ret = xbee_conEnd(con)) != XBEE_ENONE) {
   xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
  }

        return;

}

// Send AT Command
void send_at_command(int xbee_id, unsigned char *at_command) {
        int i = 0, j=0;
        char txRet;
  xbee_err ret;
        my_xbee[xbee_id].status = 0;

        printf("end_AT_command  function:  xbee_id  %d  :  AT
command %s\n",xbee_id,at_command);
        switch(xbee_id) {
                case 0:
                        if ((ret = xbee_conNew(xbee, &con,
"Remote AT", &address1)) != XBEE_ENONE)
                                xbee_log(xbee,      -1,
"xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
                        break;
                case 1:
                        if ((ret = xbee_conNew(xbee, &con,
"Remote AT", &address2)) != XBEE_ENONE)
                                xbee_log(xbee,      -1,
"xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
                        break;
                case 2:
                        if ((ret = xbee_conNew(xbee, &con,
"Remote AT", &address3)) != XBEE_ENONE)
                                xbee_log(xbee,      -1,
"xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
                        break;
        }

        if  ((ret  =  xbee_conCallbackSet(con,  myCB_remoteAT,
NULL)) != XBEE_ENONE) {
        xbee_log(xbee,  -1,  "xbee_conCallbackSet()  returned:  %d",
ret);
        return;
  }

        ret = xbee_conTx(con, &txRet, "%s", at_command);
  printf("tx: %d\n", ret);
  if (ret) {
                        my_xbee[xbee_id].status = 0;
   printf("txRet: %s\n", xbee_errorToStr(ret));

  } else {
   sleep(2);
                        my_xbee[xbee_id].status = 1;
                        printf("aux_array: %s\n", aux_array);

  }

        if ((ret = xbee_conEnd(con)) != XBEE_ENONE) {
      xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
  }
```

```
        return;
}

int main(void) {

    char txRet;
    xbee_err ret;

                                        FILE *fp;
                                        int i = 0, j=0, count=0;
                                        int  output_mask[4] =
{0,0,0,0};

                                        // INIT // Input
                                        memset(&address1,  0,
sizeof(address1));
    address1.addr64_enabled = 1;
    address1.addr64[0] = 0x00;  my_xbee[0].SH[0] = 0x00;
    address1.addr64[1] = 0x13;  my_xbee[0].SH[1] = 0x13;
    address1.addr64[2] = 0xA2;  my_xbee[0].SH[2] = 0xA2;
    address1.addr64[3] = 0x00;  my_xbee[0].SH[3] = 0x00;
    address1.addr64[4] = 0x40;  my_xbee[0].SL[0] = 0x40;
    address1.addr64[5] = 0x5C;  my_xbee[0].SL[1] = 0x5C;
    address1.addr64[6] = 0xA2;  my_xbee[0].SL[2] = 0xA2;
    address1.addr64[7] = 0x65;  my_xbee[0].SL[3] = 0x65;

                                        // Addr2 // Output
                                        memset(&address2,  0,
sizeof(address2));
    address2.addr64_enabled = 1;
    address2.addr64[0] = 0x00;  my_xbee[1].SH[0] = 0x00;
    address2.addr64[1] = 0x13;  my_xbee[1].SH[1] = 0x13;
    address2.addr64[2] = 0xA2;  my_xbee[1].SH[2] = 0xA2;
    address2.addr64[3] = 0x00;  my_xbee[1].SH[3] = 0x00;
    address2.addr64[4] = 0x40;  my_xbee[1].SL[0] = 0x40;
    address2.addr64[5] = 0x5C;  my_xbee[1].SL[1] = 0x5C;
    address2.addr64[6] = 0xA2;  my_xbee[1].SL[2] = 0xA2;
    address2.addr64[7] = 0x90;  my_xbee[1].SL[3] = 0x90;

                                        // Addr3 // Router
                                        memset(&address3,  0,
sizeof(address3));
    address3.addr64_enabled = 1;
    address3.addr64[0] = 0x00;  my_xbee[2].SH[0] = 0x00;
    address3.addr64[1] = 0x13;  my_xbee[2].SH[1] = 0x13;
    address3.addr64[2] = 0xA2;  my_xbee[2].SH[2] = 0xA2;
    address3.addr64[3] = 0x00;  my_xbee[2].SH[3] = 0x00;
    address3.addr64[4] = 0x40;  my_xbee[2].SL[0] = 0x40;
    address3.addr64[5] = 0x5C;  my_xbee[2].SL[1] = 0x5C;
    address3.addr64[6] = 0xC1;  my_xbee[2].SL[2] = 0xC1;
    address3.addr64[7] = 0x50;  my_xbee[2].SL[3] = 0x50;

                                        // Opens the conection
to the usb
    if ((ret = xbee_setup(&xbee, "xbee2", "/dev/ttyUSB0", 9600)) !=
XBEE_ENONE) {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
    }

                                        //
*****************************Create conections to the Xbee device
                                        //
*****************************IO conection to the end-devices
                                        // Address 1 - Digital
Input
                                        if ((ret =
xbee_conNew(xbee, &con_IO_1, "I/O", &address1)) != XBEE_ENONE)
{
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret,
xbee_errorToStr(ret));

        return ret;
                                        }
                                        if ((ret =
xbee_conCallbackSet(con_IO_1,  myCB_simpleIO,  NULL)) !=
XBEE_ENONE) {
                                            xbee_log(xbee,  -1,
"xbee_conCallbackSet() returned: %d", ret);
                                            return ret;
                                        }

                                        // Address 2 - Digital
Output
                                        if ((ret =
xbee_conNew(xbee, &con_IO_2, "I/O", &address2)) != XBEE_ENONE)
{
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret,
xbee_errorToStr(ret));

        return ret;
                                        }
                                        if ((ret =
xbee_conCallbackSet(con_IO_2,  myCB_simpleIO,  NULL)) !=
XBEE_ENONE) {
                                            xbee_log(xbee,  -1,
"xbee_conCallbackSet() returned: %d", ret);
                                            return ret;
                                        }

                                        //***********Remote
AT to the end-devices
                                        for (j=0;j<3;j++) {

        send_at_command(j,"NI");
                                                for
(i=0;i<20;i++) {

        my_xbee[j].NI[i] = aux_array[i];
                                                }
                                        }

                                        //***********Read the
digital status
                                        for (j=0;j<2;j++) {

        read_digital_port(j);
                                                sleep(2);

        printf("my_xbee[%d].D:%s\n",j,my_xbee[j].D);
                                        }

                                        // Iniciate Loop
                                        while (1)  {

        sleep(10);

        //sleep(60);

        printf("Verifica arquivo\n");

        // Reads new information regards the output

        fp = fopen ("php_zigbee.txt","r");

        if (fp) {

                fscanf(fp,"%d  %d  %d
%d",&temp_output[0],&temp_output[1],&temp_output[2],&temp_output
[3]);

                fclose(fp);

                printf("Novas saidas digitais: %d %d %d %d\n",


        temp_output[0],temp_output[1],temp_output[2],temp_output[
3]);

