unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

CLAUDIO MASSARU NAKAYAMA

SISTEMA DE COMUNICAÇÃO BLUETOOTH VOLTADA PARA AUTOMAÇÃO RESIDENCIAL

Guaratinguetá
2012

CLAUDIO MASSARU NAKAYAMA

SISTEMA DE COMUNICAÇÃO *BLUETOOTH* VOLTADA PARA AUTOMAÇÃO RESIDENCIAL

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Samuel E. de Lucena

Guaratinguetá

2012

Nakayama, Cláudio Massaru
N163s Sistema de comunicação bluetooth voltada para automação residencial / Cláudio Massaru Nakayama. – Guaratinguetá : [s.n], 2013
61 f. : il.
Bibliografia : f. 58-59

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2013
Orientador: Prof. Dr. Samuel Euzedice de Lucena

1. Automação residencial I .Título.
CDU 681.5

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

SISTEMA DE COMUNICAÇÃO *BLUETOOTH* VOLTADA PARA AUTOMAÇÃO RESIDENCIAL

CLÁUDIO MASSARU NAKAYAMA

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUADO EM ENGENHARIA ELÉTRICA"**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Prof. Dr. LEONARDO MESQUITA
Coordenador

BANCA EXAMINADORA:

Prof. Dr. SAMUEL E. DE LUCENA
Orientador/UNESP-FEG

Prof. Dr. LEONARDO MESQUITA
UNESP-FEG

Prof. Msc. FERNANDO RIBEIRO FILADELFO
UNESP-FEG

Dezembro de 2012

NAKAYAMA, C. M. **Sistemas de comunicação BLUETOOTH voltada para automação residencial.** 2012. 61 f. Trabalho de Graduação (Graduação em Engenharia Elétrica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2012.

RESUMO

A tecnologia *Bluetooth* presente em *smartphones* e *tablets* pode agregar uma interessante funcionalidade voltada para a automação residencial. Através da implementação de uma rede de comunicação sem fio entre um dispositivo móvel e módulos receptores localizados em diferentes pontos, ou até mesmo em um único ponto centralizado na residência, será possível gerenciar diversos elementos domésticos com apenas alguns toques no *smartphones*. Com essa aplicabilidade, o usuário possuirá maior integração com sua residência, comodidade, segurança e economia. Nesse contexto, este trabalho apresenta um projeto de sistema de comunicação *Bluetooth* de baixo custo entre um *smartphone* e um módulo receptor microcontrolado com desenvolvimento da interface de controle a partir da plataforma *Android*.

PALAVRAS CHAVE: *Bluetooth*, automação residencial, *Android*.

NAKAYAMA, C. M. **BLUETOOTH communication systems geared for home automation.** 2012. 61 f. Graduate Work (Graduate in Electrical Engineering) - Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2012.

ABSTRACT

Bluetooth technology found in smartphones and tablets can add an interesting feature focused on home automation. Through the implementation of a wireless network communication between a mobile device and receiver modules located in different points, or even on a single centralized point in residence, you can manage many household items with just a few taps on smartphones. With such applicability, allow the user greater integration with your home, comfort, safety and economy. In this context, this work presents a system design of low cost Bluetooth communication between a smartphone and a receiver module with microcontroller interface development control from the Android platform.

KEYWORDS: Bluetooth, Home Automation, Android.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Nesses últimos anos, a rápida evolução da tecnologia vem transformando em realidade a idéia da *"Casa do Futuro"* originada nos Estados Unidos na década de 20, com a promessa de facilitar as tarefas rotineiras do lar. A principal questão aborda a necessidade de se gerenciar cargas em diferentes ambientes domésticos de forma remota e simplificada (BOLZANI, 2007). Como qualquer novidade, o uso de sistemas automatizados em residências representa inicialmente, símbolo de status e modernidade. Em seguida, o conforto e a conveniência proporcionados passam a ser decisivos. E finalmente, se tornará uma necessidade e um fator de economia.

A tendência é a difusão cada vez maior deste tema no Brasil, que ainda representa um promissor mercado, sendo os Estados Unidos, o maior centro de desenvolvimento dessa área.

> Nos Estados Unidos, são aproximadamente 5 milhões de residências automatizadas e um mercado de US$ 1,6 bilhão de dólares em 1998 à US$ 3,2 bilhões para o ano de 2002 e previsão de 10,5 bilhões em 2008. No Brasil, segundo a AURESIDE, estima-se um potencial de 2 milhões de residências apenas para o estado de São Paulo e faturamento de US$ 100 milhões em 2004. (TEZA, 2002, p. 25)

A tecnologia presente no cotidiano das pessoas, que envolve avançados sistemas de comunicação sem fio, tem permitido que a automação residencial torne-se naturalmente um atrativo tecnológico. Será possível, por exemplo, ativar a iluminação de cada cômodo da casa com um simples toque no celular, como também acionar remotamente a abertura do portão da garagem, gerenciar câmeras de segurança, entre muitas outras aplicações. Entre as várias tecnologias de transmissão de dados sem fio, o *Bluetooth* apresenta grande aplicabilidade para o desenvolvimento de projetos relacionados à área de automação residencial, pois, além de estar presente em uma enorme parcela dos dispositivos móveis, a tecnologia apresenta grandes vantagens como mínimo consumo de energia e custo de implementação relativamente baixo. Baseando-se nesse contexto, o trabalho proposto resume-se inicialmente em mostrar a funcionalidade do protocolo *Bluetooth,* analisando também outras tecnologias de comunicação sem fio e posteriormente, desenvolver um protótipo de comunicação *Bluetooth*

entre um *smartphone* e um módulo receptor microcontrolado para realizar o acionamento de vários tipos de cargas de uma residência. Será desenvolvido um aplicativo gerado a partir da plataforma *Android,* utilizada nos *smartphones,* como meio de interface entre o usuário e os dispositivos.

2. SISTEMAS DE COMUNICAÇÃO SEM FIO

Os avanços tecnológicos na área de comunicações possibilitaram o surgimento de vários tipos de sistemas, que procuram atender a real necessidade de seus usuários, com a melhor qualidade possível. Nesses últimos anos, a comunicação sem fio ganhou espaço considerável entre as tecnologias de transmissão de dados. Essa tendência foi fortalecida pelo amplo investimento de instituições e empresas, o que hoje representa inserção dos sistemas de comunicação sem fio em praticamente qualquer produto eletrônico, como os celulares, computadores pessoais e periféricos, dispositivos automotivos, etc.

Cada vez mais a comunicação sem fio torna-se popular devido à conveniência e mobilidade, apresentando a grande vantagem de eliminar o uso de cabos para envio e recepção de dados entre os dispositivos. Entre as diversas tecnologias que desempenham este papel, temos o *Wi-Fi*, *IrDA* (*Infrared Data Association*), *ZigBee* e o *Bluetooth*. Cada uma dessas tecnologias apresentam características particulares, sendo assim, dependendo da aplicação, podem apresentar vantagens e desvantagens.

O *IrDA* utiliza transmissão de dados via feixes infra-vermelhos. Essa tecnologia possui grande limitação quanto à forma de transmissão, pois depende do alinhamento dos dispositivos com a tecnologia, ou seja, devem estar direcionados um para o outro, com a linha de visada livre, sem obstáculos.

Em comparação com a tecnologia *IrDA*, o *Bluetooth*, *Wi-fi* e o *ZigBee* possuem muitas vantagens para as mais variadas aplicações. Todas apresentam a capacidade de transmissão livre de interferências, utilizando técnicas que garantem a segurança no envio e recebimento de dados. A Tabela 1 mostra um comparativo entre as tecnologias de comunicação sem fio mais comuns:

	IrDA	*Bluetooth*	*Wi-Fi*	*ZigBee*
Alcance	Até 4m	1 a 100m	100 a 1000m	70 a 1600m
Consumo de energia	Baxo	Baixo	Médio	Muito Baixo
Complexidade de implementação	Baixa	Média	Alta	Baixa
Taxa de transmissão	Até 4Mbps	Até 24Mbps	Até 54Mbps	250 kbps
Numero de nós da rede	1	7	50	65535
Custo	Baixo	Baixo	Médio	Baixo

Tabela 1 - Comparativo entre as tecnologias de comunicação sem fio

Analisando-se as tecnologias, verifica-se que o *ZigBee* oferece maiores vantagens em comparação com o *Bluetooth*, *Wi-fi* e o *IrDA*. Possui maior alcance de transmissão, menor consumo de energia, custo baixo e complexidade de implementação simples. No entanto, o *ZigBee* ainda é uma tecnologia difundida apenas no campo industrial e possui grande limitação quanto à capacidade de transmissão de dados, e de forma geral, perde espaço para o *Bluetooth* e o *Wi-Fi*. Atualmente estas tecnologias estão envolvidas em muitas aplicações como em computadores pessoais, celulares, automóveis, acessórios eletrônicos, etc.

Para uma aplicação que envolva baixo custo de implementação e que atenda de forma satisfatória quanto à capacidade de transmissão de dados, o *Bluetooth* se adequa bem às necessidades para uma aplicação voltada para automação residencial. Oferece algumas vantagens como taxa de transmissão relativamente alta, consumo de energia baixo e custo do módulo de transmissão/recepção baixo. O *Wi-Fi* possui taxa de transmissão e alcance maiores, no entanto para a aplicação a qual será apresentada nesse trabalho, esses parâmetros não são fatores críticos, pois basicamente, será trabalhado apenas com dados de controle,

como por exemplo, envio de sinais para acionamento de uma lâmpada localizada em um cômodo de uma pequena residência.

Os tópicos a seguir tratarão com maiores detalhes as características de funcionamento da tecnologia *Bluetooth*.

2.1 – TECNOLOGIA *BLUETOOTH*

O *Bluetooth* é um sistema que emprega transmissão de dados sem fio a partir de ondas de rádio, seguindo um padrão global de comunicação padronizada pela *IEEE 802.15* como uma *Wireless Personal Area Network (WPAN)*. A tecnologia permite uma comunicação simples e rápida entre computadores, smartphones e outros dispositivos e periféricos, desde que estejam dentro da área de alcance da rede formada pelos mesmos.

2.1.1 - SURGIMENTO DA TECNOLOGIA

O surgimento da tecnologia *Bluetooth* se dá a partir 1994, quando a empresa Ericsson deu início ao desenvolvimento de uma tecnologia que permitisse a comunicação entre telefones celulares e acessórios utilizando sinais de rádio de baixo custo, eliminando uso de cabos. A partir desse projeto, originou-se um sistema de rádio de curto alcance que recebem o nome de *McLink*. Com o atrativo de implementação fácil e barata, o projeto começou a despertar interesse de outras empresas. Em 1998, várias empresas apoiaram a Ericsson, criando assim, o grupo *Bluetooth SIG (Special Interest Group – Ericsson, Intel, Toshiba, IBM e Nokia*). A integração entre essas empresas permitiu que fossem desenvolvidos diversos padrões a fim de garantir o uso e interoperabilidade da tecnologia nos mais variados dispositivos.

2.1.2 - HISTÓRIA DO *BLUETOOTH*

A denominação *Bluetooth* é uma homenagem ao rei dinamarquês *Harald Blatand*, mais conhecido como *Harald Bluetooth*. Um de seus grandes feitos foi a unificação da Dinamarca. Por este fato, o nome *Bluetooth* foi escolhido, pois faz alusão à integração de dispositivos de diferentes fabricantes. O logotipo é uma junção de símbolos nórdicos correspondentes às letras *"H"* e *"B",* iniciais de *Harald Bluetooth*.

Figura 1. Logotipo *Bluetooth* (*BLUETOOTH SIG*, 2012)

2.2 - FUNCIONAMENTO DO *BLUETOOTH*

2.2.1 - ARQUITETURA *BLUETOOTH*

A figura abaixo mostra a pilha de protocolo *Bluetooth*:

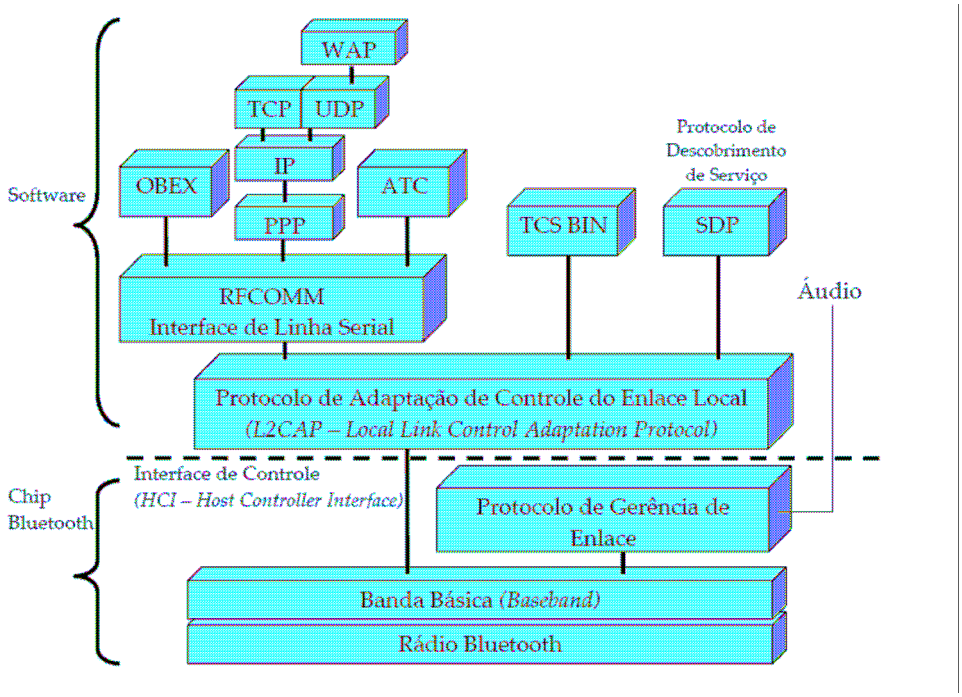


Figura 2. Arquitetura *Bluetooth* – pilha de protocolos (SACKS, 2003)

2.2.1.1 – RÁDIO

É a camada responsável por especificar detalhes como a freqüência de operação, potência, as técnicas de modulação e de transmissão utilizadas.

Como foi mencionado, a transmissão de dados é feita através de rádio freqüência, permitindo que um dispositivo detecte o outro dentro dos limites de proximidade. Para que seja possível atender aos mais variados tipos de dispositivos, o alcance máximo do *Bluetooth* foi divido em três classes de potência:

- Classe 1: potência máxima de 100mW, alcance de até 100 metros;
- Classe 2: potência máxima de 2,5mW, alcance de até 10 metros;
- Classe 3: potência máxima de 1mW, alcance de até 1 metro.

Isso significa que um dispositivo *Bluetooth* classe 3 comunica-se com outros dispositivos da mesma classe dentro do limite de alcance de 1 metro. A distância de transmissão parece ser insignificante, no entanto, para algumas aplicações como conectar um fone de ouvido em um celular, é suficiente para garantir uma boa comunicação entre os dispositivos. É importante lembrar que a comunicação entre dispositivos *Bluetooth* de diferentes classes, se limita ao alcance do dispositivo de classe inferior, ou seja, com menor alcance.

