UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

JOÃO PAULO VICENTINI FRACAROLLI

Implementação do controle remoto de uma miniatura de carro operado via computador utilizando comunicação wireless

São Carlos
2012

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Fracarolli, João Paulo Vicentini
F797i Implementação do controle remoto de uma miniatura de carro operado via computador utilizando comunicação wireless / João Paulo Vicentini Fracarolli; Orientadora Luiza Maria Romeiro Codá. 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. Controle remoto. 2. ZigBee. 3. Módulos XBee. 4. Microcontrolador PIC. 5. PySerial. 6. CPLD. I. Título.

JOÃO PAULO VICENTINI FRACAROLLI

IMPLEMENTAÇÃO DO CONTROLE REMOTO DE UMA MINIATURA DE CARRO OPERADO VIA COMPUTADOR UTILIZANDO COMUNICAÇÃO WIRELESS

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

ORIENTADORA: Luiza Maria Romeiro Codá

São Carlos
2012

FOLHA DE APROVAÇÃO

Nome: João Paulo Vicentini Fracarolli

Título: "Implementação do Controle Remoto de uma Miniatura de Carro Operado via Computador Utilizando Comunicação Wireless"

Trabalho de Conclusão de Curso defendido e aprovado em 27 / 11 / 2012,

com NOTA 9,5 (nove, meio), *pela Comissão Julgadora:*

Profa. Assistente Luiza Maria Romeiro Codá (Orientadora)
SEL/EESC/USP

Prof. Associado Evandro Luís Linhari Rodrigues
SEL/EESC/USP

Prof. Assistente Jerson Barbosa de Vargas
SEL/EESC/USP

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

Agradecimentos

Agradeço primeiramente a Deus, por me dar forças para seguir em frente. Aos meus pais, por terem me ensinado a nunca desistir e a agir sempre com ética e dignidade. À professora Luiza Maria Romeiro Codá por ter depositado sua confiança no meu trabalho e por me orientar no seu desenvolvimento, dando sugestões e tirando as dúvidas que surgiram pelo caminho. Ao professor Adilson Gonzaga por disponibilizar o uso do laboratório e de todos os equipamentos assim como o do *kit Lego Dacta* e ao professor Evandro Luís Linhari Rodrigues por emprestar seus módulos *XBee* no início do desenvolvimento do projeto. Aos meus amigos e colegas de laboratório Rafael Santos Moura e Igor Guerrero, que acompanharam o desenvolvimento do projeto e contribuíram com sugestões e ideias. Aos amigos que conheci em São Carlos e que sempre foram grandes companheiros durante a vida universitária. A toda minha família pelo apoio e também aos meus amigos de minha cidade natal, os quais considero minha segunda família. Agradeço também à Escola de Engenharia de São Carlos e a todos os professores e funcionários que contribuíram com a minha formação.

Resumo

Este trabalho consiste no desenvolvimento de um sistema para controlar uma miniatura de carro construída com peças de *Lego*. Os comandos são adquiridos pelo teclado de um computador, sendo transmitidos serialmente a um modem *XBee* que, por sua vez, se comunica com outro modem situado no carro, utilizando o protocolo de comunicação sem fio *ZigBee*. Um microcontrolador PIC (*Peripherial Interface Controller*) interpreta os comandos, gerando sinais para o controle dos sistemas responsáveis pela tração e pela esterção das rodas do carro. A interface com o usuário foi escrita utilizando a linguagem *Python* com o auxílio da biblioteca *PySerial*, que permite que dados sejam facilmente enviados ou lidos de uma porta serial. O sistema responsável pelo movimento do carro foi dividido em duas partes: o controle de tração, utilizando um *driver* do tipo ponte H conectado aos terminais de um motor de corrente contínua e o controle de direção, empregando um servomotor acionado por um sinal PWM (*Pulse Width Modulation*) gerado por um Dispositivo Lógico Programável Complexo (CPLD).

Palavras chave: Controle remoto, *ZigBee*, módulos *XBee*, microcontrolador PIC, *PySerial*, CPLD.

Abstract

This work consists in the development of a system to control a miniature car made of Lego blocks. Commands are acquired by a computer's keyboard and transmitted serially to an XBee modem which communicates with another modem located in the car using the ZigBee wireless communication protocol. A PIC (Peripheral Interface Controller) microcontroller interprets the commands, generating signals to control the systems responsible for the traction and steering of the car's wheels. The user interface was developed using the Python programming language, with the help of PySerial library, which allows data to be easily sent or read from a serial port. The system responsible for the movement of the car was divided into two parts: the traction control, using an H bridge driver connected to the terminals of a direct current motor and the steering control, using a servo motor driven by a PWM (Pulse Width Modulation) signal generated by a Complex Programmable Logic Device (CPLD).

Keywords: Remote Control, ZigBee, XBee modules, PIC microcontroller, PySerial, CPLD.

Conteúdo

Índice de figuras

Índice de tabelas

Lista de abreviações e siglas

ACK	Acknowledgement
AODV	Ad hoc On-demand Distance Vector
APS	Application Support
CI	Circuito Integrado
CPLD	Complex Programmable Logic Device
DC	Direct Current
DIP	Dual In-line Package
EEPROM	Electrically Erasable Programmable Read Only memory
FPGA	Field Programmable Gate Array
I²C	Inter-Integrated Circuit
LED	Light Emmiting Diode
PAN	Personal Area Network
PC	Personal Computer
PDA	Personal Digital Assistant
PIC	Peripheral Interface Controller
PWM	Pulse Width Modulation
RAM	Random Access Memory
RF	Radiofrequência
RISC	Reduced Instruction Set Computer
RP-SMA	Reverse Polarity SubMiniature version A
SCI	Serial Connect Interface
SMA	SubMiniature version A
SPI	Serial Peripherial Interface
USART	Universal Synchronous/Asynchronous Transmitter/Receiver
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WLAN	Wireless Local Area Network

1. Introdução

1.1. Motivação

O passo inicial para a realização deste trabalho ocorreu devido a uma visita ocasional ao Laboratório de Eletrônica Digital, onde os alunos monitores das disciplinas de eletrônica digital desenvolvem projetos para serem utilizados nos experimentos do laboratório das disciplinas de eletrônica digital, de forma a estimular o interesse por eletrônica e também incentivar o reaproveitamento de materiais que estão em desuso. Neste contexto, surgiu a possibilidade de iniciar um novo projeto utilizando um *kit Lego Dacta*. A ideia de implementar o controle remoto do carro surgiu pelo possível interesse que este tipo de aplicação poderia despertar nos alunos dos semestres iniciais da engenharia elétrica, além de ser uma forma simples de aplicar os conhecimentos adquiridos com o decorrer do curso.

Um aspecto particular deste projeto é que, diferentemente de carros de controle remoto comumente encontrados em lojas de brinquedos, os comandos são acionados utilizando-se um computador pessoal (PC). Desta forma, é possível desenvolver um *software* atuando como interface entre o usuário e o carro, o que garante maior flexibilidade e a possibilidade de incluir diferentes funções no controle remoto. A interface de comunicação entre o PC e o carro utiliza uma rede *wireless* com o protocolo *ZigBee*, por meio de *modems XBee*. Como estes *modems* já incluem todas as funcionalidades para a comunicação sem fio, a construção do *hardware* é bastante simplificada, direcionando o foco deste trabalho no desenvolvimento do sistema.

1.2. Visão geral do projeto

A Figura 1 mostra o diagrama em blocos do sistema de um ponto de vista geral. O projeto é dividido em três partes principais: a interface, a transmissão e o carro. A interface se refere à leitura de dados pelo computador e configuração do módulo *XBee* coordenador. Para enviar os comandos para o sistema, o usuário deve utilizar o terminal desenvolvido usando a linguagem *Python* e para configurar a rede e os dispositivos da mesma, utiliza-se

o software *X-CTU*, disponibilizado pelo fabricante dos módulos *XBee*. A transmissão de dados é feita através da rede sem fio pelo protocolo *ZigBee*.

Por fim, a terceira parte trata do controle dos motores do carro. São utilizados dois motores: um de corrente contínua, para a tração do carro e cuja direção é controlada por um *driver* do tipo ponte H, e um servomotor, para a direção das rodas dianteiras, onde o deslocamento angular é controlado por modulação de largura de pulsos (PWM). Os sinais de acionamento da ponte H e o do PWM são gerados pelo microcontrolador PIC, que analisa os comandos recebidos do usuário. O sinal PWM é gerado por um CPLD *Altera MAX*, da família 3000S. Cada elemento constituinte do sistema da Figura 1 está descrito detalhadamente no capítulo 2.

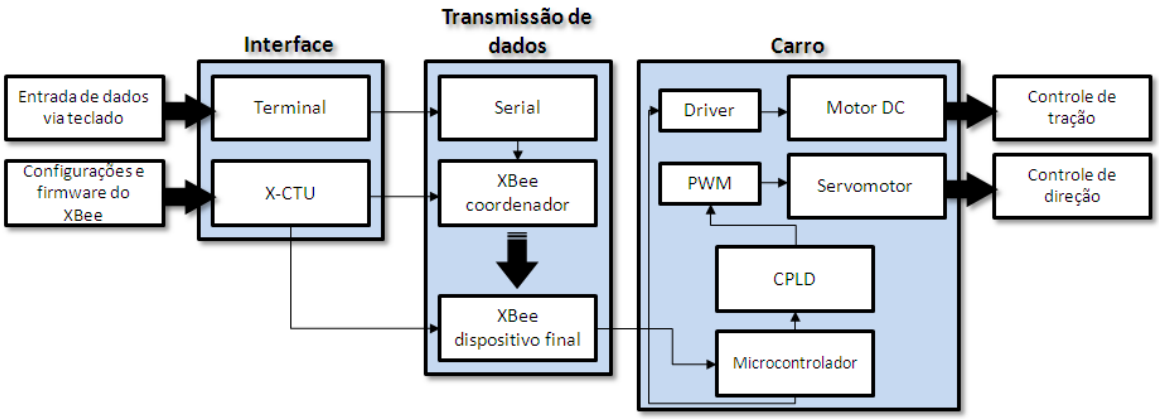

Figura 1: Representação do sistema completo em diagrama de blocos.

1.3. Estruturação do documento

Este trabalho está dividido em capítulos que descrevem as etapas do projeto e seções que tratam de assuntos específicos de cada etapa.

Capítulo 1: Introduz o projeto de forma geral, descreve a motivação e a organização da monografia.

Capítulo 2: Contém os fundamentos teóricos necessários para a contextualização e o bom entendimento das etapas realizadas durante o projeto.

Capítulo 3: Apresenta detalhadamente todas as etapas do projeto, como materiais utilizados, circuitos e softwares desenvolvidos.

Capítulo 4: Neste capítulo são mostrados e discutidos os resultados obtidos a partir dos testes realizados durante e após a elaboração do sistema.

Capítulo 5: As conclusões a respeito do trabalho desenvolvido são apresentadas, juntamente com sugestões de melhorias e adaptações.

Capítulo 6: Contém as referências bibliográficas.

Capítulo 7: Apresenta os apêndices: códigos-fonte dos programas desenvolvidos, cálculos e a lista de materiais utilizados.

Capítulo 8: Apresenta os anexos: dados relevantes extraídos de documentos utilizados como fontes bibliográficas.

2. Fundamentos teóricos

Neste capítulo é realizada uma abordagem sobre os fundamentos teóricos necessários para o desenvolvimento do projeto. Cada parte do sistema descrito no sub item 1.2, separada segundo sua função, é apresentada a seguir.

2.1. Movimentação do carro

A Figura 2, retirada do diagrama da Figura 1 do capítulo 1, mostra o bloco responsável pela movimentação do carro. Ela consiste no motor DC, servomotor (que agem diretamente na movimentação) e os meios de acioná-los (*driver* ponte H e PWM, respectivamente). Os sinais para o controle destes dispositivos são gerados pelo microcontrolador PIC e pelo CPLD.

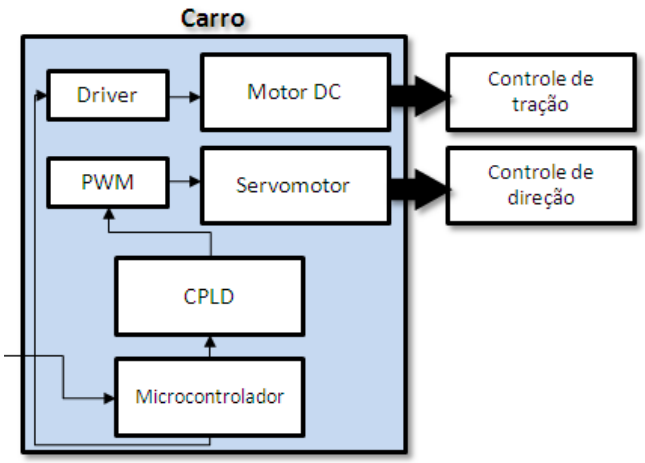

Figura 2: Diagrama em blocos do subsistema responsável pela movimentação do carro

2.1.1. Motor DC

Um motor DC ou motor CC é um tipo de máquina elétrica que opera com corrente contínua (GONZAGA e JULIANI, 2008). É composto por dois enrolamentos: o indutor, ou enrolamento de campo, que é localizado no estator (a parte fixa da máquina) e o induzido, ou enrolamento de armadura, situado no rotor (a parte girante). Para que o motor funcione,

é preciso que haja fluxo de potência para a armadura. Desta forma, são utilizadas escovas, também chamadas de comutadores, para que seja estabelecido o contato elétrico girante ao enrolamento de armadura. O funcionamento de um motor DC se baseia no magnetismo, utilizando as propriedades básicas de atração e repulsão de pólos para gerar movimento de rotação. O sentido de rotação de um motor DC é controlado pela polaridade da tensão aplicada entre os seus terminais, o que permite a utilização de um simples *driver* do tipo Ponte H para a inversão da rotação do motor.

2.1.2. Driver Ponte H

O controle de um motor DC para pequenas aplicações se resume no controle do sentido de sua rotação. Conforme já mencionado na seção 2.1.1, o sentido de rotação é diretamente influenciado pela polaridade da tensão aplicada nos terminais da armadura. A ponte H, na Figura 3, é um circuito simples composto de chaves (relês, transistores) que, sendo corretamente acionadas, podem inverter a tensão aplicada nos terminais do motor. A sua configuração tem o aspecto da letra H, de onde vem o seu nome. Para que a ponte H funcione corretamente, as chaves diagonalmente opostas devem ser acessadas simultaneamente. Desta forma, quando a chave superior esquerda e a inferior direita são acionadas, a corrente flui da esquerda para a direita. Se, por outro lado, as chaves superior direita e inferior esquerda forem fechadas, a corrente flui da direita para a esquerda (PATSKO, 2006). Estas situações são ilustradas na Figura 4.

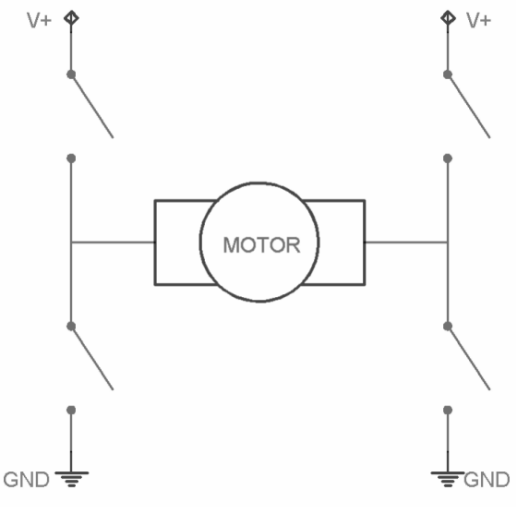

Figura 3: Ponte H (PATSKO, 2006).

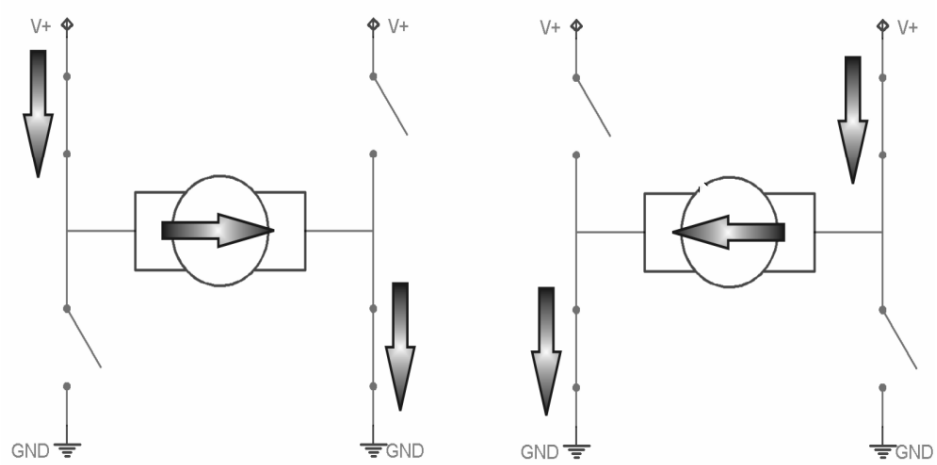

Figura 4: Possíveis estados ativos da ponte H (PATSKO, 2006).

Uma situação que nunca deve ocorrer é quando todas as chaves são fechadas, colocando os terminais positivo e negativo da fonte de alimentação em curto-circuito. Qualquer outro possível estado das chaves resulta na parada do motor. A tabela da verdade para a ponte H é mostrada na Tabela 1, onde SE, SD, IE e ID identificam as chaves superior esquerda, superior direita, inferior esquerda e inferior direita, respectivamente (PATSKO, 2006).

Tabela 1: Tabela da verdade da ponte H. 0 = Chave aberta, 1 = Chave fechada, X = qualquer estado

SE	SD	IE	ID	Ação
1	0	0	1	Motor gira num sentido S
0	1	1	0	Motor gira no sentido oposto a S
1	1	1	1	Estado proibido
X	0	X	0	Motor parado
0	X	0	X	Motor parado

2.1.3. Servomotor

Um servomotor é um pequeno dispositivo cujo eixo pode ser posicionado em ângilos específicos de acordo com um sinal de controle codificado. São dispositivos de malha fechada com controle de posição. Uma das principais diferenças entre os servomotores e os motores convencionais é o menor grau de liberdade do seu eixo. Enquanto motores DC, por

exemplo, tendem a girar indefinidamente, o eixo do servomotor é restrito a cerca de 180º. Entretanto, o seu posicionamento é mais preciso (SEATTLE ROBOTICS SOCIETY, 2004).

