DOUGLAS ALEXANDRE RODRIGUES DE SOUZA

Monitoramento de Sensores de Temperatura Utilizando Redes sem Fio - Bluetooth®

São José / SC
Setembro / 2010

DOUGLAS ALEXANDRE RODRIGUES DE SOUZA

Monitoramento de Sensores de Temperatura Utilizando Redes sem Fio - Bluetooth®

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina – Campus São José para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. Evandro Cantú, Dr. Eng.

Co-orientador:
Prof. George Henry Wojcikiewicz, Eng.

Co-orientador:
Fernando Souza de Andrade, Eng.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José / SC
Setembro / 2010

Monografia sob o título “Monitoramento de Sensores de Temperatura Utilizando Redes sem Fio - Bluetooth”, defendida por Douglas Alexandre Rodrigues de Souza e aprovada em 3 de setembro de 2010, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. Evandro Cantú, Dr.
Orientador

Prof. Mario de Noronha Neto, Dr.
IFSC – Campus São José

Prof. Ederson Torresini, M. SC.
IFSC – Campus São José

No século XI, vivia num mosteiro, perto do lago de Constança, Suíça, o monge Germano Contractus. Paralítico desde seu nascimento (18.7.1013), aos sete anos foi confiado pelos pais aos monges do Mosteiro de São Galo para ser instruído nas ciências e nas artes. Tempos depois foi admitido como monge no próprio mosteiro e ficou famoso como astrônomo, físico, matemático, teólogo, poeta e músico.

*Dia 15 de novembro de 1049, sofrendo de modo especial, rezou em sua cela, diante de um quadro de Nossa Senhora, por quem tinha uma devoção especial. Em seu coração, nasceu a prece: "**Salve, Rainha, Mãe de misericórdia, vida, doçura e esperança nossa, salve! A vós bradamos, os degredados filhos de Eva. A vós suspiramos, gemendo e chorando neste vale de lágrimas**".*

*Pouco depois, entrou em sua cela o irmão enfermeiro. Germano manifestou-lhe o desejo de ir à capela, dedicada a Nossa Senhora. Ali, continuou sua meditação e prece. Rezou: "**Eia, pois, advogada nossa, esses vossos olhos misericordiosos a nós volvei; e, depois deste desterro, mostrai-nos Jesus, bendito fruto do vosso ventre, ó clemente, ó piedosa, ó doce Maria!**"*

*A expressão "**sempre virgem**" – "**ó doce sempre Virgem Maria**" – foi acrescentada mais tarde.*

Fonte: Livro "Com Maria, A Mãe de Jesus", de Dom Murilo S.R.Krieger, scj (pág. 204)

Agradecimentos

Dedico meus sinceros agradecimentos a todos professores, colegas de trabalho e amigos que ajudaram a concluir este trabalho. Especialmente aos meus filhos Isabela e Gustavo pela alegria de me receber em casa depois de um dia cansativo de trabalho. Ao meu professor, amigo e orientador Evandro Cantú. Ao professor George Henry Wojcikiewicz co-orientador que trouxe até mim a idéia inicial do projeto. Aos professores de telecomunicações André Luiz Alves, Nilton F. Oliveira da Silva, Marcos Moecke, Pedro Armando da Silva Jr. que contribuíram em muito para execução do projeto. Aos colegas de trabalho laboratoristas Humberto, Ricardo e Gunter pela paciência e conhecimentos emprestados. Gostaria de citar também, de forma muito especial ao Dr. Fabrício de Souza Neves, reumatologista, que me trata de Artrite Reumatoide Juvenil – Doença de Still ao qual estou convalescido desde abril/2009 e a minha psicóloga Idáira Amoretti Santos que tenta organizar minha vida me motivando e me dando forças para continuar.

Resumo

Este trabalho apresenta uma proposta de utilização da tecnologia Bluetooth para transmissão de dados de grandezas físicas como pressão e temperatura a distâncias de até 10 metros. O estudo se dividiu em duas etapas. Na primeira etapa realizamos o estudo da pilha de protocolos de comunicação e envio de arquivos de texto da tecnologia IEEE 802.15 - Bluetooth. Nesse estudo também avaliamos a parte de redes sem fio utilizadas pela tecnologia em pequenas distâncias e a baixas taxas de transferências. A segunda etapa consistiu da implementação da parte eletrônica do sensor de temperatura, onde a informação é captada e transmitida para o computador e assim, enviada aos dispositivos Bluetooth encontrados.

Palavras chave: IEEE 802.15, Bluetooth.

Abstract

This paper presents a proposal for the use of technology Bluetooth for data transmission of physical quantities such as pressure and temperature at distances up to 10 meters. The study was divided into two steps. In the first stage we performed the study of the protocol stack communication and transmission of text files of IEEE 802.15 - Bluetooth. This study also evaluated the part of wireless networks technology used by small distances and low rates transfers. The second step was the implementation of part electronic temperature sensor, where information is captured and transmitted to the computer and thus sent to Bluetooth devices matches.

Keywords: IEEE 802.15, Bluetooth.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas

ACL – Asynchronous Connection-Less

ASC – Americam Standard Code

CAC – Chanel Access Code

CRC – Checagem de Redundancia Ciclica

CVSD – Continuos Variable Slope Delta Modulation

DCE – Data Communication Equipment

DIAC – Dedicated Inquiry Access Code

DTE – Data Terminal Equipment

EDR – Enhanced Data Rate

FEC – Forward Error Correction

FH-TDD – Frequency Hopping Time-Division Duplex

FM – Frequency Modulation

FSK – Frequency-Shift Keying

GFSK – Gaussian Frequency-Shift Keying

GIAC – General Inquiry Access Code

HCI – Host Controller Interface

HEC – Head Error Check

IAC – Inquiry Access Code

IEEE – Institute of Electrical and Electronics Engineers

ISM – Industrial Scientific Medicine

L2CAP – Logical Link Control and Adaptation Protocol

LAN – Local Área Network

LAP – Lower Address Part

LMP – Link Manager Protocol

MAC – Media Acess Control

OSI – Open Systems Interconnection

PCM – Pulse Code Modulation

PDA – Personal Digital Assistant

PIC – Programmable Interface Controller

PIN – Personal Identification Number

POP – Post Office Protocol

PPP – Point-to-Point Protocol

PSK – Phase-Shift Keying

RSSI – Received Signal Strength Indicator

SCL – Synchronous Connection-Oriented

SDP – Service Discovery Protocol

SIG – Special Interest Group

TCP/IP – Transfer Control Protocol / Internet Protocol

TCS – Telephony Control Specifications

UART – Universal Asynchronous Receiver/Transmitter

UHCI – Universal Host Controller Interface

USB – Universal Serial Bus

WPAN – Wireless Personal Area Network

1 Introdução

Com o avanço na utilização de sistemas de refrigeração e ar condicionado além da instalação dos aparelhos em locais de difícil acesso, como telhados, tetos de auditórios, alto de edifícios, entre outros, surgiu a necessidade de se aperfeiçoar o procedimento de avaliação da necessidade de manutenção dos aparelhos. A utilização de sensores que obtenham pressão e temperatura dos equipamentos tornaria a decisão de manutenção mais apropriada. Tal informação, captada por sensores, seria transmitida por um módulo Bluetooth para qualquer aparelho dispondo desta tecnologia. Assim, evitar-se-ia que os equipamentos fossem abertos, o que poderia ocasionar vazamento de **fluídos nocivos ao meio ambiente** como gases e óleo. Para resolver o problema, propomos o estudo de redes sem fio Bluetooth, o estudo de um *hardware* coletor e a implementação do módulo de *software* de envio. Este estudo será feito dentro do padrão IEEE 802.15.1 - Bluetooth, com especial atenção na pilha de protocolos utilizados no envio de arquivos de texto. O estudo do *hardware* coletor será feito de forma que os dados dos sensores sejam transferidos para um computador pessoal e disponibilizados para o envio pelo módulo Bluetooth. Por último, a implementação do *software* para envio das informações. Nesta fase, vamos implementar um *software* de envio de arquivos de texto, utilizando a mínima pilha de protocolos Bluetooth necessária. Utilizaremos o PC como *host*, uma vez que não se deseja, nesse momento, desenvolver um produto final, nem algo compacto, mas com o mínimo suporte de *hardware* necessário para se construir uma pilha de protocolos Bluetooth em *software*. Assim, toda pilha de protocolos Bluetooth será implementada e executada no PC conectado ao módulo Bluetooth através de USB.

1.1 Motivação

A motivação desse trabalho foi o desenvolvimento de um produto final que abrangesse várias áreas de conhecimento obtidas durante o curso, aliado a uma ótima idéia para projeto final que integra duas áreas de interesse do IFSC – Campus São José que é a de Telecomunicações e de Refrigeração e Ar Condicionado.

1.2 Objetivos

Realizar um estudo teórico sobre redes sem fio padrão IEEE 802.15 Bluetooth, focando no envio mensagens de texto através da mínima pilha de protocolos para a comunicação ponto a ponto. E ainda, um estudo da instalação e configuração do *hardware*, ou mesmo a implementação se necessário, para coletar as informações dos sensores e disponibilizá-las em formato apropriado para processamento através de uma interface com um computador pessoal. Por fim, implementar módulo de *software* necessário para obter os dados referentes à leitura dos sensores e o envio das informações por módulo Bluetooth conectado ao computador por meio de uma interface serial padrão USB.

1.3 Organização do Texto

No capítulo dois será dado um embasamento sobre a tecnologia de redes sem fio e o Bluetooth, apresentando seus principais conceitos e utilizações. No capítulo três, será explicado o desenvolvimento do protótipo e como funciona um *hardware* Bluetooth na prática. No quarto capítulo, será apresentado o *software* Bluetooth e suas funcionalidades. E, finalmente, no quinto capítulo, apresentaremos as conclusões, dificuldades encontradas e perspectivas gerais e novos projetos.

2 *Fundamentação Teórica*

Neste capítulo daremos uma visão da tecnologia de redes sem fio, seus usos e configurações, bem como o estudo mais aprofundado do próprio Bluetooth. O estudo aqui apresentado é amplamente disponível na literatura, sendo que a finalidade principal deste trabalho não é contribuir para a pesquisa na área de padrões *wireless*, mas apresentar uma comparação das principais características dos dois principais protocolos de comunicações de curto alcance terrestre que são o Wi-Fi e Bluetooth.

2.1 Redes sem Fio IEEE 802.11

Uma rede sem fio tem como principal característica fornecer uma infra-estrutura de rede onde não exista cabeamento, ou seja, a "ligação" entre os pontos se dá por meio aéreo dentro de um determinado espaço físico do qual chamamos de área de cobertura. Hoje em dia quando se fala nesse tipo LAN é indispensável falar sobre as redes de celulares com as redes 3G, e, principalmente, nas redes sem fio com a do padrão IEEE 802.11.

Nestas redes, encontramos três elementos principais para formação de uma LAN. O primeiro deles é chamado de ***host***, do inglês hospedeiro, sem fio, que como nas redes com fio (cabeadas), são os equipamentos na ponta que executam as aplicações. Este elemento pode ser um computador de mesa, um *laptop*, um *palmtop*, um PDA, ou mesmo um celular. Ainda, eles podem ou não ser equipamentos móveis. O segundo elemento chama se **enlace sem fio**, do qual um *host* faz uso para se conectar a um outro *host* por meio de uma comunicação sem a existência de um meio físico como cabos. O terceiro elemento é o que chamamos de **estação-base**, que diferentemente do *host* e do enlace sem fio, é responsável pelo envio e recebimento de dados de um *host* para a estação e da estação para um *host*. Sendo que ambos os *hosts* devem estar "associados" a essa estação-base. A estação-base é responsável pela coordenação da transmissão entre vários destes *host* sem fio devidamente associados com ela. Quando nos referimos a estarem associados, nada mais é do que dizer que esses *hosts* deverão estar dentro

da área de cobertura dessa rede através de um enlace sem fio até a estação-base. As torres de celulares e pontos de acesso sem fio 802.11 são exemplos de estações-base.

Para entendermos um pouco melhor sobre as redes sem fio, no caso a 802.11, vamos imaginar a nossa rede residencial cabeada, que dispõe de placas de rede do tipo Ethernet conectando os vários computadores em si. Ao substituirmos a placa Ethernet por uma placa de rede sem fio e alocação de um ponto de acesso substituindo o comutador Ethernet, estaríamos viabilizando uma rede sem fio sem nenhuma mudança significativa nas camadas rede ou acima dela.

As principais diferenças entre enlaces com ou sem fio são: **a redução da força do sinal**, as ondas eletromagnéticas sofrem atenuação quando atravessam algum tipo de material como uma parede por exemplo ou mesmo ao ar livre; **a interferência de outras fontes**, várias ondas de rádio quando transmitidas na mesma banda de freqüência provocam interferência uma nas outras, assim, um usuário de uma rede sem fio 802.11 que estiver falando por um telefone sem fio de 2,4 GHz pode esperar que nem a rede nem o telefone funcionem adequadamente; e **a propagação multivias**, onde parte das ondas eletromagnéticas reflete em obstáculos e no solo tomando caminhos de diferentes comprimentos entre o emissor e o receptor, causando um embaralhamento do sinal recebido.

Nos dias de hoje, podemos facilmente observar a larga utilização das redes sem fio em casa, em locais de trabalho, em rodoviárias, aeroportos, livrarias, cafés e até supermercados. As redes IEEE 802.11 consolidam-se como a principal tecnologia de acesso a Internet sem fio. Dentre as várias classes de tecnologias de LAN sem fio, o IEEE 802.11, mais conhecida como **Wi-Fi**, se destacou como a mais popular e utilizada. Existem vários padrões para o 802.11, dentre eles podemos mencionar 802.11a, 802.11b e 802.11g. A Tabela 2.1 apresenta as principais características desses padrões.

Tabela 2.1: Resumo dos padrões IEEE 802.11

Padrão	***Faixa de freqüência***	***Taxa de dados***
802.11b	2,4 - 2,485 GHz	Até 11 Mbps
802.11a	5,1 - 5,8 GHz	Até 54 Mbps
802.11g	2,4 - 2,485 GHz	Até 54 Mbps

O padrão Wi-Fi IEEE 802.11 é especialmente utilizado para a comunicação entre equipamentos diferenciados num raio de até 100 metros. Outros dois protocolos IEEE 802 nos padrões 802.15 – Bluetooth e 802.16 – WiMAX são padrões para se comunicar a distâncias mais curtas e mais longas, respectivamente.

2.2 Bluetooth IEEE 802.15.1

Bluetooth é um padrão para comunicação sem fio através de um sistema de rádio de curto alcance utilizado para interconectar com segurança os mais variados dispositivos eletrônicos como televisores, *home theaters*, *smartfones*, celulares, impressoras, fones de ouvido, *joysticks*, mouses, teclados, etc. Também pode ser utilizado para a comunicação entre computadores portáteis atuando como ponte entre outras redes como a Internet. Esta série de aplicações é conhecida como *Wireless Personal Area Network* (WPAN) que veremos mais adiante. Entre as principais características dessa tecnologia, destacamos a robustez, o baixo custo, o pequeno consumo de energia e a capacidade de trabalhar com transmissões de dados e de voz ao mesmo tempo.