A velocidade de transmissão pode atingir até 24 Mbps na versão 3.0 do *Bluetooth*. Existem versões anteriores como a 1.2 com a taxa de transmissão de 1Mbps e a versão 2.0, atingindo até 3Mbps.

2.2.1.2 - TÉCNICA DE *FREQUENCY HOPPING SPREAD SPECTRUM* (*FHSS)*

A tecnologia *Bluetooth* opera na faixa de freqüência *ISM (Industrial, Scientific, Medical)* que varia de 2,4 a 2,485 GHz. O uso dessa faixa para comunicação qualifica a tecnologia como um padrão global, pois é um padrão aberto utilizado em vários países. Uma vez que a faixa *ISM* pode ser utilizada por qualquer sistema de comunicação, é necessário garantir que o sinal do *Bluetooth* não sofra e nem gere interferências. Para que não ocorra esse problema, o *Bluetooth* usa uma técnica chamada salto de freqüência de espalhamento

espectral (*FHSS*). Esta técnica consiste em dividir a banda existente em canais independentes que mudam a frequência de transmissão dos dados ao longo do tempo. De maneira pseudoaleatória, o canal de freqüência é variado rapidamente, na qual cada valor é utilizado por um curto período de tempo chamado *dwell time* que pode ser ajustado e não deve ser superior a 400ms. No *Bluetooth*, o *dwell time* é de 625us, ou seja, 1600 mudanças de canais freqüência por segundo. A transferência de pacotes deve ocorrer dentro desses intervalos de tempo, não podendo ocorrer variação de freqüência durante o envio deles. Na maioria dos países, a faixa de freqüência *ISM* é dividida em 79 canais com banda de 1 MHz cada, de acordo com a especificação da tecnologia. Em geral todos os 79 canais disponíveis são utilizados e a freqüência pode ser determinada de acordo com a expressão da Equação 1:

$$f = (2402 + k)\ MHz, \qquad k = 0, 1, ..., 78 \qquad (1)$$

Essa técnica garante segurança criptográfica à transmissão, pois só poderá ser interceptada caso se conheça a seqüência de variação de canais de freqüência.

A Figura 3 mostra um exemplo da técnica *FHSS*.

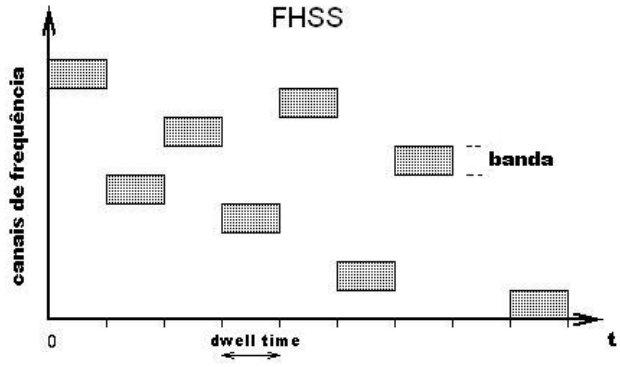


Figura 3. Técnica *FHSS* (MIR;ROBLIN,1997)

Em cada canal de frequência, são transmitidos bits de dados convertidos em sinais de rádio através da modulação *Gaussian Frequency Shift Keying (GFSK).*

2.2.1.3 - MODULAÇÃO *GFSK*

A modulação *Frequency Shift Keying* (*FSK)* atribui freqüências diferentes para a portadora em função do bit que é transmitido. Portanto, quando o bit 0 é transmitido, a portadora assume uma freqüência correspondente a um bit 0, ou seja, atribui uma variação negativa da freqüência durante o período de duração de um bit. Quando o bit 1 é transmitido, a freqüência da portadora é modificada para um valor correspondente a um bit 1, ou seja, varia para uma freqüência maior e permanece nesta freqüência durante o período de duração de um bit, como mostrado na Figura 4:

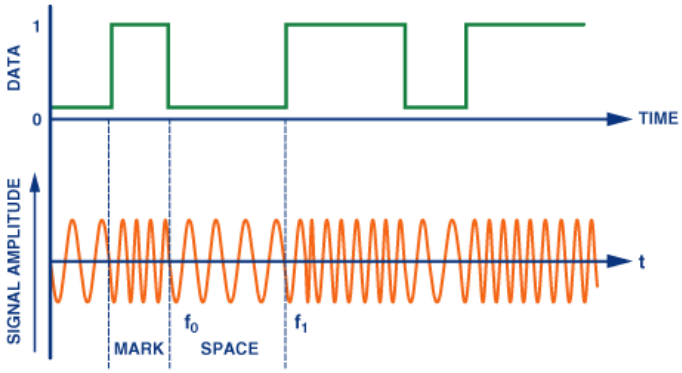


Figura 4. Modulação *FSK* (MURPHY; SLATTERY, 2005)

Na modulação *4FSK*, temos uma freqüência atribuída para cada conjunto de 2 bits (00, 01, 10, 11) totalizando quatro freqüências diferentes e possibilitando uma taxa de transmissão maior, mas, em contrapartida, provoca um aumento na largura de banda. A variação brusca de freqüência faz com que este tipo de modulação apresente grandes bandas.

Para reduzir a largura de banda, o *Bluetooth* utiliza o *GFSK* , o qual corresponde à modulação *FSK* com a introdução de um filtro gaussiano. Esse filtro funciona como uma espécie de formatador de pulsos que serve para suavizar a transição entres os valores de pico. Desta forma, com a redução das variações bruscas de freqüência, reduz-se significativamente a largura de banda.

Figura 5. Diagrama *GFSK (MALBURG, 2008)*

2.2.2 - *BASEBAND*

Representa a camada física do *Bluetooth*. Ela especifica as regras de acesso ao meio e procedimentos de camada física entre dispositivos distintos, como a localização de dispositivos na rede, estabelecimento de conexão entre dispositivos, tipo e formato do pacote, modo de endereçamento, criptografia e correção de erros, transmissão e retransmissão de pacotes, papéis que um determinado dispositivo pode assumir na rede, etc.

2.2.2.1 - TOPOLOGIA / CANAL FÍSICO

Em uma rede *Bluetooth*, cada dispositivo conectado à mesma, assume uma determinada função. Cada terminal poderá desempenhar o papel de mestre ou escravo. O mestre é aquele que iniciou a comunicação e gerencia toda a troca de informações. Uma de suas funções principais é determinar as seqüências de saltos de canais, a fim de criar uma conexão segura com um dispositivo escravo.

Uma rede formada por um mestre e um escravo, damos o nome de ponto a ponto *(point-to-point).* Quando dois ou mais escravos estão conectados a um único mestre, forma-se um tipo de rede denominado ponto a multiponto (*piconet)*. Existe a possibilidade de interação de dispositivos de *piconets* diferentes presentes em uma mesma área de atuação. Isso ocorre, quando as piconets possuem dispositivos *Bluetooth* em comum, formando uma rede chamada de *scatternet*. As *piconets* de uma *scatternet* devem possuir seqüências de saltos em freqüências distintas. Um mesmo dispositivo pode atuar como escravo em mais de uma *piconet*, mas poderá ser mestre em apenas uma delas.

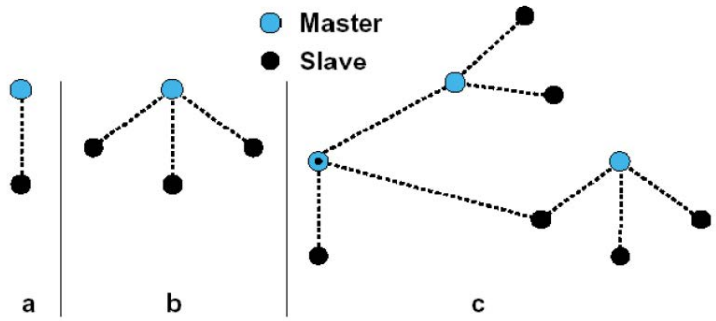


Figura 6. a) ponto a ponto ; b) *Piconet*; c) *Scatternet* (PALO WIRELESS, 2012)

Em uma *piconet* podem coexistir 7 dispositivos escravos em modo ativo e até 255 dispositivos escravos não ativos, que são definidas na especificação como *parked slave.* Cabe ao mestre gerenciar, de maneira conveniente, os dispositivos escravos ativando e desativando-os de modo que estejam ativos no máximo sete de cada vez.

A seqüência de saltos de uma *piconet* é única e é determinado pelo endereço do dispositivo *Bluetooth* do mestre e a fase na seqüência de saltos é determinado pelo relógio do mesmo. Os *times slots* são enumerados de acordo com o relógio do mestre na *piconet*. O dispositivo mestre só pode transmitir seus dados nos *time slots* pares, enquanto que os escravos só podem transmitir nos *time slots* ímpares. A transmissão e recepção dos pacotes são feitas de forma alternada no tempo, operação conhecida como *Time Division Duplex (TDD),* resultando em uma comunicação *full duplex* entre os dispositivos.

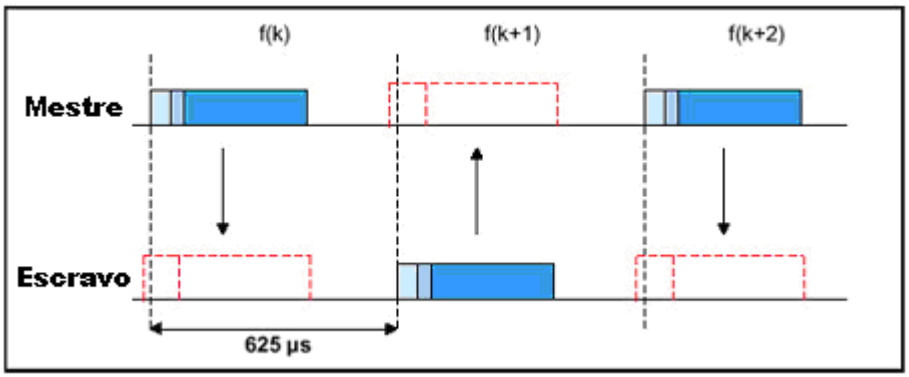


Figura 7. Divisão do canal (PALO WIRELESS, 2012)

Como visto na figura 7, o canal é dividido em *time slots*. Cada time slot tem a duração de até 625us. No *slot* f(k), o mestre transmite seus pacotes. Em f(k+1), o escravo transmite seus pacotes e assim sucessivamente.

O endereço ou identificador de um dispositivo Bluetooth possui 48 bits e é similar ao endereço *MAC* (*Media Access Control)* das placas de rede dos computadores. Ele é expresso na forma de 12 caracteres hexadecimais e é gravado na memória *ROM (Read Only Memory)* de cada dispositivo *Bluetooth*. Há uma padronização dos endereços *MAC* administrada pela *IEEE* (*Institute of Electrical and Electronics Engineers*) que define que os três primeiros bytes, são destinados a identificação do fabricante - eles são fornecidos pela própria *IEEE*. Os três últimos bytes são definidos pelo fabricante, sendo este responsável pelo controle da numeração de cada placa que produz. Apesar de ser único e gravado em hardware, o endereço *MAC* pode ser alterado através de técnicas específicas.

2.2.2.2 - ENLACE FÍSICO

A especificação do *Bluetooth* define dois tipos de enlaces entre os dispositivos mestre e escravo: *SCO (Synchonous Connection-Oriented)* e *ACL (Asynchronous Connection-Less).*

O enlace *SCO* é um link ponto-a-ponto simétrico entre um mestre e um único escravo em uma *piconet*, onde os *time slots* ficam reservados em intervalos de tempo fixo. A reserva destes slots faz com que o link se comporte como uma conexão comutada por circuito, sendo, portando ideal para a transferência de informações com restrições de tempo. O link *SCO* transmite principalmente áudio. De fato, cada canal *SCO* pode transmitir a uma taxa máxima de 64kbps.

Pacotes *SCO* nunca são retransmitidos. Uma vez que não existe retransmissão de dados no caso de erro, os dados de um pacote podem ser enviados com redundância para permitir a correção de uma parcela dos erros durante a transmissão de dados.

Um mestre pode ter até três links deste tipo, que podem ser empregado para a comunicação com um único escravo ou com escravos distintos. Um escravo pode ter até três links deste tipo, caso a comunicação seja com um mesmo mestre, ou até dois links, caso dois mestre estejam envolvidos.

O link *ACL* é um link ponto a multiponto entre um mestre e todos seus escravos ativos participando de uma *piconet*. O link *ACL* transmite principalmente dados. Os links *ACL*, ao contrário dos links *SCO*, fazem retransmissão de pacotes, assegurando assim, a integridade dos dados enviados. O link *ACL* seria análogo a uma conexão comutada por pacotes.

Entre um mestre e um escravo pode existir um único link *ACL*, independente de haver conexões *SCO* estabelecidas.

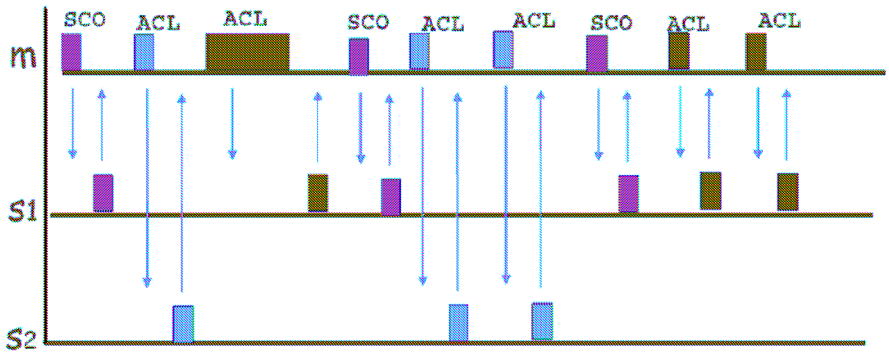


Figura 8. Links *SCO* e *ACL (PALO WIRELESS, 2012)*

2.2.2.3 - PACOTES

Todos os dados no canal de uma *piconet* são transportados em pacotes. O formato geral de um pacote *Bluetooth* é mostrado na figura 9:

Figura 9. Formato do pacote *Bluetooth (PALO WIRELESS, 2012)*

Conforme pode ser observado, o pacote é composto por três entidades: código de acesso (68/72 bits), cabeçalho do pacote (54 bits) e campo de *payload* (0 a 2745 bits).

Código de acesso: utiliza-se para a sincronização de tempo dos dispositivos, compensação do offset, fazer *paging* e *inquiry* e identificar uma *piconet* ou dispositivo endereçado.

Cabeçalho do pacote (*Header*): contém informações sobre o tipo do pacote, numeração do pacote para ordenação, fluxo de controle, endereço do escravo e para checagem de erro do cabeçalho.

Campo de *Payload*: o conteúdo desse campo depende do tipo de pacote enviado. De forma geral, quando o pacote em questão não é de controle, este campo pode ser composto

por um campo de voz, um campo de dados ou por ambos. Se o campo é de dados, então o *payload* também conterá um cabeçalho.