        }



        //***********Remote AT to the end-devices

        if (count >= 6) {

                count = 0;

                for (j=0;j<3;j++) {

                        my_xbee[j].status = 0;

                        send_at_command(j,"NI");

                        for (i=0;i<20;i++) {

                                my_xbee[j].NI[i]  =
aux_array[i];
```

```
                }
            }
        } else {
            count++;
        }

        // Checks for changes in the outputs
        need_output_update = 0;

        for (i=0;i<4;i++) {
            output_mask[i]=0;
            if (temp_output[i]-4 != my_xbee[1].D[i]-0x30) {
                need_output_update = 1;
                output_mask[i]=1;
            }
        }

        //debug
        printf("output_mask = %d %d %d %d\n",output_mask[0],output_mask[1],output_mask[2],output_mask[3]);
        printf("atuais saidas = %d %d %d %d\n",my_xbee[1].D[0]-0x30,my_xbee[1].D[1]-0x30,my_xbee[1].D[2]-0x30,my_xbee[1].D[3]-0x30);
        printf("Novas saidas digitais: %d %d %d %d\n",temp_output[0]-4,temp_output[1]-4,temp_output[2]-4,temp_output[3]-4);

        // Changes the output  my_xbee[1]
        if (need_output_update) {
            save_on_file = 1;
            printf("updating outputs:");
            for (i=0;i<4;i++) {
                if (output_mask[i]) {
                    char buf[10];
sprintf(buf,"D%d%c",i,temp_output[i]);
send_at_command(1,buf);
                }
            }
            // Re-reads the digital port
            read_digital_port(1);
            printf("my_xmee[0].D:%s\n",my_xbee[1].D);
        }
        if (save_on_file) {
            // Saves on file
            fp = fopen ("zigbee.txt","w");
            if (fp) {
fprintf(fp,"1.%d.%X%X%X%X.%X%X%X%X.%s.%s\n",my_xbee[0].status,
my_xbee[0].SH[0],my_xbee[0].SH[1],my_xbee[0].SH[2],my_xbee[0].SH[3],
my_xbee[0].SL[0],my_xbee[0].SL[1],my_xbee[0].SL[2],my_xbee[0].SL[3],
my_xbee[0].NI, my_xbee[0].D);
fprintf(fp,"2.%d.%X%X%X%X.%X%X%X%X.%s.%s\n",my_xbee[1].status,
my_xbee[1].SH[0],my_xbee[1].SH[1],my_xbee[1].SH[2],my_xbee[1].SH[3],
my_xbee[1].SL[0],my_xbee[1].SL[1],my_xbee[1].SL[2],my_xbee[1].SL[3],
my_xbee[1].NI,my_xbee[1].D);
fprintf(fp,"3.%d.%X%X%X%X.%X%X%X%X.%s.%s\n",my_xbee[2].status,
my_xbee[2].SH[0],my_xbee[2].SH[1],my_xbee[2].SH[2],my_xbee[2].SH[3],
my_xbee[2].SL[0],my_xbee[2].SL[1],my_xbee[2].SL[2],my_xbee[2].SL[3],
my_xbee[2].NI,my_xbee[2].D);
                fclose(fp);
            }
            save_on_file = 0;
        }

        // Close all conections
        if ((ret = xbee_conEnd(con_IO_1)) != XBEE_ENONE) {
            xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
            return ret;
        }
        if ((ret = xbee_conEnd(con_IO_2)) != XBEE_ENONE) {
            xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
            return ret;
        }

        xbee_shutdown(xbee);

        return 0;
}
```

10 Apêndice C – Código-fonte implementado no servidor *web*.

ARQUIVO INDEX.HTML

```
<HEAD>
<TITLE>LinuXZigbee - Tiny6410 - SEL/EESC/USP</TITLE>
	<link rel="stylesheet" href="styles.css">
	<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.min.js"></script>
	<script type="text/javascript" src="linuxzigbee.js"></script>
</HEAD>

<BODY>
<div class="menu_line">
	<div class="menu_line_black">
	</div>
</div>
<div class="all">
	<div class="top_banner">
		<IMG id="tux" SRC="images/freddyart-bee-tux-1816.png">
		<IMG id="logo" SRC="images/logo.png">
	</div>

		<div class="all menu">
			<ul>
				<li><a id="link-inicio" href="#">INICIO</a></li>
				<li><a id="link-projeto" href="#">PROJETO</a></li>
				<li><a id="link-rede" href="#">REDE</a></li>
				<li><a id="link-contato"href="#">CONTATO</a></li>
			</ul>
		</div>

	<div class="all main">
		<p>Este projeto foi desenvolvido dentro do âmbito do Trabalho de Conclusão de Curso em engenharia elétrica da
		Escola de Engenharia de São Carlos - USP, pelo aluno Fausto Perez Rodrigues. O projeto tem por objetivo mostrar
		a integração entre as tecnologias de sistemas embarcados e dispositivos utilizando o protocolo Zigbee/IEEE802.15.4
		como solução para a área de automação.</p>
		<p>Na seção de projeto, são disopnibilizados links relacionados, o código-fonte e diagramas utilizados no projeto, assim
		como o trabalho final em formato PDF. Na seção de rede está um exemplo de implementação em topologia árvore, para breve
		apresentação.</p>
		<p>Toda a utilização do conteudo neste projeto é livre para uso/modificação, levando em conta que o autor não fornece
		qualquer garantia de qualquer tipo.</p>
	</div>
</div>
</body>
```

ARQUIVO: PROJETO.HTML

```
<HEAD>
<TITLE>LinuXZigbee - Tiny6410 - SEL/EESC/USP</TITLE>
	<link rel="stylesheet" href="styles.css">
	<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.min.js"></script>
	<script type="text/javascript" src="linuxzigbee.js"></script>
</HEAD>

<BODY>
<div class="menu_line">
	<div class="menu_line_black">
	</div>
</div>
<div class="all">
	<div class="top_banner">
		<IMG id="tux" SRC="images/freddyart-bee-tux-1816.png">
		<IMG id="logo" SRC="images/logo.png">
	</div>

		<div class="all menu">
			<ul>
				<li><a id="link-inicio"href="#">INICIO</a></li>
				<li><a id="link-projeto" href="#">PROJETO</a></li>
				<li><a id="link-rede"href="#">REDE</a></li>
				<li><a id="link-contato"href="#">CONTATO</a></li>
			</ul>
		</div>


	<div class="all main">
		<a href="http://www.zigbee.org" target="_blank">
			<IMG id="logo-zigbee" SRC="images/logos/zigbee.png">
		</a>
		<a href="http://www.linux.org" target="_blank">
			<IMG id="logo-linux" SRC="images/logos/linux.png">
		</a>
		<IMG id="bg-projeto" SRC="images/projeto.png">

	</div>
</div>
</body>
```

ARQUIVO: REDE.PHP

```
<?php
require 'xbee.php';