2.2.1 Surgimento do Bluetooth

A idéia original do Bluetooth surgiu em 1994, quando a empresa Ericsson Móbile Communications começou a estudar e a desenvolver uma tecnologia que permitisse a comunicação entre telefones celulares e outros acessórios utilizando sinais de rádio de baixa potência para a substituição dos cabos. Com base no estudo dos mecanismos de comunicação em redes de telefonia celular, a empresa criou um sistema de rádio de curto alcance denominado MCLink. Mais tarde, a Ericsson percebeu que o projeto poderia dar certo já que a tecnologia era de implementação relativamente fácil e de baixo custo.

Em 1997, outras empresas começaram a interessar-se pelo projeto. Um ano mais tarde, foi criado o Bluetooth *Special Interest Group* (SIG), consórcio formado pelas empresas Ericsson, Intel, IBM, Toshiba e Nokia. Estas empresas, estavam entre as maiores empresas na área de telecomunicações, fabricação de PCs e no desenvolvimento de chips e processadores.

Com tal diversidade, o desenvolvimento do padrão tornou possível o uso e a interoperabilidade dos mais variados dispositivos dentro da tecnologia.

Em 1999 foi criada a primeira versão do protocolo Bluetooth. Já em 2000, outras empresas se juntaram ao grupo SIG: a 3Com, Agere (Lucent Technologies), Microsoft e Motorola. Desde março de 2002, o grupo de trabalho IEEE 802.15 aprovou sem grandes alterações e tornou o Bluetooth versão 1.2 um padrão oficial, o IEEE 802.15.1.

Figura 2.1 Logotipo Bluetooth

O nome Bluetooth foi uma homenagem a um rei dinamarquês chamado Harald Blåtand, também conhecido como Harald Bluetooth (Haroldo Dente-Azul). Um dos seus grandes feitos foi à unificação da Dinamarca, e, com alusão e este fato, o nome Bluetooth foi escolhido. Logo, uma tecnologia que unificasse os mais variados dispositivos das diversas empresas. O logotipo do Bluetooth que pode ser visto na Figura 2.1, é a reunião de dois símbolos nórdicos que correspondem às iniciais de Harald.

2.2.2 Potência e Alcance

O Bluetooth é baseado numa comunicação sem fio e baixo consumo de energia que permite a transmissão de dados entre os mais variados dispositivos dotados desta tecnologia. No entanto, uma combinação entre *hardware* e *software* se faz necessária para que a comunicação, através de radiofreqüência, permita que um dispositivo detecte o outro independente de suas posições, dentro de um limite de proximidade. A Tabela 2.2 mostra a potência e o alcance conforme as classes de seus transmissores.

Tabela 2.2: Potência e Alcance Máximos

Transmissor	***Potência Máxima***	***Alcance***
Classe 1	100 mW (20 dBm)	Até 100 metros
Classe 2	2,5 mW (4 dBm)	Até 10 metros
Classe 3	1 mW (0 dBm)	Até 1 metro

Baseado nas classes de potência e alcance, por exemplo, um dispositivo classe 3, poderá se comunicar com qualquer outro dispositivo de qualquer classe se a distância entre eles for de até 1 metro. Parece pouca distância, mas é o suficiente para conectar um celular no bolso da calça a um fone de ouvido. Portanto, vale a regra da menor distância de alcance entre dispositivos de classes diferentes.

2.2.3 Versões do Bluetooth

A tecnologia Bluetooth esta em constante evolução e faz com que suas especificações mudem com as novas versões.

Bluetooth 1.0: representa as primeiras especificações. Os fabricantes encontravam problemas na implementação e a interoperabilidade entre dispositivos Bluetooth desta versão.

Bluetooth 1.1: lançada em fevereiro de 2001, representava um avanço na solução dos problemas da primeira versão e foi implementado o suporte ao sistema RSSI.

Bluetooth 1.2: lançada em novembro 2003, tem como principal característica, taxas de transmissão máximas próximas a 1 Mbps. Melhor proteção contra interferências, processamento de voz aprimorado e suporte aperfeiçoado a scatternets que veremos posteriormente.

Bluetooth 2.0: lançada em novembro de 2004, traz mudanças com relação ao consumo de energia e aumento na velocidade de transmissão de dados chegando próximo dos 3 Mbps, além de correção de algumas falhas persistentes da versão 1.2.

Bluetooth 2.1: lançada em agosto de 2007, destacam-se as melhorias nos procedimentos de segurança, melhorias no consumo de energia e revisão com acréscimo de informação nas rotinas de *Inquiry*.

Bluetooth 3.0: versão lançada em abril de 2009, tem como principal característica taxas ainda mais altas na transmissão de dados chegando a 24 Mbps. Para atingir taxas tão elevadas, foram integrados padrões de comunicação IEEE 802.11.

A **versão 4.0** do **Bluetooth** foi anunciada para dezembro de 2009 e poderá estar disponível até o final de 2010. Entre outras inovações desta versão, estariam a sempre preocupante melhoria na relação de consumo de energia e a utilização de 128 bits de dados para segurança.

O fato de haver várias versões não implica que um dispositivo não possa funcionar com uma versão anterior, salvo algumas exceções, um dispositivo da versão 1.2 conectado a um da versão 2.0 terá sua taxa de transmissão limitada pelo primeiro. Portanto, as novas versões Bluetooth são, na grande maioria das vezes, totalmente compatíveis com versões anteriores.

2.2.4 Freqüência

Os dispositivos Bluetooth utilizam a banda de 2,4 GHz, que é uma freqüência de rádio aberta na maioria dos países e que nos EUA é conhecida como banda ISM. Nos países europeus e nos EUA, são atribuídos 79 canais de 1 MHz de largura, enquanto que apenas 23 canais são alocados na França, Espanha e Japão. Os canais são acessados usando uma técnica chamada *Frequency Hopping Spread Spectrum* (FHSS), com uma taxa de 1 Mbps e modulação GFSK. Esse esquema de comunicação utilizado pelo Bluetooth (FH), permite, mesmo utilizando uma freqüência aberta, garantir que o sinal Bluetooth não sofra ou gere interferências. O “salto de freqüência” consiste em acessar os canais de rádio diferentes de maneira muito rápida, fazendo com que a largura de banda fique muito pequena e diminuindo consideravelmente a chance gerar interferência.

Um dispositivo Bluetooth pode transmitir ou receber dados (modo *full-duplex*), a transmissão é alternada entre *slots* para transmitir e *slots* para receber dados utilizando um esquema chamado *Frequency Hopping Time-Division Duplex* (FH-TDD). Esses *slots* são canais divididos em períodos de 625 **µ**s (microssegundos), ou seja, em 1 segundo temos até 1600 saltos. O dispositivo denominado **mestre** inicia suas transmissões nos *slots* impares, enquanto que os **escravos** utilizam os *slots* pares. A mensagem poderá durar de 1, 3 ou 5 slots consecutivos. A Figura 2.2 exemplifica como é feita a multiplexação no tempo.

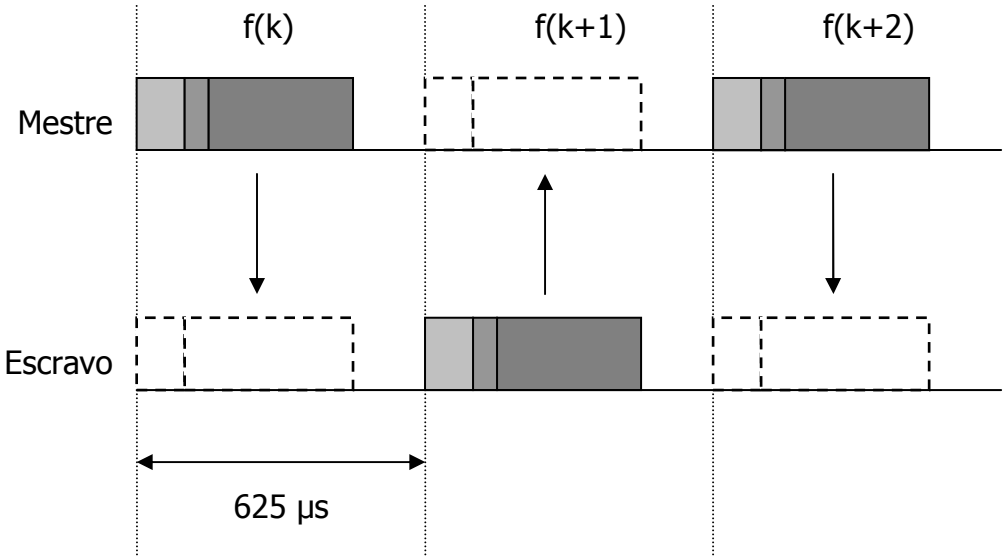


Figura 2.2 Multiplexação no Tempo.

O canal é dividido em *slots* de tempo com duração de 625 µs. No slot f(k) o mestre transmite seus pacotes. Em f(k+1), o escravo transmite seus pacotes e assim sucessivamente. Veremos mais adiante que para uma piconet com vários escravos, o mesmo raciocínio é aplicado.

2.2.5 Comunicação

O Bluetooth faz uso de dois padrões de comunicação, o padrão síncrono conhecido *Synchronous Connection-Oriented* (SCO) e assíncrono *Asynchronous Connection-L*ess (ACL). O *link* físico SCO é um *link* simétrico, ponto-a-ponto entre o dispositivo mestre e um escravo específico. É utilizado para a transmissão de dados contínuos, como por exemplo, a voz. Os *slots* são pré-reservados para cada dispositivo Bluetooth envolvido. Nesse padrão a retransmissão do pacote não é realizada. Quando ocorrer a perda de dados de voz, como no exemplo, o receptor acaba reproduzindo o som como um ruído. A taxa de transmissão de dados no modo SCO é de 432 Kbps, sendo de 64 Kpbs para voz.

Já o *link* assíncrono ACL, provê uma conexão assimétrica, ponto-multiponto e tira proveito dos *slots* não usados pelas conexões síncronas para transmissão dos dados. É

considerada uma conexão de chaveamento por pacotes. Ao contrário do SCO, o ACL permite o reenvio de pacotes de dados perdidos. É utilizado por aplicações que precisam garantir a integridade dos dados como a transferência de arquivos. A taxa de transmissão de dados no modo ACL pode alcançar 721 Kbps.

2.2.6 Modulação

O Bluetooth utiliza basicamente a modulação FSK. Esta técnica é semelhante ao FM, porém FSK desloca a freqüência através de dois pontos separados e fixos. A freqüência mais alta é chamada de marca, enquanto que a freqüência menor é chamada de espaço. Existe também um modo opcional chamado de *Enhanced Data Rate* (EDR) que usa modulação PSK na seqüência de sincronização, carga útil e *trailer* e GFSK no código de acesso e no cabeçalho. Para ambos os modos, a taxa de transmissão de símbolos é de 1 Msps (mega símbolos por segundo), sendo que, a taxa de transmissão de bits é aproximadamente 1 Mbps no modo FSK e de até 3 Mbps no modo EDR.

2.2.7 Número de Identificação Pessoal – PIN

Os dispositivos Bluetooth utilizam uma chave de acesso denominada *Personal Identification Number* (PIN) para autenticação. Sempre que um dispositivo deseja estabelecer uma conexão, um pedido de permissão deverá ser enviado. Se a chave de acesso digitada pelo usuário corresponder à chave de acesso do dispositivo detectado, a autenticação será realizada com êxito. Se não corresponder, a autenticação não acontecerá e um aviso de falha será enviado ao dispositivo solicitante.

2.3 Redes Bluetooth

Quando dois ou mais dispositivos compartilham o mesmo canal físico de uma conexão Bluetooth, eles formam uma rede chamada **piconet**. Um dos dispositivos Bluetooth será o mestre (*master*) , enquanto que os demais dispositivos, quando houver, tornar-se-ão escravos (*slave*). Até 7 escravos podem estar ativos numa piconet, e, ainda, muitos outros escravos podem se encontrar conectados em um modo estacionado (*park)*. Cada escravo estacionado recebe um endereço de 8 bits e leva apenas 2 ms para tornar-se ativo. Cabe ao mestre a tarefa de gerenciar a transmissão de dados na rede e o sincronismo entre os dispositivos.

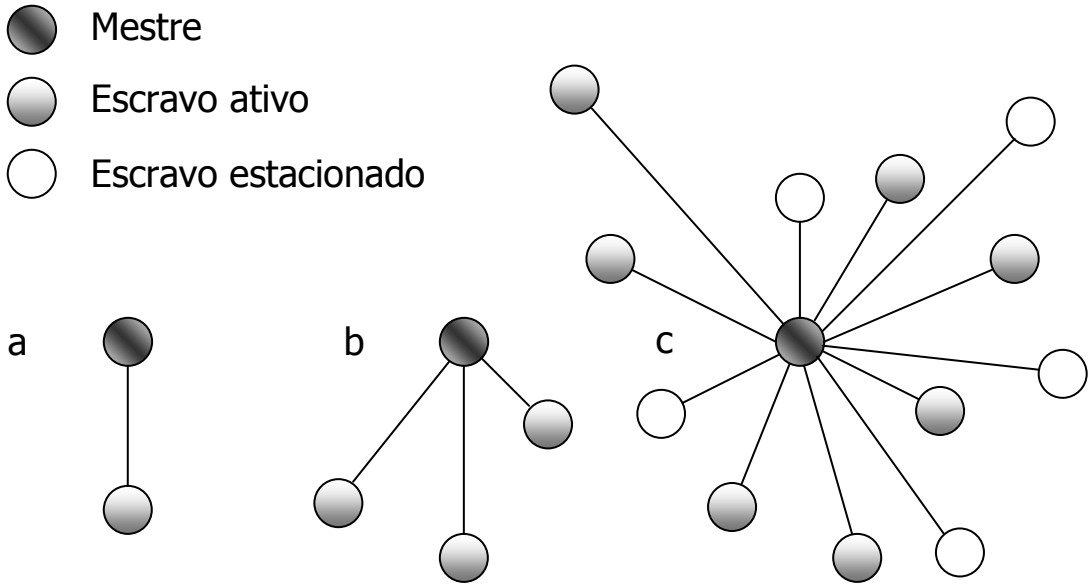


Figura 2.3 Configurações de Piconets.

Como podemos observar na Figura 2.3, as operações básicas dentro de um piconet são: a) mestre com apenas um escravo operando; b) mestre com múltiplos escravos operando; c) mestre com múltiplos escravos operando e outros escravos em modo estacionado. Em todas as configurações é possível observar que existe somente um mestre para um ou mais escravos. Isso se explica para que não existam duas piconets com a mesma seqüência de saltos de freqüência, mas o mestre de uma piconet pode ser escravo em outra piconet.