2.2.2.4 - CONTROLE DO CANAL

O controlador do *Bluetooth* opera em basicamente em dois estados: Espera (*standby*) e Conexão (*Connection*). Existem também sete sub estados que são usados para adicionar escravos a uma *piconet* ou para estabelecer conexões. Estes são os estados de: Pesquisa (*Inquiry*), Escuta de Pesquisa (*Inquiry Scan*), Resposta da pesquisa, chamada (*page*), escuta de chamada (*page scan*), resposta do escravo e resposta do mestre.

A figura 10 mostra um esquema de funcionamento do controle de estados:

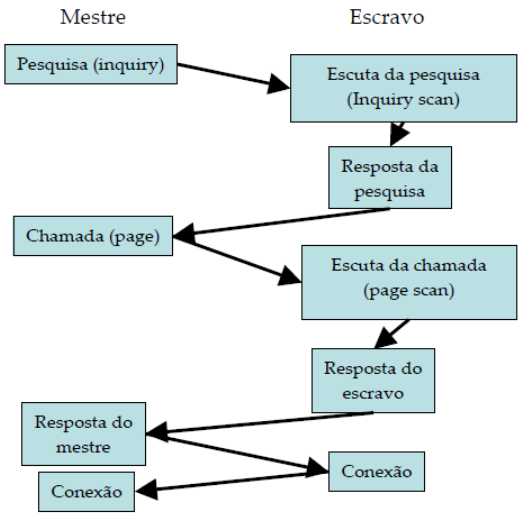


Figura 10. Controle de estados (SACKS, 2003)

A conexão entre dois dispositivos é estabelecida através da realização dos procedimentos de *inquiry* e *page*. Quando nada é conhecido sobre o dispositivo remoto. Ambos procedimentos serão necessários para que não seja estabelecida a conexão. Porém, se alguns detalhes sobre o dispositivo são conhecidos, somente o procedimento de *page* é necessário.

O procedimento de *inquiry* permite que sejam descobertos os dispositivos *Bluetooth* que se encontram ao alcance do dispositivo fonte e determinar o endereço e relógio desses dispositivos. Este procedimento tem inicio com a fonte entrando no estado de *inquiry*, no qual ela difunde pacotes do tipo *IAC (Inquiry Access Code* – Código de acesso de pesquisa

utilizado quando o mestre deseja descobrir quais dispositivos *Bluetooth* estão presentes em seu raio de ação). Estes pacotes são enviados seguindo uma seqüência de freqüências denominada *inquiry hopping sequence*.

Para receber este pacote, o dispositivo destino deve se encontrar no estado de *inquiry scan*. Ao receber o pacote *IAC*, ele passa para o estado *inquiry response* e envia um pacote de resposta contendo seu endereço e seu relógio. Este pacote é enviado utilizando uma seqüência de freqüências denominada *inquiry response hopping sequence.*

Após esse procedimento de *inquiry*, a fonte tem os endereços e relógios dos dispositivos ao seu alcance. E a conexão pode ser estabelecida usando o procedimento de *page*. O dispositivo fonte passar a ser o máster da conexão.

No procedimento de *page*, a fonte envia pacotes do tipo *DAC (Device Access Code* – Código de acesso ao dispositivo) para os dispositivos alvo. Para que esses dispositivos possam receber a mensagem de *page* (pacote *DAC*), eles deverão se encontrar no estado de *page scan*, na qual ele escuta o canal esperando por uma mensagem com o seu endereço. Ao receber essa mensagem, o dispositivo entra no estado de *slave* response e envia uma mensagem de reposta contendo o seu endereço, utilizando para isso uma seqüência de freqüências denominada *page resposnse hopping sequence.* Após receber a resposta, a fonte entra no estado de máster response e envia ao dispositivo destino de saltos de freqüência, utilizando uma seqüência de freqüências denominada *page hopping sequence.* Ao receber esse pacote, o dispositivo destino deve enviar um pacote de resposta com seu endereço. A partir daí está estabelecida a conexão e o dispositivo de destino passa a utilizar o relógio e a sequência de saltos de frequências do canal do dispositivo fonte.

2.2.3 - *LMP (Link Manager Protocol* – Protocolo de Gerenciamento de Enlace)

Utilizado para o estabelecimento e segurança de conexão, configuração entre dois dispositivos. Ele também realiza o gerenciamento da *piconet*, como por exemplo, controlando os estados de conexões de uma *piconet* e os modos de energia de um dispositivo. O dispositivo *Bluetooth* pode estar em um dos seguintes modos de energia:

-*Active Mode*: modo no qual o dispositivo participa ativamente no canal;

- *Sniff Mode*: dentre os modos de economia de energia, é o que possui maior consumo.

Neste modo, os dispositivos escravos escutam o canal a uma taxa reduzida. Eles escutam apenas durante alguns *time slots* e começam a escuta de novo em intervalos de tempo.

O intervalo de *sniff* é programável, sendo dependente da aplicação. Quando o escravo está neste modo, o mestre só pode começar uma transmissão em um *sniff slot*.

- *Hold Mode*: modo em que o consumo de energia é intermediário. O mestre pode colocar um escravo ou um escravo pode querer entrar neste modo por um intervalo de tempo especificado. Durante este intervalo nenhum pacote *ACL* será transmitido do mestre.

Neste modo, somente o relógio interno está rodando, economizando energia. Um dispositivo entra neste modo quando não existe necessidade de enviar ou receber dados por um tempo relativamente longo ou quando um dispositivo quer descobrir ou ser descoberto por outros dispositivos *Bluetooth*, ou quer se juntar a outros *piconets*.

- *Park Mode*: modo em que possuir o menor consumo de energia. Apesar de permanecer sincronizado com a *piconet*, os dispositivos escravos não participam do trafego do canal.

O único procedimento realizado por um escravo é a transmissão do sincronismo com o mestre, onde ele acorda em alguns instantes em intervalos de tempo pré-determinados para escutar mensagens. Os escravos que atuam neste modo de operação perdem seus endereços na *piconet*, devendo reobtê-los para participar novamente no canal. Um *parked slave* pode ser ativado de novo pelo mestre somente quando o escravo está acordado.

O *LM* descobre outros *LM's* remotos e se comunica com eles através do *LMP*, no qual consiste essencialmente em um numero de mensagens que são enviados de um dispositivo para o outro.

Os sinais de radio podem ser facilmente interceptados, devido a isso, dispositivos *Bluetooth* tem embutido segurança para prevenir que alguém leia ou altere a mensagem original. Como dito acima, a segurança inclui aspectos como autenticação e fazer criptografia.

A autenticação restringe o acesso a dados e funções privadas. O procedimento envia uma mensagem *LMP* que contém um numero aleatório para o requisitor. O requisitor calcula uma resposta que é uma função do numero aleatório enviado, do endereço do dispositivo do requisitor e de uma chave secreta. A resposta é enviada de volta para o verificador, que checa

se a resposta é correta ou não. A criptografia dos dados previne que alguém leia a mensagem e mantém a privacidade do link.

2.2.4 - *HCI (Host Controller Interface)*

O *HCI* provê uma interface de comando ao controlador de banda básica e ao gerenciador de enlace e acessa o status do hardware e dos registros de controle. Essencialmente, essa interface promove a abstração entre o hardware e a pilha de protocolos de nível superior. Possui três entidades funcionais:

- *HCI firmware*: implementa os comandos de acesso à banda básica e ao status e registros do hardware, e está localizado no dispositivo de hardware.
- *HCI driver*: está inserido na entidade de software e recebe notificações assíncronas de eventos HCI.
- *Host Controller Transport Layer*: camada por onde o *HCI* firmware e driver se comunicam. Existem três tipos inicialmente definidos: *USB (Universal Serial Bus), UART (Universal Asynchronous Receiver Transmitter) e RS232*.

2.2.5 - VISÃO GERAL DOS PROTOCOLOS DE CAMADAS SUPERIORES

2.2.5.1 - *L2CAP (Logical Link Control and Adaptation Layer Protocol)*

O *L2CAP* controla enlaces lógicos e executa um protocolo de adaptação de camada, que é a interface entre o padrão de protocolos de transporte de dados e o protocolo *Bluetooth*, provendo serviços de dados orientados e não orientados à conexão para camadas de protocolo de nível superior através da multiplexação de protocolos, segmentação e remontagem. Ela é usada apenas para enlaces tipo *ACL* e é baseada no conceito de canais, podendo ter vários abertos ao mesmo tempo.

2.2.5.2 - *SDP (Service Discovery Protocol)*

O descobrimento de serviços em um ambiente *Bluetooth*, onde o conjunto de serviços disponíveis muda dinamicamente, é qualitativamente diferente do descobrimento de serviço

em ambientes de redes tradicionais. A especificação do *SDP* foi definida com o intuito de atender às características especificas do *Bluetooth*.

O *SDP* provê meios de aplicações clientes descobrirem quais os serviços disponíveis de aplicações servidoras e determinar as suas características, como tipo ou classe dos serviços e os mecanismos necessários para utilização desses. Existe no máximo um servidor *SDP* por dispositivo, que pode operar como cliente e servidor *SDP* simultaneamente. Quando um servidor torna-se disponível, ele avisa o cliente em potencial, e se este estiver interessado, emite um pedido *SDP* (*SDP request*).

2.2.5.3 - RFCOMM

O *RFCOMM* é um protocolo simples de transporte que provê emulação de porta serial sobre o protocolo *L2CAP*. Suporta até 60 conexões simultâneas entre dois dispositivos *Bluetooth*. Na conexão direta, um dispositivo é conectado diretamente a um '*end-point*' (ou sistema ponto final), que pode ser outro dispositivo, um computador ou uma impressora, dentre outros. Existem também conexões entre dispositivos e equipamentos de rede, como por exemplo, um modem.

Vários protocolos são suportados através do *RFCOMM*. O *OBEX* oferece as mesmas características para aplicações como as da hierarquia do protocolo de *IrDA* (Infra Vermelho), viabilizando a comunicação entre aplicações.

ATC vem de *AT Commands*, que é a linguagem padrão *de facto* usada para controlar *modems*. O conjunto de comandos *AT* é reconhecido por praticamente todos os *modems* existentes no mercado.

PPP (Point to Point Protocol) é um protocolo de acesso remote que especifica completamente a transmissão de datagramas entre equipamentos de comunicação de dados de diferentes fabricante para ligações ponto-a-ponto do tipo seriais discadas, seriais dedicadas (enlaces telefônicos, satélite, rádio), *ISDN* e outras.

3.DESENVOLVIMENTO DO PROJETO

3.1 – O Protótipo

A principal idéia deste projeto é criar um sistema de comunicação via Bluetooth que seja capaz de simular uma interface a partir de um celular, o qual permitirá ao usuário ter maior controle dos dispositivos de sua residência. A implementação desse sistema em uma residência poderá ser feita de diversas formas. Como existem 3 classes diferentes de dispositivos *Bluetooth*, nas quais se distinguem pela potência de transmissão e principalmente pelo alcance, podemos ter dois tipos de topologias da rede de controle. A primeira opção é utilizar um controlador central conectado a um módulo *Bluetooth*, onde todos os elementos de controle se interligarão através do mesmo. Nessa topologia, consideramos que seja preciso um módulo *Bluetooth* com alcance de transmissão de 100 m (classe 3) para que haja a possibilidade de acessar os parâmetros de controle no celular em todos os ambientes de uma residência. Mesmo com um custo inicial relativamente alto do módulo Bluetooth de classe 3, seria necessário apenas um para integração de todos os elementos de controle. No entanto, existe a desvantagem em relação à sobrecarga de informação sobre o módulo, já que a taxa de transmissão é relativamente baixa, característica bastante crítica caso for utilizado comunicação *half duplex* no módulo *Bluetooth*. Outro fator importante seria a dificuldade na instalação, sendo que existe complexidade no cabeamento se for colocada em uma instalação elétrica tradicional.

A topologia distribuída, ou seja, com os controladores localizados em pontos “comuns” ou em cada ambiente da residência, mostra-se mais prática e viável para uma implementação do sistema *Bluetooth* em instalações tradicionais. Neste caso, podemos ter módulos *Bluetooth* de classe 2 com alcance de 10 m localizados em cada ambiente, sendo que estes podem formar uma pequena rede (*piconet*) entre os módulos, criando uma integração “virtual” dos elementos de controle de toda a residência. Sabendo que o celular atuará como dispositivo mestre do sistema, seria possível ter 7 módulos *Bluetooth* escravos para recepção. Cada um dos controladores espalhados pela residência poderão realizar diferentes funções dependendo do ambiente na residência:

a) Iluminação: acionamento de lâmpadas, com possibilidade de ajuste de intensidade de luz;
b) Motores: comando de portões automáticos, persianas e ventiladores;

c) Segurança: implantação de câmeras de segurança e alarmes;
d) Sensores: medição de temperatura ambiente, detectores de umidade dedicados a descobrir vazamentos, detecção de fumaça em caso de incêndio, sensores de presença, etc;
e) Economia de energia: implantação de contadores para tomadas elétricas, ligando e desligando-as em função dos parâmetros de economia de energia.

Em função de cada aplicação, deverá existir um circuito apropriado conectado ao módulo, como por exemplo, no controle de intensidade de iluminação. Neste caso seria necessário a integração de "*dimmers*" para controle em corrente alternada, visto que, o módulo trabalha em corrente contínua com níveis de tensão de 3.3 e 5V.

Com a finalidade de exemplificar uma aplicação do módulo *Bluetooth*, será simulado um ambiente doméstico, como a sala de estar, no qual possuirá controle de iluminação por *LEDs* e ventilação.

Nesse protótipo será desenvolvida uma placa microcontrolada utilizando PIC para recepção de dados via serial através do módulo *Bluetooth*. O microcontrolador será responsável por interpretar os dados recebidos a partir do módulo Bluetooth e efetuar o acionamento de suas saídas para ligar e desligar *LEDs*, controlar por *PWM (Pulse Width Modulation)* a velocidade de um motor CC e a intensidade de iluminação de *LEDs*. Lembrando que em situação real, utilizando-se produtos comerciais, como as lâmpadas *LEDs* e ventiladores (motores CA) será necessário chaves estáticas ou até mesmo relês eletromecânicos para realizar o acionamento dos mesmos.

Para a interface de controle dos parâmetros de acionamento será utilizado o celular smartphone *LG E612F* como dispositivo mestre para a comunicação *Bluetooth* com a placa receptora. Essa interface se baseará no aplicativo desenvolvido na plataforma *Android* do celular, no qual possuirá acesso à conexão com a placa de recepção por *Bluetooth*, e enviará informações para o acionamento das cargas de acordo com a vontade do usuário.

Nos próximos tópicos serão detalhados cada componente do projeto.

3.2 – O Módulo *Bluetooth*

O módulo *Bluetooth* está presente em qualquer aparelho que estabelece comunicação *Bluetooth* com qualquer outro dispositivo com a mesma característica. Atualmente, o módulo

Bluetooth pode ser encontrado facilmente no mercado, sendo vendidos separadamente com custo acessível (média de R$ 50,00).

Foi escolhido o módulo *Bluetooth* da fabricante *BOLUTEK* modelo *BLK BC04B* (Figura 11). Este dispositivo é de classe 2 e especificação de versão 2.0 com taxa máxima de transmissão de 2Mbps. A potência máxima é de 2,5mW (dBm) com alcance máximo de 10 metros.