$root_file_path = "/var/www/fausto/Codigo/";
$xbee = Xbee::factory('order','status', 'SH', 'SL', 'NI','D');

//echo 'Debug php<br>';
//Opens the file zigbee.txt to read the information
try {
	$handle = fopen($root_file_path.'zigbee.txt', 'r');

	if ($handle) {
		$i = 0;
		while(!feof($handle)) {
			$full_data = fgets($handle);
			//echo $full_data.'<br />';
			$data = explode(".",$full_data);
			$a = new Xbee;
			$my_xbee[$i] = call_user_func_array(array($xbee, 'create'),$data);
			$i++;
		}
		fclose($handle);
	}
} catch (Exception $e) {
	echo 'Exception:', $e->getMessage(), "\n";
}
$total_devices = count($my_xbee)-1;

for($i=0;$i<$total_devices;$i++) {
	$my_xbee[$i]->SH = '0'.$my_xbee[$i]->SH.'0';
}
/*
echo 'Total devices in the Network: '.$total_devices.'<br/>';
echo '<br>'.$my_xbee[0]->NI;
echo '<br>'.$my_xbee[1]->NI;
echo '<br>'.$my_xbee[2]->NI;
*/
?>

<HEAD>
<TITLE>LinuXZigbee - Tiny6410 - SEL/EESC/USP</TITLE>
	<link rel="stylesheet" href="styles.css">
	<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.min.js"></script><!-- -->
	<script type="text/javascript" src="linuxzigbee.js"></script>
</HEAD>

<BODY>
<div class="menu_line">
	<div class="menu_line_black">
	</div>
</div>
<div class="all">
	<div class="top_banner">
		<IMG id="tux" SRC="images/freddyart-bee-tux-1816.png">
		<IMG id="logo" SRC="images/logo.png">
	</div>

		<div class="all menu">
			<ul>
```

```
                                        <li><a id="link-
inicio"href="#">INICIO</a></li>
                                        <li><a id="link-projeto"
href="#">PROJETO</a></li>
                                        <li><a id="link-
rede"href="#">REDE</a></li>
                                        <li><a id="link-
contato"href="#">CONTATO</a></li>
                                    </ul>
                            </div>


                <div class="all main">
                            <IMG id="bg-rede" SRC="images/rede.png">
                            <h3>Rede implementada em topologia árvore
(cluster tree)</h3>
                            <div class="xbee_data" id="coordinator">
                                    <img class="xbee_status"
id="coordinator" src="images/network/status_on.png"></img>
                                    <div class="xbee_at">
                                            NI:
COORDINATOR<br/>
                                            SH: 0013A200<br/>
                                            SL: 405CC166<br/>
                                    </div>
                            </div>
                            <div class="xbee_data" id="router">
                                    <img class="xbee_status" id="router"
src=
                                    <?php
                                            $my_xbee[2]->status ?
$status = "on" : $status = "off";
                                            echo
"images/network/status_".$status.".png"
                                    ?>
                                    ></img>
                                    <div class="xbee_at">
                                            STATUS: <?= $status
?><br/>
                                            NI: <?= $my_xbee[2]-
>NI ?><br/>
                                            SH: <?= $my_xbee[2]-
>SH ?><br/>
                                            SL: <?= $my_xbee[2]-
>SL ?><br/>
                                    </div>
                            </div>
                            <div class="xbee_data" id="end1">
                                    <img class="xbee_status" id="end1"
src=
                                    <?php
                                            $my_xbee[0]->status ?
$status = "on" : $status = "off";
                                            echo
"images/network/status_".$status.".png"
                                    ?>
                                    ></img>
                                    <div class="xbee_at">
                                            STATUS: <?= $status
?><br/>
                                            NI: <?= $my_xbee[0]-
>NI ?><br/>
                                            SH: <?= $my_xbee[0]-
>SH ?><br/>
                                            SL: <?= $my_xbee[0]-
>SL ?><br/>
                                    </div>
                            </div>
                            <div class="xbee_data" id="end2">
                                    <img class="xbee_status" id="end2"
src=
                                    <?php
                                            $my_xbee[1]->status ?
$status = "on" : $status = "off";
                                            echo
"images/network/status_".$status.".png"
                                    ?>
                                    ></img>
                                    <div class="xbee_at">
                                            STATUS: <?= $status
?><br/>
                                            NI: <?= $my_xbee[1]-
>NI ?><br/>
                                            SH: <?= $my_xbee[1]-
>SH ?><br/>
                                            SL: <?= $my_xbee[1]-
>SL ?><br/>
                                    </div>
                                    <div class="output">
                                            <?php
for($i=0;$i<4;$i++) { ?>
                                                    <img style=
                                                    <?php
                                                            $left_push = 70*$i;
                                                            echo "'position:absolut;left:'.$left_push.'px;"';
                                                    ?>
                                                    id=<?= ""'.$i.""; ?>
                                                                    src=
                                                    <?php
                                                            $my_xbee[1]->D[$i] ? $status = "on" : $status =
"off";
                                                            echo "images/network/bulb_".$status.".png"
                                                    ?>
                                                                    ></img>
                                                            <?php } ?>
                                                    </div>
                                            </div>
                                    </div>
</div>
</body>
```

ARQUIVO: XBEE.PHP

```
<?php
class Xbee
{
   /**
    * Define a new struct object, a blueprint object with only empty properties.
    */
   public static function factory()
   {
      $struct = new self;
      foreach (func_get_args() as $value) {
         $struct->$value = null;
      }
      return $struct;
   }

   /**
    * Create a new variable of the struct type $this.
    */
   public function create()
   {
      // Clone the empty blueprint-struct ($this) into the new data $struct.
      $struct = clone $this;

      // Populate the new struct.
      $properties = array_keys((array) $struct);
      foreach (func_get_args() as $key => $value) {
         if (!is_null($value)) {
            $struct->$properties[$key] = $value;
         }
      }

      // Return the populated struct.
      return $struct;
   }
}
```

ARQUIVO: UPDATE_OUTPUT.PHP

```
<?php
//Set some variables
$root_file_path = "/var/www/fausto/Codigo/";

// Gets the parameter (output_id)
$output_id =  urldecode($_GET['output_id']);

//Opens the file php_zigbee.txt to read the information
while(!is_readable($root_file_path.'php_zigbee.txt')) ;
try {
        $handle = fopen($root_file_path.'php_zigbee.txt','r');

        if ($handle) {
                $full_data  = fgets($handle);
                fclose($handle);
                $data = explode(" ", $full_data);
                //var_dump($full_data);
                foreach ($data as $key => $elemment) $data[$key] =
trim($elemment);
                $data[$output_id] == "4" ? $data[$output_id] = "5" :
$data[$output_id] = "4";
                $full_data = implode(" ",$data);
                //var_dump($full_data);
```