O servomotor possui junto ao seu eixo um potenciômetro, que indica ao controlador qual é o ângulo em que este se encontra. Desta forma, o sistema de controle atua no motor até que o erro entre o ângulo atual e o ângulo desejado seja o menor possível. Em muitos modelos de servomotores (inclusive no modelo escolhido para este projeto) para indicar ao sistema qual o ângulo desejado, utiliza-se um sinal com largura de pulso modulada (PWM). Este sinal deve ser periódico, com período de 20ms, sendo que o tempo em nível lógico alto varia entre 1 e 2ms. A Figura 5 mostra os sinais de controle aplicados e o seu respectivo efeito no eixo do servomotor (SANTOS, 2007).

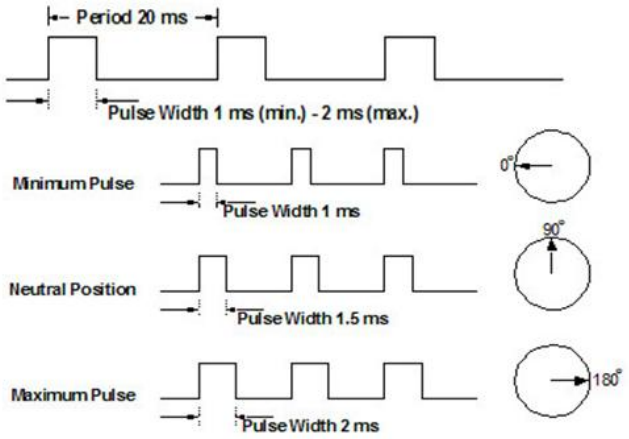

Figura 5: Sinal de PWM para o controle de um servomotor (ELECTROONS, 2012).

Pela Figura 5, é possível perceber que a largura do pulso influencia diretamente no ângulo do eixo do motor. Assim, pode-se definir qualquer largura intermediária para que a posição seja fixada em outros ângulos.

2.2. Transmissão de dados

Esta subseção aborda os meios de transmissão de informações utilizados para a comunicação do computador com o carro. O sistema de transmissão envolve comunicação serial e comunicação *wireless*, utilizando a tecnologia *ZigBee*, conforme mostra a Figura 6.

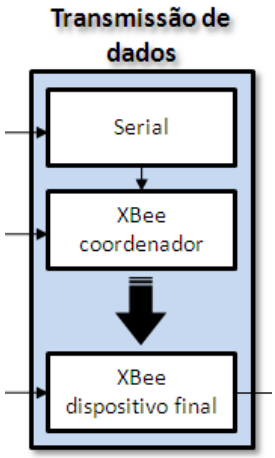

Figura 6: Diagrama em blocos do subsistema responsável pela transmissão de dados

2.2.1. Comunicação serial

A comunicação de dados em microprocessadores e dispositivos similares pode ser realizada de forma paralela ou serial. Na comunicação paralela, cada bit é transmitido por meio de uma linha particular. Já em uma comunicação serial, os bits são enviados, um após o outro, pela mesma linha. As figuras 7 e 8 representam os dois tipos de comunicação entre dois dispositivos.

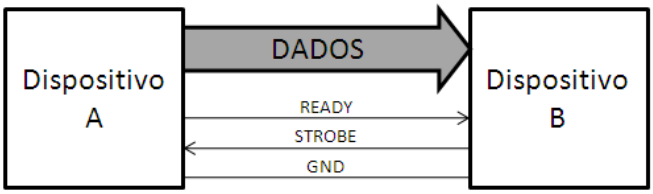

Figura 7: Comunicação paralela

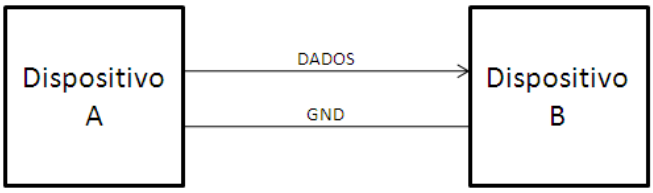

Figura 8: Comunicação serial

A interface paralela é mais veloz, porém é mais complexa, requerendo maior quantidade de conexões. Já a interface serial é simples, entretanto perde em velocidade quando comparada à comunicação paralela (KRUTZ, 1988). A comunicação serial pode ser síncrona ou assíncrona. No primeiro caso, há a necessidade de utilizar um sinal de *clock*

associado à informação a ser transmitida para que a sincronização entre o transmissor e o receptor seja realizada. No segundo caso, o sincronismo é feito por sinais especiais juntamente com a informação (MONTEIRO, 2007). A Tabela 2 apresenta uma comparação entre a comunicação serial síncrona e assíncrona.

Tabela 2: Comparação entre a transmissão serial assíncrona e a síncrona (WIKIPEDIA, 2012).

Transmissão	Vantagens	Desvantagens
Assíncrona	• Simples: não requer sincronização de ambos os lados; • Barato: como o tempo não é algo crítico, o *hardware* tem custo menor; • A configuração é mais rápida do que em outros tipos de transmissão.	• Alta proporção de bits de controle, que não estão relacionados com a informação a ser transmitida.
Síncrona	• Maior rendimento	• Maior complexidade; • O *hardware* tem maior custo.

Neste projeto, foi decidido trabalhar com comunicação assíncrona, já que não há a necessidade de alto rendimento na comunicação, devido à grande quantidade de dados a serem enviados para os dispositivos (longas sequências de caracteres do teclado). Apesar de este modo possuir uma quantidade considerável de bits de controle, não é obrigatório o uso de todos estes bits. O formato geral de um caractere na transmissão assíncrona é mostrado na Figura 9.

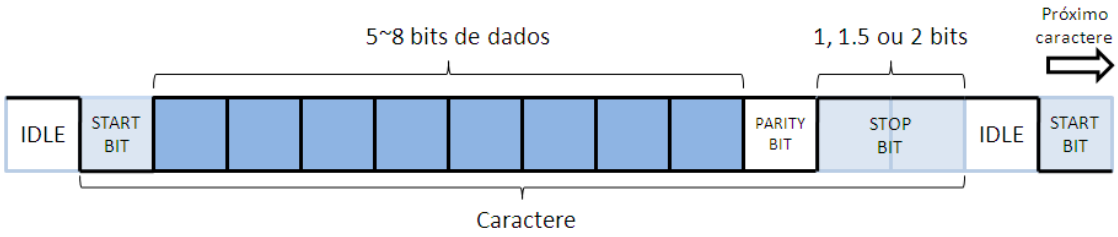

Figura 9: Formato geral de um caractere na comunicação serial assíncrona

No estado de repouso, a linha de transmissão permanece em nível lógico alto. Uma mudança de estado sinaliza que uma informação está prestes a ser transmitida. Este sinal é denominado *Start Bit* e tem a duração de um bit. Logo em seguida são transmitidos os dados, que podem variar de 5 a 8 bits. Um bit de paridade pode ser utilizado para verificar se houve algum erro nos dados enviados. Por fim, é transmitido um sinal em nível alto (*Stop*

Bit) com duração de 1, 1,5 ou 2 bits, sinalizando o fim da transmissão. A linha então retorna ao estado de repouso.

2.2.2. Comunicação *Wireless*

Atualmente, existe uma infinidade de dispositivos que se comunicam utilizando redes sem fio (*wireless*). A tecnologia *wireless* está cada dia mais presente no cotidiano das pessoas, por se mostrar uma solução mais simples, econômica e viável que o uso de cabos, principalmente quando há a necessidade de muitos dispositivos na rede (MATTOS, 2006). Dentre os padrões de comunicação sem fio existentes, destacam-se o *Wi-Fi*, o *Bluetooth* e o *ZigBee*. Estes padrões utilizam métodos diferentes de transmissão e são indicados para diferentes aplicações (SENA BLOG, 2010). A Tabela 3 mostra uma comparação entre estas três tecnologias, tomando como base seus principais aspectos.

Tabela 3: Comparação entre as tecnologias ZigBee, Wi-Fi e Bluetooth (SENA BLOG, 2010)

	ZigBee™	Wi Fi	Bluetooth
Alcance	10 ~100m*	50 ~100m	10 ~100m
Topologia de rede	Ad-hoc, P2P, estrela, mesh	Point-to-hub	Ad-hoc, redes muito pequenas
Frequência de operação	868 MHz (Europa), 900~928 MHz (América do Norte), 2,4 GHz (Global)	2,4 e 5 GHz	2,4 GHz
Complexidade	Baixa	Alta	Alta
Consumo	Muito baixo	Alto	Médio
Segurança	Camada de segurança 128 AES		Criptografia de 64 e 128 bits
Aplicações	Controle e monitoramento industrial, sensores, automação residencial e predial, brinquedos, jogos.	Conectividade em WLANs, acesso à Internet.	Conectividade wireless entre dispositivos (celulares, PDAs, laptops, fones de ouvido)

*Alcance *indoor* médio

Pela análise da Tabela 3, percebe-se que a tecnologia *ZigBee* possui um baixo consumo de energia e baixa complexidade, sendo vantajosa em sistemas simples e alimentados por baterias. Em comparação com as demais tecnologias, *ZigBee* mostra-se a mais adequada a este projeto.

2.2.3. ZigBee

ZigBee é uma tecnologia *wireless* desenvolvida pela *ZigBee Alliance* (ZIGBEE ALLIANCE, 2012) com o objetivo de operar em redes com dispositivos que requerem baixo consumo, baixo custo e baixas taxas de transmissão de dados. O protocolo *ZigBee* opera a uma frequência de 2,4 GHz e é baseado na norma IEEE 802.15.4. Dispositivos que utilizam o protocolo ZigBee podem facilmente constituir uma rede *mesh*, ou rede de malha que é um tipo de topologia constituída por pontos de acesso (clientes e roteadores *mesh*) interconectados. Os roteadores formam a estrutura da rede, garantindo acesso aos nós clientes (AKYILDIZ, WANG e WANG, 2005).

2.2.4. Redes *ZigBee* (DIGI INTERNATIONAL INC., 2008)

As redes *ZigBee* são conhecidas como PANs (*Personal Area Network*) e cada rede possui um identificador de 16 bits chamado PAN ID. Nestas redes podem conter três classes de dispositivos distintos:

- O coordenador (C);
- Os roteadores (R);
- Os dispositivos finais (E).

Em uma PAN só pode haver um coordenador, mas é possível a presença de vários roteadores e dispositivos finais. Cada classe de dispositivos possui tarefas específicas. A Tabela 4 apresenta as características de cada uma dessas classes.

Tabela 4: Classes de dispositivos numa rede ZigBee

Coordenador	• Seleciona o canal de operação e o PAN ID; • Inicia uma nova PAN; • Permite que roteadores e dispositivos finais entrem em uma PAN; • Pode transmitir e receber dados RF; • Pode auxiliar no roteamento de dados; • Não é projetado para operar com bateria; • Não pode operar no modo *sleep*.
Roteador	• Precisa entrar em uma PAN para começar a operar; • Pode permitir a entrada de outros roteadores e dispositivos finais na rede; • Pode transmitir e receber dados RF; • Realiza o roteamento dos pacotes na rede; • Não pode operar no modo *sleep*.
Dispositivo Final	• Precisa entrar em uma PAN para começar a operar; • Não pode permitir a entrada de outros dispositivos na rede; • Não pode auxiliar no roteamento; • Pode transmitir e receber dados RF; • É projetado para operar com bateria; • Pode operar no modo *sleep*; • O roteador ou coordenador que permite a entrada de um dispositivo final na rede é chamado de nó pai, e o dispositivo final é o nó filho.

Uma rede *ZigBee* é iniciada quando o coordenador escolhe um PAN ID. Assim que roteadores e dispositivos entram na rede, eles herdam o PAN ID do coordenador. Cada novo dispositivo na rede recebe um endereço de 16 bits, e o endereço do coordenador é sempre 0. Para criar a rede, o coordenador deve realizar os processos de *energy scan* e *PAN scan*.

- ***Energy Scan*** – varredura feita pelo coordenador em múltiplos canais (frequências) para detectar os níveis de energia em cada canal. Canais com níveis de energia em excesso são removidos da lista de potenciais canais para iniciar a PAN.
- ***PAN Scan*** – ocorre após o *energy scan*. O coordenador varre os canais restantes em busca de PANs existentes, utilizando um sinal *broadcast* chamado *Beacon Request*. Quaisquer roteadores ou dispositivos finais por perto respondem com um *Beacon Frame*, que contém informações sobre a PAN que o dispositivo se encontra, incluindo o PAN ID e se o dispositivo permite ou não a entrada de novos nós.

Terminado o *PAN Scan*, o coordenador analisa os frames recebidos e tenta iniciar a rede em um PAN ID e um canal que não estejam sendo utilizados.

Quando roteadores e dispositivos finais são inicializados, eles devem encontrar e se juntar a uma PAN. Para isso, deve-se realizar um *PAN Scan*, o que resulta em uma lista de *beacons* de dispositivos próximos. O roteador ou dispositivo final então analisa esta lista a procura de uma rede *ZigBee* válida. Estes dispositivos podem ser configurados para entrar em qualquer rede ou somente em uma rede com um PAN ID específico. De qualquer forma, devem sempre encontrar um coordenador ou roteador que permita sua entrada. Assim que um dispositivo encontra uma PAN válida, ele envia um *Association Request* ao nó correspondente. A Figura 10 ilustra o processo de junção de um dispositivo a uma rede.

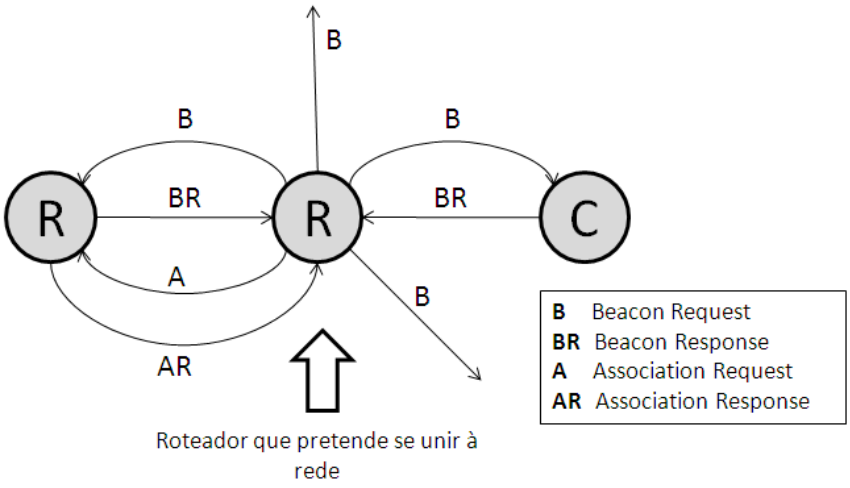

Figura 10: União de um dispositivo a uma PAN ZigBee

2.2.5. Comunicações em uma rede *ZigBee* (DIGI INTERNATIONAL INC., 2008)

Dispositivos *ZigBee* suportam endereçamento de dispositivos e endereçamento na camada de aplicação. O endereçamento de dispositivos tem como função especificar para onde os pacotes devem ser enviados e quem os enviou. São especificados dois tipos de endereços:

- **Endereço de rede de 16 bits:** este endereço é atribuído a um nó quando este se une a uma rede e é exclusivo para cada novo dispositivo da PAN. Entretanto, não são estáticos: um mesmo dispositivo recebe um endereço de rede diferente toda vez que entra em uma PAN.
- **Endereço de 64 bits:** este endereço é permanente para cada dispositivo.

Já o endereçamento na camada de aplicação indica um recipiente de aplicação particular chamado *ZigBee Endpoint*, juntamente com uma mensagem chamada *Cluster ID*.

- ***Endpoint*:** é uma tarefa ou aplicação que roda em um dispositivo *ZigBee*, similar a uma porta TCP. Cada dispositivo pode suportar um ou mais *endpoints*.
- ***Cluster ID*:** define uma função ou ação particular em um dispositivo.

Os pacotes em uma rede *ZigBee* são endereçados utilizando tanto o endereçamento de dispositivos quanto o endereçamento na camada de aplicação. A transmissão de dados pode ser feita via mensagens *broadcast* ou *unicast*. Mensagens *broadcast* são enviadas para todos os nós da rede e mensagens *unicast* são sempre destinadas a um endereço de rede de 16 bits específico. Contudo, como os endereços de 16 bits são mutáveis, é preciso realizar um processo de descoberta de endereço de rede a partir do endereço de 64 bits conhecido. Dispositivos *ZigBee* podem realizar algoritmos descoberta de endereço e descoberta de rota até um determinado nó.

- **Descoberta de endereço de rede:** o objetivo deste processo é descobrir o endereço de 16 bits que corresponde a um endereço de 64 bits conhecido de um dispositivo. Para que isto seja possível, o dispositivo que inicia a transmissão deve enviar uma mensagem *broadcast* contendo o endereço de 64 bits do nó com o qual deseja se comunicar. Cada dispositivo que recebe a mensagem compara o endereço da mensagem com o seu próprio endereço. Quando os endereços forem iguais, o nó destino envia seu endereço de 16 bits para quem enviou o *broadcast*. Assim que a resposta é obtida, o iniciador da comunicação começa a transmitir os dados.
- **Descoberta de rota:** o estabelecimento da rota entre os nós fonte e destino é feito através de roteamento *mesh*, que permite que pacotes viajem por múltiplos nós até chegar ao destino. A passagem do pacote de um nó para outro é chamado *hop*. A descoberta de rota é um processo baseado no protocolo AODV (*Ad hoc On-demand Distance Vector*).

2.2.6. Interface serial e modos de operação (DIGI INTERNATIONAL INC., 2008).

Dispositivos *ZigBee* podem usar dois tipos de protocolos de interface serial.

- **Modo transparente:** a comunicação atua como em uma linha serial. Os dados a serem transmitidos entram no módulo fonte pelo pino de entrada de dados, são transmitidos via RF pela antena e são enviados para o pino de saída de dados do

módulo destino. Este modo pode ser configurado através de comando especiais (comandos AT).

- **Modo API (*Application Programming Interface*):** os dados são contidos em frames, que podem ser do tipo *Transmit Data Frames* (dados de entrada) ou *Receive Data Frames* (dados de saída). Este modo permite que a transmissão de dados para múltiplos destinos seja feita sem a necessidade de entrar em modo de comando. Existem indicações de sucesso ou falha de envio de pacotes e, para cada pacote, há a identificação do endereço fonte.

Os módulos podem operar em diferentes modos: o modo de repouso, modo de transmissão, modo de recepção, modo de comandos e modo sleep.

- **Modo de repouso (*idle*):** modo em que o dispositivo se encontra quando não está transmitindo nem recebendo dados. Assim, o módulo checa se existe algum dado RF a ser transmitido ou para ser recebido nos *buffers* de recebimento serial ou transmissão, respectivamente.
- **Modo de transmissão:** o módulo entra neste modo quando um dado serial é recebido e está pronto para ser empacotado. Antes de transmitir o dado pela antena, o módulo verifica se o endereço de 16 bits e a rota até o destino são conhecidos. No caso em que o endereço é desconhecido, dá-se início ao descobrimento de endereço, e posteriormente, se a rota for desconhecida, inicia-se o descobrimento de rota. Caso estes procedimentos falhem, os dados são descartados. As figuras 11 e 12 ilustram o processo de transmissão de dados recebidos serialmente.