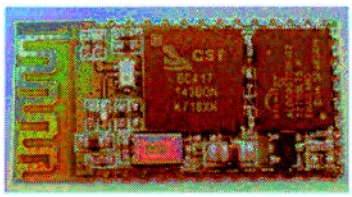


Figura 11. Módulo *Bluetooth* BC04B (BOLUTEK, 2010)

Para a confecção do protótipo foi adaptado uma placa com o módulo Bluetooth escolhido com especificação de interface *UART*. Essa adaptação (Figura 12) utiliza regulador interno de tensão que permite alimentação com 5Vdc e uma saída regulada de 3.3Vdc com corrente máxima de 50mA. Apresenta os pinos de transmissão (*TXD*) e recepção (*RXD*) serial com configuração de taxa de transmissão de 1200 a 1382400 *bps* através de comandos *AT*. A transmissão e recepção serial operam com nível 3.3Vdc.

O módulo pode ser configurado tanto como dispositivo mestre (nível lógico alto) ou como escravo (nível lógico baixo), sendo que, a configuração é feita a partir do pino *SET*.

Figura 12. Módulo *Bluetooth* com placa adaptadora (Própria Autoria)

Os demais pinos. *CLR* e *INT*, não serão utilizados na aplicação.

3.3 – Microcontrolador *PIC 16F876A*

Para desenvolvimento do protótipo, o microcontrolador *PIC16F876A* apresenta características interessantes para a aplicação proposta.

Principais características:

- 8kb de memória de programa (*FLASH*);
- 22 pinos configuráveis I/O;
- 2 saídas configuráveis *PWM* com 10 bits de resolução;
- Comunicação Serial *USART* (*Universal Synchronous Asynchronous Receiver)*

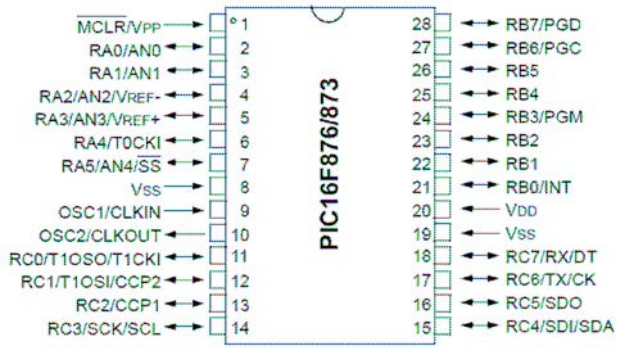


Figura 13. Microcontrolador PIC16F876A (MICROCHIP, 2003)

Este modelo da família 16F de microcontroladores PIC é de fácil procura no mercado, apresentando custo relativamente baixo (média de R$ 20,00).

3.3.1 - COMUNICAÇÃO SERIAL *UART*

A maioria das mensagens digitais são longas e por questões de praticidade a transferência de dados não é realizada enviando todos os bits simultaneamente. A transmissão bit-serial converte a mensagem em um bit por vez através de um canal, sendo que a comunicação pode ocorrer com a sincronização do transmissor e do receptor, ou com a inserção de bits para indicar o início e o fim da seqüência do dado transmitido. A primeira técnica é conhecida como transmissão serial síncrona, já a segunda é denominada transmissão serial assíncrona.

A transmissão serial assíncrona (*UART*) é utilizado apenas um canal para a transmissão dos dados e dos bits de sincronização. Isto é feito acrescentando um bit de começo start bit e um bit de parada stop bit; o canal de comunicação fica em nível alto até um start bit ser transmitido, forçando o valor no canal para nível baixo; após este bit segue a informação, com um tamanho de palavra pré-configurado no receptor e no transmissor.

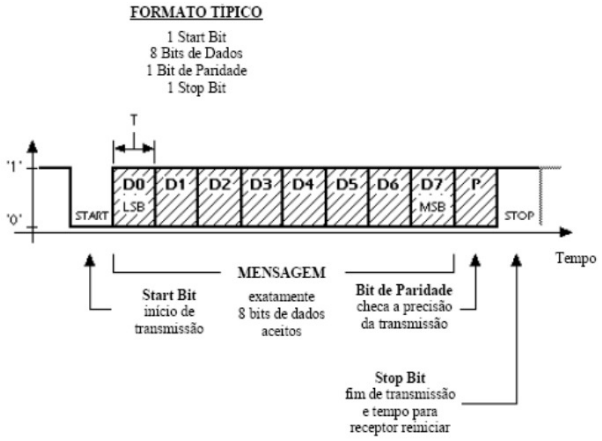


Figura 14. Formato do envio de dados serial *UART* (MIYADAIRA, 2009)

A taxa de transferência refere-se a velocidade com que os dados são enviados através de um canal e é medido em transições elétricas por segundo. Na norma *EIA232*, ocorre uma transição de sinal por bit, e a taxa de transferência e a taxa de bit são idênticas. Nesse caso, uma taxa de 9600 bps corresponde a uma transferência de 9600 dados por segundo, ou um período de aproximadamente, 104ms (1/9600 s). Outro conceito é a eficiência do canal de comunicação que é definido como o número de bits de informação utilizável enviados pelo canal por segundo. Ele não inclui bits de sincronismo, formatação, e detecção de erro que podem ser adicionados à informação antes da mensagem ser transmitida, e sempre será no máximo igual a um.

No *PIC 16F876A*, o módulo que realiza a comunicação serial é conhecido como *USART*. Ele pode ser configurado para operar nos seguintes modos: Assíncrono (*full-duplex*), Síncrono Mestre (*half duplex*) e Síncrono Escravo (*Half Duplex*). A diferença entre os modos Síncronos Mestre e Escravo é que o Mestre gera o *clock* para os Escravos (MIYADAIRA, 2009).

No desenvolvimento abordaremos apenas a recepção de dados no modo assíncrono da comunicação serial *USART*. Neste modo, os pinos RC6 e RC7 deixam de operar como saída ou entrada comum, e passam a ser pinos de transmissão e recepção de dados serial respectivamente.

Para configurar a transmissão e recepção são utilizados dois registros *TXSTA* e *RCSTA*.

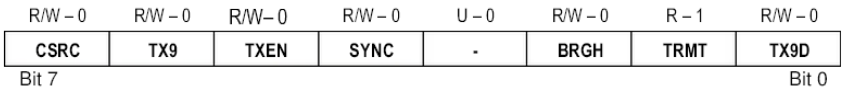


Figura 15. Registrador *TXSTA* (MICROCHIP, 2003)

CSRC: Seleção de *clock*.

Modo assíncrono: não tem efeito

TX9: Habilitação de transmissão de 9 bits.

1 = Transmissão de 9 bits

0 = Transmissão de 8 bits

TXEN: Habilitação de transmissão

1 = Transmissão ligada

0 = Transmissão desligada

SYNC: Seleção de modo *USART*

0 = modo assíncrono

BRGH: Seleção de taxa de transmissão

1 = alta velocidade

0 = modo baixa velocidade

TRMT: Status do buffer de transmissão

1 = buffer vazio

0 = buffer cheio

TX9D: 9° bit de transmissão, quando TXEN for usado

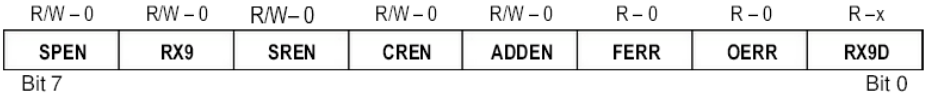


Figura 16. Registrador *RCSTA* (MICROCHIP, 2003)

SPEN: Habilitação da porta serial

1 = Habilitada

0 = Desabilitada

RX9: Habilitação de recepção de 9 bits

1 = Recepção de 9 bits

0 = Recepção de 8 bits

SREN: Habilitação de recepção única

Modo assíncrono: não tem efeito

CREN: Habilitação de recepção contínua

1 = Habilita recepção contínua

0 = Desabilita recepção contínua

ADDEN: Habilitação de detector de endereço

1 = Habilita detecção de endereço

0 = Desabilita detecção de endereço

FERR: Erro de enquadramento

1 = Houve erro de enquadramento (dado recebido com taxa diferente da configurada)

0 = Não houve erro de enquadramento

OERR: Erro de sobrescrita no buffer de recepção

1 = Houve sobrescrita no buffer de recepção

0 = Não houve sobrescrita no buffer de recepção

RX9D: 9° bit recebido, se RX0 = 1.

Para recepção de dados de forma serial, o circuito interno comporta-se como apresentado na figura abaixo:

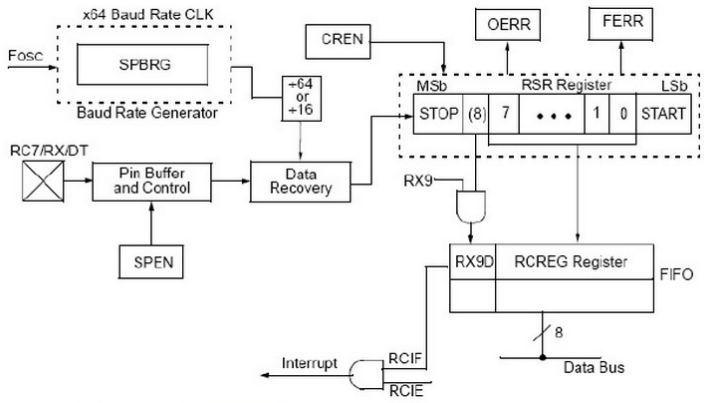


Figura 17. Circuito interno do módulo Serial *USART* (MICROCHIP, 2003)

Os dados recebidos pela porta serial são rotacionados e alojados no registro interno RSR. Uma vez verificado a coerência de start bit e stop bit, os dados são transferidos para o buffer de recepção, isto é, o registro RCREG e o *flag* RCIF é setado. Caso haja alguma incoerência com os bits de start e stop, o indicador de erro de enquadramento FERR é setado. O *flag* RCIF é zerado toda quando o registro RCREG é lido. Podem ser recebidos até dois bytes sem que faça a leitura do registro RCREG, sendo que o primeiro recebido fica armazenado em RCREG e o segundo em RSR, sendo transferido para RCREG assim que este é lido. Caso chegue um terceiro byte RSR é sobrescrito e o bit de erro OERR é setado (MICROCHIP, 2003). Para realizar recepção de dados enviados pela serial é necessário seguir os seguintes passos:

1 - Carregar SPBRG com o valor correto para a taxa de transmissão desejada;
2 - Habilitar comunicação serial síncrona: TXSTA, SYNC = 0 e RCSTA, SPEN = 1;
3 - Habilitar a transmissão: RCSTA, RXEN = 1.

Feito isso, basta monitorar o bit RCIF, que será setado toda vez que existir um dado válido no registro RCREG.

3.3.2 - MÓDULOS CCP *(Capture, Compare and PWM)*

No *PIC16F876A* existem dois módulos *CCP*, chamados *CCP1* e *CCP2*. Aqui será tratado o *CCP1*, mas todo o raciocínio é válido para *CCP2*. O módulo *CCP* realizar uma série de funções por hardware. Para exemplificar a versatilidade desse periférico, podemos citar como exemplos de sua utilização: geração de sinais de *PWM*, geração de sinais analógicos, medida de freqüência, medida de largura de pulso, dentre várias outras aplicações.

Existem três modos de operação: Captura, Comparação e *PWM*. O modo *PWM* utiliza o *TIMER 2* do PIC como base de tempo.

Nesse projeto será utilizado apenas o modo *PWM* dos módulos *CCP1* e *CCP2*.

3.3.2.1 - *TIMER 2*

No microcontrolador PIC, os *TIMERs* são aplicados fundamentalmente como temporizadores ou contadores. Eles podem ser configurados para serem incrementados por

um sinal de *clock* interno (ciclo de máquina), para o primeiro caso ou simplesmente por uma fonte externa acoplada a um pino específico para o segundo.

Quando configurados como temporizadores, como o próprio nome sugere, são usados para realizar contagem de tempo. No modo contador, eles contam a quantidade de vezes que um determinado evento ocorre, baseado na borda de subida ou descida do sinal externo.

Os *TIMERs* possuem um bloco *prescaler*, e no caso do *TIMER2*, possui um *postscaler*. O *prescaler* tem a função de definir o número de vezes que um determinado evento deve ocorrer, antes de o registro ser incrementado. Por exemplo, suponha que o módulo *TIMER* reconheça um evento na borda de subida do sinal de entrada; logo, se o *prescaler* associado for igual a 2, significa que a cada duas bordas de subida, o bloco fornece uma. Normalmente é o sinal de saída do bloco *prescaler* que incrementa o registro do *TIMER*.

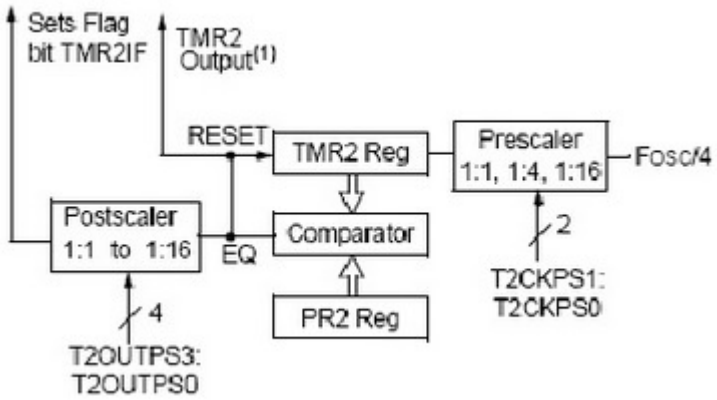


Figura 18. Diagrama interno do *TIMER 2* (MICROCHIP, 2003)

O bloco *postscaler* funciona como um contador de sinais enviados por um bloco comparador, o qual está presente no módulo *TIMER2* do PIC 16F876A, cuja função é comparar o valor de registro *TMR2* com um valor de referência *PR2*, e enviar um sinal para o bloco, caso seja verificada igualdade. O valor definido no *postscaler* determina a quantidade de sinais que o bloco deve receber, para que o *Flag bit* de interrupção seja setado (MIYADAIRA, 2009). Por exemplo, se o *postscaler* estiver configurado como 1:5, somente no quinto sinal enviado pelo bloco comparador o *Flag bit* de interrupção será setado.

Características do *TIMER2*:

- Opera como temporizador de 8 bits;
- A fonte de *clock* do sistema (Fosc/4);

- Utilizado pelo módulo *CCP* como base de tempo para o *PWM*;
- Esse módulo conta com um *period register PR2* de 8 bits que é utilizado como um limitador de contagem do *TMR2*. Este limitador é comparado com o valor de PR2 a cada ciclo de *clock*, e quando ambos forem iguais, o comparador envia um sinal que reinicia o *TMR2* e incrementa o *postscaler*.

3.3.2.2 - MODO *PWM*

O modo *PWM* permite utilizar sinais modulados em largura de pulso, que consiste em representar um valor pelo *duty-cycle* de um trem de pulsos de frequência fixa. Por exemplo, trabalhando com o *PWM* do *PIC 16F876A*, sua resolução máxima é de 10bits, ou seja, 1023 corresponde a 100% do *duty-cycle*. Assim, podemos determinar através da resolução, o *duty-cycle* correspondente a 5%, 30%, 99%, etc.