```
                        while(!is_writable($root_file_path.'php_zigbee.txt'))
;
                        $handle                                         =
fopen($root_file_path.'php_zigbee.txt','w');
                        fprintf($handle,"%s",$full_data);
                        fclose($handle);
                }
} catch (Exception $e) {
                echo 'Exception:',  $e->getMessage(), "\n";
}

echo json_encode(Array("id" => $output_id, "value" => $data[$output_id]));

?>
```

ARQUIVO: AJAX_UPDATE.PHP

```
<?php
require 'xbee.php';

$root_file_path = "/var/www/fausto/Codigo/";
$xbee = Xbee::factory('order','status', 'SH', 'SL', 'NI','D');

//echo 'Debug php<br>';
//Opens the file zigbee.txt to read the information
while(!is_readable($root_file_path.'zigbee.txt')) ;
try {
                $handle = fopen($root_file_path.'zigbee.txt', 'r');

                if ($handle) {
                        $i = 0;
                        while(!feof($handle)) {
                                $full_data  = fgets($handle);
                                //echo $full_data.'<br />';
                                $data = explode(".",$full_data);
                                $a = new Xbee;
                                $my_xbee[$i]                    =
call_user_func_array(array($xbee, 'create'),$data);
                                $i++;
                        }
                        fclose($handle);
                }
} catch (Exception $e) {
                echo 'Exception:',  $e->getMessage(), "\n";
}
$total_devices = count($my_xbee)-1;

for($i=0;$i<$total_devices;$i++) {
                $my_xbee[$i]->SH = '0'.$my_xbee[$i]->SH.'0';
}
var_dump($my_xbee);
//echo json_encode($my_xbee);

?>
```

ARQUIVO: STYLES.CSS

```
body {
                font-family:calibri, sans-serif;
                color:#DFDFDF;
                background:#1C1C14;
                color:#         bbb;
}

.border {
                border: 1px solid white;
}

.all {
                height:100%;
                width:900px;
                margin-left:auto;
                margin-right:auto;
}

.top_banner {
                z-index:11;
                height:100px;
}

.menu_line {
                background:#FCF18D;
                z-index:-2;
                left:0;
                top:155px;
                height:30px;
                width:100%;
                position:absolute;
}
.menu_line_black {
                background:#383429;
                z-index:-1;
                left:0;
                top:25px;
                height:3px;
                width:100%;
                position:absolute;
}

#tux {
                z-index:11;
                margin-left:20px;
                width:183px;
}

#logo {
                float:right;
                margin-right:20px;
                margin-top:20px;
                width:40%;
                z-index:100;
}

.menu {
                font-size:30px;
                margin-left:200px;
                top:155px;
                height:30px;
                width:100%;
                position:absolute;
}

.menu ul {
   padding-left:0px;
   margin:0px;
   float: left;
   width: 100%;
   list-style:none;
}

.menu ul li { display: inline; }

.menu ul li a {
                                margin-top:-7px;
   padding: 0px 40px;
   float:left;
   color: #333;
   text-decoration: none;
   border-bottom:2px solid #1C1C14;
}

.menu ul li a:hover {
   color: #333;
   border-bottom:4px solid #FCF18D;
}

.main {
                position:absolute;
                top:200px;
}

#bg-projeto {
                z-index:-1;
                position:absolute;
                left:0px;
                top:10px;
}

#logo-zigbee {
                width: 190px;
                position:absolute;
                top:190px;
                left:40px;
}

#logo-linux {
                                height: 140px;
                                position:absolute;
                                top:40px;
                                left:70px;
}

#bg-rede {
                z-index:-1;
                position:absolute;
                left:0px;
                top:35px;
}

.xbee_data {
                position:absolute;
                width:200px;
                height:200px;
}
```

```
.xbee_data#coordinator {
		top:350px;
		left:50px;
}

.xbee_data#router {
		top:350px;
		left:350px;
}

.xbee_data#end1 {
		top:200px;
		left:630px;
}

.xbee_data#end2 {
		top:482px;
		left:630px;
}

.xbee_status {
		position: absolute;
		width: 45px;
		top: 27px;
		left: 45px;

}

.xbee_at {
		position:absolute;
		top:100px;
		left:50px;
}

.output {
		position:absolute;
		top:200px;
}

.output img{
		position:absolute;
		width:50px;
		cursor: hand;
		cursor: pointer;
}

.links {
		z-index:1;
}

.footer {
		text-align:center;
   bottom: 0;
   position: absolute;

}
```

ARQUIVO: LINUXZIGBEE.JS

```
// Waits until the whole document is loaded
$(document).ready(function() {

		function callback_timeout(){
		   return function(){
				//window.location.reload();
			}
		}

		//setTimeout(callback_timeout(), 60000);
 //setTimeout(callback_timeout(), 15000);

		$(".output img").on("click",function (e) {

		   $.ajax({
			url: 'update_output.php',	//the script to call to get data
			data: "output_id="+$(this).attr("id"),			//you can
insert url argumnets here to pass to api.php			//for example
"id=5&parent=6"
			dataType: 'json',		//data format
			success: function(data)  {

		//alert(data.id);

		//alert(data.value);

								if
(data.value == "5") {

		src = "on";
								} else {

		src = "off";
								}
								$(".output
img#"+data.id).attr("src","images/network/bulb_"+src+".png")

		   }
		  });/**/

					$(this).attr("src","images/loader.gif");
		 });

		var production = true;
		var path;

		if (production) {
			path = "http:///143.107.235.36/fausto/site/tcc/";
		} else {
			path = "http://localhost/~f_rodrigues/tcc/";
		}

		// Action to the menu buttons
		$("#link-inicio").click(function() {
		  window.location = path;
		});

		$("#link-projeto").click(function() {
		   window.location = path+"projeto.html";
		});

		$("#link-rede").click(function() {
		   window.location.replace(path+"rede.php");
		});