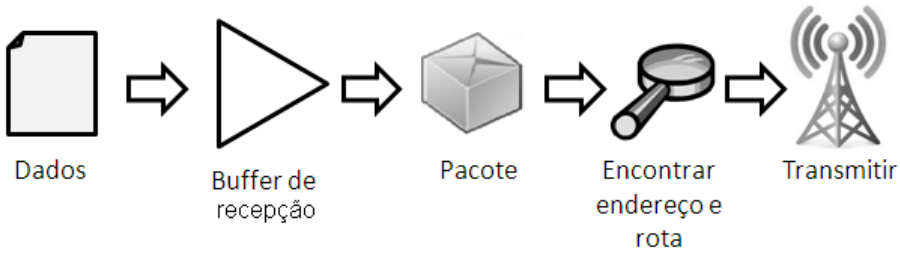

Figura 11: Transmissão serial em um dispositivo ZigBee

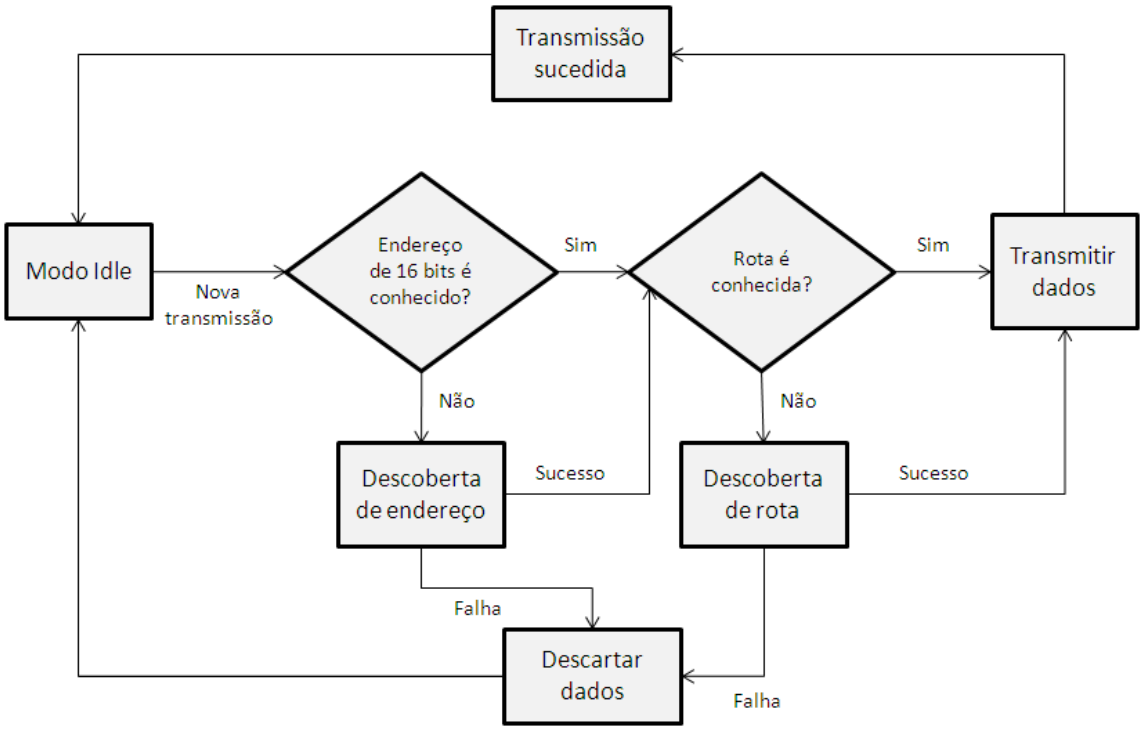

Figura 12: Ilustração do processo de procura de endereço e rota para transmissão de dados

- **Modo de recepção:** quando um pacote RF é recebido, os dados são transferidos para o *buffer* de transmissão serial. A Figura 13 ilustra o processo de recepção de dados.

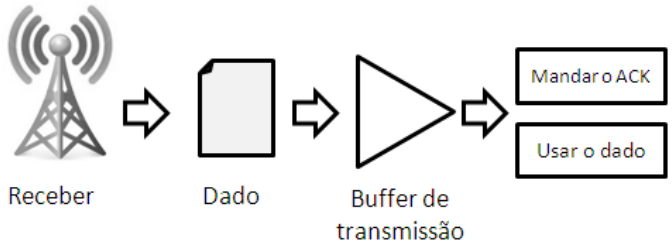

Figura 13: Recepção serial de dados em um dispositivo ZigBee

- **Modo de comandos:** é um modo especial onde caracteres seriais enviados são interpretados como comandos que podem modificar ou ler parâmetros do módulo. Este modo é conhecido como modo de comandos AT. Para ativar o modo de comandos, deve-se proceder da seguinte forma:

- Não enviar nenhum caractere por 1 segundo;
- Enviar os caracteres “+++” dentro de 1 segundo;
- Esperar 1 segundo.

Os tempos mencionados acima podem ser modificados pelos comandos GT e CC (Anexo I – Comandos AT). Se a sequência acima estiver correta, o módulo envia a *string* “OK\r” pelo pino de saída de dados. Ao entrar no modo de comando, o módulo pode receber comandos AT pelo pino de entrada de dados. A sintaxe dos comandos AT segue o padrão indicado na Figura 14.

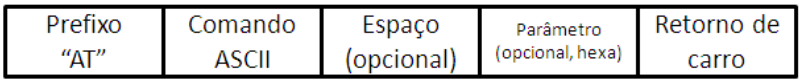

Figura 14: Estrutura de um comando AT

Se o parâmetro de um comando AT for omitido, será retornado o valor do parâmetro atual. Para confirmar alterações realizadas nos parâmetros do módulo, deve-se utilizar o comando WR. Para deixar o modo de comandos, pode-se utilizar o comando “ATCN” seguido de um retorno de carro, ou não enviar nenhum comando dentro do tempo definido pelo parâmetro CT.

- **Modo *sleep*:** é um modo disponível para dispositivos finais que entram em estado de baixo consumo de energia quando não estão sendo utilizados. Pode ser ativado por transição de pino ou utilizando *sleep* cíclico. O dispositivo é acordado quando seu nó pai possui dados destinados a ele.

2.2.7. Comissionamento e diagnóstico de rede (DIGI INTERNATIONAL INC., 2008)

Uma forma de testar o *link* entre dois módulos é o envio de dados para o *cluster ID* de *loopback*, no *endpoint* de dados. Assim, os dados enviados serão ecoados e pode-se verificar a integridade da conexão através do monitoramento da linha de dados. Este teste pode ser feito tanto sob o *firmware* AT quanto sob o *firmware* API. No *firmware* AT, deve-se configurar os parâmetros abaixo no modo de comandos:

- Ativar o parâmetro que habilita endereçamento na camada de aplicação (ZA=1);
- Configurar o parâmetro do *cluster ID* (CI = 0x12);
- Configurar os *endpoints* da fonte e do destino (SE = DE = 0xE8);

- Configurar o endereço do destino (DH e DL)
- Sair do modo de comando.

No *firmware* API, o processo acima pode ser feito utilizando um único *frame*, “*Explicit Addressing ZigBee Command*” (0x11), usando 0x12 como *cluster ID* e 0xE8 como *endpoint* destino. Os dados do pacote serão ecoados.

3. Metodologia de construção

Neste capítulo são descritos os métodos empregados para a elaboração do projeto, baseando-se nas informações teóricas apresentadas no capítulo 2. Cada componente utilizado na montagem do sistema é apresentado nas subseções seguintes.

3.1. Módulos *XBee ZNet 2.5*

O protocolo de comunicação escolhido para este projeto foi o *ZigBee*. Suas principais vantagens são o baixo consumo e a baixa complexidade (SENA BLOG, 2010). Dentre os dispositivos que utilizam o protocolo *ZigBee*, necessitava-se de um componente pequeno, customizável e que abrangesse as funcionalidades do protocolo da forma mais geral o possível. O produto mais comum encontrado foi o módulo *XBee*, fabricado pela *Digi International*. Trata-se de um pequeno *modem* que pode se comunicar com outros módulos *XBee* de forma que uma rede *ZigBee* possa ser construída. A Figura 15 mostra um módulo igual aos utilizados, com conector RP-SMA. Desta forma, foram necessárias duas antenas semelhantes às usadas em roteadores.

Figura 15: Módulo XBee ZNet 2.5 com conector RP-SMA (http://www.digiwireless-solutions.com)

Os módulos *XBee* podem ser encontrados nas versões padrão e PRO. A versão PRO difere da versão padrão por possuir maior desempenho, porém maior consumo de energia (DIGI INTERNATIONAL INC., 2008). A Tabela 5 mostra uma comparação entre as características principais das duas versões. Neste projeto, foram utilizados módulos na versão padrão, pois suas propriedades se mostraram suficientes para esta aplicação, além de possuírem um preço menor.

Tabela 5: Comparação de desempenho entre as versões padrão e PRO dos módulos XBee ZNet 2.5 (DIGI INTERNATIONAL INC., 2008)

Table 1-01. Specifications of the XBee /XBee-PRO ZNet 2.5 OEM RF Module

Specification	XBee ZNet 2.5	XBee PRO ZNet 2.5
Performance		
Indoor/Urban Range	up to 133 ft. (40 m)	up to 300 ft. (100 m)
Outdoor RF line-of-sight Range	up to 400 ft. (120 m)	up to 1 mile (1.6 km)
Transmit Power Output	2mW (+3dBm), boost mode enabled 1.25mW (+1dBm), boost mode disabled	63mW (+18 dBm) 10mW (+10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 - 230400 bps (non-standard baud rates also supported)	1200 - 230400 bps (non-standard baud rates also supported)
Receiver Sensitivity	-96 dBm, boost mode enabled -95 dBm, boost mode disabled	-102 dBm
Power Requirements		
Supply Voltage	2.1 - 3.6 V	3.0 - 3.4 V
Operating Current (Transmit, max output power)	40mA (@ 3.3 V, boost mode enabled) 35mA (@ 3.3 V, boost mode disabled)	295mA (@3.3 V)
Operating Current (Receive))	40mA (@ 3.3 V, boost mode enabled) 38mA (@ 3.3 V, boost mode disabled)	45 mA (@3.3 V)
Idle Current (Receiver off)	15mA	15mA
Power-down Current	< 1 uA @ 25°C	< 1 uA @ 25°C
General		
Operating Frequency Band	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960" x 1.087" (2.438cm x 2.761cm)	0.960 x 1.297 (2.438cm x 3.294cm)
Operating Temperature	-40 to 85° C (industrial)	-40 to 85° C (industrial)
Antenna Options	Integrated Whip, Chip, RPSMA, or U.FL Connector*	Integrated Whip, Chip, RPSMA, or U.FL Connector*
Networking & Security		
Supported Network Topologies	Point-to-point, Point-to-multipoint, Peer-to-peer, and Mesh	Point-to-point, Point-to-multipoint, Peer-to-peer, and Mesh
Number of Channels	16 Direct Sequence Channels	13 Direct Sequence Channels
Addressing Options	PAN ID and Addresses, Cluster IDs and Endpoints (optional)	PAN ID and Addresses, Cluster IDs and Endpoints (optional)
Agency Approvals		
United States (FCC Part 15.247)	OUR-XBEE2	MCQ-XBEEPRO2
Industry Canada (IC)	4214A-XBEE2	1846A-XBEEPRO2
Europe (CE)	ETSI	ETSI
RoHS	Compliant	Compliant

Para possibilitar a comunicação sem fio, foram utilizados dois módulos *XBee*: um coordenador, ligado ao computador e um dispositivo final, no carro. Para configurar os

módulos, foi utilizado o programa X-CTU, disponível para *download* no site da *Digi International*. O X-CTU é usado para a gravação do *firmware* no módulo, configuração de parâmetros e testes. Primeiramente, o módulo deve ser ligado a uma porta USB (Universal Serial Bus) do computador. Para isso, foi preciso adquirir um adaptador. Dentre os vários adaptadores encontrados, o CON-USBBEE, desenvolvido pela Rogercom mostrou-se o mais viável e barato, já que é um produto nacional. A Figura 16 mostra o adaptador, sem o módulo *XBee*.

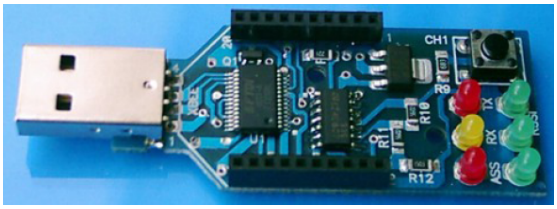

Figura 16: Adaptador CON-USBBEE (ROGERCOM, 2012)

Após instalar o *driver* do adaptador, sempre que este for ligado a uma porta USB é criada uma porta virtual COMx. O *software* X-CTU pode então acessar a porta serial correspondente àquela conectada ao módulo. A Figura 17 mostra a aba de seleção de portas do X-CTU. Para testar se a comunicação com o módulo está funcionando corretamente, basta clicar no botão "*Test / Query*". Se a comunicação estiver correta, uma mensagem com o número de série do *modem* deve aparecer. A gravação de *firmware* e a configuração dos parâmetros do módulo *XBee* são feitas na aba "*Modem Configuration*", conforme mostra a Figura 18. Ao clicar no botão "*Read*", é possível acessar todos os parâmetros de configuração do módulo. O *firmware* pode ser escolhido no menu "*Function Set*" e para gravar as alterações deve-se clicar no botão "*Write*".

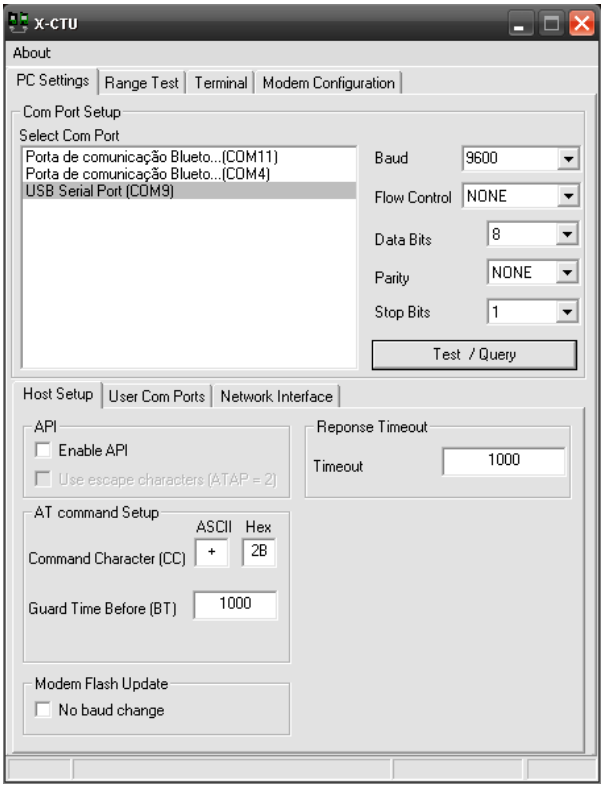

Figura 17: Seleção de portas no software X-CTU

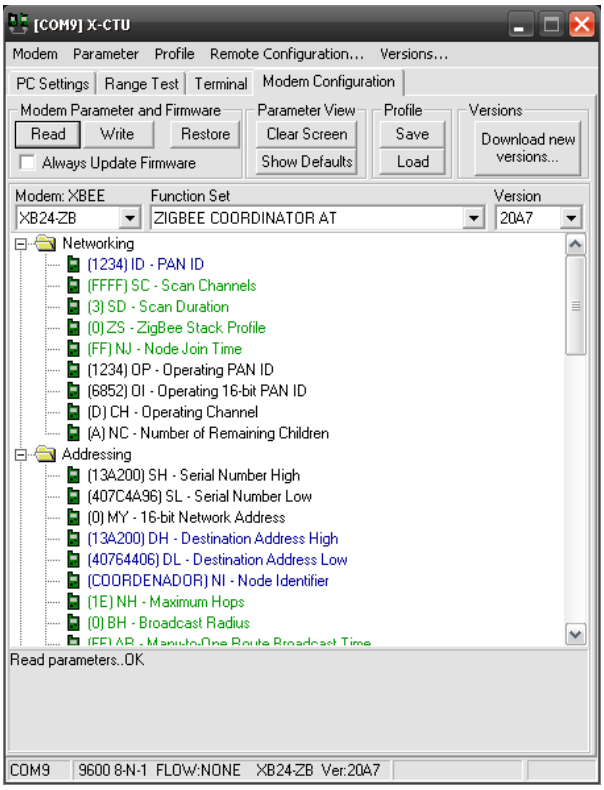

Figura 18: Configuração do módulo XBee no software X-CTU

Os módulos utilizados neste projeto foram configurados de acordo com a Tabela 6. O PAN ID identifica a rede à qual os módulos pertencem. Os parâmetros DH (*destination high*) e DL (*destination low*) indicam o endereço de destino dos dados enviados serialmente ao módulo correspondente. No módulo coordenador, DH e DL correspondem ao endereço de 64 bits (número de série) do módulo dispositivo final e vice-versa. Por fim, o parâmetro NI (*node identifier*) é a *string* de identificação do *modem*.

Tabela 6: Parâmetros de configuração dos módulos XBee

	Módulo 1 (Coordenador)	**Módulo 2 (Dispositivo Final)**
Firmware	ZIGBEE COORDINATOR AT	ZIGBEE END DEVICE AT
PAN ID	1234	1234
DH	13A200	13A200
DL	40794406	407C4A96
NI	COORDENADOR	CARRO

Com os módulos configurados desta maneira, eles irão operar no modo transparente, ou seja, todos os caracteres enviados para o coordenador são encaminhados para o dispositivo final e em seguida para o seu pino DOUT, sendo acessíveis para um circuito externo. Neste projeto, o formato do caractere adotado é composto de um *Start Bit*, 8 bits de dados e um *Stop Bit*. Optou-se pela não utilização de um bit de paridade porque eventuais erros de transmissão não são críticos para o envio de comandos para o carro.

Para auxiliar no diagnóstico da rede utilizou-se um botão conectado ao pino 20 e um LED (*light emitting diode*) indicador conectado ao pino 15, conforme a Figura 19.