A maior parte das aplicações de *PWM* para microcontroladores se aproveita da propriedade da energia de um sinal retangular ser proporcional ao seu *duty*-cycle. No caso da implementação para o projeto, temos o exemplo do sinal *PWM* aplicado a um *LED*. Um *duty-cycle* de 100% irá acender o *LED* com potência máxima; já um sinal um sinal de 70% de tempo em alto, fornece ao *LED* 70% da potência máxima, e assim por diante.

O modo *PWM* do PIC necessita de uma base de tempo que dará a freqüência do sinal. O módulo *CCP* utiliza o *TIMER 2* para conseguir essa base. Isso pode ser observado no diagrama em blocos do *PWM* na figura abaixo:

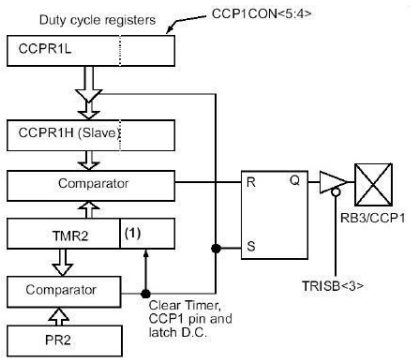


Figura 19. Diagrama em blocos do modo *PWM* (MICROCHIP, 2003)

A geração do sinal *PWM* se dá da seguinte forma. Cada vez que *TMR2* coincide com *PR2*, o pino *CCP1* é setado e *TMR2* é resetado. Isso nos dá a freqüência do sinal. O *duty-cycle*

é conseguido comparando o *CCPR1H* concatenados com dois bits de *latch* com *TMR2* concatenado com mais dois bits da pré escala ou gerados pelos ciclos Q. Quando há a coincidência, o pino *CCP1* é zerado. As concatenações nos dão 10 bits de resolução. O *duty-cycle* é configurado através de *CCP1L* e dos bits 4 e 5 de *CCP1CON*. Esse valor é atualizado em *CCPR1H* e nos bits de *latch* a cada período (MICROCHIP, 2003).

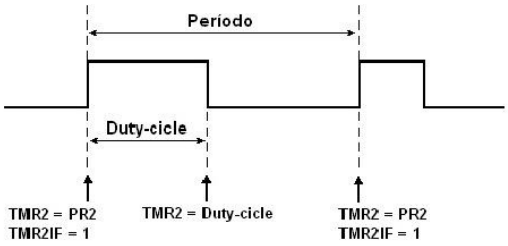


Figura 20. Sinal *PWM* (MIYADAIRA, 2009)

O período é o inverso da freqüência e é configura através de *PR2* e da pré-escala do *TMR2* e pode ser calculado através da Equação 2:

$$T_{PWM} = \frac{[(PR2)+1]*4*(TMR2PS)}{F_{OSC}} \qquad (2)$$

Onde:

Tpwm: período do *PWM*;
Fosc: freqüência do oscilador;
TMR2PS: fator de pré-escala do *TIMER2*.

Para encontrarmos o valor de *PR2* a partir de um dado valor de período.

$$PR2 = \frac{T_{PWM} * F_{OSC}}{4*(TMR2PS)} - 1 \qquad (3)$$

O *duty-cycle* por sua vez é configurado através de *CCPR1L* e dos bits 4 e 5 de *CCP1CON*. Geralmente se especifica o *duty-cycle* em porcentagem do tempo total. Assim, dado um *duty-cycle* em porcentagem (DC%), o tempo correspondente é encontrado pela Equação 4:

$$T_{DC} = \frac{DC[9:0] * (TMR2PS)}{F_{OSC}} \qquad (4)$$

Onde:

T_{DC} = Tempo de duty-cycle;

TMR2PS = fator de pré-escala do timer 2;

DC[9:0] = valor de 10 bits obtido acrescentando os bits 4 e 5 a esquerda de CCP1RL.

3.4 – DESENVOLVIMENTO DO CIRCUITO E *FIRMWARE*

3.4.1 – DESCRIÇÃO DO CIRCUITO

O protótipo proposto utilizará três opções de acionamento pela interface. Uma das opções se baseará no acionamento simples de 4 *LEDs* compondo a Iluminação Central do ambiente sala. Os *LEDs* são conectados aos pinos RA0 (pino 2) , RA1 (pino 3), RA2 (pino 4) e RA3 (pino 5) do microcontrolador *PIC16F876A*. Em cada pino conectado a um *LED*, é colocado em série, resistor de 330 Ohms para limitar a corrente em 13mA, já que cada pino configurado como saída suporta até 25mA de corrente.

A segunda opção proposta é a Iluminação Auxiliar. Este circuito é composto por 2 LEDs conectados a um driver transistorizado (BC548), assim podemos alimentar os LEDs sem risco de sobrecarregar a saída do PIC. Os LEDs possuem controle de intensidade luminosa através da modulação por largura de pulso (PWM). A base do transistor é conectado ao pino CCP1 (pino 13). Desta forma, saturamos e cortamos o transistor de acordo com o sinal PWM enviado para a saída CCP1.

De forma análoga ao sistema de Iluminação Central, temos o controle de velocidade do ventilador. No protótipo utilizamos motor cc retirado de um tocador de fitas como exemplo de aplicação. O controle de velocidade é feito a partir da saída CCP2 (pino 12) configurada como PWM no PIC. Assim, a partir do transistor (BC547) saturamos e cortamos de acordo com o sinal PWM acoplado à base. É importante lembrar que inserimos o diodo ligado paralelamente ao motor para evitar picos de tensão reversa sobre o transistor causado pelo descarregamento rápido das bobinas do motor.

Como forma de interface para a placa de controle, inserimos um display LCD 16x2. Utilizamos o modo de transferência por *nibble* afim de diminuir o número de entradas. Quando trabalhamos com a comunicação através de 2 nibbles colocamos dividimos o byte que desejamos em 2 *nibbles*. Então enviamos o *nibble* mais significativo (fazendo *enable* = 1 e *enable* = 0) e então enviamos o *nibble* menos significativo. Assim, as entradas de dados D7, D6, D5 e D4 da placa do LCD serão conectados ao pinos RB7 (pino 28), RB6 (pino 27), RB5 (pino 26) e RB4 (pino 25), respectivamente. As entradas de controle *Enable* (E) e RS, são conectadas, respectivamente, aos pinos RB1 (pino 22) e RB0 (pino 21) do PIC.

3.4.1.1 – *PROTEUS*

Ferramenta disponibilizada pela *Labcenter Eletronics*, O *Proteus* é um software utilizado para o desenho de circuitos analógicos e digitais, esquemas elétricos e placas de circuitos impressos. Possibilita a modelagem e simulação de circuitos e vêm com vários dispositivos comerciais em seus repositórios.

A partir deste software, desenvolveremos o circuito da placa controladora e simularemos as funcionalidades principais do projeto.

A figura abaixo mostra o circuito proposto implementado no *PROTEUS*.

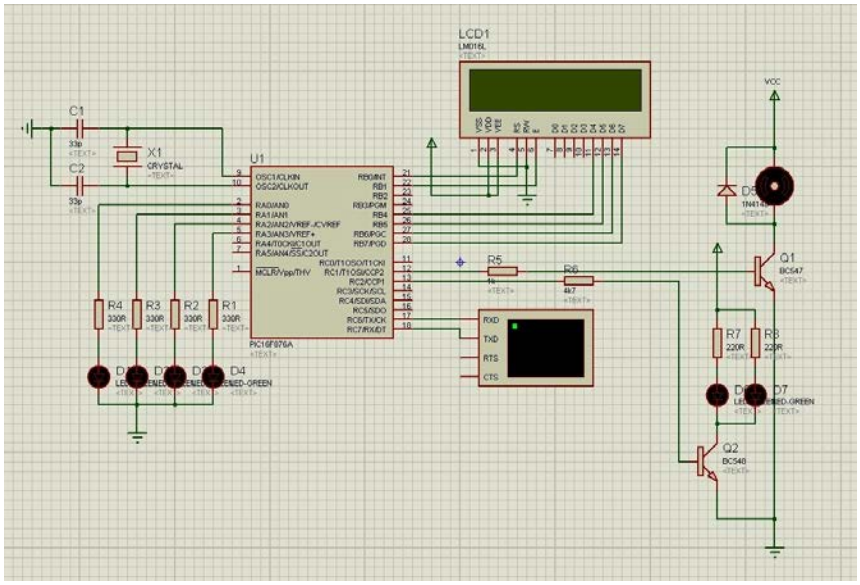


Figura 21. Circuito desenvolvido no *PROTEUS* (Própria Autoria)

Para realizar a simulação do circuito, o *PROTEUS* permite que seja carregado no microcontrolador o *firmware* a partir das propriedades do componente. Esse *firmware* deve ser um arquivo com a extensão ".hex" obtido através de um compilador.

Nos próximos tópicos serão detalhados o desenvolvimento do *firmware*.

3.4.2 – O *FIRMWARE*

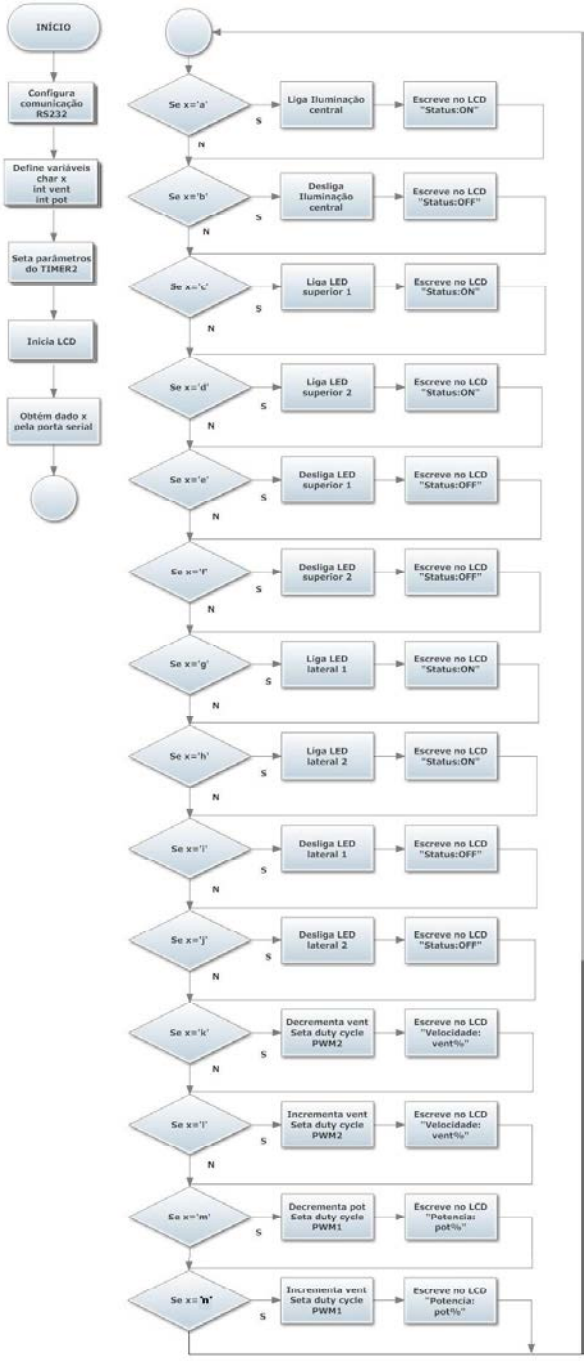


Figura 22. Fluxograma do *Firmware* (Própria Autoria)

Para o desenvolvimento do programa para o microcontrolador, utilizamos o *CCS* (*Custom Computer Services* – sigla adotada pelo desenvolvedor) que é um compilador em linguagem em C que oferece uma ferramenta completa de desenvolvimento e depuração de aplicações embarcadas em execução no PIC Microchip® MCUs e dsPIC® DSCs. Possui biblioteca de funções e arquivos de códigos para cada microcontrolador prontos para serem incluídos no programa.

Todo o Firmware deste projeto foi desenvolvido em C, o que facilita a manutenção e a portabilidade para outro microcontrolador.

Inicialmente configuramos os parâmetros essenciais para o funcionamento correto do controlador, como a comunicação serial e o *TIMER2* para aplicações de sinal *PWM* a partir dos módulos *CCP*. Para a comunicação serial *RS-232*, utilizamos a seguinte diretiva:

Sintaxe: #use rs232 (***options***)

Em *options*, setamos o *baud rate* para 9600bps, sem uso do bit de paridade e determinação dos pinos de transmissão (RC6) e recepção (RC7).

#use rs232(baud=9600, PARITY=N, xmit=PIN_C6, rcv=PIN_C7)

Para obter os parâmetros do *TIMER2* utilizados como base de tempo dos módulos *PWMs*, temos a Equação 2. Determinamos primeiramente o Prescaler 1:16 (TMR2PS = 16) utilizando cristal de 10Mhz. Desta forma, o sinal de clock (Fosc/4) será dividido por 16, sendo necessário 16 ciclos completos para que seja incrementado o TMR2. Assim, para um sinal PWM de freqüência 1250Hz, o valor de PR2 será dado por 124.

Portanto, usando a diretiva em C para configurar o *TIMER 2*:

Sintaxe: setup_timer_2 (***prescale***, ***PR2***, ***postscale***)

setup_timer_2(T2_DIV_BY_16,124,1);

Posteriormente, é configurado o módulo *CCP1* e *CCP2* para operar em modo *PWM*. Utilizamos a seguinte diretiva:

setup_ccp1(CCP_PWM);

```
setup_ccp2(CCP_PWM);
```

Declaramos uma variável caractere “x”, representando o dado de 8 bits recebido pelo pino de recepção serial (RC7), a qual será lida constantemente para determinarmos o comando aplicado a partir de testes condicionais *if* de acordo com fluxograma apresentado na figura. Desta forma, para acionarmos os *LEDs* da Iluminação central, cada pino definido como LED_s1, LED_s2, LED_l1 e LED_l2, poderemos colocá-los em nível lógico alto ou baixo utilizando as diretivas *output_high(pino)* ou *output_low(pino).*

Na Iluminação auxiliar e ventilador, Quando o módulo *CCP1* é colocado em modo *PWM*, é possível setar o *duty_cycle* a partir da diretiva *set_pwm1_duty(duty_cycle)*, na qual o valor do *duty_cycle* varia de 0 à 100, representando a porcentagem em nível lógico alto em relação ao período *PWM*. A função configura os registradores *CCPR1L* e os bits 4 e 5 de *CCP1CON* a partir do valor do *duty-cycle*.

3.4.3 – SIMULAÇÕES

Ao compilarmos o *firmware* no *CCS*, é gerado um arquivo .hex. Esse arquivo é carregado na propriedade do PIC do circuito desenvolvido no *PROTEUS*. Desta forma, o firmware é carregado no microcontrolador e pronto para simular o circuito.