As Piconets reunidas, que são interligadas através de dispositivos Bluetooth que fazem parte de uma ou mais piconets formam uma **scatternet**. Nessas piconets deve existir sempre uma seqüência de saltos diferentes para que uma piconet não interfira na outra. À medida que novas piconets se juntem, à possibilidade de utilização da mesma freqüência aumenta e o desempenho pode cair. Na Figura 2.4 podemos observar a intrincada rede formada pela reunião de piconets.

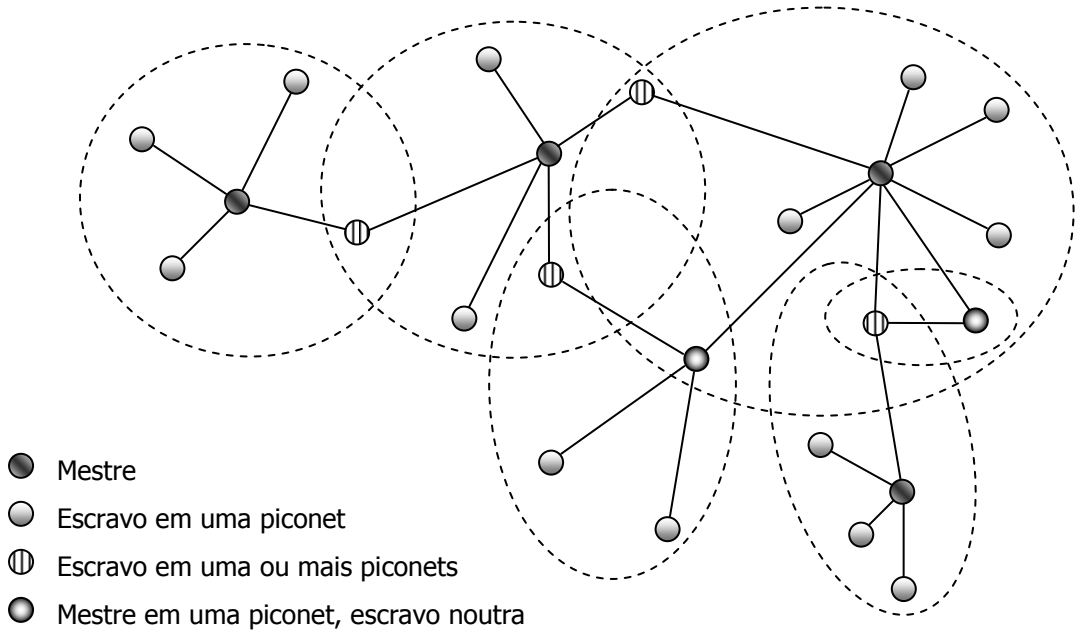


Figura 2.4 Complexa Configuração da Scatternet.

Essas “redes espalhadas” chamadas de scatternet quando interligadas, formam um sistema **ad-hoc** composto de múltiplas redes. As redes ad-hoc, não possuem um nó ou terminal encarregado de receber e encaminhar todas as comunicações, sendo que, qualquer terminal pode exercer essas funções. Portanto, uma scatternet é uma rede ad-hoc e todas são WPANs.

2.3.1 Formato Geral de Pacotes

O formato geral dos pacotes no modo básico do Bluetooth é dividido em três partes: 1) O *Access Code* (código de acesso), que pode possuir 68 ou 72 bits; 2) *Header* (cabeçalho) com 54 bits; 3) *Payload* (carga útil) com até 2745 bits.

Figura 2.5 Formato Geral do Pacote – Modo Básico.

O **código de acesso** constitui a primeira parte do pacote. Permite a sincronização das unidades da piconet, a compensação do componente DC e a identificação do dispositivo endereçado ou da piconet. O formato geral deste campo é apresentado na Figura 2.6.

Figura 2.6 Formato do Código de Acesso.

O preâmbulo (*preamble*) do código de acesso é composto por 4 bits numa seqüência fixa de zeros e uns, utilizada para a compensação DC. O campo de sincronismo (*sync word*) é formado por um código de 64 bits, derivado da parte baixa do endereço do dispositivo denominada LAP (*Lower Address Part*). Quando o código de acesso é seguido por um cabeçalho (*header*), o campo *trailer* é formado por 4 bits (zeros e uns alternados) e é enviado a fim de melhorar ainda mais a compensação DC.

Os tipos de código de acesso são:

CAC – *Chanel Access Code* (código de acesso ao canal): Identifica a piconet de forma única e é incluído em todos os pacotes transmitidos através do canal. O campo de sincronismo do CAC é derivado do LAP da unidade mestre.

DAC – *Device Access Code* (código de acesso ao dispositivo): É utilizado para o procedimento de *paging*, que será descrito mais adiante no que trata da parte de

estabelecimento de conexão. O campo de sincronismo do DAC é derivado do LAP do dispositivo que esta sendo acessado.

IAC – *Inquiry Access Code* (código de acesso de interrogação): Utilizado quando o dispositivo mestre deseja descobrir quais os dispositivos Bluetooth estão presentes na sua área de cobertura. Ainda, esta interrogação pode ser geral **GIAC** – *General Inquiry Access Code*, ou dedicada **DIAC** – *Dedicated Inquiry Access Code*. Enquanto o GIAC é direcionado a todos os dispositivos Bluetooth, o DIAC é endereçado a um subgrupo de dispositivos Bluetooth que possuem alguma característica em comum.

A segunda parte do pacote é o **cabeçalho** (*header*). Por utilizar um código corretor de erro chamado *Forward Error Correction* (FEC) de 1/3, somente 18 bits dos 54 são significativos. Os 3 primeiros bits endereçam os escravos na piconet (limitando em sete escravos). Seguem 4 bits do tipo de pacote e enlace. Em seguida vem um bit de controle de fluxo que indica *buffer* cheio em enlaces ACL, outro bit de confirmação e um outro de controle de seqüência. Por fim, 8 bits para checagem de erro. O restante dos bits é utilizado pelo FEC permitindo uma eventual correção de erros do cabeçalho no receptor. A Figura 2.7 dá uma visão geral do cabeçalho do pacote.

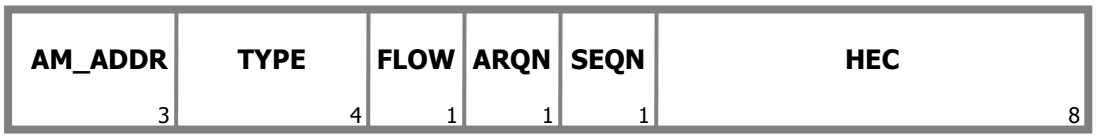


Figura 2.7 Formato do Cabeçalho.

AM_ADDR: É composto de 3 bits únicos para cada escravo ativo na piconet. O endereço “000” é utilizado para mensagens de *broadcast*. O dispositivo mestre não possui endereço próprio pois todo pacote em um piconet é trocado entre o mestre e um escravo que possui AM_ADDR.

TYPE: São atribuídos 4 bits para identificar ao todo 16 tipos de pacote, sendo que a interpretação do seu significado depende se a conexão é SCO ou ACL.

FLOW: Bit utilizado para o controle de fluxo em conexões ACL. O valor 0 (STOP) indica que a transmissão deve ser interrompida temporariamente e o valor 1 (GO) indica que a transmissão pode prosseguir.

ARQN: Esse bit indica sucesso quando o valor for 1 (ACK) ou falha valor 0 (NAK) no envio de um pacote ACL de informação com CRC. O não recebimento de um ACK indica falha no recebimento de um pacote.

SEQN: É o bit que determina a seqüência do pacote de informação com CRC. O método utilizado é o da alternância, assim, o receptor pode identificar as retransmissões de pacotes devido à perda do ACK.

HEC: Este campo é composto por 8 bits, utilizado com verificador de erro de cabeçalho enviado.

A última parte do pacote básico Bluetooth e o da carga útil (*payload*). De modo geral, quando o pacote não é de controle ele pode ser de voz , dados ou ambos. Quando forem dados, ele deve ficar dividido entre um campo de cabeçalho, um corpo de mensagem e um código CRC.

2.3.2 Tipos de Pacotes

Como visto no item 2.3.1 o tipo de um pacote é representado pelo campo TYPE do cabeçalho do pacote e está diretamente associado ao tipo de conexão utilizada, síncrona SCO ou assíncrona ACL. Os pacotes podem ser divididos em 3 grupos, os pacotes SCO, os pacotes ACL ou os pacotes de ambos os tipos de *links*.

Pacotes comuns:

ID: Pacote de identidade utilizado nas rotinas de interrogação, resposta e *paging*. Contém apenas o código de acesso que pode ser do tipo DAC ou IAC.

NULL: Numa referência a nulo, é utilizado para reportar o sucesso de uma transmissão prévia (ARQN), ou o estado do *buffer* de recepção (FLOW). É utilizado quando não há dados a serem transmitidos e é constituído pelo CAC e pelo cabeçalho do pacote.

POLL: Pacote de *polling*. Verifica se existe algo a ser transmitido. O dispositivo escravo ao receber um pacote desse tipo, deve responder obrigatoriamente a ele, mesmo que não haja nada para ser transmitido. Esse pacote é formado pelo CAC e pelo cabeçalho somente.

FHS: Utilizado para controles especiais a fim de responder a pacotes de interrogação e de *paging*, para informar o endereço do mestre e, ainda, fornecer o AM_ADDR dos escravos. O campo *payload* dos pacotes FHS são divididos em vários campos menores e ocupam apenas um *slot* de tempo.

Pacotes SCO: Os pacotes desse tipo não possuem CRC e não são retransmitidos. É especialmente utilizado para a transmissão de voz a 64 Kbps.

HV1: Transporta 10 bytes de informação enviados com FEC 1/3 e ocupa 30 bytes. Pacotes desse tipo devem ser enviados a cada 2 *slots* de tempo (TSCO=2).

HV2: Transporta 20 bytes de informação enviados com FEC 2/3 e ocupa 30 bytes. Esses pacotes devem ser enviados a cada 4 *slots* de tempo (TSCO=4).

HV3: Transporta 30 bytes de informação sem FEC. Pacotes desse tipo devem ser enviados a cada 6 *slots* de tempo (TSCO=6).

DV: Pacotes deste tipo podem transmitir dados e voz conjuntamente. O *payload* é dividido de forma que 80 bits são para voz, sem FEC, e outros 150 bits de dados codificados com FEC 2/3 e com CRC.

Pacotes ACL: Os pacotes desse tipo são utilizados para envio de dados sem restrições de tempo.

DM1: Pacote somente de dados. Pode conter até 18 bytes de informação incluindo o cabeçalho do *payload* e 2 bytes de CRC devidamente codificados com FEC 2/3. Este tipo de pacote ocupa somente um *slot* de tempo.

DH1: Idem ao DM1, só que não é codificado com FEC. Possui até 28 bytes de informação e ocupa um *slot* de tempo.

DM3: Similar ao DM1. Pode conter até 123 bytes de informação, incluindo 2 bytes de cabeçalho, outros 2 bytes de CRC e codificado com FEC 2/3. Pode ocupar até 3 *slots* de tempo.

DH3: Idem ao DM3, só que não é codificado com FEC. Pode conter até 185 bytes de informação e pode ocupar até 3 *slots* de tempo.

DM5: Similar aos pacotes DM1 e DM3. Pode conter até 226 bytes de informação, com 2 bytes de cabeçalho, outros 2 bytes de CRC e é codificado com FEC 2/3. Pode ocupar até 5 *slots* de tempo.

DH5: Idem ao DM5, só que não é codificado com FEC. Pode conter até 341 bytes de informação e pode ocupar até 5 *slots* de tempo.

AUX1: Similar ao DH1, sem FEC e sem CRC. Pode conter até 30 bytes de informação e ocupa apenas 1 slot de tempo.

2.3.3 Áudio

A codificação de áudio no Bluetooth, interface aérea, pode utilizar uma taxa 64 Kbps com modulação log *Pulse Code Modulation* (PCM) formato lei-A ou lei-μ, ou 64 Kbps com modulação *Continuos Variable Slope Delta Modulation* (CVSD). Esse último formato, aplica um algoritmo de modulação delta adaptativa com *companding* silábica. A codificação de voz sobre a interface de linha é projetada para ter uma qualidade melhor ou igual a 64 Kbps PCM log. A Tabela 2.3 apresenta um resumo de esquemas de codificação de voz suportados na interface aérea.

Tabela 2.3: Esquemas de Codificação de Voz

Codecs de Voz	
Linear	CVSD
8-bit logarítmica	Lei-A Lei-μ

2.4 Estados de Operação

O dispositivo Bluetooth pode estar em dois estados principais de operação denominados de espera (*standby*) e de conexão (*connection*). O estado de *standby* é o estado padrão de baixo consumo de energia, onde apenas o relógio nativo do dispositivo permanece ativo. No estado de conexão o dispositivo Bluetooth pode interagir diretamente com outros dispositivos. A Figura 2.8 mostra o diagrama de estados de um controlador Bluetooth.