		$("#link-contato").click(function() {
		   window.location.replace(path+"contato.html");
		});
});
```

11 ANEXO A – Lista de comandos AT

Este anexo contém a informação retirada do *datasheet* dos dispositivos Xbee, fornecido pela Digi Internetional, a partir da página 129.

TABELA 8 Lista de comandos AT de *Addressing*

AT	Name and Description	Node	Parameter Range	Default
DH	**Destination Address High**.Set/Get the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the 64-bit destination address for data transmission. Special definitions for DH and DL include 0x000000000000FFFF (broadcast) and 0x0000000000000000 (coordinator).	CRE	0 - 0xFFFFFFFF	0
DL	**Destination Address Low**. Set/Get the lower 32 bits of the 64-bit destination address. When combined with DH, it defines the 64-bit destination address for data transmissions. Special definitions for DH and DL include 0x000000000000FFFF (broadcast) and 0x0000000000000000 (coordinator).	CRE	0 - 0xFFFFFFFF	0xFFFF(Coordinator) 0 (Router/End Device)
MY	**16-bit Network Address**. Read the 16-bit network address of the module. A value of 0xFFFE means the module has not joined a ZigBee network	CRE	0 - 0xFFFE [read-only]	0xFFFE
MP	**16-bit Parent Network Address**. Read the 16-bit network address of the module's parent. A value of 0xFFFE means the module does not have a parent.	E	0 - 0xFFFE [read-only]	0xFFFE
NC	**Number of Remaining Children**. Read the number of end device children that can join the device. If NC returns 0, then the device cannot allow any more end device children to join.	CR	0 - MAX_CHILDREN (maximum varies)	read-only
SH	**Serial Number High**. Read the high 32 bits of the module's unique 64-bit address.	CRE	0 - 0xFFFFFFFF [read-only]	factory-set
SL	**Serial Number Low**. Read the low 32 bits of the module's unique 64-bit address.	CRE	0 - 0xFFFFFFFF [read-only]	factory-set
NI	**Node Identifier.** Stores a string identifier. The register only accepts printable ASCII data. In AT Command Mode, a string can not start with a space. A carriage return ends the command. Command will automatically end when maximum bytes for the string have been entered. This string is returned as part of the ND (Node Discover) command. This identifier is also used with the DN (Destination Node) command. In AT command mode, an ASCII comma (0x2C) cannot be used in the NI string	CRE	20-Byte printable ASCII string	ASCII space character (0x20)
SE	**Source Endpoint**. Set/read the ZigBee application layer source endpoint value. This value will be used as the source endpoint for all data transmissions. SE is only supported in AT firmware.The default value 0xE8 (Data endpoint) is the Digi data endpoint	CRE	0 - 0xFF	0xE8
DE	**Destination Endpoint**. Set/read Zigbee application layer destination ID value. This value will be used as the destination endpoint all data transmissions. DE is only supported in AT firmware.The default value (0xE8) is the Digi data endpoint.	CRE	0 - 0xFF	0xE8

CI	**Cluster Identifier**. Set/read Zigbee application layer cluster ID value. This value will be used as the cluster ID for all data transmissions. CI is only supported in AT firmware.The default value0x11 (Transparent data cluster ID).	CRE	0 - 0xFFFF	0x11
NP	**Maximum RF Payload Bytes**. This value returns the maximum number of RF payload bytes that can be sent in a unicast transmission. If APS encryption is used (API transmit option bit enabled), the maximum payload size is reduced by 9 bytes. If source routing is used (AR < 0xFF), the maximum payload size is reduced further. **Note**: NP returns a hexadecimal value. (e.g. if NP returns 0x54, this is equivalent to 84 bytes)	CRE	0 - 0xFFFF	[read-only]
DD	**Device Type Identifier**. Stores a device type value. This value can be used to differentiate different XBee-based devices. Digi reserves the range 0 - 0xFFFFFF. For example, Digi currently uses the following DD values to identify various ZigBee products: 0x30001 - ConnectPort X8 Gateway 0x30002 - ConnectPort X4 Gateway 0x30003 - ConnectPort X2 Gateway 0x30005 - RS-232 Adapter 0x30006 - RS-485 Adapter	CRE	0 - 0xFFFFFFFF	0x30000

TABELA 9 Lisa de comandos AT de *Networking*

AT	Name and Description	Node	Parameter Range	Default
CH	**Operating Channel**. Read the channel number used for transmitting and receiving between RF modules. Uses 802.15.4 channel numbers. A value of 0 means the device has not joined a PAN and is not operating on any channel.	CRE	**XBee** 0, 0x0B - 0x1A (Channels 11-26) **XBee-PRO (S2)** 0, 0x0B - 0x18	[read-only]
ID	**Extended PAN ID**. Set/read the 64-bit extended PAN ID. If set to 0, the coordinator will select a random extended PAN ID, and the router / end device will join any extended PAN ID. Changes to ID should be written to non-volatile memory using the WR command to preserve the ID setting if a power cycle occurs.	CRE	0- 0xFFFFFFFFFFFFFFFF	0
OP	**Operating Extended PAN ID**. Read the 64-bit extended PAN ID. The OP value reflects the operating extended PAN ID that the module is running on. If ID > 0, OP will equal ID.	CRE	0x01 - 0xFFFFFFFFFFFFFFFF	[read-only]
NH	**Maximum Unicast Hops**. Set / read the maximum hops limit. This limit sets the maximum broadcast hops value (BH) and determines the unicast timeout. The timeout is computed as (50 * NH) + 100 ms. The default unicast timeout of 1.6 seconds (NH=0x1E) is enough time for data and the acknowledgment to traverse about 8 hops.	CRE	0 - 0xFF	0x1E
BH	**Broadcast Hops.** Set/Read the maximum number of hops for each broadcast data transmission. Setting this to 0 will use the maximum number of hops.	CRE	0 - 0x1E	0

OI	**Operating 16-bit PAN ID**. Read the 16-bit PAN ID. The OI value reflects the actual 16-bit PAN ID the module is running on.	CRE	0 - 0xFFFF	[read-only]
NT	**Node Discovery Timeout**. Set/Read the node discovery timeout. When the network discovery (ND) command is issued, the NT value is included in the transmission to provide all remote devices with a response timeout. Remote devices wait a random time, less than NT, before sending their response.	CRE	0x20 - 0xFF [x 100 msec]	0x3C (60d)
NO	**Network Discovery options.** Set/Read the options value for the network discovery command. The options bitfield value can change the behavior of the ND (network discovery) command and/or change what optional values are returned in any received ND responses or API node identification frames. Options include: 0x01 = Append DD value (to ND responses or API node identification frames) 002 = Local device sends ND response frame when ND is issued.	CRE	0 - 0x03 [bitfield]	0
SC	**Scan Channels**. Set/Read the list of channels to scan. ***Coordinator*** - Bit field list of channels to choose from prior to starting network. ***Router/End Device*** - Bit field list of channels that will be scanned to find a Coordinator/ Router to join. Changes to SC should be written using WR command to preserve the SC setting if a power cycle occurs. Bit (Channel): 0 (0x0B) 4 (0x0F) 8 (0x13) 12 (0x17) 1 (0x0C) 5 (0x10) 9 (0x14) 13 (0x18) 2 (0x0D) 6 (0x11) 10 (0x15) 14 (0x19) 3 (0x0E) 7 (0x12) 11 (0x16) 15 (0x1A)	CRE	**XBee** 1 - 0xFFFF [bitfield] **XBee-PRO (S2)** 1 - 0x3FFF [bitfield] (bits 14, 15 not allowed) **XBee-PRO (S2B)** 1-0x7FFF (bit 15 is not allowed)	1FFE