O *PROTEUS* possui uma ferramenta chamada *Virtual Terminal* que simula uma comunicação serial tanto assíncrona como síncrona. Ela tem uma interface da qual podemos enviar dados a partir do teclado do computador e visualizar os dados recebidos. Na simulação feita, conectamos os pinos *RXD* e *TXD* do *Virtual Terminal* nos pinos 17 e 18 do PIC respectivamente, como foi mostrado na figura. Com essa ferramenta, simulamos a porta serial do módulo *Bluetooth* para verificar o funcionamento do circuito e do *firmware* do PIC.

Também foi verificado os sinais *PWM* gerados dos módulos *CCP1* e *CCP2* utilizando a ferramenta *Virtual Oscilloscope.* Essa ferramenta simula um osciloscópio, permitindo a visualização das formas dos sinais de entrada acoplados aos canais. A figura abaixo mostra a interface do osciloscópio mostrando os sinais *PWM* dos módulos *CCP1* e *CCP2*.

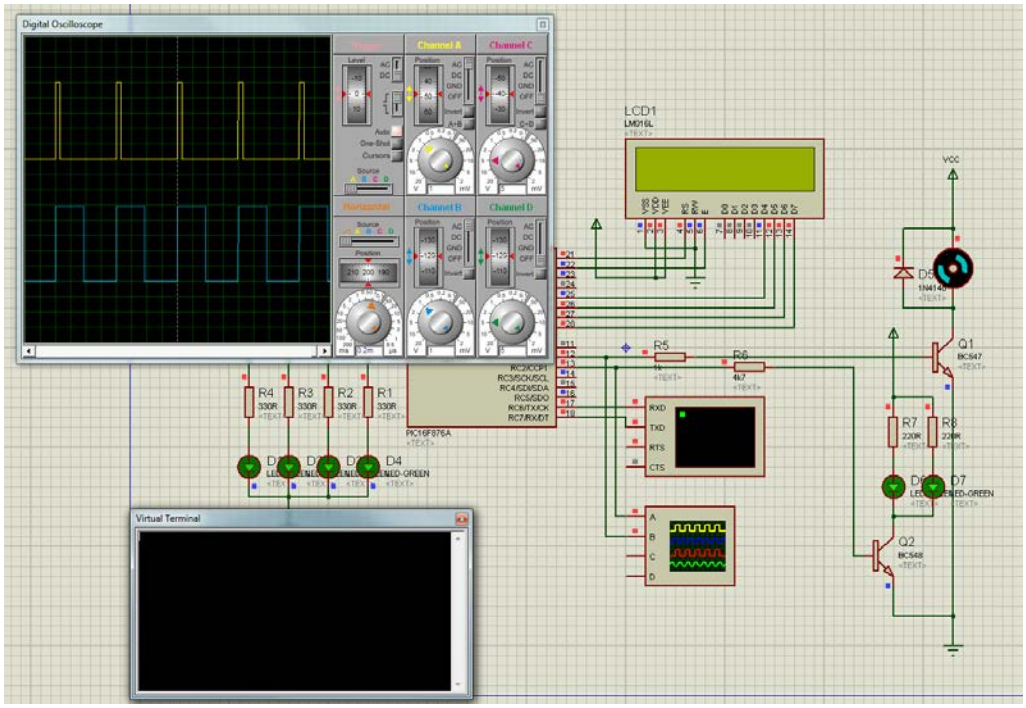


Figura 23. Interface de simulação do *PROTEUS* (Própria Autoria)

3.4.4 – CONFECÇÃO DA PLACA CONTROLADORA

Como já foi mencionado, o módulo *Bluetooth* possui a porta serial operando com nível 3.3Vdc. No entanto, o *PIC16F876A*, utiliza níveis de 5Vdc para a recepção serial. Neste caso, propomos a utilização de transistores para regular o nível de tensão de recepção no PIC, de 3.3Vdc do módulo *Bluetooth* para 5Vdc.

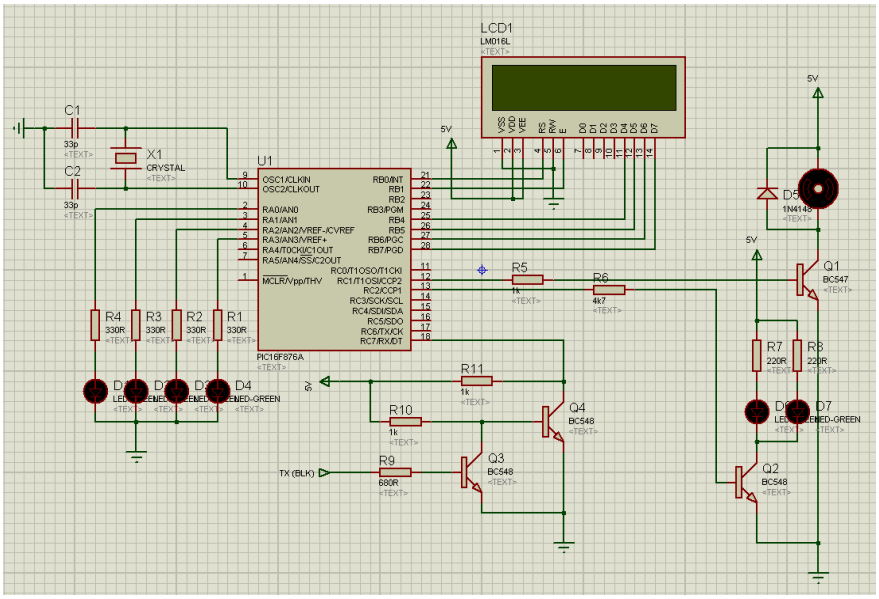


Figura 24. Circuito Final da placa controladora (Própria Autoria)

O circuito é montado sobre uma placa de fenolite ilhada feito a partir do processo de solda das trilhas.

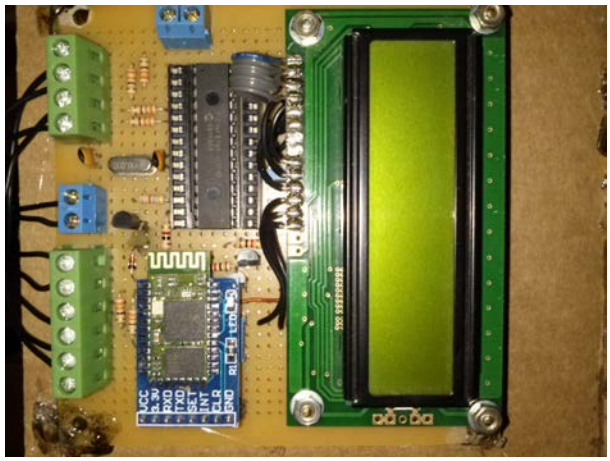


Figura 25. Protótipo da Placa controladora (Própria Autoria)

O módulo *Bluetooth* utilizado possui as configurações de transmissão e recepção serial padrões de fábrica dados por:

- velocidade de transmissão de 9600bps;
- sem bit de paridade;
- nome do dispositivo: *BOLUTEK*;
- senha de pareamento: 1234;

O último processo para a conclusão do módulo de processamento é a gravação do firmware dentro da memória flash do microcontrolador. Para tanto é necessária a utilização de um gravador compatível com o microcontrolador. No projeto proposto foi utilizado o gravador *PKBurnner*2 fabricado pela *Carelle Engenharia* que possui compatibilidade com *PicKIT2* da *Microchip*.

A partir do arquivo .hex obtido na compilação do firmware, utilizamos a plataforma *PicKIT2 Programmer* para gravar o firmware na memória flash do microcontrolador.

Figura 26. *PKBurnner2* (Própria Autoria)

4 – INTERFACE BASEADA EM APLICATIVO *ANDROID*

A interface de comunicação com o controlador foi criada baseado em um aplicativo desenvolvido a partir do software ***App Inventor*** sobre a plataforma *Android* da *Google*. Basicamente, o aplicativo instalado no smartphone, terá funções de acionamento dos *LEDs* e do ventilador com interface amigável.

4.1 – SISTEMA OPERACIONAL *ANDROID*

O *Android* é um sistema operacional móvel criado para *smartphones* e *tablets*. Foi incialmente desenvolvido pelo *Google*, com a compra da empresa *Android Inc*. em 2005. O sistema é baseado no *Linux*, com o objetivo de ser uma plataforma aberta, devido ao código livre (gratuito), permite a oportunidade de personalização das aplicações e componentes presentes em seu sistema, desenvolvimento rápido e moderno de aplicações corporativas, uma vez que sua plataforma é moderna e flexível. Essas características permitem que o sistema seja utilizado em aparelhos de várias marcas, o que o auxiliou em sua consolidação. Tudo isso tendo em vista o grande crescimento do uso de *smartphones* e *tablets*, sendo que atualmente, até setembro de 2012, aproximadamente 500 milhões de aparelhos ativados no mundo possuem o *Android*, representando uma parcela de 72,4% do mercado (GARTNER INC, 2012).

Como foi mencionado, o *Android* opera como base o sistema *Linux*. Além de permitir uma abstração entre o hardware e o software, ele é o responsável pelos principais serviços do sistema como gerenciamento de memória e gerenciamento de processos (drive de *USB*, áudio, *Bluetooth*, *Wi-fi*, câmera, etc). As bibliotecas C/C++ utilizadas pelo sistema atuam em diversas funções como implementação da biblioteca padrão do C com licença e otimização para dispositivos embarcados, bibliotecas para suporte a formatos de áudio, vídeo e imagens, gerenciador de acesso ao display, composição de camadas de imagens 2D e 3D, renderizador de fontes bitmaps e vetoriais e o banco de dados relacional SQLite (GESTWICKI, 2011).

No *Android*, as aplicações são escritas em *Java* e executam em sua própria máquina virtual, que por sua vez é executada em seu próprio processo no *Linux*, isolando-a de outras aplicações e facilitando o controle de recursos. Possuí ainda, uma framework de aplicações, que fornece todas as funcionalidades necessárias para a construções de aplicativos através das bibliotecas nativas, sendo a base do *App Inventor*.

4.2 – APP INVENTOR

O *App Inventor* foi originalmente criado no *Google Labs*, mas atualmente pertence ao *MIT Labs*, do *Massachussets Institute of Technology*. É desenvolvido em código aberto, gratuito, permitindo que qualquer pessoa possa criar um aplicativo para o sistema operacional *Android* através dele (WOLBER, 2011).

O *App Inventor* roda através de um navegador pelo endereço http://www.appinventor.mit.edu/, no qual dispõem-se de servidores para salvar os projetos para cada usuário criado e disponibiliza biblioteca de funções em blocos. O ambiente de desenvolvimento do *App Inventor* é destinado a facilitar a composição de aplicações envolvendo interfaces gráficas, sendo que o código de programação é baseado em *Java*.

4.2.1 – CONSTRUÇÃO DO APLICATIVO

Primeiramente, foi criado uma conta através da página inicial do ambiente de desenvolvimento e dado um nome ao projeto: *Autohome*. Após criado o projeto, temos a página *Component Designer.* Através dessa tela inserimos os componentes necessários para a aplicação proposta. Como mostra a figura, do lado esquerdo temos a guia *Palette.* Nesta guia estão presentes os componentes disponíveis a serem inseridos ao aplicativo. Inserimos os seguintes itens:

- *Botão Conectar*: conexão com módulo Bluetooth, pareamento;
- *Botão Sair*: desconectar módulo Bluetooth e sair do aplicativo;
- *Botão IlumCentral*: liga/desliga iluminação central;
- *Botão Sup1*: liga/desliga LED superior 1;
- *Botão Sup2*: liga/desliga LED superior 2;
- *Botão Lat1*: liga/desliga LED lateral 1;
- *Botão Lat2*: liga/desliga LED lateral 2;
- *Botão AuxMais*: incrementa valor relacionado ao LED auxiliar;
- *Botão AuxMenos*: decrementa valor relacionado ao LED auxiliar;
- *Botão VentMais*: incrementa valor relacionado ao ventilador;
- *Botão VentMenos*: decrementa valor relacionado ao ventilador;
- *BluetoothClient1*: insere biblioteca de funções para conexão Bluetooth e envio e recebimento de dados;

- *Labels*: caixas de textos para representação gráfica no aplicativo sem nenhuma função específica;
- *TableArrangement1*: permite criar formatos de exibição dos componentes inseridos, como em tabelas, divisão em linhas ou colunas, etc;
- *Image*: foi anexado logotipo da *UNESP*.

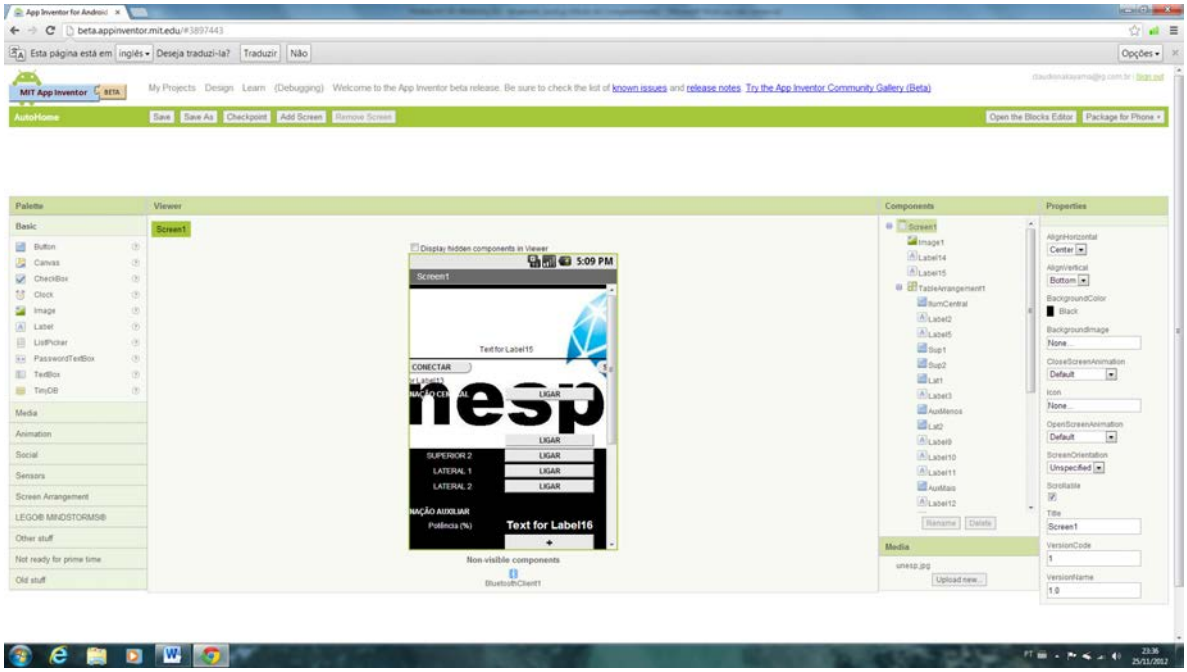


Figura 27. *Página Component Designer* (Própria Autoria)

Ao centro, em *Viewer*, mostra a representação da tela inicial do aplicativo. Nela arrastamos os componentes a serem inseridos. Cada componente inserido, será relacionado na guia *Components.*

Na guia *Properties,* podemos alterar as propriedades dos componentes, como fonte de texto, dimensões, parâmetros de segurança Bluetooth, etc. Em *Media* podemos importar arquivos de imagens, sons e vídeos para o desenvolvimento do aplicativo.