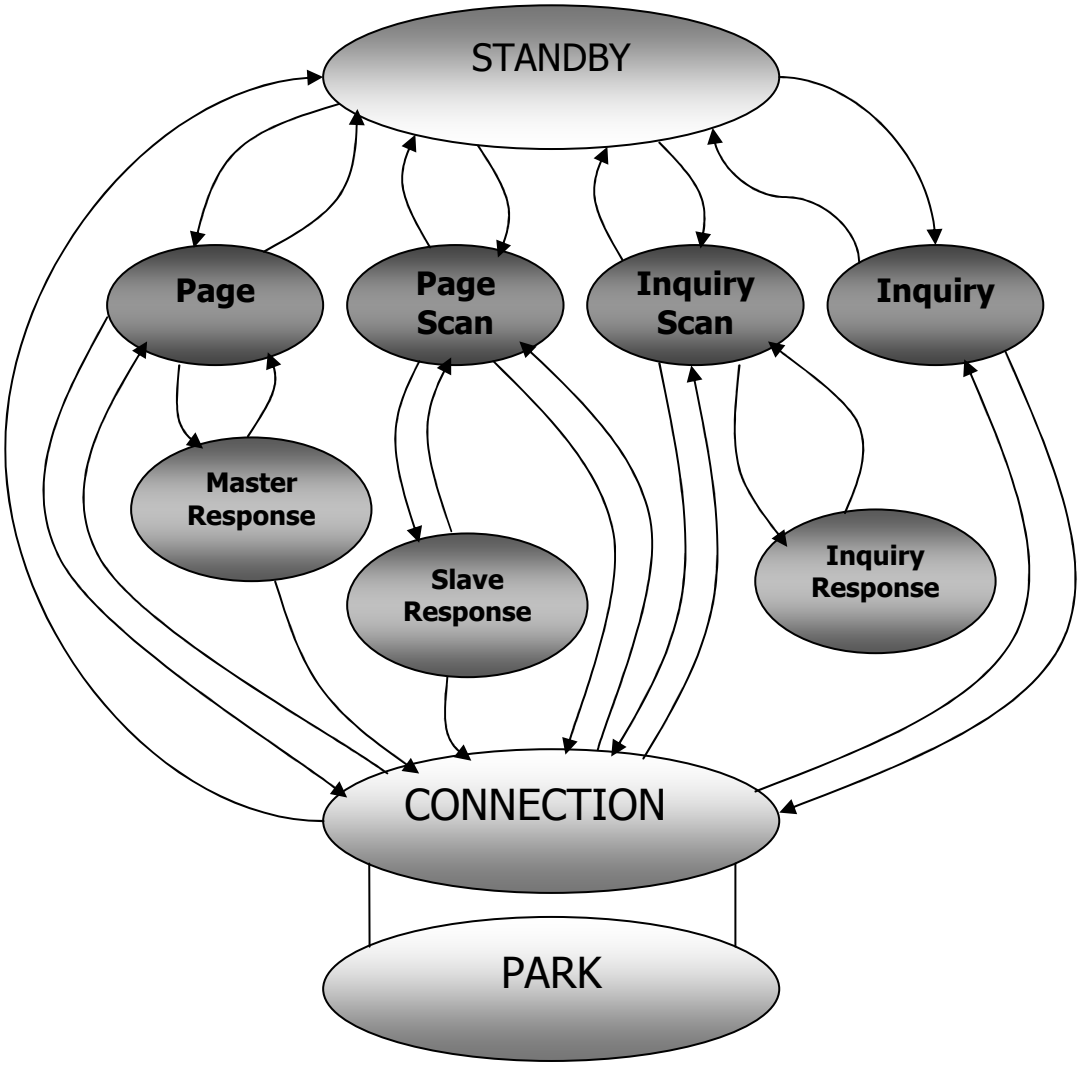


Figura 2.8 Diagrama de Estados de Conexão.
[Adaptado de BLUETOOTH, 2010]

O estado de conexão se divide em quatro modos: ativo, *sniff*, *hold* e *park*. No modo **ativo** o mestre manda pacotes de sincronização periodicamente e os escravos recebem e transmitem pacotes. No modo ***sniff*** a transmissão entre mestre e escravo se dá através de *slots* de tempo específicos. No modo ***hold*** o escravo fica sem receber e enviar pacotes por um tempo pré-determinado. Depois pode sincronizar-se novamente com o mestre e espera por instruções. No modo ***park*** o escravo também não transmite nem recebe pacotes, mas mantém o sincronismo com o mestre e recebe um endereço global de estacionamento (*parked address*), onde o dispositivo passa a usar um endereço global em vez do seu endereço físico. O modo *park* é de baixíssimo consumo de energia e em tempos regulares, o dispositivo escuta o canal para verificar eventuais transmissões.

A Figura 2.8 apresenta um diagrama de estados ilustrando os diferentes estados utilizados no tratamento do *link*. Existem outros sub estados dentro do estado de conexão. São eles *inquiry*, *inquiry scan*, *page*, *page scan*, *master response*, *slave response* e *inquiry response*.

Inquiry: utilizado para iniciar um procedimento. Permite que sejam descobertos novos dispositivos Bluetooth na área de cobertura. Os pacotes de *inquiry* são do tipo ID contendo IAC.

Inquiry scan: para responder ao pacote de *inquiry* é preciso que o outro dispositivo se encontre no modo de *inquiry scan*. Imediatamente ao receber o pacote ele passa ao modo de *inquiry response*, enviando um pacote de resposta do tipo FHS contendo seu endereço e algumas vezes o seu relógio.

Paging: passados os procedimentos de inquiry, segue o procedimento de *paging*, onde resulta numa conexão entre o mestre e o escravo. A unidade fonte passa a ser o mestre da conexão e envia pacotes *page* do tipo ID com DAC do dispositivo escravo. Neste momento mestre e escravo ainda não estão sincronizados.

Page scan: para responder à mensagem de *page*, o dispositivo deverá se encontrar no estado de *page scan*, no qual ela escuta o canal e espera por mensagens com seu DAC. Ao receber o pacote o dispositivo entra imediatamente em estado de *slave response*.

Slave response: Envia uma mensagem de resposta contendo o DAC desta unidade. Ao receber esta resposta, a fonte entra no estado de *page master response*, enviando ao dispositivo de destino um pacote do tipo FHS.

Master response: Ao receber pacotes FHS, o dispositivo destino envia um pacote de resposta com o seu ID e DAC. A partir desse momento, o dispositivo de destino passa a

utilizar os parâmetros do canal da fonte tornando-se seu escravo. Paralelo a isso, ao receber a resposta, a fonte passa a agir como mestre do dispositivo recém adicionado a piconet.

Os pacotes trocados no estado de conexão estabelecem o tipo de *link* se SCO ou ACL. Uma vez estabelecido, a troca de informações pode ser realizada. Durante este estado, o mestre envia pacotes de POLL para verificar seu link com os escravos. Se algum dispositivo não responder a esses pacotes, então a conexão deve ser restabelecida com o procedimento de *paging*.

2.5 Controle de Acesso ao Meio

O controle de acesso ao meio denominado *Media Acess Control* (MAC), é uma das subcamadas da camada de enlace da arquitetura OSI. É responsável por determinar quem tem acesso o meio de cada vez. No Bluetooth o MAC é dividido em duas partes principais: O protocolo de gerência de enlace *Link Manager Protocol* (LMP), e o protocolo de controle e adaptação de enlace lógico chamado *Logical Control and Adaptation Protocol* (L2CAP). A Figura 2.9 mostra a pilha de protocolos sugerida pelo SIG.

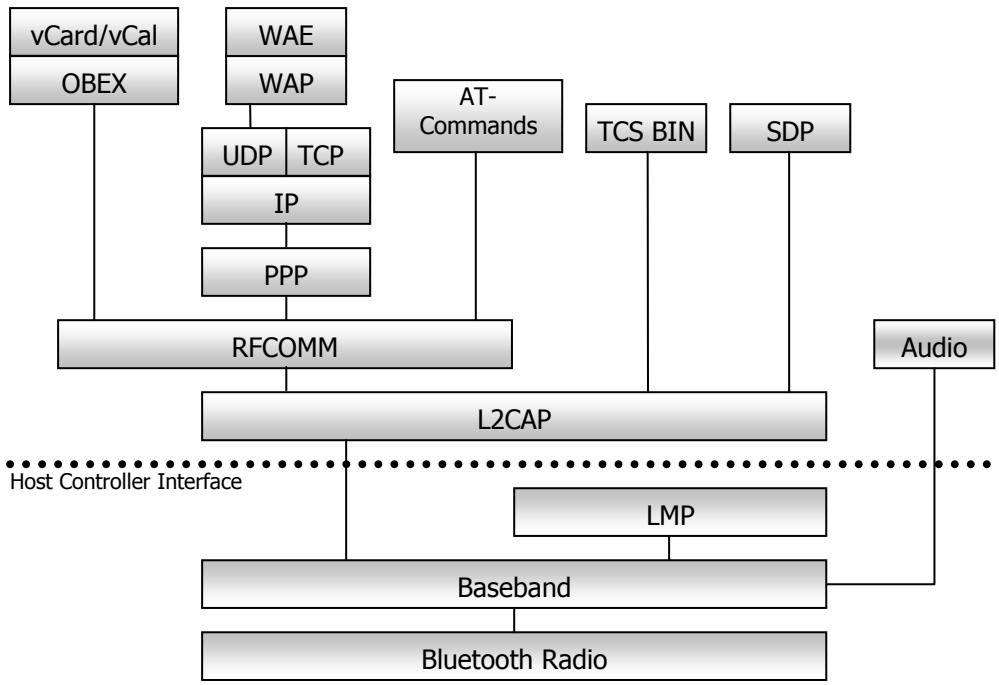


Figura 2.9 Pilha de Protocolos Sugerida pelo SIG.

2.5.1 Protocolo de Gerência de Enlace – LMP

O protocolo LMP nada mais é do que um software que roda no microprocessador do dispositivo Bluetooth e permite gerenciar a comunicação entre os dispositivos. No momento que dois dispositivos Bluetooth ficam próximos, o LMP de cada um deles descobre um ao outro, estabelecendo assim, uma comunicação pontual com mensagens sendo trocadas através do LMP. Essas mensagens têm a função de configurar e controlar enlaces, o transporte lógico e controlar enlaces físicos. A configuração inclui mecanismos de segurança como autenticação e criptografia.

Como forma de exemplificar o funcionamento do protocolo LMP, vamos apresentar a transação que faz o estabelecimento da conexão que ocorre depois dos procedimentos na banda básica. A Figura 2.10 mostra o diagrama de como essa configuração é feita.

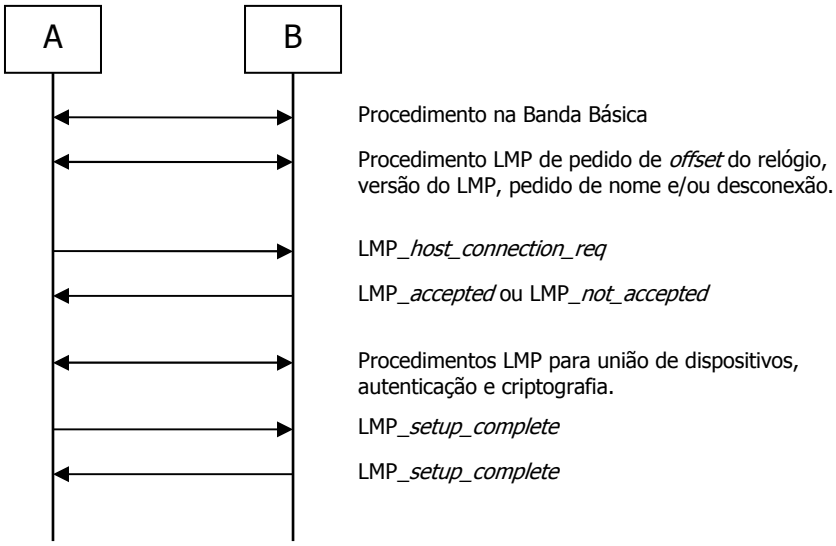


Figura 2.10 Diagrama do Estabelecimento de Conexão.
[Adaptado de BLUETOOTH, 2010]

Os primeiro procedimentos vistos na Figura 2.10 são informativos. Caso o dispositivo A queira criar uma conexão que envolva camadas superiores do LMP, ele envia um LMP_*host_connection_req*, que pode ser aceito ou não, indicando se haverá ou não o envolvimento dessas camadas. Logo após, serão realizados procedimentos de segurança. Para finalmente, os dispositivos enviarem a confirmação de configuração concluída de um para outro.

2.5.2 Protocolo de Controle e Adaptação de Enlace Lógico – L2CAP

O protocolo L2CAP faz parte da subcamada MAC no Bluetooth. Esse protocolo permite a multiplexação de protocolos de nível superior e os processos de segmentação e remontagem de pacotes, além de transmitir informações sobre QoS (qualidade de serviço). De forma análoga ao LMP, o L2CAP está acima do protocolo de Banda Básica. Este protocolo, permite, entre outras coisas, que as aplicações de camadas superiores transmitam e recebam pacotes L2CAP de 64 Kb. O L2CAP utiliza apenas enlaces ACL.

Dentre as principais características do protocolo L2CAP estão à simplicidade e baixo *overhead*, o que o torna apropriado para dispositivos com pouca memória e capacidade de processamento limitada, como celulares, *laptops*, PDAs, entre outros. Tal característica mencionada acima, faz com que dispositivos Bluetooth possuam boa eficiência energética, pois permitem um alto aproveitamento da banda sem com isso consumir muita energia.

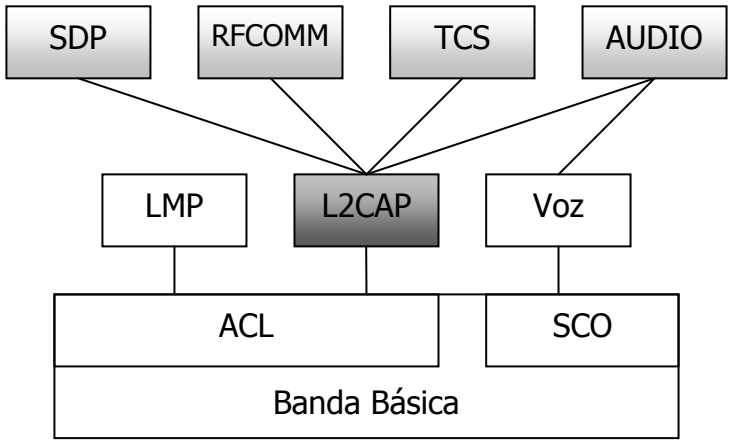


Figura 2.11 Comunicação L2CAP com outros Protocolos.
[Adaptado de Muller, Nathan J., 2001.]

Como visto na Figura 2.11 a capacidade de comunicação com diversas camadas exige uma série de funcionalidades do L2CAP. Podemos observar que L2CAP se comunica diretamente com o protocolo de descoberta de serviço *Service Discovery Protocol* (SDP), com o *Radio Frequency Communication* (RFCOMM), e especificações de controle de telefonia o *Telephony Control Specifications* (TCS). Apesar das aplicações que envolvem áudio e

telefonia estarem associadas e trabalharem em enlaces SCO da banda básica, comunicações que trabalham com áudio em forma de pacotes, como por exemplo telefonia IP, também pode comunicar-se com o L2CAP e os pacotes tratados como dados comuns.

3 *Desenvolvimento do Protótipo*

Agora que já há um embasamento teórico sobre o sistema Bluetooth que iremos utilizar, podemos começar a detalhar o nosso objetivo principal do trabalho: implementar uma solução para o monitoramento de sensores de pressão e temperatura e o envio das informações por Bluetooth utilizando a mínima pilha de protocolos. Vale ressaltar que a idéia sempre foi utilizar os recursos materiais do IFSC – Campus São José, portanto nenhum material, como circuitos integrados, resistores, *protoboard*, entre outros, foram comprados.

3.1 Arquitetura Geral do Sistema

A Figura 3.1 apresenta uma visão geral do sistema proposto. O **equipamento** mostrado consiste e um aparelho de refrigeração e ar condicionado, por exemplo, um ar condicionado. A este equipamento será acoplado **sensor** de pressão e de temperatura que vão enviar as informações por um *hardware* coletor para o PC. Nessa arquitetura, utilizaremos o **PC** como *host*, uma vez que se trata de um protótipo e não se deseja, nesse momento, desenvolver algo miniaturizado. Em seguida, enviaremos as informações através de uma interface USB para um **módulo Bluetooth** e por fim, para um outro dispositivo Bluetooth como um **celular**, por exemplo.

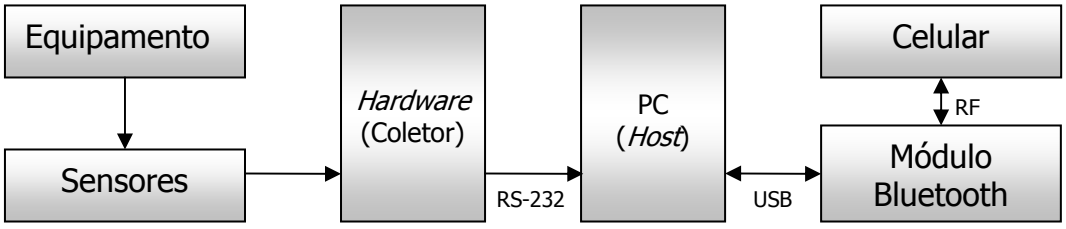


Figura 3.1 Visão Geral do Sistema.