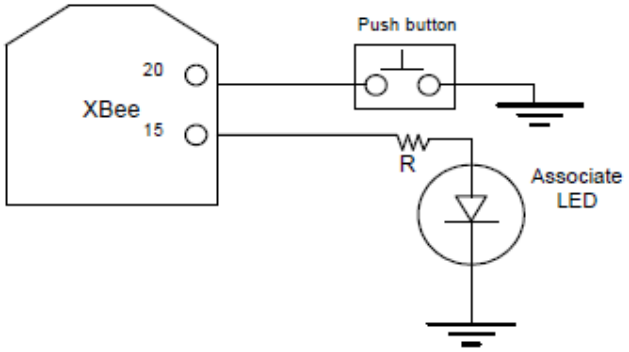

Figura 19: Botão de comissionamento e LED associado (DIGI INTERNATIONAL INC., 2008)

Algumas portas do módulo XBee possuem várias funções multiplexadas. Estas funções podem ser selecionadas a partir da configuração dos parâmetros Dn, onde n é um número de 0 a 7, correspondendo a cada *bit* da porta. Com o comando D5 = 1 (*bit* 5), o LED indicará o *status* do módulo e com o comando D0 = 1 (*bit* 0) o botão realiza ações diferentes dependendo de quantas vezes ele for acionado. A Tabela 7 mostra as ações possíveis dependendo do número de vezes que o botão é pressionado e se o módulo se encontra ou não em uma PAN.

Tabela 7: Funcionamento do botão de comissionamento (DIGI INTERNATIONAL INC., 2008)

Nº de vezes que o botão é pressionado	**Se o módulo já se encontra em uma PAN**	**Se o módulo não se encontra em uma PAN**
1	• Acorda um dispositivo final por 30 s • Manda uma transmissão broadcast de identificação de nó	• Acorda um dispositivo final por 30 s • Envia um código numérico ao pino associado indicando o que causou a falha de entrada na rede
2	• Manda uma mensagem broadcast para que o coordenador habilite a entrada de dispositivos na rede durante 1 minuto	• N/A
4	• Faz com que o dispositivo deixe a PAN • Restaura os valores dos parâmetros para o padrão • O dispositivo tenta entrar em uma rede baseando-se nos valores dos parâmetros ID e SC	• Restaura os valores dos parâmetros para o padrão • O dispositivo tenta entrar em uma rede baseando-se nos valores dos parâmetros ID e SC

O LED associado é ligado no pino associado, que indica o *status* de rede do dispositivo. Se o módulo não se encontra em uma rede, o estado do pino associado é alto. Quando o módulo entra em uma PAN, o LED associado pisca num intervalo de tempo regular, com período definido pelo comando LT. Se LT = 0, o dispositivo usa o tempo padrão (500ms para coordenador e 250ms para roteadores e dispositivos finais). O LED associado ainda pode indicar códigos de erro. O número de vezes que o LED piscará é dado pelo valor do parâmetro AI subtraído de 0x20. Quando há transmissão *broadcast* através do

acionamento do botão de comissionamento, o dispositivo que receber a mensagem piscará seu LED associado rapidamente durante 1 segundo.

3.2. Interface com o usuário

Para controlar o carro, necessitou-se de uma interface com o usuário pela qual fosse possível a leitura de caracteres enviados pelo teclado e o acesso à porta serial onde o módulo *XBee* coordenador estivesse conectado. Muitos programas com essa finalidade se encontram disponíveis, mas optou-se pelo desenvolvimento de uma interface própria para este projeto. Existem várias bibliotecas de interface serial para as mais diversas linguagens de programação e ambientes, como por exemplo, C++, *Python*, *Processing* (baseado em *Java*) e MATLAB. Todas as bibliotecas pesquisadas apresentam classes que tornam o uso das portas seriais bastante simples e intuitivo. Neste projeto, foi escolhida a linguagem *Python*, por ser mais simples que C++ e não depender de um ambiente, como o *Processing* e o MATLAB.

A interface criada é executada em modo texto pelo *prompt* de comando e possui dois ambientes: o de controle do carro e o ambiente de comandos. No ambiente de controle do carro, o usuário pode movimentá-lo utilizando as teclas WASD do teclado do computador. Para iniciar a interface, deve-se executar o arquivo AT.py com o comando `python AT.py`. Para que o programa execute corretamente, é preciso que o módulo coordenador já esteja conectado à porta USB. A Figura 20 apresenta a tela inicial do programa.

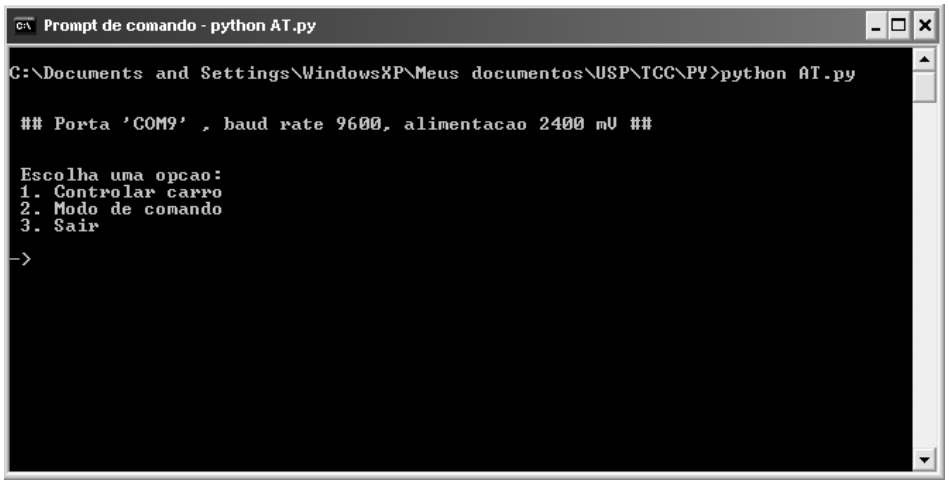

Figura 20: Tela principal do Terminal

No modo de controle do carro, o programa entra em *loop* esperando que alguma tecla seja pressionada. O *loop* é cessado quando a tecla “x” é ativada, e o programa volta para a tela inicial. Já no modo de comandos, o programa também entra em *loop*, esperando por um comando digitado pelo usuário. Após entrar com um comando e pressionar a tecla “ENTER”, a resposta ao comando é mostrada na tela. Para retornar à tela principal, deve-se entrar com o comando “sair”. A Figura 21 mostra o modo de comandos, onde o usuário digitou o comando NI (*Node Identifier*) e obteve a resposta do módulo coordenador.

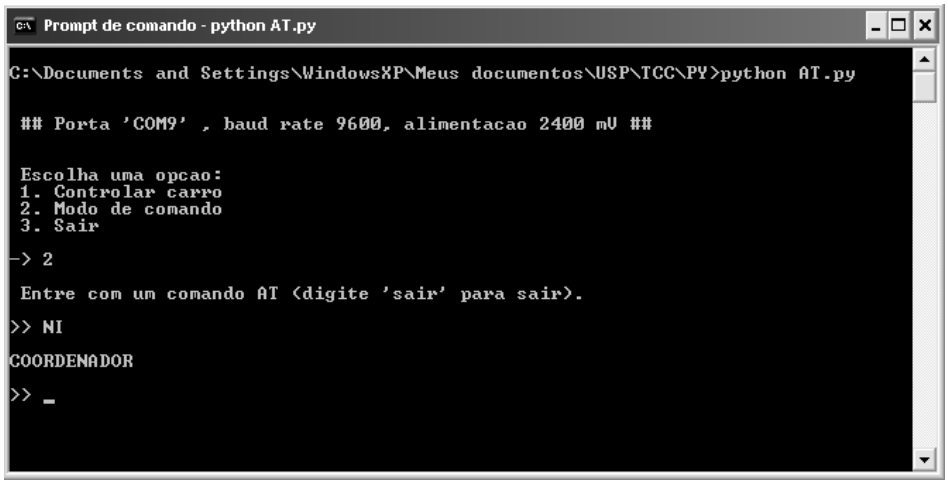

Figura 21: Terminal no modo de comandos

Para executar a interface, deve-se primeiramente instalar a versão 2.7.3 do *Python* (PYTHON SOFTWARE FOUNDATION, 2012) e a biblioteca *PySerial* (LIECHTI, 2010), que contém as funções de comunicação com portas seriais. Os arquivos para a instalação podem ser baixados diretamente das páginas oficiais do *Python* e da biblioteca *PySerial*. O código fonte do programa encontra-se no Apêndice I.

3.3. Microcontrolador

Assim que os dados são recebidos pelo módulo *XBee* dispositivo final, eles devem ser interpretados para a geração dos sinais que controlam os motores do carro. Uma solução largamente utilizada nestes casos é o emprego de microcontroladores, que apresentam grande versatilidade, tendo em vista que o problema é tratado por um *software*. Existem inúmeras opções de microcontroladores disponíveis no mercado, porém neste caso a escolha foi baseada na solução mais simples que atendesse os seguintes requisitos:

- Baixo consumo;

- Possuir interface serial para receber os dados;
- Possuir um número de portas o suficiente para a geração dos sinais de controle;
- Fácil de ser encontrado no mercado;
- Possuir boa documentação.

Dentre os microcontroladores pesquisados, os que atenderam aos critérios exigidos para o projeto foram o PIC16F628A, da *Microchip* e o MSP430F2002, fabricado pela *Texas Instruments* (a família *Freescale* foi descartada). A Tabela 8 mostra uma comparação entre as características relevantes destes dois dispositivos, de acordo com os dados obtidos dos seus respectivos *datasheets*.

Tabela 8: Comparação entre os microcontroladores PIC16F628A (MICROCHIP TECHNOLOGY INC., 2007) e MSP430F2002 (TEXAS INSTRUMENTS INC., 2011).

	PIC16F628A	**MSP430F2002**
Tensão de alimentação	2,0 ~ 5,5V	1,8 ~ 3,6V
Corrente em operação	120µA @ 1MHz, 2V	220µA @ 1MHz, 2,2V
Corrente em stand-by	0,1 µA @ 2 V	0,5 µA
Arquitetura	RISC 8-bit	RISC 16-bit
Interface serial	USART (SCI)	SPI ou I²C
Número de pinos	18	14
Número de portas I/O	16	10
Memória de programa (flash)	2KB	1KB + 256B
RAM	224B	128B
EEPROM	128B	-
Preço (Farnell Newark)	R$10,62	R$5,14
Disponibilidade (Farnell Newark)	Alta	Baixa

Pela Tabela 8, nota-se que ambos os microcontroladores possuem vantagens e desvantagens, porém neste projeto foi utilizado o PIC16F628A, levando em conta o consumo, a disponibilidade e principalmente a possibilidade de configurar a interface serial para uma comunicação assíncrona, o que não é possível no MSP430F2002. Além disso, o conhecimento que já se tinha deste microcontrolador foi outro fator determinante para a sua escolha. A pinagem do PIC utilizado é ilustrada na Figura 22.

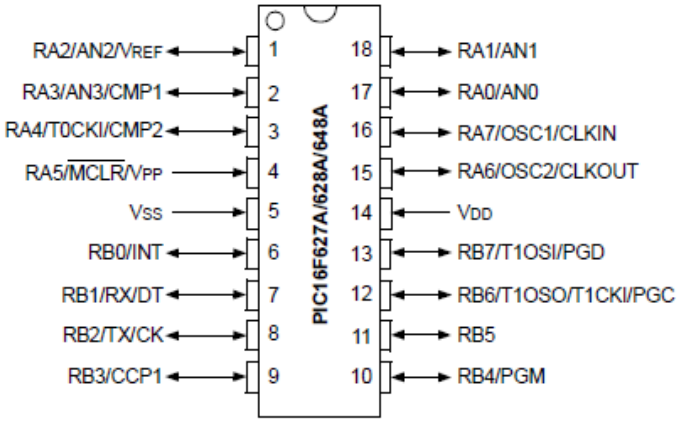

Figura 22: Pinagem do PIC16F628A (MICROCHIP TECHNOLOGY INC., 2007)

Os dados oriundos do módulo *XBee* são enviados ao PIC pelo pino 7 (RX) e são gerados 5 sinais de saída, segundo a Tabela 9.

Tabela 9: Sinais de controle gerados pelo PIC

Nome do sinal	Pino	Nome do pino
bitFrente	9	RB3
bitTras	13	RB7
bitDireita	12	RB6
bitCentro	11	RB5
bitEsquerda	10	RB4

Todos os pinos utilizados como sinais de controle pertencem à porta de 8 bits PORTB do microcontrolador. O PIC possui dois registradores, TRISA e TRISB, que possibilitam configurar a direção das suas respectivas portas, PORTA e PORTB. Esses registradores são de 8 bits e um bit com valor ‘0’ significa que o bit na posição correspondente da sua respectiva porta é uma saída. Caso contrário, este bit é interpretado como uma entrada. Desta forma, deve-se primeiramente configurar o registrador TRISB para que os bits na Tabela 9 sejam interpretados como saída. Observando a Figura 22, nota-se que os pinos RB1 e RB2, pertencentes à PORTB, são multiplexados com os pinos de recepção e transmissão serial, RX e TX. Para que as portas seriais do microcontrolador funcionem corretamente, deve-se definir a direção destes pinos como entrada, ou seja, as posições equivalentes no registrador TRISB devem ser colocadas em nível ‘1’. A Figura 23 representa como o registrador TRISB deve ser configurado.

PORTB							
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

TRISB							
0	0	0	0	0	1	1	X

X = Irrelevante

Figura 23: Configuração do registrador TRISB

Como a porta RB0 não foi utilizada, o valor do bit menos significativo do TRISB é irrelevante, ou seja, pode ser ‘0’ ou ‘1’. Desta forma, ao registrador foi atribuído o valor hexadecimal 0x06, cujo correspondente em binário é 00000110.

Além das saídas, é preciso configurar a comunicação serial. Para isso, são empregados três registradores: TXSTA (*Transmit Status and Control Register*), RCSTA (*Receive Status and Control Register*) e SPBRG (*Baud Rate Generator*). A Figura 24 representa o registrador TXSTA e como configurá-lo.

TXSTA							
CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D
Configuração							
X	0	1	0	X	1	X	X

X = Irrelevante

CSRC: Origem do clock (somente síncrono)
TX9: Habilita transmissão com 9 bits
TXEN: Habilita transmissão
SYNC: Habilita modo síncrono
BRGH: Modo de alta taxa de transmissão
TRMT: Indicador do status do registrador de deslocamento
TX9D: Valor do 9º bit (se TX9=1)

Figura 24: Configuração do registrador TXSTA

O registrador TXSTA foi configurado para que a transmissão utilize palavras de 8 bits (TX9=0), modo assíncrono (SYNC=0) e com a possibilidade de alta taxas de transmissão (BRGH=1). Assim, adotando o valor ‘0’ para os bits irrelevantes, foi atribuído o valor 00100100 ao registrador, que em hexadecimal é 0x24. A configuração para o registrador RCSTA é feita conforme a Figura 25.

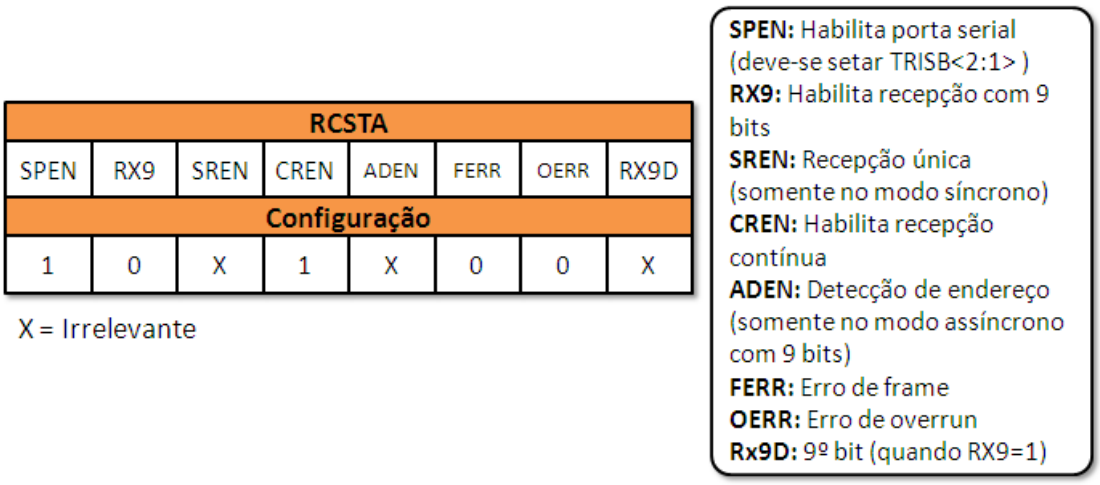

Figura 25: Configuração do registrador RCSTA

Como a recepção também utiliza palavras de 8 bits, atribuiu-se o valor '0' ao bit RX9. Os bits FERR e OERR recebem '0' pois erros de transmissão devem ser simplesmente ignorados. Assim, o registrador RCSTA recebe o valor hexadecimal 0x90, que corresponde ao binário 10010000. Para definir a taxa de transmissão (*baud rate*), deve-se atribuir um valor ao registrador SPBRG, segundo a equação 1, onde f_{osc} é a frequência de oscilação do cristal, BR é o valor da taxa de transmissão desejada e x é o valor que deve ser atribuído ao registrador.

$$BR = \frac{f_{osc}}{16(x+1)} \quad \textbf{(1)}$$

Efetuando os cálculos com BR = 9600 bps e f_{osc} = 6,144 MHz, obteve-se x = 39, ou em hexadecimal, x = 0x27. Decidiu-se usar um cristal externo para geral o *clock*, devido a possíveis interferências da temperatura no *clock* interno do microcontrolador. A Tabela 10 apresenta um resumo das configurações dos registradores utilizados neste projeto.

Tabela 10: Resumo das configurações dos registradores

Registrador	Valor (hexadecimal)
TRISB	0x06
TXSTA	0x24
RCSTA	0x90
SPBRG	0x27

Quando um dado é recebido pela porta serial, um bit indicador, RCIF (*Interrupt Flag*), é habilitado. O caractere recebido é armazenado no registrador RCREG, de onde pode ser

copiado para uma variável e ser interpretado. O *software* escrito em linguagem C funciona basicamente desta forma: enquanto nenhum dado é recebido, o programa fica em loop. Caso um caractere esteja presente (o que pode ser verificado pelo bit RCIF), o valor em RCREG é copiado para a variável `dado` e de acordo com o valor do caractere os sinais de saída são modificados, conforme as tabelas Tabela 11 e Tabela 12. Se qualquer outro caractere for recebido, os sinais `bitFrente` e `bitTras` são zerados, fazendo com que o carro pare. Pela Tabela 12, observa-se que os valores atribuídos aos sinais de saída dependem da posição anterior de servomotor. Se o servomotor está voltado para a direita, por exemplo, ao pressionar a tecla 'a', a próxima posição é o centro. Ao se pressionar a tecla 'a' mais uma vez, o servomotor vira para a esquerda. O código fonte do programa contido no microcontrolador, desenvolvido no ambiente *MikroC* (MIKROELEKTRONIKA, 2012), pode ser verificado no Apêndice II.