Após a inserção dos componentes e as devidas configurações das propriedades dos mesmos, utiliza-se o editor de blocos do App Inventor através do botão *Open the Blocks Editor* localizado no canto superior direito.

O Blocks Editor é utilizado para configurarmos o comportamento de cada componente inserido no *Component Designer.* A interface é bastante fácil de utilizar, pois todas as funções se baseiam em blocos, bastando arrastá-los para o plano do projeto.

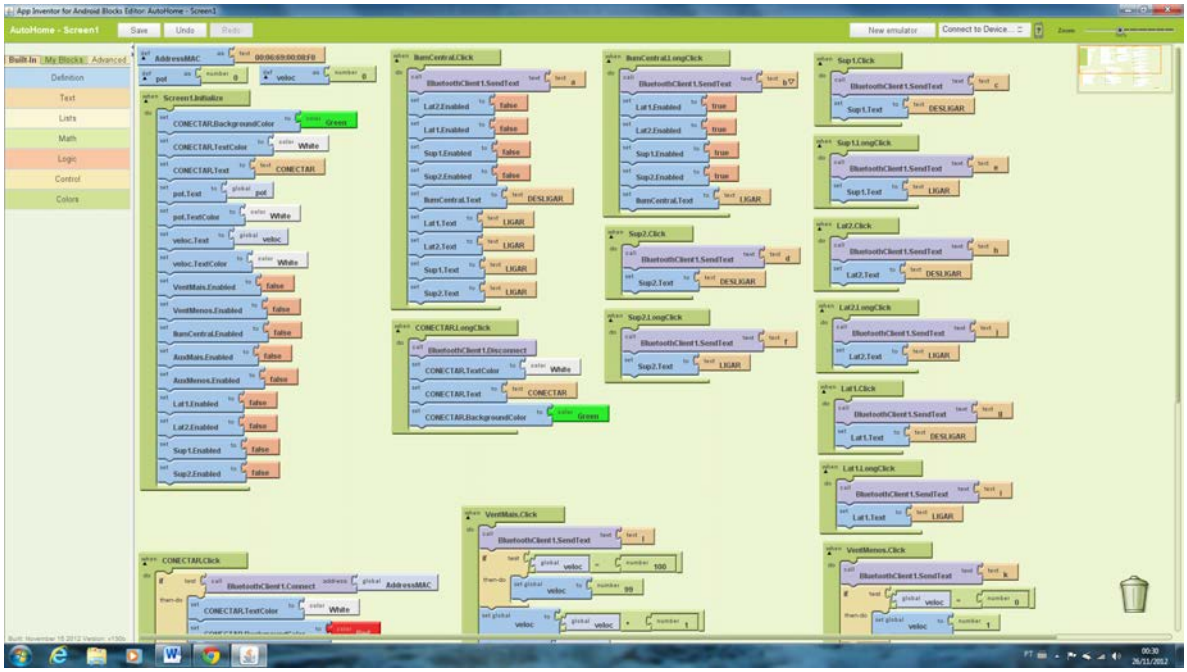


Figura 28. Página *Blocks Editor* (Própria Autoria)

Inicialmente, declaramos as variáveis a serem utilizadas no aplicativo. Através da guia *Built-In* localizada na parte esquerda da janela, temos a função *Definition.* Nela podemos inserir um bloco de definição de variável.

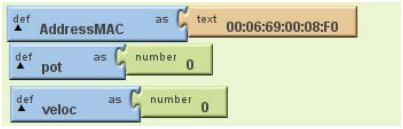
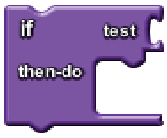


Figura 29. Definição de variáveis (Própria Autoria)

Definimos como variáveis globais através do bloco *def x as*:

- *AddressMAC :* Endereço *MAC* do módulo *Bluetooth* *"00:06:00:08:F0"* definido como texto (*text*) - o endereço foi obtido a partir do manual do dispositivo. Pode ser obtido também a partir do reconhecimento de um celular;
- *pot* : valor inicial definido como número (*number*) igual a zero para o parâmetro de potência do *LED* auxiliar;
- *veloc* : valor inicial definido como número (*number*) igual a zero para o parâmetro de velocidade do ventilador.

Inserimos blocos de texto e números e atribuímos a eles valores como mostra a figura. Para relacionarmos cada bloco com o outro, é feito o "*encaixe"* como em um *"quebra-cabeças",* desta forma representamos a atribuição da variável a um valor, relações de condição, etc. A forma como os blocos são dispostos para a montagem de uma rotina, é bastante semelhante à linguagem de programação (C/C++). A diferença é que no lugar de códigos, representamos graficamente em blocos (GESTWICKI, 2011).



= if (test) { function();}

Figura 30. Comparação entre linguagem de programação e de blocos do *App Inventor*
(Própria Autoria)

Em *Built-In* são relacionados apenas blocos básicos para definição de parâmetros (*Definition*), texto e formatação (*Text*), listas (*Lists*), operações matemáticas (*Math*), operações lógicas (*Logic*), testes condicionais e encerramento do aplicativo (*Control*) e parâmetros de cores (*Colors*).

Na guia *My Blocks* apresenta blocos específicos, onde estão relacionados todos os componentes criados na primeira etapa, como definições de comportamento. Isto é, para cada componente podemos definir diferentes tipos de comportamentos no aplicativo. Abaixo citamos os principais blocos de comportamento utilizados no projeto:

- *when Screenx.Initialize do*: ao ser inicializado a tela *x* (*screen x)* será realizado as ações dos blocos inseridos dentro do campo *do*;
- *when x.Click do:* quando o componente for pressionado por um pequeno instante (*click*) ele realiza as ações dos blocos inseridos dentro do campo *do*;
- *when x.LongClick do:* quando o componente for pressionado por um longo instante (*long click*) ele realiza as ações dos blocos inseridos dentro do campo *do*;
- *set x.Enabled to*: ativa ou desativa componente no aplicativo de acordo com a condição de verdadeiro (true) ou falso (false) respectivamente;
- *set x.TextColor to*: altera cor do texto do componente de acordo com o parâmetro de cor inserido no campo *to*;

- *set x.BackgroundColor to*: altera cor do plano de fundo do componente de acordo com o parâmetro de cor inserido no campo *to*;
- *call Bluetooth.Clientx.Connect address*: realiza conexão com dispositivo *Bluetooth x* a partir de um endereço MAC (*address*) do mesmo. Caso ocorra sucesso na conexão, ele retorna como condição verdadeira (*true*);
- *call Bluetooth.Clientx.disconnect*: encerra conexão com o dispositivo *Bluetooth x*;
- *call Bluetooth.Clientx.SendText text*: envia um texto para o dispositivo *Bluetooth x* conectado.

Observações: *x* representa valor ou parâmetro atribuído pelo desenvolvedor ou ordem do componente.

Basicamente, a lógica de funcionamento do aplicativo se resume na seguinte forma:

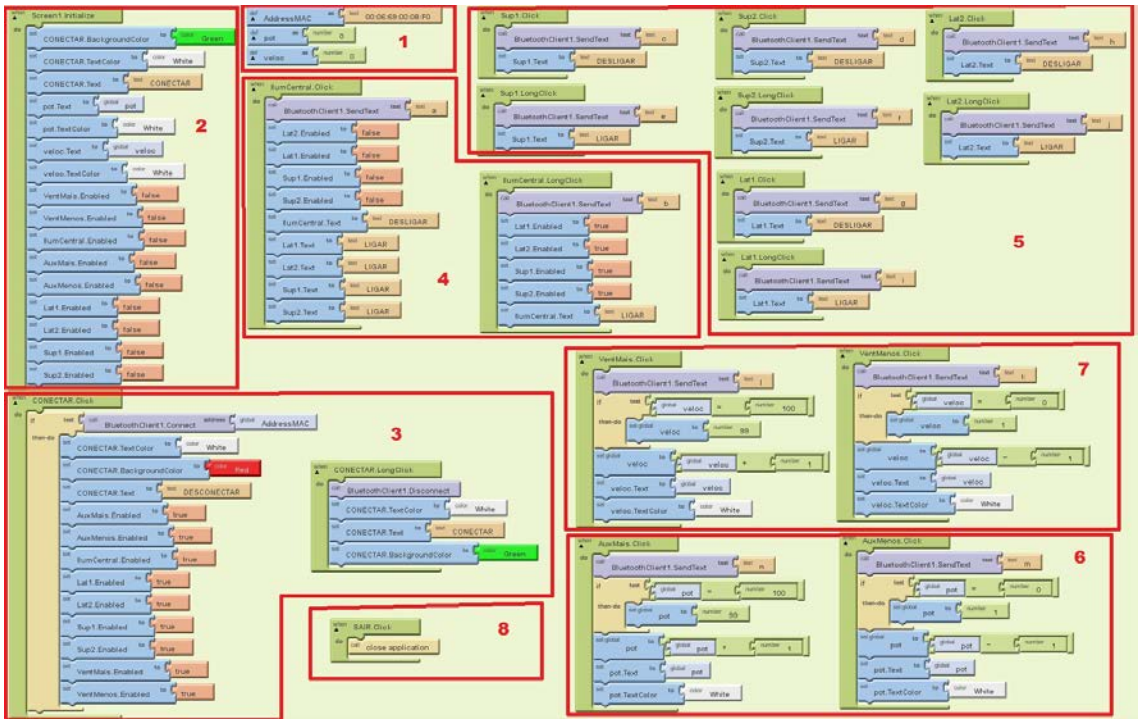


Figura 31. Visão geral dos conjuntos de blocos do aplicativo (Própria Autoria)

- Conjunto de blocos 1: atribuição das variáveis AddressMAC (endereço MAC do módulo Bluetooth), *pot* e *veloc*;

- Conjunto de blocos 2: inicializa tela1 (Screen1) mostrando o Botão CONECTAR com fundo verde, texto em branco e definido como *"CONECTAR"*. Os valores das variáveis *pot* e *veloc* são atribuídas como texto e cor Branca. Os demais botões estarão todos desativados;

- Conjunto de blocos 3: são definidos dois tipos de comportamentos para o botão CONECTAR. O primeiro é baseado no bloco *when CONECTAR.Click/do,* sendo que quando o botão CONECTAR for pressionado por um pequeno instante (click), ele irá realizar uma rotina de teste *if test/then-do.* Neste caso, através do bloco *call BluetoothClient1.Connect* o módulo *Bluetooth* é conectado a partir do endereço *MAC*, retornando uma condição verdadeira. Assim, são realizadas as ações dos blocos dentro do campo *then-do*. O botão CONECTAR mudará o fundo para vermelho e o texto passará a ser *"DESCONECTAR"*. Os demais botões serão ativados.

O segundo comportamento inserido para o botão CONECTAR é baseado no bloco *when CONECTAR.LongClick/do.* Quando for pressionado por longo tempo, será desconectado o módulo *Bluetooth* a partir do bloco *call BluetoothClient1.Disconnect;*

- Conjunto de blocos 4: O botão *IlumCentral* ao ser pressionado por um pequeno instante irá acionar a iluminação central, ativando todos os *LEDs* através do envio do caractere "a" através do bloco *call BluetoothClient1.SendText*. A iluminação central poderá ser desativada ao pressionarmos o botão por longo tempo. Será enviado o caractere "b". Ao ser ativado a Iluminação Central, todos os botões do conjunto de blocos 5 será desativado;

- Conjunto de blocos 5: de forma análoga ao botão *IlumCentral*, poderão ser ligados e desligados individualmente os *LEDs* da Iluminação Central a partir dos botões Sup1, Sup2, Lat1 e Lat2. Respectivamente, para acionar os *LEDs*, serão enviados os caracteres "c", "d", "g" e "h";

- Conjunto de blocos 6: neste conjunto mostra a forma como iremos aumentar ou diminuir o *duty cycle* do módulo *CPP1 PWM* do *PIC* para simular a Iluminação Auxiliar. Através do botão AuxMais, a cada click, será enviado o caractere "n". Desta

forma, o PIC incrementará o duty cycle a cada recepção do caractere “n”. Para visualizar o valor de *duty cycle* setado no aplicativo, atribuímos o valor incrementado para a variável *pot*. Para evitar que o valor seja maior que 100, setamos a variável *pot* como 99 sempre que a mesma for 100. Assim, o valor será incrementado posteriormente para 100, nunca maior que isso.

De forma contrária, podemos decrementar o *duty cycle* a partir do botão *AuxMenos*. É enviado o caractere “m” para o módulo *Bluetooth* sempre que o botão for pressionado por um pequeno instante. No caso de diminuirmos o valor de *pot*, este nunca será menor que zero. Pois foi implementado uma rotina em que será setado como 1 sempre que o valor de *pot* for zero. Assim, posteriormente será decrementado, mantendo-se sempre com o valor zero;

- Conjunto de blocos 7: da mesma forma que foi implementado para a Iluminação Auxiliar, a velocidade do ventilador poderá ser incrementada ou decrementada através dos botões *VentMais* e *VentMenos*, respectivamente. No primeiro , será enviado o caractere “l” a cada click no botão, e no segundo, será enviado o caractere “k”;

- Conjunto de blocos 8: ao pressionarmos o botão SAIR por um pequeno instante, será chamada a função *call close application,* fechando o aplicativo, retornando a interface comum do celular.

Com o programa pronto, instalamos o aplicativo no celular a partir do App Inventor na opção *“Package to Phone”*.

Figura 32. Visualização do aplicativo no celular (Própria Autoria)

5. RESULTADOS

Para exemplificar a aplicação, montamos uma maquete simulando uma sala de estar com os componentes da ILUMINAÇÃO CENTRAL (em vermelho), ILUMINAÇÃO AUXILIAR (em verde) e VENTILADOR (em azul), mostrado na Figura 33.

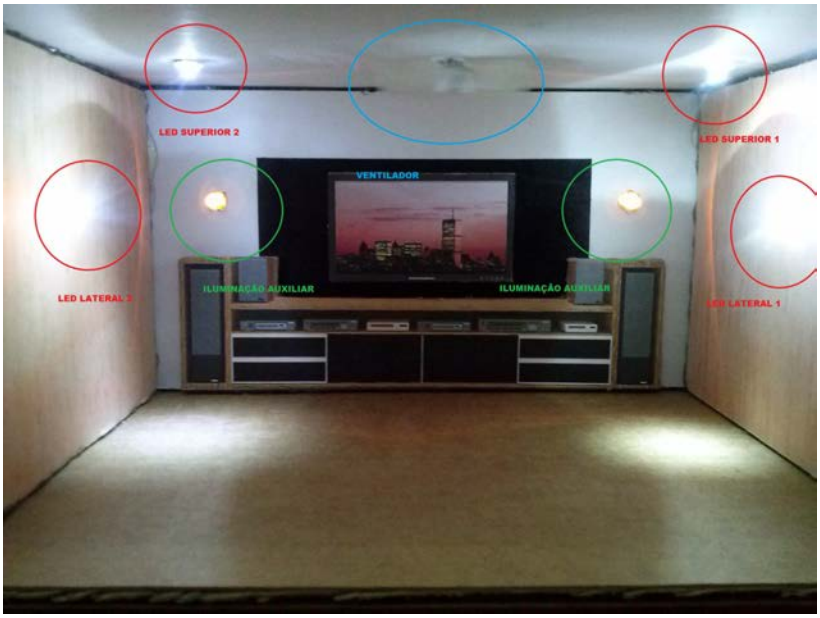


Figura 33. Maquete Sala de estar (Própria Autoria)

Utilizando o aplicativo *"Autohome"*, testamos todas as opções de acionamento. Notamos que a comunicação da placa controladora com o celular mostrou-se confiável e estável até uma distância de 10m entre eles, seguindo as especificações do módulo *Bluetooth* utilizado, apresentando resposta imediata aos comandos aplicados a partir do aplicativo. Verificou-se também o acionamento por PWM da ILUMINAÇÃO AUXILIAR e do VENTILADOR, mostrando a variação da intensidade da iluminação e velocidade de rotação do motor.