3.2 *Hardware* Coletor

O *hardware* coletor é basicamente composto por um sensor de temperatura, um conversor analógico digital (A/D) e um microcontrolador (8051), que converterá a informação coletada e enviara para o *host*. Além destes elementos indispensáveis, podemos citar ainda, a utilização de uma fonte DC com (+5V, -5V, +12V) para alimentação dos circuitos, CIs 74244 utilizados como *buffers*, para poder apresentar os dados da temperatura, tanto nos displays de 7 segmentos, como também ao microcontrolador. Os CIs 74151, multiplexadores (MUX), foram utilizados para converter os dados de saída do conversor A/D paralelo de 24 bits (3,5 dígitos), para 1 bit, utilizando 5 portas do 8051 para endereçar os MUXs e uma porta para coletar a informação de forma serial. A Figura 3.2 exibe uma comparação entre o sistema mínimo proposto e o sistema realizado.

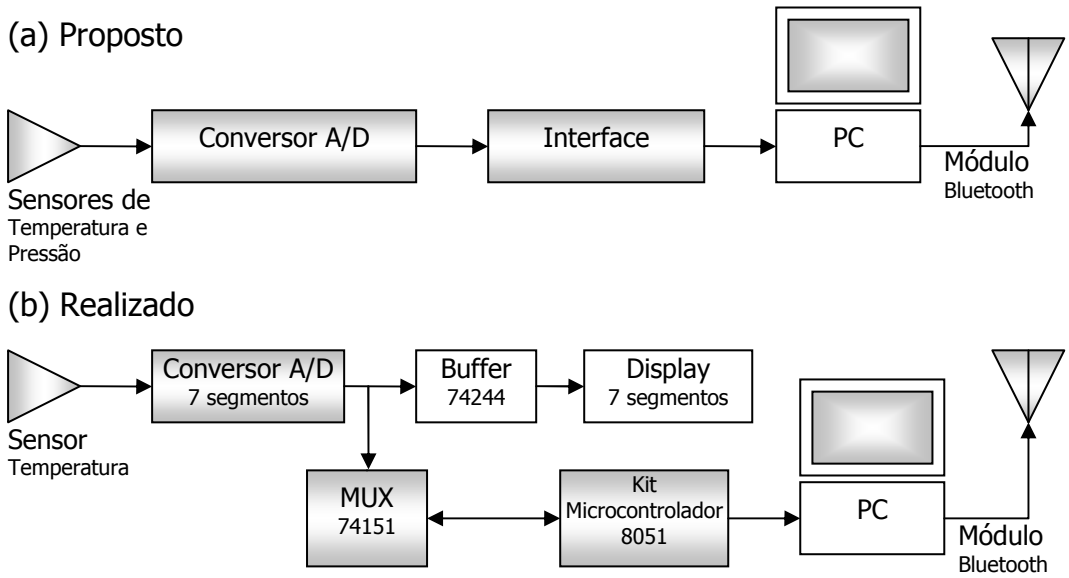


Figura 3.2 Modelo Proposto (a) e Realizado (b).

Ao analisarmos a Figura 3.2 do modelo proposto (a), percebemos que poderíamos realizar um *hardware* onde contivesse, o sensor de temperatura que poderia ser um termoresistor, um conversor A/D de 10 bits, por exemplo, uma interface com o PC que

poderia ser um microcontrolador com RS-232. Seguidos de um PC utilizado com *host* e um módulo Bluetooth que poderia ser com interface USB.

No modelo realizado (b) também visto na Figura 3.2, optamos pela utilização somente do sensor de temperatura, uma vez que, a implementação do sensor de pressão seria algo semelhante. Além do mais, informações como temperatura são melhores exemplificadas que as de pressão. Em nosso projeto, utilizamos um termoresistor KTY com resistência de 1 KΩ a 25º C. O conversor A/D utilizado foi um ICL7107 que tem como principais características possuir uma saída para 3 e ½ dígitos em *displays* de 7 segmentos, além de funcionar com um voltímetro, que pode ser configurado dentro de uma escala de 0 a 200 mV. Os *buffers* (74244) foram utilizados para apresentação das saídas do ICL7107 no *display* de 7 segmentos. O motivo que levou a utilização do 74244, foi a vontade de apresentar a informação da temperatura no próprio circuito através dos displays de 7 segmentos, para depois enviá-las para o PC a fim de serem transmitidas por Bluetooth. O conjunto de MUX (74151) utilizados no nosso sistema, tem como finalidade, acessar as informações na saída do ICL7107, multiplexando 21 bits (3 *displays* de 7 segmentos) para um bit serial. O endereçamento e a leitura dos dados serial dos MUX foi feito através do kit do microcontrolador 8051. O kit didático do microcontrolador 8051 foi a solução encontrada para a leitura das informações de temperatura, através do endereçamento dos MUXs, como também, a leitura serial, além da conversão de 7 segmentos para ASC, para depois serem transmitidas pelo próprio kit através da RS-232. O PC, utilizado como *host* em nosso projeto, fica responsável por receber a informação de temperatura, através da interface serial RS-232 e enviá-la por meio de um módulo Bluetooth conectado via USB para os dispositivos que estiverem dentro área de alcance. O módulo Bluetooth, como os demais dispositivos utilizados, serão descritos na seqüência do trabalho (v. Apêndice B, p. 95).

3.2.1 Sensor de Temperatura

O sensor de temperatura utilizado em nosso projeto foi o KTY 21-6. Faz parte do grupo de sensores de silício e possui resistência de 1000 Ω a uma temperatura de 25º C. Entre outras características estão a temperatura de operação entre -50° C a +150° C (-60 F a 300 F), saída linear, excelente estabilidade a longo prazo, polaridade independente, devido à construção

simétrica, rápido tempo de resposta e tolerâncias de resistência de ± 3% a 25º C. O *datasheet* completo pode ser encontrado em www.datasheetcatalog.com.

3.2.2 Conversor A/D

O ICL7107, utilizado em nosso projeto, contém um conversor analógico-digital com *clock* interno e tensão de referência, incluindo sete decodificadores e excitadores para displays de 7 segmentos, formando um conjunto de 3 e ½ dígitos. Associado a alguns componentes externos, pode formar instrumentos simples para a medição de tensões e outras grandezas físicas que possam ser convertidas em tensões elétricas. O ICL7107 é apropriado para displays de *leds* e precisa de uma fonte de alimentação simétrica de +5V e -5V típico. Um outro conversor o ICL7106 é projetado para interface com *display* de cristal líquido.

Este conversor A/D reuniu uma combinação de alta precisão, versatilidade e economia . Podemos ainda constatar outras características como o baixo ruído e consumo de energia tipicamente menor que 10 mW, uma boa estabilidade, e, é disponível para venda em CIs sem chumbo (Pb-Free) de acordo com *Restriction of Hazardous Substances Directive Compliant* (RoHS). O *datasheet* do componente pode ser encontrado em www.datasheetcatalog.com, e mostra todos detalhes do conversor, bem como o exemplo de implementação de circuitos utilizando o ICL7107.

Em nosso projeto, o circuito foi montado de maneira que o divisor de tensão, formado pelo nosso sensor KTY, gerasse a tensão de referência para temperatura. A calibração foi feita com ajuda de 2 potenciômetros (2K2 e 1KΩ) responsáveis respectivamente pelo ajuste do zero e ajuste de escala, utilizando os valores padrões de 0ºC e 100ºC, água com gelo e água fervente para sua calibração. O esquema elétrico completo encontra-se no Apêndice B, p. 95.

3.2.3 Multiplexador – 74151

O 74151 utilizado em nosso projeto é um seletor / multiplexador de dados que contém uma decodificação completa em um único CI para seleção da fonte de dados desejada. O CI pode selecionar uma das 8 fontes de dados disponíveis endereçadas por 3 bits. Possui uma

entrada de *strobe*, que deve estar em um nível lógico baixo para permitir que os dados possam ser acessados. Ainda, o 74151, apresenta saídas complementares W e Y. Em nosso projeto foi utilizado um conjunto de 4 CIs 74151, associados de forma tal, que 21 bits da saída do ICL7107 foram multiplexados para um bit. A Figura 3.3 mostra o diagrama de conexão e a Tabela 3.1 apresenta a tabela verdade para esse multiplexador.

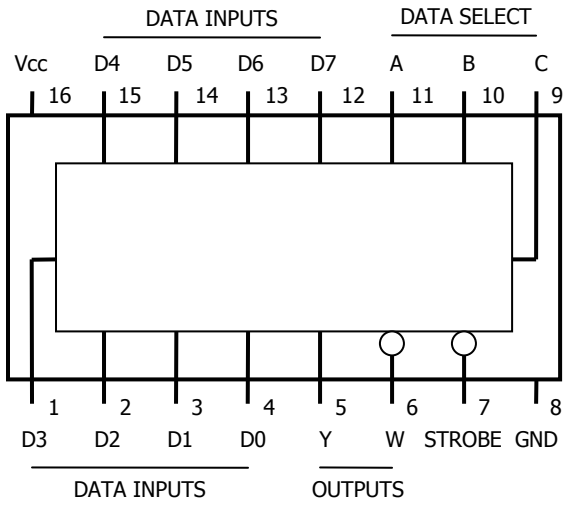


Figura 3.3 Diagrama de Conexão do 74151.

Tabela 3.1: Tabela Verdade do 74151

Inputs				**Outputs**	
Select			**Strobe**	**Y**	**W**
C	**B**	**A**	**S**		
X	X	X	H	L	H
L	L	L	L	D0	D0
L	L	H	L	D1	D1
L	H	L	L	D2	D2
L	H	H	L	D3	D3
H	L	L	L	D4	D4
H	L	H	L	D5	D5
H	H	L	L	D6	D6
H	H	H	L	D7	D7

H = Nível Alto, L = Nível Baixo, X = Não importa
D0..D7 = nível das respectivas entradas D

3.2.4 *Buffer* - 74244

Os 74244, utilizados em nosso projeto, têm como principal característica funcionarem como *buffers* ou *drivers* de linha e são projetados para melhorar o desempenho em placas para PC. Possuem 8 portas de entrada e saída acionadas por gatilho que libera um terceiro estado (3-STATE). Podem ser utilizados para endereçamento de memória, *drivers* de relógio e transmissores e receptores de barramento. Possui alta impedância de entrada, o que proporciona uma melhor rejeição de ruído. Característica esta, aproveitada pelo nosso projeto, a fim de dividir o sinal de saída do ICL7107 para os MUXs e para *display* de 7 segmentos como se fosse um barramento. A Figura 3.4 exibe o diagrama de conexão e a Tabela 3.2 mostra os 3 estados possíveis de saída.

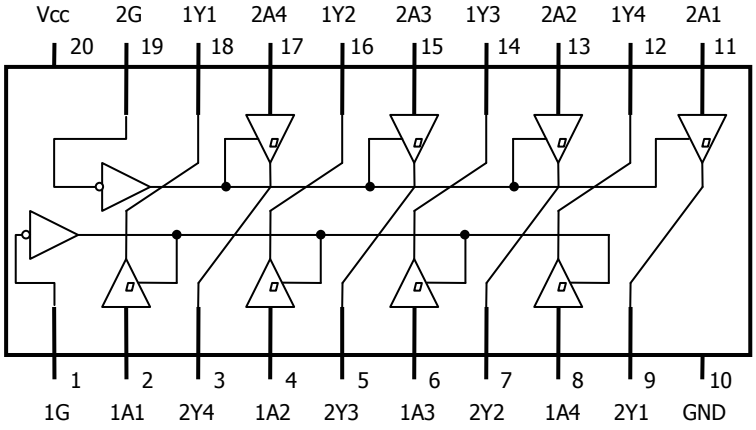


Figura 3.4 Diagrama de Conexão do 74244.

Tabela 3.2: Tabela de Funções do 74244

Inputs		**Outputs**
$\overline{G}$	**A**	**Y**
L	L	L
L	H	H
H	X	Z

L = Nível lógico baixo
H = Nível lógico alto
X = Não importa
Z = Alta impedância

3.2.5 Kit Microcontrolador 8051

O kit didático do microcontrolador, utilizado em nosso projeto, proporciona desenvolver aplicações baseadas na família 8051. Com ele podemos programar diretamente o microcontrolador sem a necessidade adicional de outros kits de gravação; além do mais, já possui interface de comunicação RS-232 integrada, facilitando a comunicação com o PC. O kit vem com processador AT89S8252, fonte de alimentação de 9V/300mA – 110/220V e cabo de gravação. Entre as principais características estão:

- Compatibilidade com a família 8051;
- 8 Kbytes de memória Flash;
- 256 bytes de memória RAM;
- 32 portas de entrada / saída;
- Modo de programação serial.

O kit permite:

- Gravação através da saída paralela do computador;
- Permite o desenvolvimento aplicando a linguagem *Assembler*;
- Possui memória *flash* que evita a perda das informações gravadas;
- Permite a utilização do kit como terminal RS-232 conectado a um PC.

O kit possui entre outras coisas:

- Cristal de 11,0592 MHz;
- Barramento com 8 *leds*;
- Teclado com 6 teclas tipo *push-botton*;
- Canal serial RS-232.

Em nosso projeto, o microcontrolador 8051 foi programado, primeiro para indexar os multiplexadores, obtendo a informação multiplexada, segundo, para converter a informação e por último enviá-la pela interface RS-232.

3.2.5.1 *Software* 8051

As Figuras 3.5, 3.6, 3.7 e 3.8 trazem os fluxogramas do programa realizado. No Apêndice A, p. 93, encontra-se o código fonte completo implementado em linguagem *Assembler*.