SD	**Scan Duration**. Set/Read the scan duration exponent. Changes to SD should be written using WR command. ***Coordinator*** - Duration of the Active and Energy Scans (on each channel) that are used to determine an acceptable channel and Pan ID for the Coordinator to startup on. ***Router / End Device*** - Duration of Active Scan (on each channel) used to locate an available Coordinator / Router to join during Association. Scan Time is measured as:(# Channels to Scan) * (2 ^ SD) * 15.36ms - The number of channels to scan is determined by the SC parameter. The XBee can scan up to 16 channels (SC = 0xFFFF). Sample Scan Duration times (13 channel scan): If SD = 0, time = 0.200 sec SD = 2, time = 0.799 sec SD = 4, time = 3.190 sec SD = 6, time = 12.780 sec	CRE	0 - 7 [exponent]	3

ZS	**ZigBee Stack Profile**. Set / read the ZigBee stack profile value. This must be set the same on all devices that should join the same network.	CRE	0-2	0
NJ	**Node Join Time**. Set/Read the time that a Coordinator/Router allows nodes to join. This value can be changed at run time without requiring a Coordinator or Router to restart. The time starts once the Coordinator or Router has started. The timer is reset on power-cycle or when NJ changes.	CR	0 - 0xFF [x 1 sec]	0xFF (always allows joining)
JV	**Channel Verification**. Set/Read the channel verification parameter. If JV=1, a router will verify the coordinator is on its operating channel when joining or coming up from a power cycle. If a coordinator is not detected, the router will leave its current channel and attempt to join a new PAN. If JV=0, the router will continue operating on its current channel even if a coordinator is not detected.	R	0 - Channel verification disabled 1 - Channel verification enabled	0
NW	**Network Watchdog Timeout**. Set/read the network watchdog timeout value. If NW is set > 0, the router will monitor communication from the coordinator (or data collector) and leave the network if it cannot communicate with the coordinator for 3 NW periods. The timer is reset each time data is received from or sent to a coordinator, or if a many-to-one broadcast is received.	R	0 - 0x64FF [x 1 minute] (up to over 17 days)	0 (disabled)
JN	**Join Notification**. Set / read the join notification setting. If enabled, the module will transmit a broadcast node identification packet on power up and when joining. This action blinks the Associate LED rapidly on all devices that receive the transmission, and sends an API frame out the UART of API devices. This feature should be disabled for large networks to prevent excessive broadcasts.	RE	0-1	0
AR	**Aggregate Routing Notification**. Set/read time between consecutive aggregate route broadcast messages. If used, AR should be set on only one device to enable many-to-one routing to the device. Setting AR to 0 only sends one broadcast	CR	0 - 0xFF	0xFF

TABELA 10 Lista de comandos AT de *Security*

AT	Name and Description	Node	Parameter Range	Default
EE	**Encryption Enable**. Set/Read the encryption enable setting.	CRE	0 - Encryption disabled 1 - Encryption enabled	0
EO	**Encryption Options.** Configure options for encryption. Unused option bits should be set to 0. Options include: 0x01 - Send the security key unsecured over-the-air during joins 0x02 - Use trust center (coordinator only	CRE	0 - 0xFF	
NK	**Network Encryption Key**. Set the 128-bit AES network encryption key. This command is write-only; NK cannot be read. If set to 0 (default), the module will select a random network key.	C	128-bit value	0

KY	**Link Key**. Set the 128-bit AES link key. This command is write only; KY cannot be read. Setting KY to 0 will cause the coordinator to transmit the network key in the clear to joining devices, and will cause joining devices to acquire the network key in the clear when joining.	CRE	128-bit value	0

TABELA 11 Lista de comandos AT para edição de Opções do modo AT

AT	Name and Description	Node	Parameter Range	Default
CT	**Command Mode Timeout.** Set/Read the period of inactivity (no valid commands received) after which the RF module automatically exits AT Command Mode and returns to Idle Mode.	CRE	2 - 0x028F [x 100 ms]	0x64 (100d)
CN	**Exit Command Mode.** Explicitly exit the module from AT Command Mode.	CRE	--	--
GT	**Guard Times**. Set required period of silence before and after the Command Sequence Characters of the AT Command Mode Sequence (GT + CC + GT). The period of silence is used to prevent inadvertent entrance into AT Command Mode.	CRE	1 - 0x0CE4 [x 1 ms] (max of 3.3 decimal sec)	0x3E8 (1000d)
CC	**Command Sequence Character**. Set/Read the ASCII character value to be used between Guard Times of the AT Command Mode Sequence (GT + CC + GT). The AT Command Mode Sequence enters the RF module into AT Command Mode. The CC command is only supported when using AT firmware: 20xx (AT coordinator), 22xx (AT router), 28xx (AT end device).	CRE	0 - 0xFF	0x2B ('+' ASCII)

TABELA 12 - Lista de comandos At da Interface com RF

AT	Name and Description	Node	Parameter Range	Default
PL	**Power Level**. Select/Read the power level at which the RF module transmits conducted power. For XBee-PRO (S2B) Power Level 4 is calibrated and the other power levels are approximate.	CRE	**XBee** (boost mode disabled) 0 = -8 dBm 1 = -4 dBm 2 = -2 dBm 3 = 0 dBm 4 = +2 dBm **XBee-PRO (S2)** 4 = 17 dBm **XBee-PRO (S2)** **(International Variant)** 4 = 10dBm	4

PM	**Power Mode**. Set/read the power mode of the device. Enabling boost mode will improve the receive sensitivity by 1dB and increase the transmit power by 2dB Note: Enabling boost mode on the XBee-PRO (S2) will not affect the output power. Boost mode imposes a slight increase in current draw. See section 1.2 for details.	CRE	0-1, 0= -Boost mode disabled, 1= Boost mode enabled.	1
DB	**Received Signal Strength**. This command reports the received signal strength of the last received RF data packet. The DB command only indicates the signal strength of the last hop. It does not provide an accurate quality measurement for a multihop link. DB can be set to 0 to clear it. The DB command value is measured in -dBm. For example if DB returns 0x50, then the RSSI of the last packet received was -80dBm. As of 2x6x firmware, the DB command value is also updated when an APS acknowledgment is received.	CRE	0 - 0xFF Observed range for XBee-PRO: 0x1A - 0x58 For XBee: 0x 1A - 0x5C	
PP	**Peak Power.** Read the dBm output when maximum power is selected (PL4).	CRE	0x0-0x12	[read only]

TABELA 13 Lista de comandos AT de Interface Serial

AT	Name and Description	Node	Parameter Range	Default
AP	API Enable. Enable API Mode. The AP command is only supported when using API firmware: 21xx (API coordinator), 23xx (API router), 29xx (API end device).	CRE	1 - 2 1 = API-enabled 2 = API-enabled (w/escaped control	1
AO	**API Options**. Configure options for API. Current options select the type of receive API frame to send out the Uart for received RF data packets.	CRE	0 - Default receive API indicators enabled 1 - Explicit Rx data indicator API frame enabled (0x91) 3 - enable ZDO passthrough of ZDO requests to the UART which are not supported by the stack, as well as Simple_Desc_req, Active_EP_req, and Match_Desc_req.	0