Tabela 11: Relação entre os caracteres e os sinais de saída de controle da tração

Caractere (hexa)	Caractere	bitFrente	bitTras
0x77	w	1	0
0x74	s	0	1

Tabela 12: Relação entre os caracteres e os sinais de saída de controle da direção

Caractere (hexa)	Caractere	Posição anterior do servomotor	bitEsquerda	bitCentro	bitDireita
0x61	a	Esquerda	1	0	0
		Centro	1	0	0
		Direita	0	1	0
0x64	s	Esquerda	0	1	0
		Centro	0	0	1
		Direita	0	0	1

3.4. Sistema de tração

Os sinais `bitFrente` e `bitTras` (Tabela 9) gerados pelo PIC, são utilizados para acionar o motor DC responsável pela tração do carro, por intermédio de um *driver* do tipo ponte H. Foi escolhido o *driver* L293D, que contém duas pontes H em um único dispositivo, o que facilita bastante a construção do *hardware*, além de reduzir a área da placa de circuito

impresso. O L293D é fabricado pela *Texas Instruments*, e sua pinagem é ilustrada na Figura 26 (TEXAS INSTRUMENTS INCORPORATED, 2002).

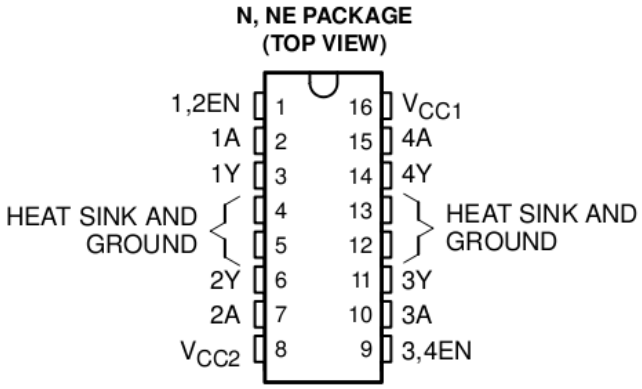

Figura 26: Pinagem do driver L293D (TEXAS INSTRUMENTS INCORPORATED, 2002)

O L293D possui dois pinos para alimentação. O pino V_{CC1} corresponde à alimentação lógica, já o pino V_{CC2} refere-se à alimentação do motor DC, que é conectado diretamente entre as saídas 1Y e 2Y. Os sinais de entrada oriundos do microcontrolador são conectados nos pinos 1A e 2A. Cada lado do CI corresponde a uma ponte H, que devem ser habilitadas pelos pinos 1,2EN ou 3,4EN. Os pinos centrais (4, 5, 13 e 12) são aterrados. A Tabela 13 mostra como os pinos foram conectados.

Tabela 13: Conexões do driver L293D. NC = Não Conectado

Pino	Nome	Conexão	Pino	Nome	Conexão
1	1,2EN	V_{CC} (5V)	**9**	3,4EN	Terra
2	1A	bitFrente	**10**	3A	NC
3	1Y	Motor DC	**11**	3Y	NC
4	GND	Terra	**12**	GND	Terra
5	GND	Terra	**13**	GND	Terra
6	2Y	Motor DC	**14**	4Y	NC
7	2A	bitTras	**15**	4A	NC
8	Vcc2	V_{CC} do motor (5V)	**16**	Vcc1	V_{CC} (5V)

Neste projeto foi utilizado o *Mini-Motor 71427*, fornecido pela empresa *Lego*. Este motor DC foi escolhido por ser bastante simples, pequeno, leve (42 gramas) e principalmente por possuir um *case* plástico com formato compatível com as peças de *Lego*, facilitando a montagem do carro. A Figura 27 ilustra o aspecto externo do motor.

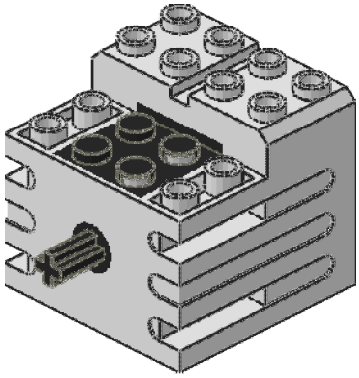

Figura 27: Mini-Motor Lego 71427 (PEERON, 2011)

A Tabela 14, obtida através das medidas realizadas pelo site *Philohome*, mostra as características do motor 71427 alimentado com 4,5V e com uma carga de 2,25 N.cm em seu eixo. Estes valores fornecem uma base para a aplicação deste projeto, onde o motor foi alimentado com 5V a fim de reduzir o consumo.

Tabela 14: Características do Mini-Motor Lego 71427 testado a 4,5 V e com carga de 2,25 N.cm (PHILOHOME, 2012).

Velocidade (rpm)	57
Corrente (A)	0,12
Potência mecânica (W)	0,13
Potência elétrica (W)	0,54
Eficiência	24%

Os terminais do motor são acessados pelos encaixes presentes na porção preta da face superior do motor. Esses pinos possuem pequenos contatos metálicos em suas laterais, permitindo o acesso elétrico através de um cabo especial, fornecido com o *kit* (Figura 28). Para conectar o motor aos pinos do L293D foi necessário modificar uma das pontas do cabo, encaixando os fios em um conector *modu* de 2 vias. A Figura 29 representa um diagrama em blocos do sistema de tração, desde a geração dos sinais de controle pelo microcontrolador até a conexão do motor DC nos terminais da ponte H do circuito integrado L293D.

Figura 28: Cabo de alimentação do motor 71427 (rebrickable.com)

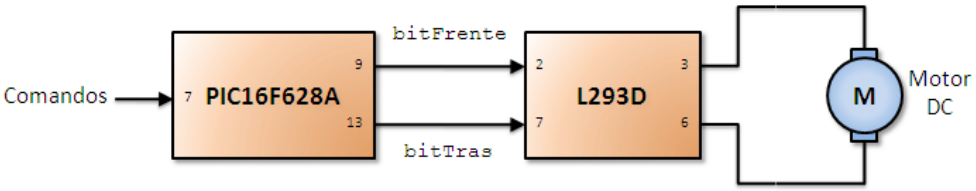

Figura 29: Diagrama em blocos do sistema de tração

3.5. Sistema de direção

O sistema de direção consiste no controle do ângulo de curvatura das rodas dianteiras do carro de acordo com os valores dos sinais `bitDireita`, `bitEsquerda` e `bitCentro` (Tabela 9). O motor responsável por este controle deve ter alto torque, suportando o peso do carro, o posicionamento deve ser preciso e de rápida resposta. Os servomotores atendem a estas características, com a vantagem de serem facilmente encontrados no mercado e alguns possuírem tamanho e peso bastante reduzidos.

O servomotor escolhido para este projeto foi o *Hextronik HXT900* (Figura 30), que já estava disponível no laboratório e apresenta as características necessárias ao projeto. Suas principais características são apresentadas na Tabela 15, obtida do site *ServoDatabase.com*.

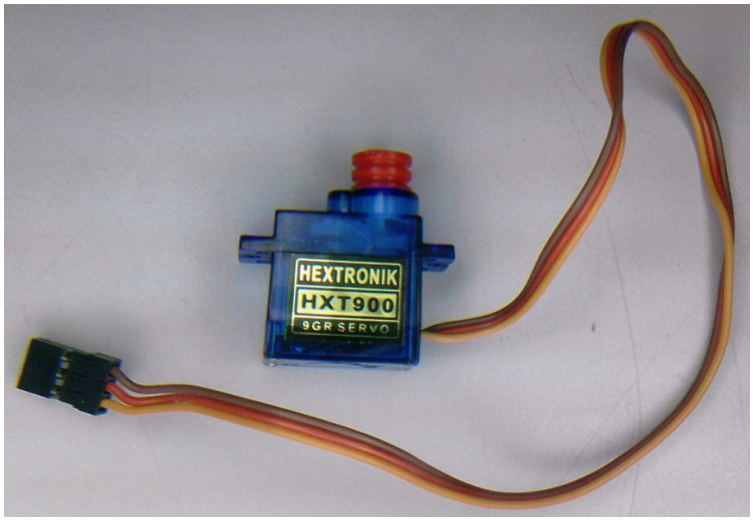

Figura 30: Servomotor Hextronik HXT900

Tabela 15: Características do servomotor Hextronik HXT900 (SERVO DATABASE, 2012)

Torque	1,60 kg.cm @ 4,8V
Velocidade	60º/0,12s @ 4,8V
Peso	9,1g
Dimensões (c x l x a)	21 x 12 x 22 mm
Alcance rotacional	±90º
Preço médio	US$ 3,65

Para acionar o servomotor, é necessário um sinal modulado por largura de pulsos (PWM). Como a posição do servomotor é controlada pela largura do pulso em nível alto, houve a necessidade de desenvolver um sistema que recebesse os três sinais provenientes do PIC e gerasse o PWM correspondente a cada sinal. A primeira ideia foi utilizar o CI LM555, com componentes externos sendo chaveados por um *driver* ULN2003. Este método se mostrou ineficiente por dois motivos: 1) a abordagem puramente analógica e com dependência dos valores dos componentes eletrônicos fez com que o sinal de PWM gerado apresentasse ruídos e valores imprecisos, causando tremulações nas rodas dianteiras; 2) o uso do *driver* ULN2003 acarretou em um consumo alto, inviabilizando o uso de baterias. O PWM então foi gerado por um CPLD via codificação em VHDL (*VHSIC Hardware Description Language*), a fim de se obter um sinal preciso, formando os ângulos de 0º, -45º e +45º, correspondendo às larguras de pulso em nível alto de 1.5, 1.25 e 1.75ms, respectivamente. Estas três posições se mostraram suficientes para o controle de direção

do carro. Foi utilizado o CPLD *EPM3064ALC44*, fabricado pela *Altera*. A Tabela 16 mostra algumas de suas características, extraídas do *datasheet*.

Tabela 16: Características do CPLD Altera EPM3064ALC44 (ALTERA CORPORATION, 2002)

Altera EPM3064A PLCC 44 Pinos	
Macrocélulas	64
Nº máximo de portas I/O	34
Tensão de alimentação	3,0 ~ 3,6V
Tensão de entrada	-0,5 ~ 5,75V
Tensão de saída	0 ~ Tensão de alimentação (V)
Corrente de alimentação	Aprox. 25mA @ 10MHz
Preço (Farnell Newark)	R$ 12,33

O objetivo do *software* escrito em VHDL é gerar um sinal PWM com a largura de pulso variável de acordo com os valores de entrada. A Tabela 17 ilustra como deve ser o sinal de saída para as três combinações possíveis dos sinais vindos do microcontrolador.

Tabela 17: Entradas e saída do gerador de PWM

Entradas (níveis lógicos)			**Saída (PWM)**	
bitEsquerda	bitCentro	bitDireita	Período	Largura do pulso (nível alto)
1	0	0	20 ms	1,25 ms
0	1	0	20 ms	1,50 ms
0	0	1	20 ms	1,75 ms

Além dos sinais de controle, foi necessária uma entrada de *clock* (gerado por um circuito oscilador de 10 MHz). A partir deste sinal periódico foram realizadas contagens de pulsos para gerar o sinal de PWM. Assim, a entidade descrita pelo *software* possui 4 entradas (`clock`, `dir`, `cen` e `esq`) do tipo bit e somente uma saída (`pwm`), também do tipo bit. Primeiramente, o sinal de clock passa por um divisor de frequência, de onde é obtido um sinal de 4 kHz. Isto foi feito para que os valores das contagens não sejam muito altos, já que se deseja valores da ordem de milissegundos. O divisor de frequência foi construído com as *Megafunctions* disponibilizadas pelo ambiente de programação *Quartus II* (ALTERA CORPORATION, 2012), que são entidades pré-definidas comumente utilizadas. Nesta entidade, é possível definir o módulo e o número de *flip-flops* do contador. O módulo foi obtido simplesmente dividindo-se a frequência do *clock* de entrada pela frequência do sinal

desejado, ou seja, 10 MHz dividido por 4 kHz. Assim, obteve-se que o contador deve ser módulo 2500.

Com o sinal de 4 kHz, gerou-se o PWM por meio de contagens de subidas de borda. A cada subida, são transcorridos 250 µs, então o período de 20 ms é obtido após 80 contagens. Os pulsos de 1,25 ms, 1,5 ms e 1.75 ms são gerados com contagens até 5, 6 e 7, respectivamente. Os códigos em VHDL da entidade principal e do divisor de frequência estão disponíveis no Apêndice III. A Figura 31 representa o diagrama em blocos do sistema de direção, com as pinagens de entrada e saída.

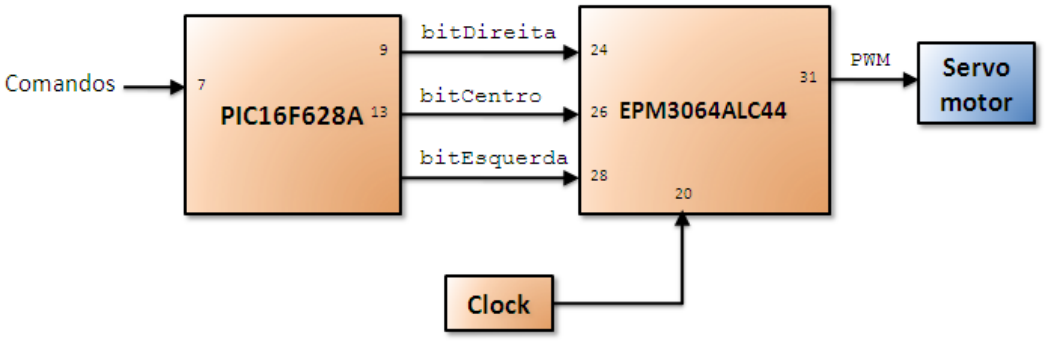

Figura 31: Diagrama em blocos do sistema de direção

3.6. Construção do *hardware*

Esta seção trata da elaboração da placa que integra todos os sistemas apresentados nas seções anteriores. Todo o *hardware* foi projetado para operar com duas baterias de 9V. Como foram utilizados dispositivos que requerem diferentes tensões de alimentação, tornou-se necessário empregar reguladores de tensão de 5V e de 3,3V. O circuito pode ser aberto ou fechado com uma chave, e foram utilizados LEDs indicadores para cada uma das baterias. A Figura 32 representa o esquemático do estágio de alimentação do circuito, com 3 sinais de alimentação gerados: Vcc1 (5V), utilizado para alimentar os componentes da placa e o servomotor; Vcc2 (5V), para alimentar o motor DC e Vcc3 (3,3V), proveniente de Vcc1, usado para alimentar o módulo *XBee*.

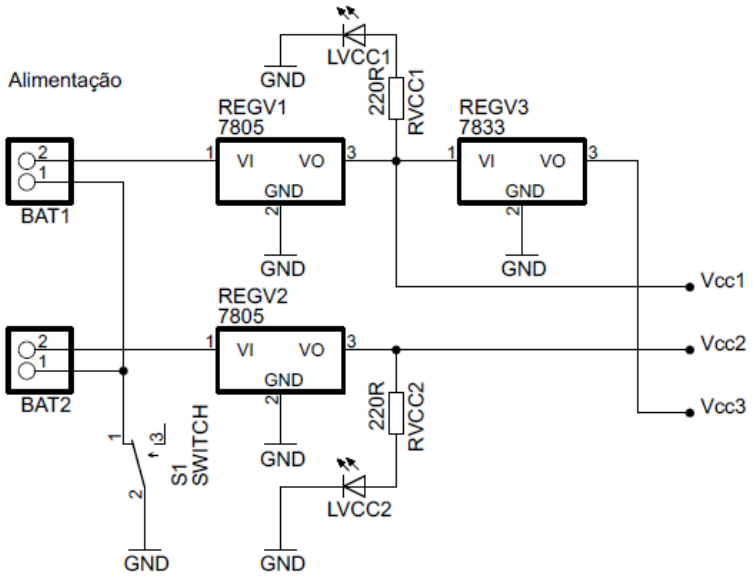

Figura 32: Esquemático do estágio de alimentação

A Figura 33 mostra o esquemático referente ao módulo *XBee*. Um LED foi conectado ao pino 15 com a função de LED Associado, para mostrar o *status* do modem. Uma chave táctil foi ligada ao pino 20, com a função de botão de comissionamento. Além disso, um LED indicativo foi conectado ao pino de saída de dados, piscando rapidamente sempre que um dado for transmitido com sucesso a partir do módulo coordenador.

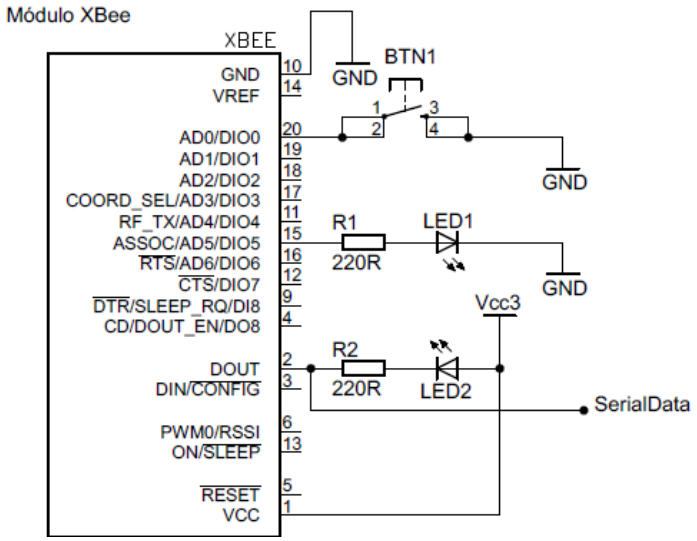

Figura 33: Esquemático das conexões do módulo XBee

O circuito relativo ao microcontrolador PIC está representado na Figura 34. Pode-se observar que uma chave táctil foi ligada ao pino de *reset* e o circuito oscilador conectado entre os pinos OSC1 e OSC2 foi montado segundo as especificações do *datasheet*. Além

disso, um capacitor de 100nF foi inserido entre a alimentação e o terra, com o objetivo de filtrar eventuais ruídos.