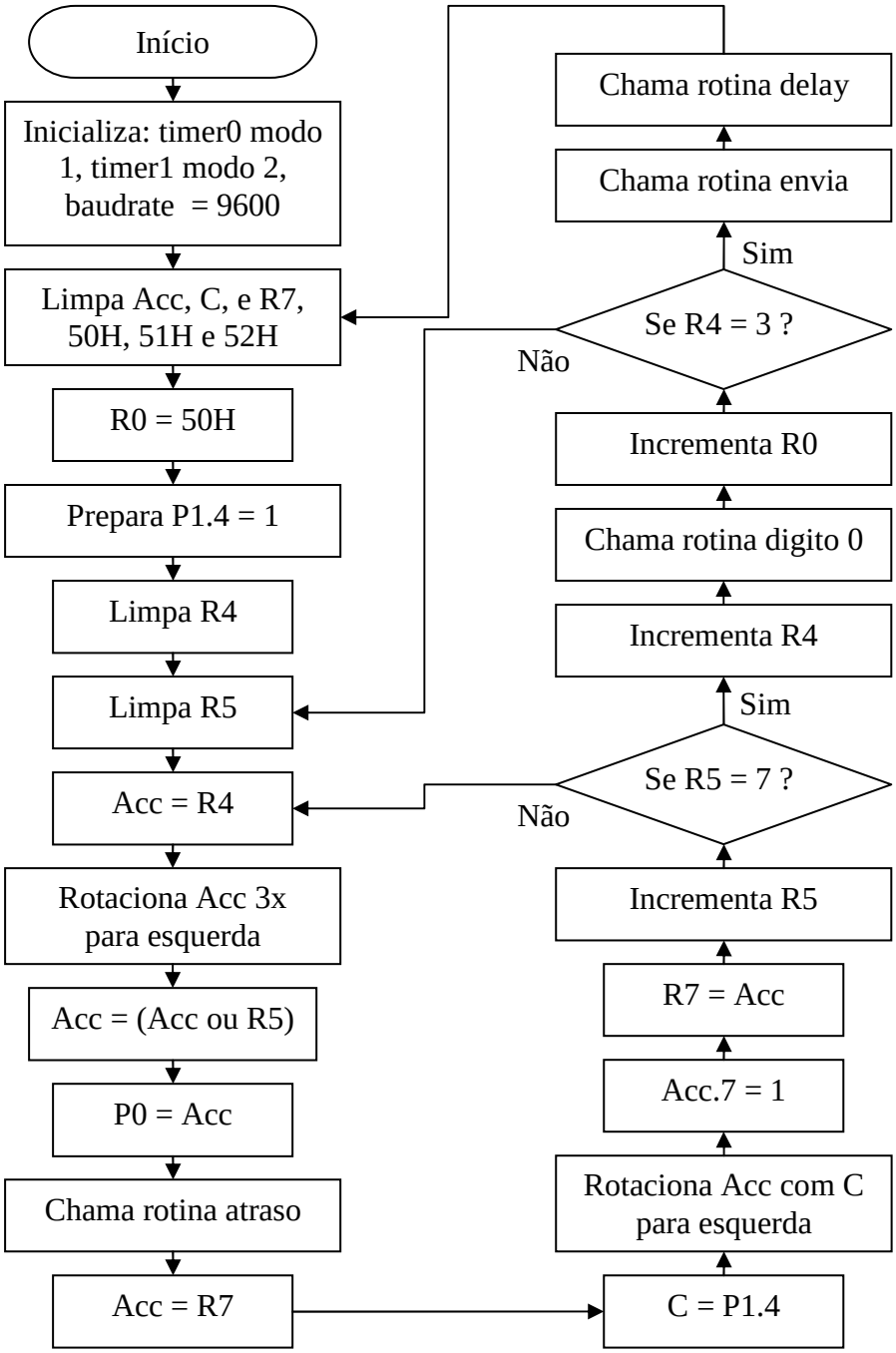


Figura 3.5 Fluxograma *Software* 8051 – Parte I.

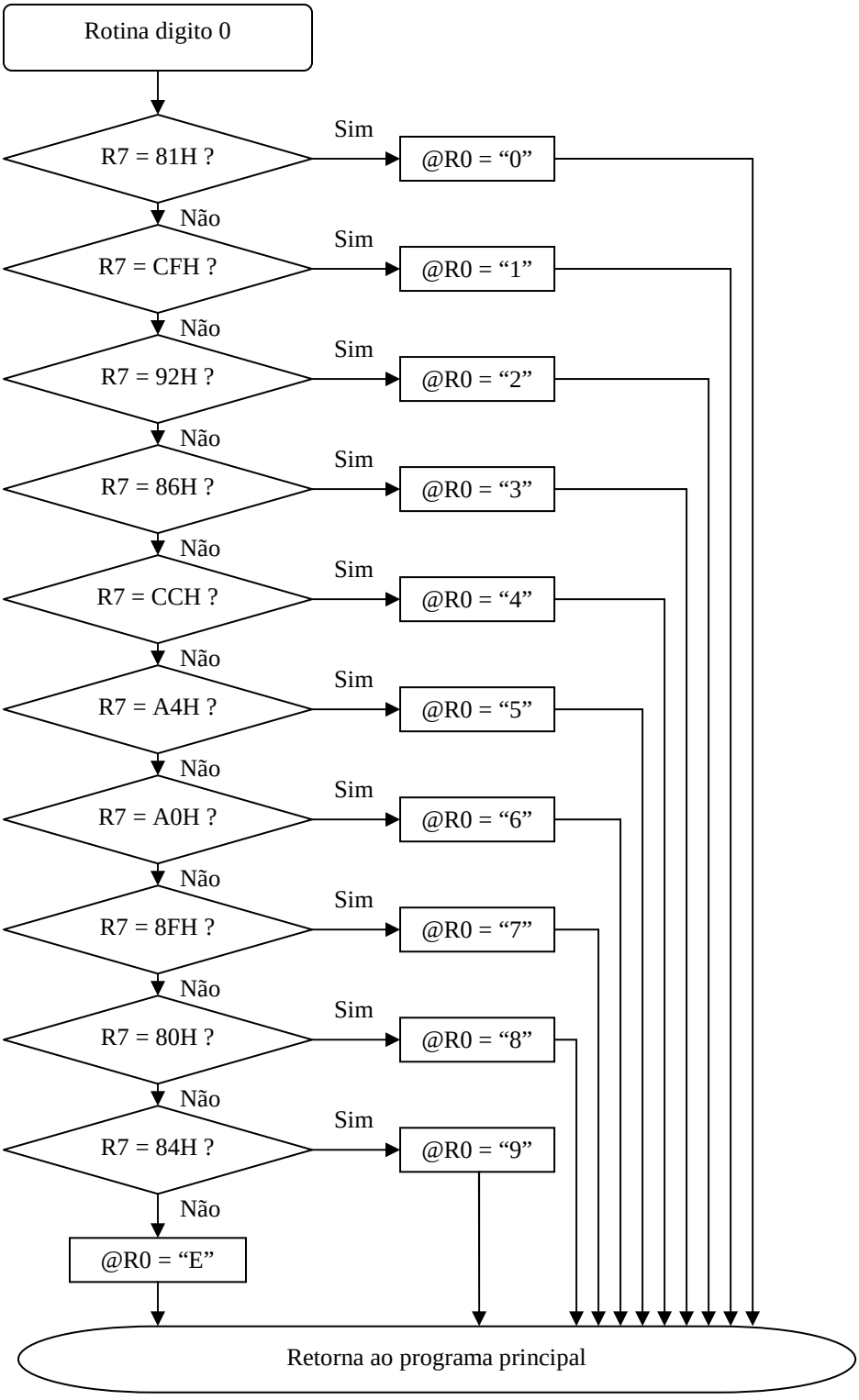


Figura 3.6 Fluxograma *Software* 8051 – Parte II.

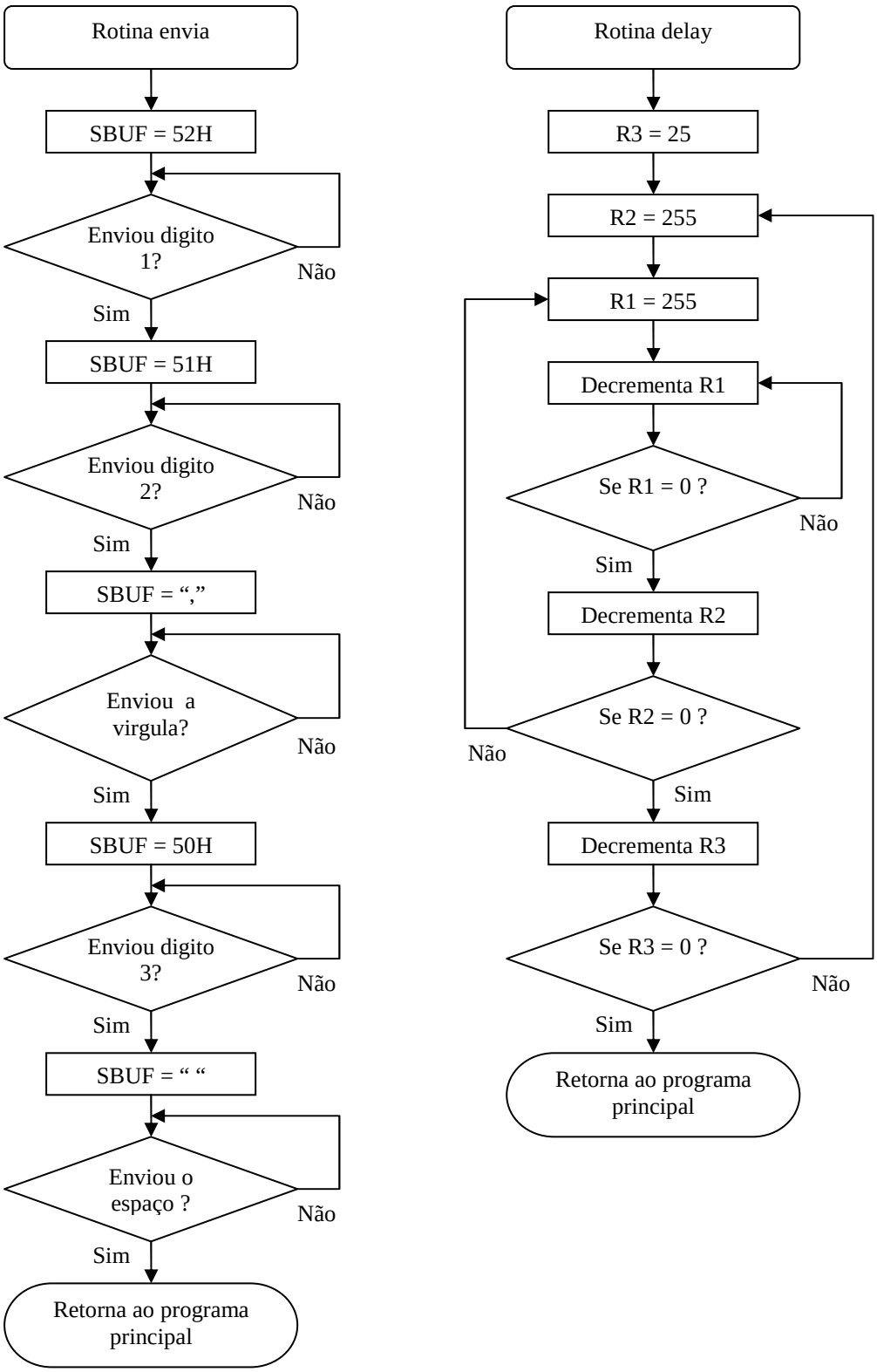


Figura 3.7 Fluxograma *Software* 8051 – Parte III.

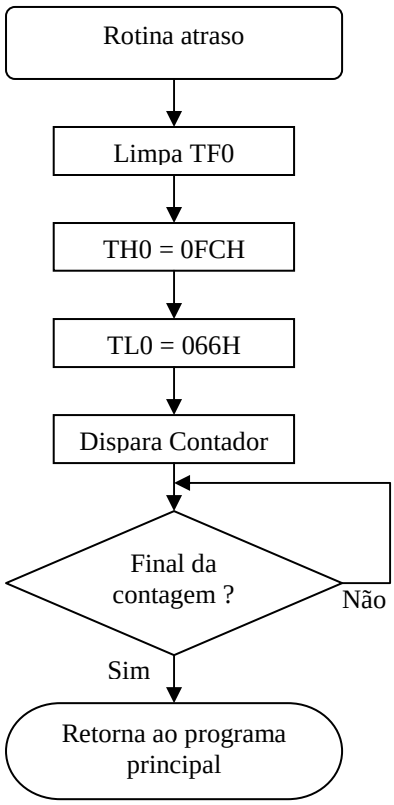


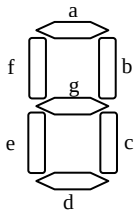
Figura 3.8 Fluxograma *Software* 8051 – Parte IV.

O *software* gravado e executado pelo microcontrolador 8051 é composto por 3 partes principais e 2 auxiliares. A primeira parte, observada no fluxograma da Figura 3.5, trata da inicialização dos temporizadores e contadores, e, também da configuração da taxa de transmissão de dados (*baudrate*), utilizada no envio das informações através da comunicação serial RS-232, que foi configurada em 9600 bps. Ainda, podemos destacar, a preparação da porta P1.4, “setada” para 1 e que será utilizada para a leitura dos dados multiplexados. Basicamente, a rotina principal faz uma contagem de 0 a 7, endereçando os 3 três primeiros multiplexadores, e outra contagem de 0 a 2 endereçando o último MUX, cuja saída Y vai ser lida através da porta P1.4. As 5 portas de saídas utilizadas do kit do 8051 para endereçar os multiplexadores foram as de P0.0 a P0.4.

A segunda parte do *software* que podemos ver no fluxograma da Figura 3.6, faz a conversão de símbolos do *display* de 7 segmentos, para ASC. Tal conversão é realizada, a partir da comparação do dado lido e armazenado em um determinado local da memória com o símbolo esperado, se a comparação for verdadeira, um conjunto de bits que representa um

símbolo ASC será armazenado na memória para posterior transmissão. A Tabela 3.3 apresenta a conversão de 7 segmentos para ASC.

Tabela 3.3: Conversão de 7 Segmentos para ASC

Display 7 segmentos Bits: xabcdefg	***Display***	***ASC Hexadecimal***
10000001	0	30H
11001111	1	31H
10010010	2	32H
10000110	3	33H
11001100	4	34H
10100100	5	35H
10100000	6	36H
10001111	7	37H
10000000	8	38H
10000100	9	39H
	Led acesso bit 0 Led apagado bit 1	

A terceira parte do sistema, que pode ser observada no fluxograma “rotina envia” da Figura 3.7. Como o próprio nome já diz, é à parte do sistema utilizada para enviar a informação recebida e armazenada na memória. A variável SBUF corresponde a um endereço

de memória responsável pelo envio da informação. Quando um determinado byte de dados é armazenado em SBUF, o mesmo é imediatamente enviado a uma taxa de *baudrate* pré-determinada. Na nossa aplicação foi de 9600 bps. O bit TI é “setado” para 1 se o envio ocorreu com sucesso. Essa rotina, envia os dígitos 1 e 2, a vírgula, o dígito 3 e por último, espaço em branco. Essas informações correspondem à dezena, unidade e fração da temperatura. Portanto, para simplificar nossa aplicação à faixa de temperatura ficou limitada ente 00,0ºC. a 99,9ºC. Lembrando que o nosso circuito pode operar entre -50ºC. e +150ºC. que é a faixa de operação do sensor de temperatura utilizado.

A Figura 3.7 mostra ainda, o fluxograma da rotina de *delay* típica, responsável por configurar o tempo entre uma varredura e outra das informações. O tempo médio configurado ficou em entre 3 e 4 segundos. Nesse tipo de rotina, não se exige uma grande precisão de tempo.