BD	**Interface Data Rate**. Set/Read the serial interface data rate for communication between the module serial port and host. Any value above 0x07 will be interpreted as an actual baud rate. When a value above 0x07 is sent, the closest interface data rate represented by the number is stored in the BD register.	CRE	0-7 (standard baud rates) 0 = 1200 bps 1 = 2400 2 = 4800 3 = 9600 4 = 19200 5 = 38400 6 = 57600 7 = 115200	3
NB	**Serial Parity**. Set/Read the serial parity setting on the module.	CRE	0 = No parity 1 = Even parity 2 = Odd parity 3 = Mark parity	0
SB	**Stop Bits**. Set/read the number of stop bits for the UART. (Two stop bits are not supported if mark parity is enabled.)	CRE	0 = 1 stop bit 1 = 2 stop bits	0
RO	**Packetization Timeout**. Set/Read number of character times of inter-character silence required before packetization. Set (RO=0) to transmit characters as they arrive instead of buffering them into one RF packet The RO command is only supported when using AT firmware: 20xx (AT coordinator), 22xx (AT router), 28xx (AT end device).	CRE	0 - 0xFF [x character times]	3
D7	**DIO7 Configuration**. Select/Read options for the DIO7 line of the RF module.	CRE	0 = Disabled 1 = CTS Flow Control 3 = Digital input 4 = Digital output, low 5 = Digital output, high 6 = RS-485 transmit enable (low enable) 7 = RS-485 transmit enable (high enable)	1
D6	**DIO6 Configuration.** Configure options for the DIO6 line of the RF module.	CRE	0 = Disabled 1 = RTS flow control 3 = Digital input 4 = Digital output, low 5 = Digital output, high	0

TABELA 14 Lista de comandos AT de *I/O control*

AT	Name and Description	Node	Parameter Range	Default
IR	**IO Sample Rate**. Set/Read the IO sample rate to enable periodic sampling. For periodic sampling to be enabled, IR must be set to a non-zero value, and at least one module pin must have analog or digital IO functionality enabled (see D0-D8, P0-P2 commands). The sample rate is measured in milliseconds.	CRE	0, 0x32:0xFFFF (ms)	0
IC	I**O Digital Change Detection**. Set/Read the digital IO pins to monitor for changes in the IO state. IC works with the individual pin configuration commands (D0-D8, P0-P2). If a pin is enabled as a digital input/output, the IC command can be used to force an immediate IO sample transmission when the DIO state changes. IC is a bitmask that can be used to enable or disable edge detection on individual channels. Unused bits should be set to 0.	CRE	: 0 - 0xFFFF	0
P0	**PWM0 Configuration**. Select/Read function for PWM0.	CRE	0 = Disabled 1 = RSSI PWM 3 - Digital input, monitored 4 - Digital output, default low 5 - Digital output, default high	1
P1	**DIO11 Configuration**. Configure options for the DIO11 line of the RF module.	CRE	0 - Unmonitored digital input 3- Digital input, monitored 4- Digital output, default low 5- Digital output, default high	0
P2	**DIO12 Configuration**. Configure options for the DIO12 line of the RF module.	CRE	0 - Unmonitored digital input 3- Digital input, monitored 4- Digital output, default low 5- Digital output, default high	0

P3	**DIO13 Configuration**. Set/Read function for DIO13. This command is not yet supported.	CRE	0, 3-5 0 – Disabled 3 – Digital input 4 – Digital output, low 5 – Digital output, high	
D0	**AD0/DIO0 Configuration**. Select/Read function for AD0/DIO0.	CRE	1 - Commissioning button enabled 2 - Analog input, single ended 3 - Digital input 4 - Digital output, low 5 - Digital output, high	1
D1	**AD1/DIO1 Configuration**. Select/Read function for AD1/DIO1.	CRE	0, 2-5 0 – Disabled 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0
D2	AD2/DIO2 Configuration. Select/Read function for AD2/DIO2.	CRE	0, 2-5 0 – Disabled 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0
D3	**AD3/DIO3 Configuration**. Select/Read function for AD3/DIO3.	CRE	0, 2-5 0 – Disabled 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0

D4	**DIO4 Configuration**. Select/Read function for DIO4.	CRE	0, 3-5 0 – Disabled 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0
D5	**DIO5 Configuration**. Configure options for the DIO5 line of the RF module.	CRE	0 = Disabled 1 = Associated indication LED 3 = Digital input 4 = Digital output, default low	1
D8	**DIO8 Configuration**. Set/Read function for DIO8. This command is not yet supported.	CRE	0, 3-5 0 – Disabled 3 – Digital input 4 – Digital output, low 5 – Digital output, high	
LT	**Assoc LED Blink Time**. Set/Read the Associate LED blink time. If the Associate LED functionality is enabled (D5 command), this value determines the on and off blink times for the LED when the module has joined a network. If LT=0, the default blink rate will be used (500ms coordinator, 250ms router/end device). For all other LT values, LT is measured in 10ms.	CRE	0, 0x0A - 0xFF (100 - 2550 ms)	0
PR	**Pull-up Resistor**. Set/read the bit field that configures the internal pull-up resistor status for the I/O lines. "1" specifies the pull-up resistor is enabled. "0" specifies no pullup.(30k pull-up resistors)	CRE	0 - 0x3FFF	0 - 0x1FFF
RP	**RSSI PWM Timer.** Time the RSSI signal will be output on the PWM after the last RF data reception or APS acknowledgment.. When RP = 0xFF, output will always be on.	CRE	0 - 0xFF [x 100 ms]	0x28 (40d)
%V	**Supply Voltage.** Reads the voltage on the Vcc pin. Scale by 1200/1024 to convert to mV units. For examplee, a %V reading of 0x900 (2304 decimal) represents 2700mV or 2.7OV.	CRE	-0x-0xFFFF [read only]	-

V+	**Voltage Supply Monitoring.** The voltage supply threshold is set with the V+ command. If the measured supply voltage falls below or equal to this threshold, the supply voltage will be included in the IO sample set. V+ is set to 0 by default (do not include the supply voltage). Scale mV units by 1024/1200 to convert to internal units. For example, for a 2700mV threshold enter 0x900. Given the operating Vcc ranges for different platforms, and scaling by 1024/1200, the useful parameter ranges are: XBee 2100-3600 mV 0,0x0700-0x0c00 PRO 3000-3400 mV, 0,0x0a00-0x0b55	CRE	0-0xFFFF	0
TP	Reads the module temperature in Degrees Celsius. Accuracy +/- 7 degrees. 1° C = 0x0001 and -1° C = 0xFFFF. Command is only available in PRO S2B.	CRE	0x0-0xFFFF	-

TABELA 15 Lista de comandos AT de diagnósticos

AT	Name and Description	Node	Parameter Range	Default
VR	**Firmware Version**. Read firmware version of the module. The firmware version returns 4 hexadecimal values (2 bytes) "ABCD". Digits ABC are the main release number and D is the revision number from the main release. "B" is a variant designator. XBee and XBee-PRO ZB modules return: 0x2xxx versions.	CRE	0 - 0xFFFF [read-only]	Factory-set
HV	***Hardware* Version**. Read the *hardware* version of the module.version of the module. This command can be used to distinguish among different *hardware* platforms. The upper byte returns a value that is unique to each module type. The lower byte indicates the *hardware* revision. XBee ZB and XBee ZNet modules return the following (hexadecimal) values: 0x19xx - XBee module 0x1Axx - XBee-PRO module	CRE	0 - 0xFFFF [read-only]	Factory-set