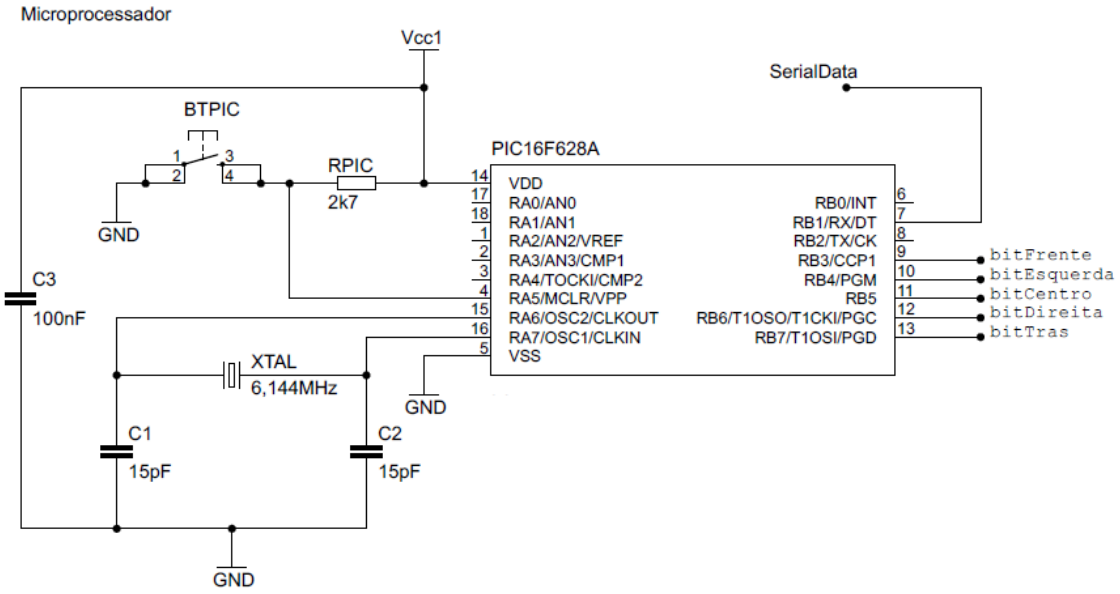

Figura 34: Esquemático das conexões do PIC

O esquemático da Figura 35 mostra como foram feitas as ligações do *driver* L293D, segundo a Tabela 13. Já a Figura 36 mostra as conexões no CPLD *EPM3064A*. Além dos sinais de I/O e alimentação, é necessário um sinal de *clock*, gerado por um oscilador DIP (*Dual In-line Package*) de 10MHz.

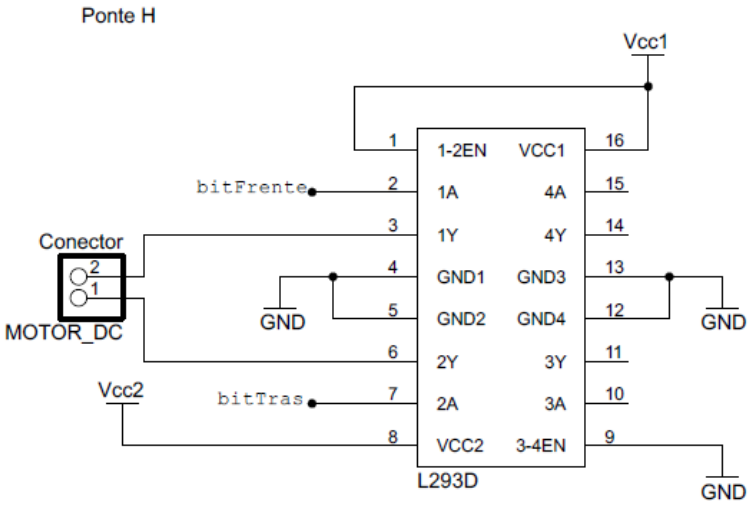

Figura 35: Esquemático das conexões da ponte H

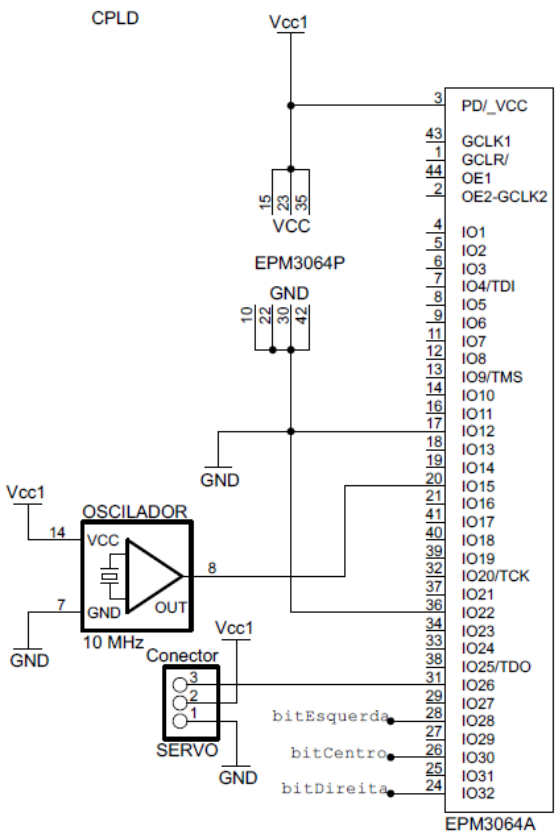

Figura 36: Esquemático das conexões do CPLD

Utilizando o *software EAGLE* (CADSOFT COMPUTER, 2011), foi criado o *layout* da placa de circuito impresso correspondente aos esquemáticos apresentados. As figuras 37 e 38 ilustram as camadas superior e inferior da placa e a Figura 39 apresenta uma foto da placa concluída.

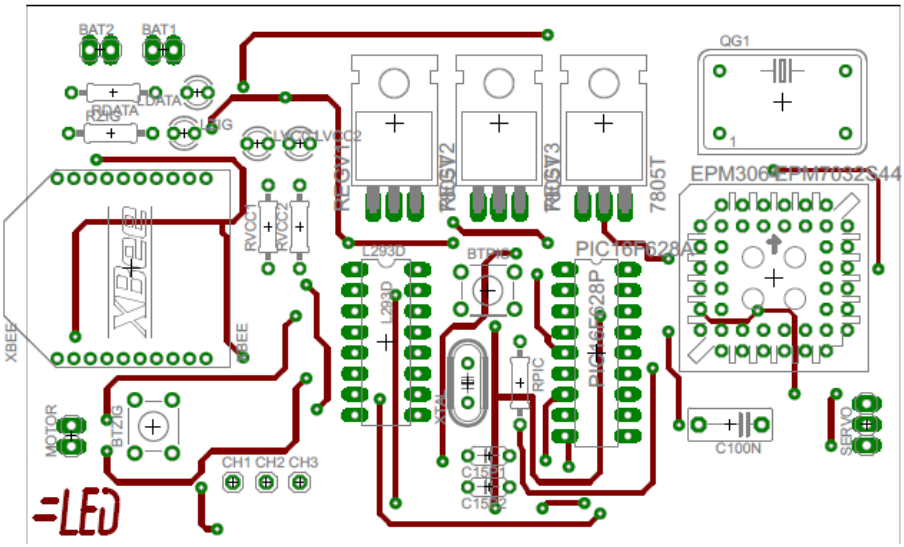

Figura 37: Layout da camada superior da placa

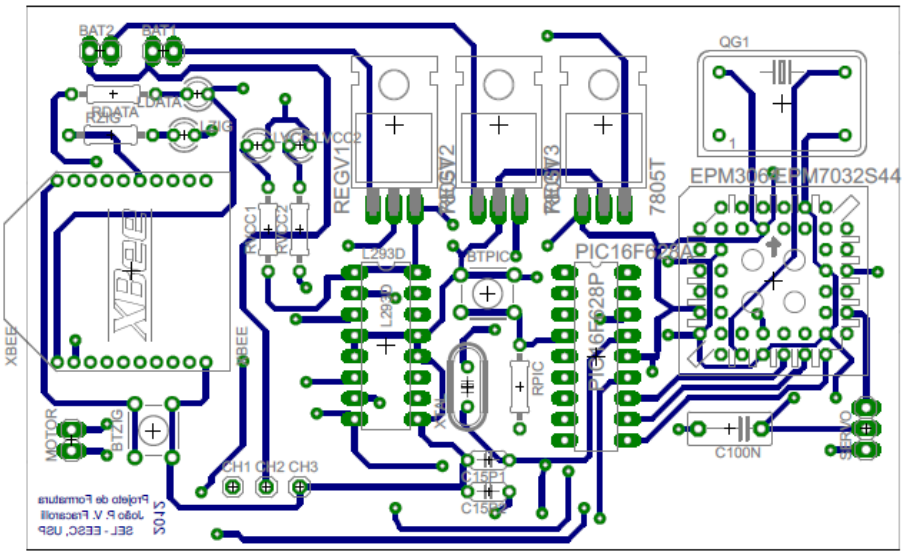

Figura 38: Layout da camada inferior da placa

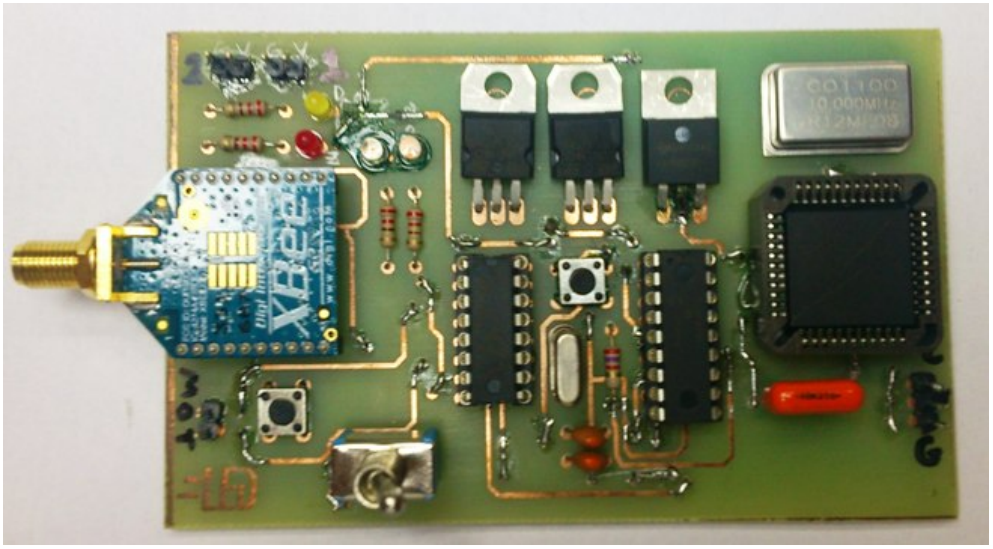

Figura 39: Placa de circuito impresso finalizada

A estrutura do carro foi montada utilizando peças de *Lego*, o que garantiu uma grande flexibilidade e a capacidade de ajustar o formato do carro para que a placa de circuito impresso fosse encaixada de forma conveniente no topo do veículo. As baterias foram inseridas em duas cavidades montadas abaixo da placa, e foram conectadas ao

hardware com o uso de cabos de dois terminais com conectores *modu*. O Apêndice IV apresenta a lista de materiais utilizados neste projeto.

4. Resultados

Nesta seção, são apresentados os resultados obtidos após a concepção do projeto, cuja metodologia de construção foi descrita no capítulo 3. As figuras 40 e 41 mostram o carro finalizado.

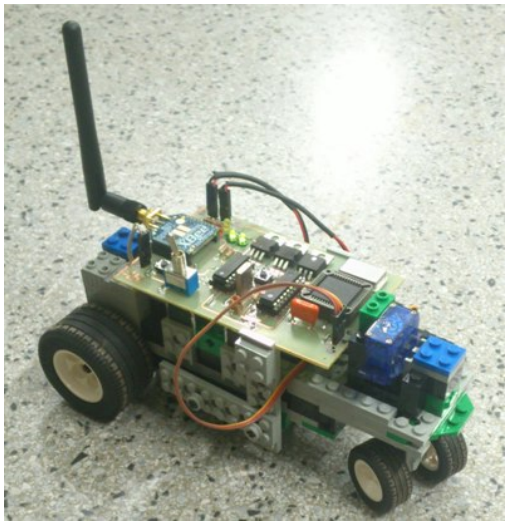

Figura 40: Carro finalizado (a)

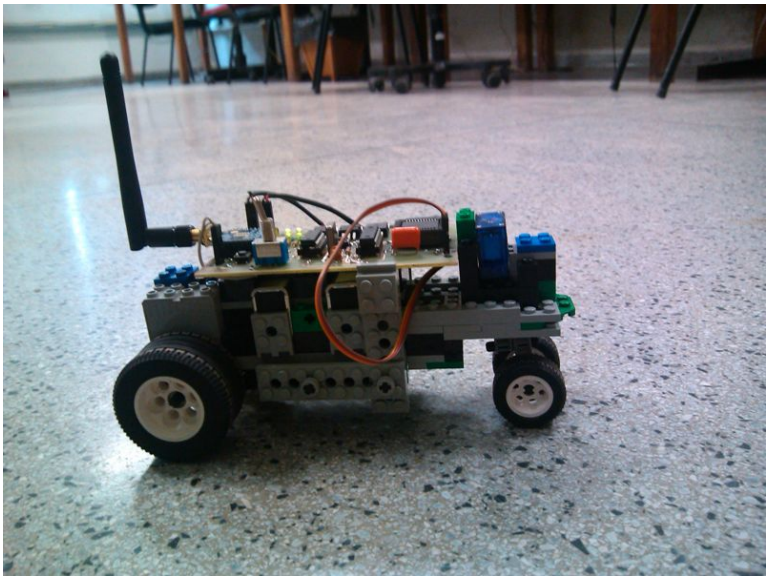

Figura 41: Carro finalizado (b)

Primeiramente, observou-se o funcionamento do carro, enviando comandos a partir do PC e verificando a resposta dos motores. A Tabela 18 mostra as ações correspondentes a cada comando que pode ser enviado para o carro.

Tabela 18: Resumo dos comandos do carro

Comando	Ação do carro
W	Avançar
A	Virar para esquerda
S	Recuar
D	Virar para direita
Soltar botão	Parar

O carro pôde ser controlado livremente e todos os comandos obtiveram respostas imediatas. Após esta observação inicial, foram realizados os testes de alcance e consumo, que são grandezas de interesse para este projeto. Por fim, são discutidas algumas aplicações didáticas que podem ser extraídas deste trabalho.

4.1. Teste de alcance

A fim de verificar o alcance dos módulos *XBee*, foi realizado um teste no campo de futebol do *campus* da Universidade. A Figura 42 ilustra o campo de futebol com as suas dimensões e os pontos em que o alcance foi testado. Este teste foi realizado com as baterias completamente carregadas.

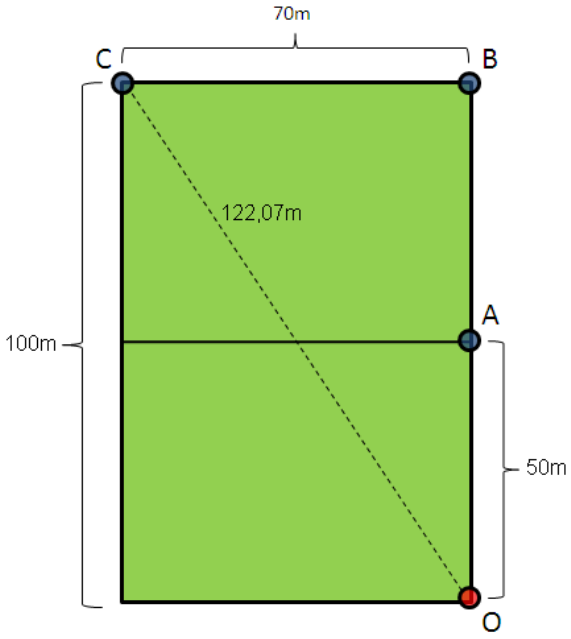

Figura 42: Teste de alcance do carro

O computador com o módulo coordenador foi fixado no ponto O, e o carro foi posicionado em três pontos diferentes, A, B e C. Os comandos foram enviados para o módulo dispositivo final e foi observada a resposta do sistema para cada ponto de teste. A Tabela 19 resume os resultados obtidos.

Tabela 19: Resultados do teste de alcance no campo de futebol

Ponto testado	**Distância de O (m)**	**Houve resposta correta aos comandos?**
A	50	Sim
B	100	Sim
C	122,07	Sim

É possível perceber que o sistema responde para todas as distâncias testadas, o que é condizente com o *datasheet* do módulo *XBee ZNet 2.5* (alcance de cerca de 120 m), conforme pode ser verificado na Tabela 5 do capítulo 3. Para testar o alcance máximo, o módulo coordenador foi posicionado no estacionamento do prédio da engenharia elétrica e o carro foi afastado em direção aos prédios do instituto de química até os comandos pararem de obter resposta. A Figura 43, retirada do *Google Maps*, mostra uma foto de satélite da região do teste, assinalando os pontos de teste e a distância máxima atingida.

Figura 43: Teste de alcance máximo

Na Figura 43, o ponto O representa onde foi fixado o módulo coordenador, e o ponto F representa o local onde o carro parou de funcionar. A distância foi medida com base na escala fornecida pelo *Google Maps*, obtendo-se 123,5 m. Observando a maior distância testada no campo de futebol (122,07 m, Tabela 19), verifica-se que este valor é bem próximo ao máximo encontrado (123,5 m). A medida obtida no segundo teste também é coerente com os dados apresentados na Tabela 5 (120 m).

4.2. Teste de consumo

Com as duas baterias completamente carregadas, o carro funciona normalmente, como o esperado. Como a bateria dedicada à alimentação geral do circuito também alimenta o servomotor, quando este é acionado há momentaneamente um aumento na corrente consumida pelo circuito. Com o descarregamento desta bateria, atinge-se um ponto em que ao ser acionado o servomotor, a bateria não consegue suprir a alimentação dos outros componentes, que são desligados rapidamente. Isso faz com que o microcontrolador sofra um *reset*, levando as rodas dianteiras à posição central novamente. Observa-se então

que nesta situação, ao enviar um comando de posicionamento das rodas dianteiras, estas imediatamente retornam ao centro.

Foi desenvolvido um programa em *Python (Apêndice V)* para testar o tempo de autonomia do carro, acionando-se o servomotor a cada 0,5s. Este programa incrementa um contador a cada 1s e mostra o seu valor na tela do computador, sendo fácil verificar o tempo decorrido até que o carro pare de responder normalmente. Os resultados obtidos estão dispostos na Tabela 20.

Tabela 20: Resultados do teste de autonomia do carro

	Início	**Fim**
Tensão da bateria	9,05 V	8,16 V
Horário	11h35min	11h45min
Valor do contador	0	607

Pelas medidas realizadas, percebe-se que o tempo de autonomia do carro é de cerca de 10 minutos, com o servomotor sendo acionado aproximadamente 1200 vezes. Neste intervalo de tempo, a tensão fornecida pela bateria caiu de 9,05 V para 8,16 V, ou seja, houve uma queda de 9,8%. Uma possível solução para esse problema é o acréscimo de uma terceira bateria, dedicada exclusivamente à alimentação do servomotor.

Para a segunda parte do teste, foi utilizada a fonte de tensão de bancada para alimentar o circuito do carro, ligado em série com um amperímetro. O problema deste método foi observar um valor médio para a corrente medida, uma vez que o valor oscilava a partir do primeiro dígito após a vírgula. Decidiu-se então retirar o amperímetro e, utilizando uma resistência de 8,2 ohms ligada em série com o circuito, mediu-se a tensão sobre este resistor no osciloscópio. Com o auxílio das funções de medidas de média e os cursores do aparelho, o valor pôde ser melhor observado. A Tabela 21 apresenta os valores medidos por este método, com o carro em repouso.