Por último, na Figura 3.8, apresentamos o fluxograma com a configuração típica de rotina de atraso, utilizando os temporizadores / contadores do 8051. Nessa rotina de atraso, o temporizador *timer* 0 foi previamente configurado no programa principal em modo 1 de 16 bits. Por estar diretamente relacionado com a freqüência do *clock* do cristal utilizado, o tempo que a rotina fica aguardando possui uma precisão muito maior do que em outras rotinas de atraso. No kit do 8051 utilizado, o cristal é de 11,0592 MHz. Os contadores TH0 e TL0 foram configurados para um tempo de 1 ms (1 KHz). Ou seja, precisaremos apenas de 21 ms para fazer uma leitura completa da temperatura, diminuindo assim a possibilidade de o sistema ler em alguma transição (mudança de temperatura). As equações (3.1) e (3.2) apresentam o cálculo do valor do contador para gerar um determinado atraso.

$$N = \frac{CLOCK}{12 * F} \tag{3.1}$$

$$F = \frac{CLOCK}{12 * N} \tag{3.2}$$

Para uma freqüência de 1 KHz, com *clock* = 11,0592 MHz, então N = 922. Como este é um contador que conta para cima, precisamos subtrair este valor de 65536. Com isto, N = 65536-922 = 64614 ou FC66H.

3.2.6 Computador Pessoal – PC

A utilização de computador pessoal como *host* foi assim decidida por se tratar de um protótipo e de não haver necessidade de se desenvolver algo compacto ou que podemos chamar de “embarcado”. É o PC com o mínimo suporte de *hardware* necessário, que irá executar toda pilha de protocolos necessária para o envio de informações pelo Bluetooth. Algumas características básicas do PC utilizado são: memória RAM 500MB, HD de 40GB, sistema operacional Linux – Ubuntu 10.04 (abril de 2010). Para entendermos melhor a escolha da utilização do PC como *host*, apresentaremos algumas configurações possíveis para esse sistema.

3.2.6.1 PC como *Host* e Módulo Bluetooth Externo

É a típica configuração em se tratando de protótipos. Nessa configuração, toda a pilha de protocolos Bluetooth é implementada em *software* e executada em um PC com módulo Bluetooth conectado através de USB ou *Universal Asynchronous Receiver / Transmitter* (UART). É a configuração que utilizamos em nosso projeto e a Figura 3.9 mostra como fica esta configuração.

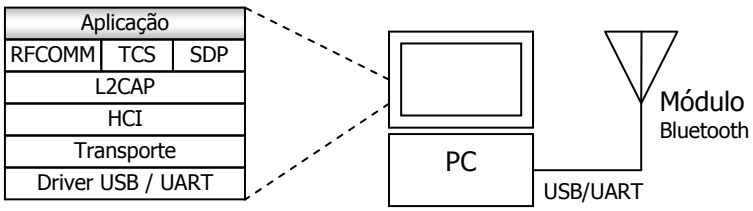


Figura 3.9 Configuração com PC como *Host*.

3.2.6.2 Microcontrolador como *Host* e Módulo Bluetooth Externo

Essa configuração é idêntica a anterior, porém, a pilha de protocolos é executada em um microcontrolador embutido. É a típica configuração utilizada em sistemas embutidos. A Figura 3.10 exibe esta configuração utilizando o microcontrolador como *Host*.

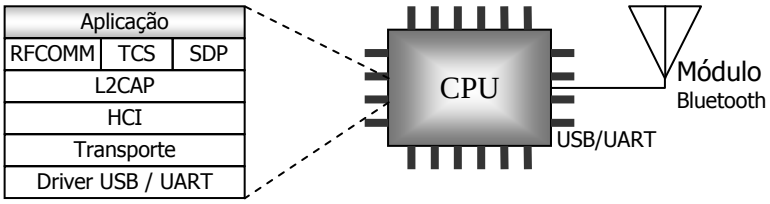


Figura 3.10 Configuração com Microcontrolador como *Host*.

3.2.6.3 Aplicação Integrada ao Módulo Bluetooth

Nesta configuração, a aplicação é executada a partir da pilha de protocolos implementada no próprio módulo Bluetooth. Podemos ainda utilizar um microcontrolador de entrada / saída para interfaces externas. A Figura 3.11 da uma idéia da configuração com aplicação integrada ao módulo Bluetooth.

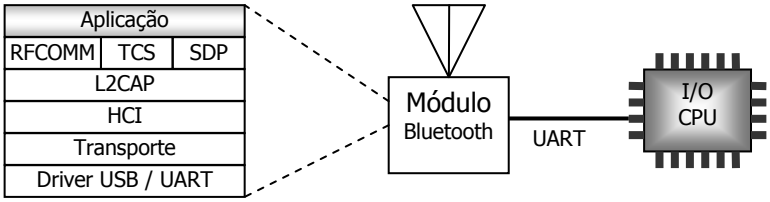


Figura 3.11 Configuração com Aplicação Integrada ao Módulo.

3.2.6.4 Aplicação Integrada ao Microprocessador

Nesta configuração, a aplicação é executada em um microprocessador com todas as funcionalidades Bluetooth embutidas nele. Ainda, podem ter módulo RF interno ou externo. A Figura 3.12 mostra um modelo com módulo RF externo.

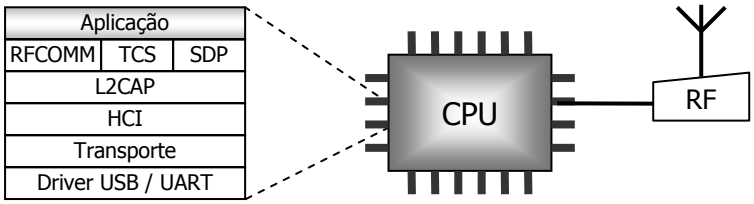


Figura 3.12 Configuração com Aplicação Integrada ao Microprocessador.

3.2.7 Módulo Bluetooth

Em nosso projeto utilizamos 4 módulos Bluetooth USB de 3 modelos diferentes. Quando se fala de diferentes, isso pode servir para designar formatos, tamanhos e cores. Mas é na versão do Bluetooth e sua classe é que sua característica é mais significativa. A Figura 3.13 mostra a foto dos modelos de módulo Bluetooth utilizados nos testes.

Figura 3.13 Foto dos Módulos Bluetooth Utilizados.

Todos os módulos Bluetooth devem seguir as especificações definidas pelo grupo SIG para que seus chips consigam comunicar-se com os demais chips existentes no mercado. Internamente, um módulo Bluetooth provê uma Interface de rádio e um enlace físico entre dois ou mais dispositivos.

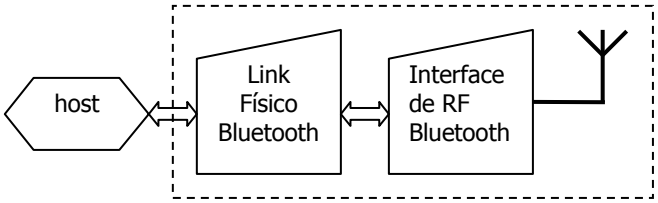


Figura 3.14 Blocos Funcionais de um Módulo Bluetooth.

O funcionamento básico de um sistema microcontrolado, sinaliza a interface de rádio (partes que efetivamente realizam a comunicação: módulos de transmissão / recepção, antena), para que seja estabelecido um link físico entre dois ou mais dispositivos.

3.2.8 Outros Materiais Utilizados

Além dos elementos descritos até agora, foram utilizados em nosso projeto, uma fonte AT, dois *protoboards*, resistores e capacitores de valores diversos, um cabo serial RS-232, dois potenciômetros de 1K e 2K2Ω e fios de cores variadas. Foi também utilizado um multímetro digital, aparelhos de celulares diversos, especialmente um Sony Ericsson K660i. A Figura 3.15 traz a foto de uma fonte AT semelhante à utilizada.

Figura 3.15 Foto de uma Fonte AT.

As fontes AT são o modelo mais simples, predominando até o final da década de 90, quando foram substituídas pelas ATX. Antes de utilizar a fonte é preciso identificar as tensões que ela fornece. Existem alguns conjuntos de cabos saindo da fonte de cores diferenciadas. Cada cor corresponde a uma tensão diferente. Essa relação entre cores e tensões obedece a um padrão, que podemos verificar na Tabela 3.4.

Tabela 3.4: Cores dos Cabos – Fonte AT

Nome	***Cor***	***Descrição***
GND	Preto	Terra 0V
+5V	Vermelho	+5V DC
-5V	Branco	-5V DC
+12V	Amarelo	+12V DC
-12V	Azul	-12V DC
PG	Laranja	“Power Good”

O cabo laranja, chamado de “Power Good”, é um recurso da fonte AT que têm a função de indicar quando todas as outras tensões estão estabilizadas. Quando isso ocorre, a tensão nessa saída é de +5V. Em nosso projeto utilizamos a fonte para a alimentação simétrica de +5V e -5V do ICL7107, +5V nos 74151 e 74244, além de fornecer os +12V para o kit do microcontrolador 8051.

Outro equipamento importante utilizado em nosso projeto, responsável pela ligação física entre o microcontrolador 8051 e o PC é o cabo RS-232. Também conhecido por EIA RS-232C ou V.24, que é um padrão para troca serial de dados binários entre um DTE (terminal de dados) e um DCE (comunicador de dados). É comumente usado nas portas seriais dos PCs.

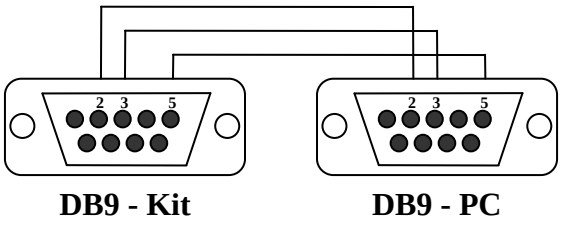


Figura 3.16 Cabo RS-232 Ponto a Ponto.

O cabo RS-232, utilizado em nosso projeto, foi confeccionado de modo especial diferente dos cabos chamados NULL MODEM onde o TX pino 3 de um conector inverte-se com RX pino 2 do outro conector. Como o kit do 8051 utilizado, já transmite TX pelo pino 2 e recebe RX pelo pino 3, fazendo o chamado *cross-over*, tivemos então, que confeccionar um cabo de modo que as ligações fossem feitas ponto a ponto entre os pinos 2, 3 e 5 de ambos os lados. A Figura 3.16 exibe como ficou o cabo da nossa aplicação.

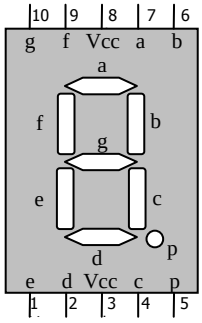


Figura 3.17 Diagrama de Conexão do *Display* CTK D166A.

Outro material utilizado e já referenciado no texto, foi o *display* de 7 segmentos. Em nosso projeto foi utilizado o CTK D166A. É um *display* anodo comum com alimentação de +5V. Os 4 *displays* utilizados em nosso projeto, foram conectados as saídas dos CI 74244 (*buffers*) através de um resistor de 470Ω utilizado para limitar a corrente, controlando a luminosidade dos *leds*. A Figura 3.17 exibe o diagrama de conexão para esse modelo de *display* de 7 segmentos.

3.3 Visão do *Hardware* Coletor

Podemos dividir a visão do *hardware* coletor em quatro partes. A Figura 3.18 mostra a foto com a realização do circuito. No Apêndice B, p. 95, encontra-se o diagrama elétrico do circuito coletor de temperatura. Na parte de cima da foto, observa-se um fio enrolado, trata-se do sensor de temperatura. No *protoboard* abaixo do sensor, onde estão os *leds* marcando a temperatura de 20,2 graus, estão, o conversor A/D ICL7107, os 3 buffers 74244, e todos os componentes resistivos e capacitivos necessários para a polarização dos circuitos. Na terceira parte, segundo *protobord*, estão os 4 multiplexadores 74151, configurados como podemos observar, de forma que os 3 primeiros multiplexadores pegam a informação da parte de cima, saída do ICL7107, e se ligam a 3 entradas do último multiplexador, que por sua vez, é endereçado pelo kit do 8051, logo abaixo, e devolve um bit (um fio), ao próprio kit, última parte vista na foto. Do kit, podemos falar que as informações seguem até o computador por meio de um cabo serial ponto a ponto que ligam as duas interfaces RS-232.

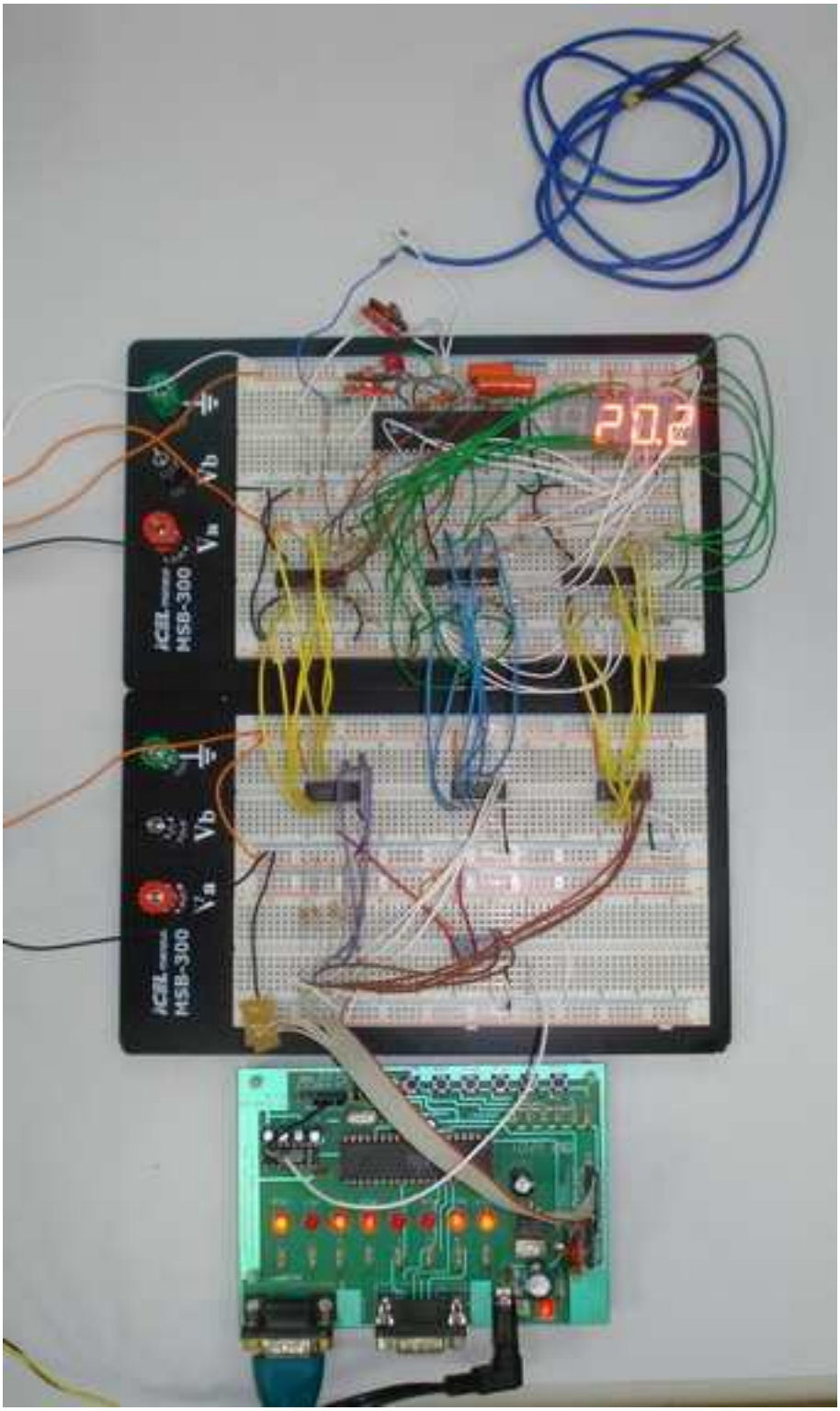


Figura 3.18 Foto do *Hardware* Coletor.