A cada acionamento aplicado, temos diferentes comportamentos da placa controladora, como mostra a Figura 34:

Figura 34. Placa controladora Ativa (Própria Autoria)

De acordo com *Firmware*, em cada acionamento é mostrado no *LCD*, o estado (*status*) da opção, se ligado é *ON* e desligado é *OFF*.

Ao sairmos do aplicativo, a comunicação *Bluetooth* com a placa controladora é desfeita, sendo que o módulo entra em modo de espera novamente. O acionamento é mantido mesmo com a desconexão da comunicação, por exemplo, quando acionamos a *Iluminação Central*, se sairmos do aplicativo, será desconectado a comunicação *Bluetooth* com a placa controladora, mas será mantida o último comando efetuado até que seja estabelecida nova conexão e processados novos comandos.

A partir das simulações do circuito via software (*PROTEUS*) e da realização de testes reais com a placa e a interface de controle desenvolvida, foi possível comprovar a eficiência da comunicação *Bluetooth* entre o celular e a placa controladora. Podemos afirmar que a implementação do módulo *Bluetooth* facilitou consideravelmente no desenvolvimento do projeto, já que o módulo utiliza padrão de comunicação serial para fazer interface com o microcontrolador PIC. Assim, não foi necessário nenhum comando específico para o funcionamento dele.

O custo para implementação da placa controladora está relacionado na Tabela 2:

Material	**Custo (R$)**
Módulo Bluetooth classe 2 modelo BLK	45,00
Microcontrolador PIC 16F876A	18,00
Componentes (Capacitores, transistores, resistores, etc)	15,00
Placa, solda, cabos, etc.	5,00
Total	**83,00**

Tabela 2 – Relação de Materiais/Custo

Analisando o custo final deste projeto, é interessante que ao considerarmos a produção em larga escala dessas placas, o custo unitário pode ser ainda menor. Logicamente que a placa desenvolvida nesse projeto é apenas uma representação criada para realizarmos testes do módulo *Bluetooth*, devendo ser considerada circuitos para acionamento de cargas em corrente alternada. Mas ainda, seria válido pensar em baixo custo do projeto, já que a interface criada é baseada em ambientes de desenvolvimento gratuitos de código livre.

6. CONCLUSÃO

Ao final deste trabalho, podemos afirmar que a utilização da comunicação Bluetooth apresenta muitas vantagens em uma aplicação desenvolvida para automação residencial. Além do baixo consumo de energia e do baixo custo, o módulo usa interface serial, permitindo simples implementação do circuito e do *firmware* do protótipo. E ainda, no desenvolvimento do aplicativo *Android*, não foi envolvido nenhum custo, já que foi utilizado o ambiente de desenvolvimento *App Inventor*, totalmente gratuito e com funcionalidades simples. Como atualmente, o uso de *smartphones* e *tablets* com sistema operacional *Android* tem aumentado ao longo desses últimos 2 anos, a tendência será utilizar cada vez mais aplicativos para uma infinidade de funções. Nesse contexto, o desenvolvimento do aplicativo *Android* para a interface de controle sem fio de ambientes residenciais, poderá se tornar uma eficiente ferramenta que alia as funcionalidades comuns de um *smartphone*, por exemplo, com aplicações voltadas para a automação residencial, podendo também entrar no contexto de aplicações para usuários com limitações físicas, simplificando o cotidiano dentro de uma residência.

Esse exemplo de projeto poderá ser implementado com novas versões da tecnologia *Bluetooth* (versão 3.0), com maior capacidade de transmissão, maior alcance, permitindo ampla integração da residência utilizando dispositivos de classe 1. Em futuros projetos, podemos incluir a leitura de sensores, como detectores de presença, sensores de temperatura, monitoramento de câmeras de segurança, com transmissão de dados em formato de vídeo e som, e ainda, na economia de energia.

REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM, E. Tecnologia Bluetooth. Disponível em: <www.infowester.com/bluetooth.php>. Acessado em 10 out. 2012.

ALIEVI, C. A. **Automação residencial com utilização de controlador lógico.** 2008. 84 f. Trabalho de Conclusão de Curso (Especialização) – Instituto de Ciências Exatas e Tecnológicas, Centro Universitário FEEVALE, Novo Hamburgo, 2008.

AURESIDE, Associação Brasileira de Automação Residencial. **A casa da Microsoft.** Disponível em: <http://www.aureside.org.br/temastec/default.asp?file=concbasicos03.asp>. Acessado em 02 out. 2012.

BLUETOOTH SIG; Specification of the Bluetooth System. Disponível em: <www.bluetooth.org>. Acessado em 11 out. 2012.

BOLUTEK – DATASHEET BC04B – 2010. Disponível em: <http://www.bolutek.com/Products_info.asp?id=278>. Acessado em: 13 out. 2012.

BOLZANI, C.; Desmistificando a Domótica. AURESIDE. 2007. Artigo disponível em: <http://www.aureside.org.br/artigos/default.asp?file=01.asp&id=74>. Acessado em: 20 out. 2012.

GARTNER INC. 2012. Disponível em: <www.gartner.com>. Acessado em 23 out. 2012.

GESTWICKI, P. - App inventor for Android with studio-based learning – 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=2037164>. Acessado em: 12 out. 2012.

LAYTON, J.; FRANKLIN, C.; HowStuffWorks. Como funciona o Bluetooth. Disponível em: <http://informatica.hsw.uol.com.br/bluetooth2.htm>. Acessado em: 10 out. 2012.

MARBURG, M. Modulação, Grupo de Teleinformática e Automação. 2008. URFJ. Artigo disponível em: <www.gta.ufrj.br >. Acessado em 10 out. 2012.

MICROCHIP TECNOLOGY INC; PIC 16F87XA DATA SHEET – 2003. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>. Acessado em: 12 out. 2012.

MIR, R. M.; ROBLIN, P.; Implementation and testing of a frequency hopping spread spectrum wireless link for data transmission between vehicle*; 1997. Columbus, Ohio, EUA*. Department of Electrical Engineering, The Ohio State University, *1997*.

MIYADAIRA, A. NOBORU MICROCONTROLADORES - PIC18: Aprenda e programe em linguagem em C. 1ª Edição – São Paulo 2009 Editora Érica.

MURPHY, E.; SLATTERY, C.; *Direct Digital Synthesis (DDS) Controls Waveforms in Test, Measurement, and Communications.* 2005. Artigo disponível em: <**http://www.analog.com/library/analogDialogue/archives/39-08/dds_apps.html**>. Acessado em: 20 out. 2012.

PALO WIRELESS; BLUETOOTH RESOURCE CENTER. 2012. Disponível em: <http://www.palowireless.com/bluetooth/>. Acessado em: 20 out. 2012.

PEREIRA, F. - Microcontroladores PIC - Técnicas Avançadas - 1ª Edição – São Paulo 2006 Editora Érica.

SACKS, A. G.; **Sistema de Gerenciamento de redes e processos através de computadores portáteis via *Bluetooth***. 2003. 98 f. Trabalho de Graduação (Graduação em Engenharia da Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2003.

TEZA, V. R.; **Alguns aspectos sobre a Automação Residencial - Domótica**. 2002. 106 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Santa Catarina, Santa Catarina, 2002.

WOLBER, D. - App inventor and real-world motivation -2011. Disponível em: <http://dl.acm.org/citation.cfm?id=1953329>. Acessado em: 12 out. 2012.

ANEXO

FIRMWARE - Código em C

```
#include <16f876A.h> //inclui biblioteca de definições do
dispositivo
#fuses HS,NOWDT,NOPROTECT //alta frequencia de clock,
Watchdog Timer desabilitado
                    //sem proteção do programa
#use delay(clock=10000000)//clock de 10MHz

#include "mod_lcd.c" //inclui biblioteca de funções para o
display LCD
#use rs232(baud=9600, PARITY=N, xmit=PIN_C6,
rcv=PIN_C7, ERRORS, DISABLE_INTS)
//setado baud rate de 9600 bps, sem paridade, pino c6 como
transmissão e pino c7
//como recepção, detecção de erros, e interrupções desabilitadas

#define LED_s1 PIN_A0 //definição do pino A0 como saída
para o LED sup1
#define LED_s2 PIN_A1 //definição do pino A1 como saída
para o LED sup2
#define LED_l1 PIN_A2 //definição do pino A2 como saída para
o LED lat1
#define LED_l2 PIN_A3 //definição do pino A3 como saída para
o LED lat2



unsigned char x;
int lamp, vent;

void main(void){

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_spi(SPI_SS_DISABLED); //desabilita comunicação SPI
  setup_timer_1(T1_DISABLED); //desabilita TIMER1
  setup_timer_2(T2_DIV_BY_16,124,1); //prescaler=16, PR2
=124, postscaler=1
  setup_ccp1(CCP_PWM); //configura CCP1 como PWM
  setup_ccp2(CCP_PWM); //configura CCP2 como PWM

  set_pwm1_duty(0);
  set_pwm2_duty(0);

  vent=0;
  lamp=0;

  lcd_ini();// Inicializa o LCD
  lcd_escreve ('\f'); // limpa o display
  lcd_pos_xy (5, 1);
  printf  (lcd_escreve, "AUTOHOME");
  lcd_pos_xy (3, 2); // Posiciona cursor no LCD
  delay_ms(100);
  printf  (lcd_escreve, "TG_NAKAYAMA");
  delay_ms(2000);

  lcd_escreve ('\f'); // limpa o display
  printf  (lcd_escreve, "   FEG UNESP");
  lcd_pos_xy (3, 2); // Posiciona cursor no LCD
  printf  (lcd_escreve, "aguardando...");

   while(1){

   x=getc();
   delay_ms(10);

     if(x=='a'){ output_high(LED_s1); //liga iluminação central
              output_high(LED_s2);
              output_high(LED_l1);
              output_high(LED_l2);

             lcd_escreve ('\f'); // limpa o display
             lcd_pos_xy (1, 1);
             printf  (lcd_escreve, "Ilum. central");
             lcd_pos_xy (1, 2);
             printf  (lcd_escreve, "status: ON");

             }

     if(x=='b'){ output_low(LED_s1); //desliga iluminação
central
              output_low(LED_s2);
              output_low(LED_l1);
              output_low(LED_l2);

             lcd_escreve ('\f'); // limpa o display
             lcd_pos_xy (1, 1);
             printf  (lcd_escreve, "Ilum. central");
             lcd_pos_xy (1, 2);
             printf  (lcd_escreve, "status: OFF");}

     if(x=='c'){ output_high(LED_s1); // liga LED sup1
             lcd_escreve ('\f'); // limpa o display
             lcd_pos_xy (1, 1);
```

```
        printf  (lcd_escreve, "Ilum. sup 1");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: ON");}

if(x=='d'){ output_high(LED_s2); //liga LED sup2
        lcd_escreve ('\f'); // limpa o display
        lcd_pos_xy (1, 1);
        printf  (lcd_escreve, "Ilum. sup 2");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: ON");}


if(x=='e'){ output_low(LED_s1); //desliga LED sup1
        lcd_escreve ('\f'); // limpa o display
        lcd_pos_xy (1, 1);
        printf  (lcd_escreve, "Ilum. sup 1");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: OFF");}

if(x=='f'){ output_low(LED_s2); //desliga LED sup2
        lcd_escreve ('\f'); // limpa o display
        lcd_pos_xy (1, 1);
        printf  (lcd_escreve, "Ilum. sup 2");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: OFF");}

if(x=='g'){ output_high(LED_l1); //liga LED lat1
        lcd_escreve ('\f'); // limpa o display
        lcd_pos_xy (1, 1);
        printf  (lcd_escreve, "Ilum. lat 1");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: ON");}

if(x=='h'){ output_high(LED_l2); //liga LED lat2
        lcd_escreve ('\f'); // limpa o display
        lcd_pos_xy (1, 1);
        printf  (lcd_escreve, "Ilum. lat 2");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: ON");}

if(x=='i'){ output_low(LED_l1); //desliga LED lat1
        lcd_escreve ('\f'); // limpa o display
        lcd_pos_xy (1, 1);
        printf  (lcd_escreve, "Ilum. lat 1");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: OFF");}

if(x=='j'){ output_low(LED_l2); //desliga LED lat2
        lcd_escreve ('\f'); // limpa o display
        lcd_pos_xy (1, 1);
        printf  (lcd_escreve, "Ilum. lat 2");
        lcd_pos_xy (1, 2);
        printf  (lcd_escreve, "status: OFF");}


if(x=='k'){ //decrementa velocidade do ventilador
   lcd_escreve ('\f'); // limpa o display
   lcd_pos_xy (1, 1);
   printf  (lcd_escreve, "Ventilador");
   if(vent==0){vent=1;}
   vent--;
   lcd_pos_xy (1, 2);
   printf  (lcd_escreve,"Velocidade: %d%%",vent);
   set_pwm2_duty(vent);}

if(x=='l'){ //incrementa velocidade do ventilador
   lcd_escreve ('\f'); // limpa o display
   lcd_pos_xy (1, 1);
   printf  (lcd_escreve, "Ventilador");
   vent++;
   if(vent==101){vent=100;}
   lcd_pos_xy (1, 2);
   printf  (lcd_escreve,"Velocidade: %d%%",vent);
   set_pwm2_duty(vent);}


if(x=='m'){ //decrementa potencia do LED aux
   lcd_escreve ('\f'); // limpa o display
   lcd_pos_xy (1, 1);
   printf  (lcd_escreve, "Ilum. Aux");
   if(lamp==0){lamp=1;}
   lamp--;
   lcd_pos_xy (1, 2);
   printf  (lcd_escreve,"Potencia: %d%%",lamp);
   set_pwm1_duty(lamp);}

if(x=='n'){ //incrementa potencia do LED aux
   lcd_escreve ('\f'); // limpa o display
   lcd_pos_xy (1, 1);
   printf  (lcd_escreve, "Ilum. Aux");
   lamp++;
   if(lamp==101){lamp=100;}
   lcd_pos_xy (1, 2);
   printf  (lcd_escreve,"Potencia: %d%%",lamp);
   set_pwm1_duty(lamp);}

}
}
```