Tabela 21: Medidas realizadas no teste com a fonte de bancada

Tensão medida da fonte, V_F	9,09 V
Tensão medida no osciloscópio, V_R	0,76 V
Resistência em série, R	8,2 Ω
Corrente calculada, $I_R = V_R/R$	92,68 mA

Com os valores obtidos na Tabela 21, é possível calcular a resistência equivalente do circuito do carro, a partir da equação 2.

$$R_C = \frac{V_F}{I_R} \quad \textbf{(2)}$$

Assim, foi obtido R_C = 98,08 Ω. O *datasheet* da bateria *Energizer* utilizada apresenta gráficos da carga *versus* tempo de operação para algumas aplicações comuns do produto, inclusive para brinquedos, que é a categoria em que este projeto se enquadra. O gráfico da Figura 44, retirado da folha de dados, apresenta a curva para um brinquedo de 270 Ω. Nota-se que neste caso, o fornecimento da bateria cai de 9V para 8V em cerca de três horas.

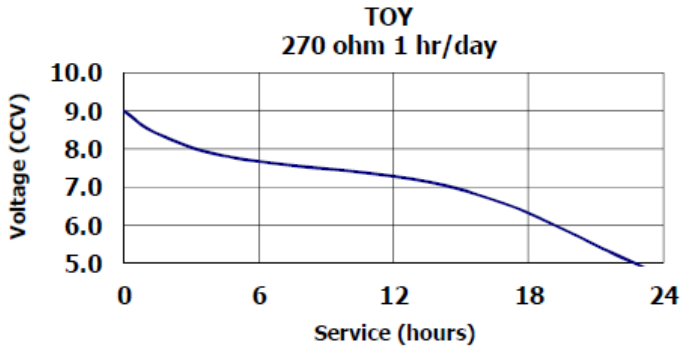

Figura 44: Gráfico de tensão fornecida versus tempo de operação para um brinquedo de 270 ohms (ENERGIZER HOLDINGS INC., 2012).

Assumindo que para qualquer outro dispositivo a curva de descarga *versus* tempo tem o aspecto da Figura 44 e que o consumo depende proporcionalmente da resistência, é possível estimar o tempo de autonomia do carro na situação de repouso. Observa-se que 98,08 Ω equivalem a 36,3% de 270 Ω. Logo, o tempo de autonomia será 36,3% de 3 horas, que é aproximadamente 65 minutos.

A Tabela 22 mostra as correntes exigidas da bateria 2, que se dedica ao motor DC, nas possíveis situações em que o carro se encontra. Estes valores foram medidos em bancada com o amperímetro digital. Neste caso foi possível medir a corrente diretamente, pois o valor não sofreu oscilações e como o motor DC permanece em funcionamento ao ser acionado, é mais fácil medir a corrente com este em funcionamento.

Tabela 22: Correntes exigidas da bateria 2 de acordo com a situação do sistema.

Situação	**Bateria 2 (V_{CC2})**
Repouso	23 mA
Acionando motor DC	33 mA

Pela tabela 22, é possível perceber que quando o sistema está em repouso, há um consumo de 23 mA. As rodas traseiras não são desligadas para garantir que o carro permanecerá parado mesmo em um aclive ou declive.

4.3. Aplicações didáticas

Elementos deste projeto podem ser utilizados com fins didáticos nas disciplinas de laboratório que envolvem eletrônica digital e microprocessadores. Um projeto de microcontroladores, por exemplo, pode envolver tanto o acionamento de um motor DC, como o controle direto do posicionamento de um servomotor. Microcontroladores como o PIC16F628A possuem uma função para gerar um PWM internamente. No presente trabalho, esta função foi substituída pelo CPLD para que os conhecimentos adquiridos sobre dispositivos programáveis pudessem ser aplicados, tornando o projeto mais interdisciplinar.

Na disciplina de Laboratório de Sistemas Digitais II, onde os alunos desenvolvem projetos usando a linguagem VHDL, é possível utilizar o próprio programa desenvolvido neste projeto (o acionamento de um servomotor) como uma prática inicial, por ser algo simples de ser implementado e que envolve ferramentas importantes como as *Megafunctions*, além de ser uma forma de introduzir blocos sequenciais, com um exemplo simples do uso da sintaxe *if else* da linguagem. Pode-se também aproveitar os conceitos aqui apresentados sobre o protocolo *ZigBee* e unir o uso de CPLDs com comunicação sem fio, desenvolvendo, por exemplo, um projeto que consiste na programação da comunicação entre dois CPLDs usando *modems XBee*. Assim, o aluno teria que desenvolver um código que gerasse a informação serial a ser transmitida de um dispositivo para o outro e o receptor interpretaria estes sinais, transmitindo-os a um terminal ou a um *display*. Enfim, este trabalho pode gerar o interesse sobre os assuntos aqui abordados, tais como protocolos de comunicação sem fio, aplicação de microprocessadores, dispositivos lógicos programáveis e linguagens de programação como *Python*.

5. Conclusões

Foi possível observar que o sistema funcionou conforme o projetado, como pode ser verificado na Tabela 18 do capítulo 4. Os comandos foram transmitidos de forma rápida e com o alcance dentro da faixa especificada pelo fabricante dos *modems XBee* (aproximadamente 120 metros). Verificou-se que mesmo a uma distância maior que 100 metros (Tabela 19) a resposta aos comandos foi em imediata. Não foram observadas falhas de transmissão, de interpretação dos dados e geração dos sinais de controle. Os motores foram acionados corretamente, permitindo que o carro fosse controlado com total liberdade. Conforme o desenvolvimento do projeto optou-se sempre pela utilização de circuitos integrados ou dispositivos programáveis, como o microcontrolador e o CPLD, garantindo uma maior flexibilidade ao sistema, que pode ser modificado mais facilmente. Desta forma, o *hardware* construído mostrou-se bastante simples e de fácil compreensão, podendo ser usado para fins didáticos, além de ser uma aplicação interessante de eletrônica e sistemas digitais.

Observou-se um alto consumo proporcionado pelos motores e reguladores que tornaram o uso de baterias convencionais ineficiente, o que acarretou na necessidade de melhorias quanto à alimentação. Um dos principais problemas encontrados foi o tempo de autonomia do carro, que passa a responder inadequadamente quando a tensão da bateria dedicada ao servomotor e à alimentação dos componentes cai até cerca de 90% de seu valor com carga total. Apesar deste problema, foi possível verificar o funcionamento de cada subsistema do projeto. Seguem abaixo algumas sugestões para otimizar o sistema em trabalhos futuros.

- Aprimorar o sistema de alimentação, estudando outras formas de alimentar o sistema, como por exemplo o uso de baterias de alta densidade de carga, como a *Swing 4400*, desenvolvida pela *Boston Power* (BOSTON-POWER INC., 2010).
- Analisar o uso de outros tipos de componentes, visando otimizar o projeto. Por exemplo: eliminar a necessidade de utilizar o CPLD com a geração do PWM em um microprocessador; substituir os reguladores de tensão utilizados por outros que consumam menos, como o LP2954, da *National Semiconductor* (NATIONAL SEMICONDUCTOR CORPORATION, 2005)*;* usar o driver DRV8836, da *Texas Instruments* (TEXAS INSTRUMENTS INCORPORATED, 2012) *ao invés do L293D.*
- Adicionar sensores ao carro, de modo que ele possa medir a distância entre obstáculos e evitá-los;
- Desenvolver uma interface gráfica para o programa desenvolvido em Python;

- Procurar soluções alternativas às utilizadas (microcontrolador, linguagens de programação, tipos de motores diferentes);
- Desenvolver uma interface que independa da plataforma utilizada no computador (Windows, Linux, etc.);
- Procurar adaptar o projeto para o uso de FPGAs (*Field Programmable Gate Arrays*)

Referências

AKYILDIZ, I. F.; WANG, X.; WANG, W. **Wireless mesh networks: a survey**. Computer Networks, 1 Janeiro 2005. 445-487.

ALTERA CORPORATION. **MAX 3000A Programmable Logic Device Family**. [S.l.]. 2002.

ALTERA CORPORATION. **Quartus II Web Edition Software**. Altera.com, 2012. Disponivel em: <http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html>. Acesso em: 30 Novembro 2012.

BOSTON-POWER INC. **Swing 4400 Rechargeable Lithium-Ion Cell**. [S.l.]. 2010.

CADSOFT COMPUTER. **Cadsoft EAGLE PCB Design Software - EAGLE Shop, Support, Tutorials**. Cadsoft USA, 2011. Disponivel em: <http://www.cadsoftusa.com/>. Acesso em: 24 outubro 2012.

DIGI INTERNATIONAL INC. **XBee ZNet 2.5 / XBee-PRO ZNet 2.5 OEM RF Modules**. [S.l.]. 2008.

ELECTROONS. **Servo Motor Control**. Electroons.com, 2012. Disponivel em: <http://www.electroons.com/electroons/servo_control.html>. Acesso em: 27 Junho 2012.

ENERGIZER HOLDINGS INC. **Energizer 522 - Product Datasheet**. [S.l.], p. 2. 2012.

GONZAGA, D. P.; JULIANI, A. D. P. **Notas de aula em Máquinas Elétricas de Corrente Contínua**. São Carlos: [s.n.], 2008.

KRUTZ, R. L. **Interfacing Techniques in Digital Design With Emphasis On Microprocessors**. 1ª Edição. ed. New York: John Wiley & Sons, Inc, v. I, 1988.

LIECHTI, C. **Welcome to pySerial's documentation**. pySerial, 2010. Disponivel em: <http://pyserial.sourceforge.net/>. Acesso em: 08 Agosto 2012.

MATTOS, G. M. **Redes de Acesso em Banda Larga utilizando Sistemas VSAT e WiFi**. Pontifícia Universidade Católica. Rio de Janeiro, p. 64-65. 2006.

MICROCHIP TECHNOLOGY INC. **PIC16F627A/628A/648A Data Sheet**. [S.l.]. 2007.

MIKROELEKTRONIKA. **mikroC PRO for PIC - C compiler for Microchip PIC microcontrollers**. Mikroe.com, 2012. Disponivel em: <http://www.mikroe.com/mikroc/pic/>. Acesso em: 30 Novembro 2012.

MONTEIRO, M. A. **Introdução à Organização de Computadores**. 5ª Edição. ed. Rio de Janeiro: LTC, v. 1, 2007.

NATIONAL SEMICONDUCTOR CORPORATION. **LP2954/LP2954A 5V and Adjustable Micropower Low-Dropout Voltage Regulators**. [S.l.]. 2005.

PATSKO, L. F. **Tutorial Montagem da Ponte H**. Maxwell Bohr Instrumentação Eletrônica. [S.l.]. 2006.

PEERON. **Sets that have 'Electric Technic Mini-Motor 9v' (71427c01)**. Peeron.com, 2011. Disponivel em: <http://www.peeron.com/inv/parts/71427c01>. Acesso em: 19 Setembro 2012.

PHILOHOME. **LEGO® 9V Technic Motors compared characteristics**. Philo's Home Page, 2012. Disponivel em: <http://www.philohome.com/motors/motorcomp.htm>. Acesso em: 19 Setembro 2012.

PYTHON SOFTWARE FOUNDATION. **Python Programming Language - Official Website**. Python Programming Language - Official Website, 2012. Disponivel em: <http://www.python.org/>. Acesso em: 3 Outubro 2012.

ROGERCOM. **Manual do Adaptador CON-USBBEE**. [S.l.]. 2012.

SANTOS, A. **Servomotores**. www.sumoderobos.org. Porto Alegre. 2007.

SEATTLE ROBOTICS SOCIETY. **Whats a servo: A quick tutorial**. seattlerobotics.org, 2004. Disponivel em: <http://www.seattlerobotics.org/guide/servos.html>. Acesso em: 01 Dezembro 2012.

SENA BLOG. **The comparison of Wi-Fi, Bluetooth and ZigBee**. Sena Blog, 2010. Disponivel em: <http://www.sena.com/blog/?p=359>. Acesso em: 20 Setembro 2012.

SERVO DATABASE. **Hextronik HXT900 Servo Specifications and Reviews**. ServoDatabase.com, 2012. Disponivel em: <http://www.servodatabase.com/servo/hextronik/hxt900>. Acesso em: 8 Outubro 2012.

TEXAS INSTRUMENTS INC. **MSP430F20x3, MSP430F20x2, MSP430F20x1 Mixed Signal Microcontroller**. [S.l.]. 2011.

TEXAS INSTRUMENTS INCORPORATED. **L293, L293D Quadruple Half-H Drivers**. [S.l.]. 2002.

TEXAS INSTRUMENTS INCORPORATED. **DRV8836. Dual Low Voltage H-Bridge IC**. [S.l.]. 2012.

WIKIPEDIA. **Comparison of synchronous and asynchronous signalling**. Wikipedia, the Free Encyclopedia, 2012. Disponivel em: <http://en.wikipedia.org/wiki/Comparison_of_synchronous_and_asynchronous_signalling>. Acesso em: 2 Julho 2012.

ZIGBEE ALLIANCE. **ZigBee Technology**. ZigBee Alliance, 2012. Disponivel em: <www.zigbee.org>. Acesso em: 16 Julho 2012.

Apêndices

Apêndice I – Código fonte do programa AT.py

```
# Interface para o controle e envio de comandos para o carro
# Autor: Joao Paulo V. Fracarolli
# TCC - Engenharia Eletrica - Eletrônica
# SEL, EESC - USP  08/08/12

import serial    #classe serial do PySerial
import time      #classe com rotinas de delay
import msvcrt    #obtencao de caracteres diretamente. Funciona no Windows

# Constantes

modo_comando = '+++'
nome_porta = 'COM9'
baud_rate = 9600
timeout = 1 #1s de timeout

# Funcoes

def entrarModoDeComando(porta_serial):
        porta_serial.write(modo_comando)
        time.sleep(1.5) #espera 1.5 s
        porta_serial.readline()


def enviarComando(porta_serial,comando):
        entrarModoDeComando(porta_serial)
        msg = 'AT' + comando + '\r'
        porta_serial.write(msg)
        time.sleep(timeout)
        return porta_serial.readline()[0:-1]


def tensaoDeAlimentacao(porta_serial):
        tensao = int(enviarComando(porta_serial,'%V'),16)
        return (tensao/1023)*1200


# Conectando a porta

xbee = serial.Serial(nome_porta,baud_rate,8,'N',1,0,1,1,1) #8 bits, 1 stop bit, sem paridade

key = 'nope'
opt = 'nope'
cmd = 'nope'

tensao = tensaoDeAlimentacao(xbee)
```

```
print " \n\n ## Porta %r , baud rate %r, alimentacao %r mV ##\n" %(xbee.name, xbee.baudrate,
tensao)

while opt != '3':
        print "\n Escolha uma opcao: \n 1. Controlar carro\n 2. Modo de comando\n 3. Sair\n"
        opt = raw_input("-> ")
        if opt == '1':
               key = 'nope'
               print "Use as teclas WASD para controlar o carro. Pressione \'x\' para sair."
                      while key != 'x':
                             if msvcrt.kbhit():
                                    key = msvcrt.getch()
                                    xbee.write(key)
                                    time.sleep(0.05)
                                    xbee.write(0)

        elif opt == '2':
                print "\n Entre com um comando AT (digite \'sair\' para sair).\n"
                while cmd != 'sair':
                      cmd = raw_input(">> ")
                      if cmd != 'sair':
                              print "\n",enviarComando(xbee,cmd),"\n"
        elif opt == '3':
                print "Saindo.."
        else:
                print "Opcao invalida!"


xbee.close()
```

Apêndice II – Código fonte do programa PIC_carro.c

```
/*    ###Controle do carro de Lego ###      *
 ------------------------------------------
 * O objetivo deste programa é gerar os     *
 * sinais de controle dos motores do        *
 * carro de  Lego, de acordo com os valores *
 * dos dados recebidos via interface serial *
 * pelo módulo XBee                         *
 ------------------------------------------
 * Autor: João Paulo V. Fracarolli          *
 * Projeto de Formatura do curso de         *
 *  Engenharia Elétrica - Ênfase em         *
 *  Eletrônica, SEL, EESC - USP             *
 ------------------------------------------*/

//Definicoes

#define bitFrente RB3_bit
#define bitTras RB7_bit
#define bitEsquerda RB4_bit
#define bitCentro RB5_bit
#define bitDireita RB6_bit

void frente();
void tras();
void esquerda();
void direita();
void para();

int direcao = 0;

void main(){

    unsigned int dado = 0;

    //Configuracoes
    TRISB = 0x06;
    TXSTA = 0x24;
    RCSTA = 0x90;
    SPBRG = 0x27;

    //Condicao inicial
    para();
    bitDireita = 0;
    bitEsquerda = 0;
    bitCentro = 1;

    while(1){
        while(!RCIF_bit);
        dado = RCREG;
```

```
        // Verificacao do dado recem chegado
        switch(dado){
            case 0x77:{frente(); break;}
            case 0x61:{esquerda(); break;}
            case 0x73:{tras(); break;}
            case 0x64:{direita(); break;}
            default: para();
        }
    }

}

void frente(){
    bitFrente = 1;
    bitTras = 0;
    return;
}

void tras(){
    bitFrente = 0;
    bitTras = 1;
    return;
}

void esquerda(){
    if(direcao==0){
        bitEsquerda = 1;
        bitCentro = 0;
        bitDireita = 0;
        direcao-=1;
    }
    else if(direcao>0){
        bitEsquerda = 0;
        bitCentro = 1;
        bitDireita = 0;
        direcao-=1;
    }
    return;
}

void direita(){
    if(direcao==0){
        bitEsquerda = 0;
        bitCentro = 0;
        bitDireita = 1;
        direcao+=1;
    }
    else if(direcao<0){
        bitEsquerda = 0;
        bitCentro = 1;
        bitDireita = 0;
```

```
        direcao+=1;
    }
    return;
}

void para(){
    bitFrente = 0;
    bitTras = 0;
    return;
}
```