AI	**Association Indication**. Read information regarding last node join request: 0x00 - Successfully formed or joined a network. (Coordinators form a network, routers and end devices join a network.) 0x21 - Scan found no PANs 0x22 - Scan found no valid PANs based on current SC and ID settings 0x23 - Valid Coordinator or Routers found, but they are not allowing joining (NJ expired) 0x24 - No joinable beacons were found 0x25 - Unexpected state, node should not be attempting to join at this time 0x27 - Node Joining attempt failed (typically due to incompatible security settings) 0x2A - Coordinator Start attempt failed' 0x2B - Checking for an existing coordinator 0x2C - Attempt to leave the network failed 0xAB - Attempted to join a device that did not respond. 0xAC - Secure join error - network security key received unsecured 0xAD - Secure join error - network security key not received 0xAF - Secure join error - joining device does not have the right preconfigured link key 0xFF - Scanning for a ZigBee network (routers and end devices)	CRE	0 - 0xFF [read-only]	--

TABELA 16 Lista de comandos AT para *sleep*

AT	Name and Description	Node	Parameter Range	Default
SM	**Sleep Mode** Sets the sleep mode on the RF module. An XBee loaded with router firmware can be configured as either a router (SM set to 0) or an end device (SM > 0). Changing a device from a router to an end device (or vice versa) forces the device to leave the network and attempt to join as the new device type when changes are applied.	RE	0-Sleep disabled (router) 1-Pin sleep enabled 4-Cyclic sleep enabled 5 - Cyclic sleep, pin wake	0 - Router 4 - End Device
SN	**Number of Sleep Periods.** Sets the number of sleep periods to not assert the On/Sleep pin on wakeup if no RF data is waiting for the end device. This command allows a host application to sleep for an extended time if no RF data is present	CRE	1 - 0xFFFF	1
SP	**Sleep Period.** This value determines how long the end device will sleep at a time, up to 28 seconds. (The sleep time can effectively be extended past 28 seconds using the SN command.) On the parent, this value determines how long the parent will buffer a message for the sleeping end device. It should be set at least equal to the longest SP time of any child end device.	CRE	0x20 - 0xAF0 x 10ms (Quarter second resolution)	0x20
ST	**Time Before Sleep** Sets the time before sleep timer on an end device.The timer is reset each time serial or RF data is received. Once the timer expires, an end device may enter low power operation. Applicable for cyclic sleep end devices only.	E	1 - 0xFFFE (x 1ms)	0x1388 (5 seconds)

SO Command	**Sleep Options**. Configure options for sleep. Unused option bits should be set to 0. Sleep options include: 0x02 - Always wake for ST time 0x04 - Sleep entire SN * SP time Sleep options should not be used for most applications. See chapter 6 for more information.	E	0 - 0xFF	0
WH	**Wake Host**. Set/Read the wake host timer value. If the wake host timer is set to a non-zero value, this timer specifies a time (in millisecond units) that the device should allow after waking from sleep before sending data out the UART or transmitting an IO sample. If serial characters are received, the WH timer is stopped immediately.	E	0 - 0xFFFF (x 1ms)	
SI	**Sleep Immediately**. See Execution Commands table below..			
PO	**Polling Rate**. Sets the polling rate for the end device.	E	0 - 0x1770 (10msec)	0x00 (100 msec)

TABELA 17 Lista de comandos AT de execução

AT	Name and Description		Parameter Range	Default
AC	**Apply Changes**. Applies changes to all command registers causing queued command register values to be applied. For example, changing the serial interface rate with the BD command will not change the UART interface rate until changes are applied with the AC command. The CN command and 0x08 API command frame also apply changes.	CRE	-	
WR	**Write**. Write parameter values to non-volatile memory so that parameter modifications persist through subsequent resets. Note: Once WR is issued, no additional characters should be sent to the module until after the "OK\r" response is received. The WR command should be used sparingly. The EM250 supports a limited number of write cycles.“	CRE	--	--
RE	**Restore Defaults**. Restore module parameters to factory defaults.	CRE	--	--
FR	***Software* Reset.** Reset module. Responds immediately with an OK status, and then performs a *software* reset about 2 seconds later.	CRE	--	--
NR	**Network Reset**. Reset network layer parameters on one or more modules within a PAN. Responds immediately with an “OK” then causes a network restart. All network configuration and routing information is consequently lost. *If NR = 0*: Resets network layer parameters on the node issuing the command. *If NR = 1*: Sends broadcast transmission to reset network layer parameters on all nodes in the PAN.	CRE	0-1	--
SI	**Sleep Immediately**. Cause a cyclic sleep module to sleep immediately rather than wait for the ST timer to expire.	E	-	-

CB	**Commissioning Pushbutton**. This command can be used to simulate commissioning button presses in *software*. The parameter value should be set to the number of button presses to be simulated. For example, sending the ATCB1 command will execute the action associated with 1 commissioning button press.	CRE		
ND	**Node Discover.** Discovers and reports all RF modules found. The following information is reported for each module discovered. MY<CR> SH<CR> SL<CR> NI<CR> (Variable length) PARENT_NETWORK ADDRESS (2 Bytes)<CR> DEVICE_TYPE<CR> (1 Byte: 0=Coord, 1=Router, 2=End Device) STATUS<CR> (1 Byte: Reserved) PROFILE_ID<CR> (2 Bytes) MANUFACTURER_ID<CR> (2 Bytes) <CR> After (NT * 100) milliseconds, the command ends by returning a <CR>. ND also accepts a Node Identifier (NI) as a parameter (optional). In this case, only a module that matches the supplied identifier will respond. If ND is sent through the API, each response is returned as a separate AT_CMD_Response packet. The data consists of the above listed bytes without the carriage return delimiters. The NI string will end in a "0x00" null character. The radius of the ND command is set by the BH command.	CRE	optional 20-Byte NI or MY value	--
DN	**Destination Node.** Resolves an NI (Node Identifier) string to a physical address (case-sensitive). The following events occur after the destination node is discovered: <AT Firmware> 1. DL & DH are set to the extended (64-bit) address of the module with the matching NI (Node Identifier) string. 2. OK (or ERROR)\r is returned. 3. Command Mode is exited to allow immediate communication <API Firmware> 1. The 16-bit network and 64-bit extended addresses are returned in an API Command Response frame. If there is no response from a module within (NT * 100) milliseconds or a parameter is not specified (left blank), the command is terminated and an “ERROR” message is returned. In the case of an ERROR, Command Mode is not exited. The radius of the DN command is set by the BH command.	CRE	up to 20-Byte printable ASCII string	--

IS	**Force Sample** Forces a read of all enabled digital and analog input lines.	CRE	--	--
1S	**XBee Sensor Sample**. Forces a sample to be taken on an XBee Sensor device. This command can only be issued to an XBee sensor device using an API remote command.	RE	-	-