Apêndice III – Código em VHDL do gerador de PWM

Entidade principal:

```
-- Gerador de PWM para controle de um servomotor
-- Autor: João Paulo V. Fracarolli

entity PWM_Generator is
	port
	(
		-- Entradas
		clock : in  bit;
		dir   : in  bit := '0';
		cen   : in  bit := '1';
		esq   : in  bit := '0';
		-- Saídas
		Pwm   : out bit
	);
end PWM_Generator;

architecture func of PWM_Generator is

		-- Este divisor gera a frequencia de 4kHz a partir do clock
		--  e a saída é o q(11)
		component divisor is
		port(
			clock : in  bit ;
			q     : out bit_vector (11 downto 0)
		);
		end component;

		signal q        : bit_vector(11 downto 0);
		-- o cont vai contar ate o periodo de 20ms (20ms/250us = 80)
		signal cont     : integer range 0 to 80 := 0;
		-- o ton e o tempo em que o PWM fica em nivel alto
		signal ton      : integer range 0 to 7  := 0;
		-- sinal que eh montado com a concatenacao dos estados de entrada
		signal posicao  : bit_vector(2 downto 0);

begin

	div: divisor port map(clock,q);

	posicao <= esq & cen & dir;      -- concatena os 3 bits

	ton <= 5 when posicao="100" else -- esquerda 1.25ms (1.25ms/250us = 5)
		6 when posicao="010" else -- centro 1.5ms (1.5ms/250us = 6)
		7;                        -- direita 1.75ms (1.75ms/250us = 7)


	process(q(11),ton) -- q(11) eh a saida de 4kHz
```

```
	begin

		if(q(11)'event and q(11) = '1') then
			if(cont < 80) then
				cont <= cont+1; --conta ate 20ms e rebobina
			else
				cont <= 0;
			end if;
		end if;

	end process;

	pwm <= '1' when cont<ton else
		'0';

end func;
```

Megafunction do contador:

```
-- megafunction wizard: %LPM_COUNTER%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: LPM_COUNTER

-- ============================================================
-- File Name: divisor.vhd
-- Megafunction Name(s):
--			LPM_COUNTER
--
-- Simulation Library Files(s):
--			lpm
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 11.0 Build 157 04/27/2011 SJ Web Edition
-- ************************************************************


--Copyright (C) 1991-2011 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
```

```
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.


LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;

LIBRARY lpm;
USE lpm.all;

ENTITY divisor IS
	PORT
	(
		clock		: IN BIT ;
		q		: OUT BIT_VECTOR (11 DOWNTO 0)
	);
END divisor;


ARCHITECTURE SYN OF divisor IS

	SIGNAL sub_wire0	: BIT_VECTOR (11 DOWNTO 0);



	COMPONENT lpm_counter
	GENERIC (
		lpm_direction		: STRING;
		lpm_modulus		: NATURAL;
		lpm_port_updown		: STRING;
		lpm_type		: STRING;
		lpm_width		: NATURAL
	);
	PORT (
			clock	: IN BIT ;
			q	: OUT BIT_VECTOR (11 DOWNTO 0)
	);
	END COMPONENT;

BEGIN
	q    <= sub_wire0(11 DOWNTO 0);

	LPM_COUNTER_component : LPM_COUNTER
	GENERIC MAP (
		lpm_direction => "UP",
		lpm_modulus => 2500,
		lpm_port_updown => "PORT_UNUSED",
		lpm_type => "LPM_COUNTER",
		lpm_width => 12
```

```
	)
	PORT MAP (
		clock => clock,
		q => sub_wire0
	);



END SYN;

-- ============================================================
-- CNX file retrieval info
-- ============================================================
-- Retrieval info: PRIVATE: ACLR NUMERIC "0"
-- Retrieval info: PRIVATE: ALOAD NUMERIC "0"
-- Retrieval info: PRIVATE: ASET NUMERIC "0"
-- Retrieval info: PRIVATE: ASET_ALL1 NUMERIC "1"
-- Retrieval info: PRIVATE: CLK_EN NUMERIC "0"
-- Retrieval info: PRIVATE: CNT_EN NUMERIC "0"
-- Retrieval info: PRIVATE: CarryIn NUMERIC "0"
-- Retrieval info: PRIVATE: CarryOut NUMERIC "0"
-- Retrieval info: PRIVATE: Direction NUMERIC "0"
-- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "MAX7000S"
-- Retrieval info: PRIVATE: ModulusCounter NUMERIC "1"
-- Retrieval info: PRIVATE: ModulusValue NUMERIC "2500"
-- Retrieval info: PRIVATE: SCLR NUMERIC "0"
-- Retrieval info: PRIVATE: SLOAD NUMERIC "0"
-- Retrieval info: PRIVATE: SSET NUMERIC "0"
-- Retrieval info: PRIVATE: SSET_ALL1 NUMERIC "1"
-- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
-- Retrieval info: PRIVATE: nBit NUMERIC "12"
-- Retrieval info: PRIVATE: new_diagram STRING "1"
-- Retrieval info: LIBRARY: lpm lpm.lpm_components.all
-- Retrieval info: CONSTANT: LPM_DIRECTION STRING "UP"
-- Retrieval info: CONSTANT: LPM_MODULUS NUMERIC "2500"
-- Retrieval info: CONSTANT: LPM_PORT_UPDOWN STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_COUNTER"
-- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "12"
-- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL "clock"
-- Retrieval info: USED_PORT: q 0 0 12 0 OUTPUT NODEFVAL "q[11..0]"
-- Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
-- Retrieval info: CONNECT: q 0 0 12 0 @q 0 0 12 0
-- Retrieval info: GEN_FILE: TYPE_NORMAL divisor.vhd TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL divisor.inc FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL divisor.cmp TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL divisor.bsf FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL divisor_inst.vhd TRUE
-- Retrieval info: LIB_FILE: lpm
```

Apêndice IV – Lista de materiais utilizados no projeto

Lista de materiais	
Quantidade	**Material**
2	Modem XBee ZNet 2.5
1	Placa CON-USBBEE
1	Cabo extensor USB (macho-fêmea)
2	Antena Tenda Q2407
1	Servomotor Hextronik HXT900
1	Mini Motor DC Lego 71427
1	CI L293D (DIP)
1	Microcontrolador PIC16F628A (DIP)
1	CPLD Altera EPM3064ALC44-10N
1	Soquete DIP 18 pinos
1	Soquete DIP 16 pinos
1	Soquete PLCC 44 pinos
2	Soquete de 10 vias, espaçamento de 2 mm para XBee
2	Chave táctil 4 terminais
1	Chave alavanca 3 terminais
4	Resistor 220 ohms
1	Resistor 2700 ohms
1	LED amarelo 3 mm
1	LED vermelho 3 mm
2	LED verde 3 mm de alto brilho
2	Regulador de tensão L7805CV
1	Regulador de tensão UA78M33C
1	Oscilador C01100 10MHz
1	Cristal abc xtal 6,144MHz
1	Capacitor 100nF
2	Capacitor 15pF
-	Peças de Lego

Apêndice V – Código fonte do programa de teste de autonomia do carro

```
import serial    #classe serial do PySerial
import time      #classe com rotinas de delay

# Constantes

modo_comando = '+++'
nome_porta = 'COM9'
baud_rate = 9600
timeout = 1 #1s de timeout


# Conectando a porta

xbee = serial.Serial(nome_porta,baud_rate,8,'N',1,0,1,1,1) #8 bits, 1 stop bit, sem paridade

print "\n\n Enduro!\n\n"

opt = 'no'
cont_s = 0

while opt!='x':
    time.sleep(0.5)
    xbee.write('a')
    time.sleep(0.5)
    xbee.write('d')

    cont_s = cont_s + 1
    print "\n",cont_s

    time.sleep(0.5)
    xbee.write('d')
    time.sleep(0.5)
    xbee.write('a')

    cont_s = cont_s + 1
    print "\n",cont_s
```

Anexos

Anexo I – Comandos AT

Special

Table 7-01. Special Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
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	--	--
WB	**Write Binding Table:** Writes the current binding table to non-volative memory.	CRE	--	--
RE	**Restore Defaults**. Restore module parameters to factory defaults. RE command does not reset the ID parameter.	CRE	--	--
FR	**Software Reset**. Reset module. Responds immediately with an "OK" then performs a reset ~2 seconds later. Use of the FR command will cause a network layer restart on the node if SC or ID were modified since the last reset.	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	--

Node types that support the command: C = Coordinator, R = Router, E = End Device

Figura 45: Comandos AT – Especiais (DIGI INTERNATIONAL INC., 2008)

Addressing

Table 7-02. Addressing Commands)

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
DH[2]	**Destination Address High**. Set/Get the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the destination address used for transmission. 0x000000000000FFFF is the broadcast address for the PAN. DH is not supported in API Mode. 0x0000000000000000 is the Coordinator's 16-bit network address.	CRE	0 - 0xFFFFFFFF	0
DL[2]	**Destination Address Low**. Set/Get the lower 32 bits of the 64-bit destination address. When combined with DH, DL defines the destination address used for transmission. 0x000000000000FFFF is the broadcast address for the PAN. DL is not supported in API Mode. 0x0000000000000000 is the Coordinator's 16-bit network address.	CRE	0 - 0xFFFFFFFF	0xFFFF(Coordinator) 0 (Router/End Device)
MY	**16-bit Network Address.** Get the 16-bit network address of the module.	CRE	0 - 0xFFFE [read-only]	0xFFFE
MP	**16-bit Parent Network Address.** Get the 16-bit parent network address of the module.	E	0 - 0xFFFE [read-only]	0xFFFE
NC	**Number of Children**. Read the number of end device children that have joined to the device. This command returns the number of child table entries currently in use.	CR	0 - 8	read-only
SH	**Serial Number High**. Read high 32 bits of the RF module's unique IEEE 64-bit address. 64-bit source address is always enabled.	CRE	0 - 0xFFFFFFFF [read-only]	factory-set
SL	**Serial Number Low**. Read low 32 bits of the RF module's unique IEEE 64-bit address. 64-bit source address is always enabled.	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.	CRE	20-Byte printable ASCII string	ASCII space character (0x20)
DD	**Device Type Identifier.** Stores a device type value. This value can be used to differentiate multiple XBee-based products.	CRE	0 - 0xFFFFFFFF [read-only]	0x20000
ZA[2]	**ZigBee Application Layer Addressing**. Set/read the Zigbee application layer addressing enabled attribute. If enabled, data packets will use the SE, DE, and CI commands to address Zigbee application layer source and destination endpoints, and the cluster ID fields in all data transmissions. ZA is only supported in the AT firmware.	CRE	0 - 1	0
SE[2]	**Source Endpoint**. Set/read the ZigBee application layer source endpoint value. If ZigBee application layer addressing is enabled (ZA command), 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	1 - 0xEF	0xE8

Figura 46: Comandos AT – Endereçamento (DIGI INTERNATIONAL INC., 2008)

Table 7-02. Addressing Commands)

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
DE[2]	**Destination Endpoint**. Set/read Zigbee application layer destination ID value. If ZigBee application layer addressing is enabled (ZA command), 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 - 0xEF	1 - 0xEF
CI[2]	**Cluster Identifier**. Set/read Zigbee application layer cluster ID value. If ZigBee application layer addressing is enabled (ZA command), 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 - 0xFF	0x11
BI[2]	**Binding Table Index**. Set/read the binding table index value. If this value is set to a valid binding table index, the addressing information at that index in the binding table will be used for all data transmissions. BI is only supported in AT firmware	CRE	0 - 0xFF	0xFF

1. Node types that support the command: C=Coordinator, R=Router, E=End Device
2. Command supported by modules using AT Command firmware only

Figura 47: Comandos AT - Endereçamento (continuação) (DIGI INTERNATIONAL INC., 2008)

Networking & Security

Table 7-03. Networking Commands

AT Command	Name and Description	Node Type[1]	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	0, 0x0B-0x1A *(XBee)* 0, 0x0C – 0x18 (XBee-PRO)	[read-only]
ID	**PAN ID**. Set/Get the PAN (Personal Area Network) ID. ***Coordinator*** - Set the preferred Pan ID. Set ID = 0xFFFF to auto-select. ***Router / End Device*** - Set the desired Pan ID. When the device searches for a Coordinator, it attempts to only join to a parent that has a matching Pan ID. Set ID = 0xFFFF to join a parent operating on any Pan ID. Changes to ID should be written to non-volatile memory using the WR command.	CRE	0 - 0x3FFF, 0xFFFF	0x0234 (291d)
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 - 0x20	0
OP	**Operating PAN ID**. Read the PAN (Personal Area Network) ID. The OP value reflects the operating PAN ID that the module is running on. If ID < 0xFFFF, OP will equal ID.	CRE	0 - 0x3FFF	[read-only]
NT	**Node Discover Timeout.** Set/Read the amount of time a node will spend discovering other nodes when ND or DN is issued.	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
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	--

Figura 48: Comandos AT – Rede (DIGI INTERNATIONAL INC., 2008)

Table 7-03. Networking Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
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	--
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. 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) Note: Setting SC to include more than 12 continuous channels could cause data to be received on incorrect frequencies due to crosstalk issues with the EM250 at certain power levels. See Appendix E for details. Changing SC may result in not being able to communicate with long-range '-PRO' modules from Digi	CRE	**XBee** 1 - 0xFFFF [bitfield] **XBee-PRO** 2 - 0x3FFE [bitfield] (bits 0, 14, 15 not allowed)	0x1FFE
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
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 – 0x40, 0xFF [x 1 sec]	0xFF (always allows joining)
JV	**Channel Verification**. Set/Read the channel verification parameter. If JV=1, and the network is an open network (NJ=0xFF), 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
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
AI	**Association Indication**. Read information regarding last node join request: 0x00 - Successful completion - Coordinator started or Router/End Device found and joined with a parent. 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) 0x27 - Node Joining attempt failed (typically due to incompatible security settings) 0x2A - Coordinator Start attempt failed' 0xFF - Scanning for a Parent 0x2B - Checking for an existing coordinator	CRE	0 - 0xFF [read-only]	--

Figura 49: Comandos AT - Rede (continuação) (DIGI INTERNATIONAL INC., 2008)

Security

Table 7-04. Security Commands

AT Command	Name and Description	Node Type[1]	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	CRE	0 - 0xFF	
KY	Encryption Key. Set the 128-bit AES encryption key. This command is read-only; KY cannot be read.	CRE	0 - 0xFFFFFFFFFFFFFFFF	0

Figura 50: Comandos AT – Segurança (DIGI INTERNATIONAL INC., 2008)

RF Interfacing

Table 7-05. RF Interfacing Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
PL	**Power Level**. Select/Read the power level at which the RF module transmits conducted power.	CRE	**XBee** (boost mode disabled) 0 = -8 dBm 1 = -4 dBm 2 = -2 dBm 3 = 0 dBm 4 = +2 dBm **XBee-PRO** 4 = 18 dBm **XBee-PRO (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 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.			

1. Node types that support the command: C = Coordinator, R = Router, E = End Device

Figura 51: Comandos AT - Interface RF (DIGI INTERNATIONAL INC., 2008)

Serial Interfacing (I/O)

Table 7-06. Serial Interfacing Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
AP[2]	**API Enable**. Enable API Mode. The AP parameter is only applicable when using modules that contain the following firmware versions:1.1xx (coordinator), 1.3xx (router/end device)	CRE	1 - 2 1 = API-enabled 2 = API-enabled (w/escaped control characters)	1
AO[2]	**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)	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 0x80 - 0x38400 (non-standard rates)	3

Figura 52: Comandos AT - Interface Serial (DIGI INTERNATIONAL INC., 2008)

Table 7-06. Serial Interfacing Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
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
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.	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	0

1. Node types that support the command: C = Coordinator, R = Router, E = End Device
2. Command supported by modules using API firmware only

Figura 53: Comandos AT - Interface Serial (continuação) (DIGI INTERNATIONAL INC., 2008)

I/O Commands

Table 7-07. I/O Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
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	-	-
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 - 0xFFFF (ms)	0
IC	**IO 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. Bit (IO pin): 0 (DIO0)4 (DIO4)8 (DIO8) 1 (DIO1) 5 (DIO5) 9 (DIO9) 2 (DIO2) 6 (DIO6) 10 (DIO10) 3 (DIO3) 7 (DIO7) 11 (DIO11)	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

Figura 54: Comandos AT – I/O (DIGI INTERNATIONAL INC., 2008)

Table 7-07. I/O Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
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	0-5 0 – Disabled 1 - Node identification 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 5 = Digital output, default high	1
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	0x14 - 0xFF (200 - 2550 ms)	0
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	

Figura 55: Comandos AT - I/O (continuação) (DIGI INTERNATIONAL INC., 2008)

Table 7-07. I/O Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
PR	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) Bits:" 0 - DIO4 (Pin 11) 1 - AD3 / DIO3 (Pin 17) 2 - AD2 / DIO2 (Pin 18) 3 - AD1 / DIO1 (Pin 19) 4 - AD0 / DIO0 (Pin 20) 5 - RTS / DIO6 (Pin 16) 6 - DTR / Sleep Request / DIO8 (Pin 9) 7 - DIN / Config (Pin 3) 8 - Associate / DIO5 (Pin 15) 9 - On/Sleep / DIO9 (Pin 13) 10 - DIO12 (Pin 4) 11 - PWM0 / RSSI / DIO10 (Pin 6) 12 - PWM1 / DIO11 (Pin 7)	CRE	0 - 0x1FFF	0 - 0x1FFF
RP	**RSSI PWM Timer.** Time RSSI signal will be output after last transmission. When RP = 0xFF, output will always be on.	CRE	0 - 0xFF [x 100 ms]	0x28 (40d)
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. (See D0 command).	CRE		

Figura 56: Comandos AT - I/O (continuação 2) (DIGI INTERNATIONAL INC., 2008)

Diagnostics

Table 7-08. Diagnostics Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
VR	**Firmware Version**. Read firmware version of the module.	CRE	0 - 0xFFFF [read-only]	Factory-set
HV	**Hardware Version**. Read hardware version of the module.	CRE	0 - 0xFFFF [read-only]	Factory-set
%V	**Supply Voltage.** Reads the voltage on the Vcc pin. To convert the reading to a mV reading, divide the read value by 1023 and multiply by 1200. A %V reading of 0x8FE (2302 decimal) represents 2700mV or 2.70V.	CRE	-	-

1. Node types that support the command:C = Coordinator, R = Router, E = End Device

Figura 57: Comandos AT – Diagnóstico (DIGI INTERNATIONAL INC., 2008)

AT Command Options

Table 7-09. AT Command Options Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
CT[2]	**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[2]	**Exit Command Mode.** Explicitly exit the module from AT Command Mode.	CRE	--	--
GT[2]	**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[2]	**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. CC command is only applicable when using modules that contain the following "AT Command" firmware versions: 8.0xx (Coordinator), 8.2xx (Router), 8.4xx (End Device)	CRE	0 - 0xFF	0x2B ('+' ASCII)

1. Node types that support the command: C = Coordinator, R = Router, E = End Device
2. Command supported by modules using AT Command firmware only

Figura 58: Comandos AT – Opções (DIGI INTERNATIONAL INC., 2008)

Sleep Commands

Table 7-010. Sleep Commands

AT Command	Name and Description	Node Type[1]	Parameter Range	Default
SM	**Sleep Mode** Sets the sleep mode on the RF module	RE	0-Sleep disabled 1-Pin sleep enabled 4-Cyclic sleep enabled Note: When SM=0, the device operates as a router. When SM changes to a non-zero value, the router leaves the network and rejoins as an end device. **Only end devices can sleep**	0
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	RE	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.	RE	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 Sleep Mode chapter for more information.	E	0 - 0xFF	0

Figura 59: Comandos AT – Sleep (DIGI INTERNATIONAL INC., 2008)