4 Software Bluetooth

Agora que já se tratou de todo *hardware* necessário para coletar as informações de temperatura, precisamos transmiti-las por meio de Bluetooth. Para isso, nesse capítulo vamos descrever os recursos de *software* utilizados para fazer a transmissão dos dados de temperatura para um dispositivo com Bluetooth integrado, como um celular por exemplo.

4.1 Pilha de Protocolos

A pilha de protocolos sugerida pelo SIG, apresentada na Figura 2.9, envolve uma série de funcionalidade que, dependendo da aplicação, não serão utilizadas. A implementação completa descrita pelo SIG geraria algo imenso em se tratando do tamanho do seu código. Por exemplo, a pilha de protocolos desenvolvida em código aberto para Linux, o BlueZ, precisaria mais de 100Kbytes de memória para que seus módulos fossem carregados e se pudesse fazer uma comunicação entre dispositivos. Em se tratando de sistemas embutidos onde a tecnologia já é bastante utilizada, a quantidade de memória requerida é muito alta. Por esse motivo, surgiu a necessidade de se utilizar somente parte da pilha de protocolos necessária para uma determinada aplicação, assim, a alternativa sugerida é desenvolver uma pilha de protocolos adaptada à aplicação e não a aplicação voltada pra pilha.

A idéia básica é aproveitar do **código BlueZ** disponível em www.bluez.org, à parte da pilha de protocolos para se desenvolver uma camada de *software* mínima necessária, de modo a resolver a comunicação entre dois dispositivos Bluetooth que atendam a aplicação. Na nossa aplicação, o problema estava em transmitir as informações de temperatura, que poderia ser em modo de texto, para um telefone celular ou qualquer dispositivo Bluetooth relativamente próximo ao equipamento. Nesta situação, observando melhor a pilha de protocolos Bluetooth, um protocolo especificamente chamou nossa atenção e poderia ser utilizado para permitir o envio de objetos, especialmente de arquivos, chamado de *Object Exchange* (OBEX). Nesse sentido, aproveitamos que o **código Obexd** com toda a pilha necessária para nossa aplicação encontra-se disponível para *download* no site: www.bluez.org, passamos a estudá-lo e utilizá-

lo nos testes de envio de um arquivo de texto contendo as informações de temperatura, base da nossa aplicação. A seguir, veremos mais sobre o BlueZ e o Obexd.

4.2 *Software* BlueZ

A pilha de protocolos BlueZ foi desenvolvida inicialmente por Max Krasnyansky da empresa Qualcomm - Quality Communications. Em maio de 2001 a empresa decidiu liberar seu código fonte e o anúncio foi enviado para a lista de discussão dos desenvolvedores Bluetooth e este foi o início oficial do Linux Bluetooth. Um mês depois o código foi pego por Linus Torvalds para inclusão no kernel Linux a partir da versão 2.4.6, possuindo sua própria pilha de protocolos Bluetooth. Em janeiro de 2004, 15 mantenedores do BlueZ entregaram toda a documentação em mãos a Marcel Holtmann para dar prosseguimento aos estudos. Isto aconteceu quase um mês depois do lançamento, por Linus Torvalds, da primeira versão da série 2.6 de kernel. O objetivo de uma qualificação oficial do BlueZ como um subsistema do SIG Bluetooth foi alcançado em abril de 2005, com a ajuda do Tom Tom BV.

O primeiro código fonte aberto do Bluetooth, foi desenvolvido pela empresa Axis Communications, e foi chamado OpenBT. Em abril de 2005, Anders Johansson anunciou que seu desenvolvimento seria interrompido e a lista de discussão encerrada.

A tecnologia sem fio Bluetooth é conhecida em todo o mundo como sendo uma a solução de rádio de baixo custo, que fornece ligações entre computadores portáteis, telefones celulares e outros dispositivos portáteis, podendo fornecer ligações com outras redes como a Internet. Toda a especificação Bluetooth é desenvolvida, publicada e promovida pelo SIG.

Entre as principais características , o BlueZ, fornece suporte para as camadas de núcleo e protocolos Bluetooth. É flexível, eficiente e utiliza uma aplicação modular. Entre outras características podemos citar:

- Implementação modular completa;
- Multiprocessamento simétrico seguro;
- Processamento de dados *Multithreaded*;
- Suporte para múltiplos dispositivos Bluetooth;
- Abstração real de *hardware;*

- Interface padrão *socket* para todas as camadas;
- Dispositivos e serviços de apoio com nível de segurança.

Atualmente o BlueZ consiste em vários módulos distintos:

- Bluetooth núcleo de subsistemas do kernel;
- L2CAP e as camadas de kernel SCO de áudio;
- RFCOMM, BNEP, CMTP e implementações HIDP do kernel;
- HCI UART, USB, PCMCIA e drivers de dispositivos virtuais;
- Bluetooth geral e bibliotecas SDP e *daemons*;
- Utilitários de testes e configurações;
- Ferramentas de análise e decodificação de protocolos.

Os módulos do kernel do BlueZ, bibliotecas e utilitários são concebidos para funcionar perfeitamente em muitas arquiteturas conhecidas do Linux. Isto também inclui as plataformas de processadores *single* e *multi*, bem como sistemas *hyper threading*:

- Intel e AMD x86;
- AMD64 e EM64T (x86-64);
- SUN SPARC 32/64bit;
- PowerPC 32/64bit;
- Intel XScale e StrongARM;
- Processadores Hitachi / Renesas SH;
- Motorola DragonBall.

O suporte para BlueZ pode ser encontrado em muitas distribuições Linux e, em geral, é compatível com qualquer sistema Linux no mercado:

- Debian GNU / Linux;
- Ubuntu Linux;
- Fedora Core / Red Hat Linux;
- OpenSuSE / SuSE Linux;
- Mandriva Linux.

4.2.1 Instalação dos Pacotes

Em nosso projeto, utilizando um computador com sistema operacional Linux Ubuntu versão 10, devidamente conectado a Internet, os pacotes Bluetooth e BlueZ puderam ser instalados a partir do terminal do sistema digitando:

```
# apt-get install bluetooth bluez
```

Nesse momento, uma série de informações é coletada e os pacotes são instalados ou atualizados conforme sua versão. Outros pacotes podem se fazer necessários e eles devem ser instalados da mesma forma como citado acima.

4.2.2 Principais Comandos

Antes de falar sobre os principais comandos, devemos verificar o estado de operação do serviço **dbus** responsável dentre outras coisas pela conexão USB do qual o nosso módulo Bluetooth faz uso. E do próprio serviço **Bluetooth. A**mbos os serviços precisam estar iniciados. Os comandos a seguir foram utilizados para iniciar, reiniciar, parar ou verificar *status*. Como usuário **root,** ou com ajuda do comando **sudo,** devemos executar o comando diretamente no terminal:

```
# /etc/init.d/dbus start
# /etc/init.d/bluetooth status
* bluetooth is running
```

As opções: *status*, *start*, *restart* e *stop*, podem ser utilizadas para o serviço Bluetooth e *start*, *restart* e *stop* para o serviço dbus. Algumas dessas opções, forçam o reinicio do sistema operacional.

4.2.2.1 Hciconfig

O comando hciconfig é usado para configurar os dispositivos Bluetooth instalados. Como principal parâmetro de configuração esta o hciX, onde X identifica o número do dispositivo Bluetooth no sistema. Quando não informado, X = 0, e informações do hci0 serão apresentadas na tela. O comando [# man hciconfig] trás outras informações sobre o comando. Abaixo, seguem alguns exemplos de comandos e suas saídas:

```
    # hciconfig hci0 -a
    hci0:   Type: USB
            BD Address: 00:11:F6:09:ED:1C ACL MTU: 310:10 SCO MTU: 64:8
            UP RUNNING PSCAN ISCAN
            RX bytes:2934 acl:0 sco:0 events:52 errors:0
            TX bytes:438 acl:0 sco:0 commands:52 errors:0
            Features: 0xff 0xff 0x8f 0xfe 0x9b 0xf9 0x00 0x80
            Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
            Link policy: RSWITCH HOLD SNIFF PARK
            Link mode: SLAVE ACCEPT
            Name: '"Douglas-0"'
            Class: 0x4a0100
            Service Classes: Networking, Capturing, Telephony
            Device Class: Computer, Uncategorized
            HCI Ver: 2.0 (0x3) HCI Rev: 0xc5c LMP Ver: 2.0 (0x3) LMP
Subver: 0xc5c
            Manufacturer: Cambridge Silicon Radio (10)
    # hciconfig hci0 voice
    hci0:   Type: USB
            BD Address: 00:11:F6:09:ED:1C ACL MTU: 310:10 SCO MTU: 64:8
            Voice setting: 0x0060 (Default Condition)
            Input Coding: Linear
            Input Data Format: 2's complement
            Input Sample Size: 16 bit
            # of bits padding at MSB: 0
            Air Coding Format: CVSD
    # hciconfig hci0 piscan
    # hciconfig hci0 revision
    hci0:   Type: USB
            BD Address: 00:11:F6:09:ED:1C ACL MTU: 310:10 SCO MTU: 64:8
```

```
        Unified 21e
        Chip version: BlueCore4-ROM
        Max key size: 128 bit
        SCO mapping: HCI
```

O primeiro comando da lista acima o [# **hciconfig hci0 –a]**, traz informações de base como o endereço do dispositivo Bluetooth (BD *Address*), status da conexão, tipo de conexões aceitas, estatísticas de dados transmitidos e recebidos, tipos de pacotes, política para os *links*, nome do dispositivo, classe, versão de HCI e LMP. O segundo comando, [# **hciconfig hci0 voice**], trás informações sobre o tipo de dado de voz que pode ser transmitido pelo dispositivo Bluetooth. O quarto comando do exemplo, [# **hciconfig hci0 piscan],** serve para configurar o dispositivo tanto no modo ***page scan***, quanto no modo ***inquiry scan***. Nesse modo o dispositivo pode ver e ser visto por outros dispositivos Bluetooth. E o ultimo comando utilizado com exemplo [# **hciconfig hci0 revision**] mostra informação sobre a versão do chip e máximo tamanho da chave de criptografia que pode ser utilizada.

4.2.2.2 Hcitool

O comando hcitool é usado para configurar as conexões Bluetooth. Servindo também para enviar alguns comandos aos dispositivos Bluetooth. Abaixo selecionamos alguns exemplos de comandos que são mais utilizados:

```
# hcitool dev
Devices:
        hci0  00:11:F6:09:ED:1C
# hcitool inq
Inquiring ...
        00:1F:81:00:02:00 clock offset: 0x20e8    class: 0x020104
        00:21:9E:21:19:1E clock offset: 0x0e46    class: 0x5a0204
# hcitool scan
Scanning ...
        00:1F:81:00:02:00       U2
        00:21:9E:21:19:1E       K660i – Douglas
# hcitool info 00:21:9E:21:19:1E
```

```
    Requesting information ...
            BD Address:  00:21:9E:21:19:1E
            Device Name: K660i - Douglas
            LMP Version: 2.0 (0x3) LMP Subversion: 0xef
            Manufacturer: ST Microelectronics (48)
            Features: 0xbf 0xec 0x8d 0xfe 0x98 0x29 0x00 0x00
                  <3-slot packets> <5-slot packets> <encryption> <slot
offset>
                  <timing accuracy> <role switch> <sniff mode> <channel
quality>
                  <SCO link> <HV3 packets> <u-law log> <A-law log> <CVSD>
                  <power control> <transparent SCO> <broadcast encrypt>
                  <EDR ACL 2 Mbps> <EDR ACL 3 Mbps> <enhanced iscan>
                  <interlaced iscan> <interlaced pscan> <inquiry with RSSI>
                  <extended SCO> <AFH cap. slave> <AFH class. slave>
                  <3-slot EDR ACL> <5-slot EDR ACL> <AFH cap. master>
                  <EDR eSCO 2 Mbps>
    # hcitool cc 00:21:9E:21:19:1E
    # hcitool auth 00:21:9E:21:19:1E
    # hcitool con
    Connections:
            > ACL 00:1F:81:00:02:00 handle 42 state 1 lm MASTER
            > ACL 00:21:9E:21:19:1E handle 43 state 1 lm SLAVE AUTH
                  ENCRYPT
```

Nos exemplos acima, temos: [# **hcitool dev**] que mostra os dispositivos locais; [# **hcitool inq**] que informa quais são os dispositivos remotos na área de cobertura. Para cada dispositivo descoberto, um endereço de dispositivo, um relógio de *offset* e de classe são mostrados; [# **hcitool scan**] que informa os dispositivos remotos na área de cobertura e o seu nome é apresentado; [# **hcitool info 00:21:9E:21:19:1E**] que exibe entre outras coisas, o nome do dispositivo, versão e características suportadas do dispositivo remoto com o endereço Bluetooth passado; [# **hcitool cc 00:21:9E:21:19:1E**] cria uma conexão em banda base com o dispositivo remoto com endereço Bluetooth passado. O comando [# **hcitool auth 00:21:9E:21:19:1E**] deve ser executado imediatamente depois do comando **hcitool cc**, solicitando autenticação para este dispositivo. E por fim o comando [# **hcitool con**] que mostra as conexões ativas em banda base.

4.2.2.3 Hcidump

É um comando que analisa os dados HCI, lendo os dados brutos que chegam ou saem de um dispositivo Bluetooth. Opcionalmente, os dados podem ser gravados em um arquivo de texto para serem analisados posteriormente. O aplicativo Wireshark – *Network Protocol Analizer* foi utilizado em nosso projeto para identificar e analisar os protocolos utilizados na comunicação Bluetooth, obtidos através de um arquivo gerado pelo comando hcidump. Abaixo a forma geral do comando direcionando a saída para um arquivo no formato ASCII.

```
# hcidump –R -a –w teste.txt
HCI sniffer – Bluetooth packet analizar ver 1.42
device: hci0 snap_len: 1028 filter: 0xffffffff
...
```

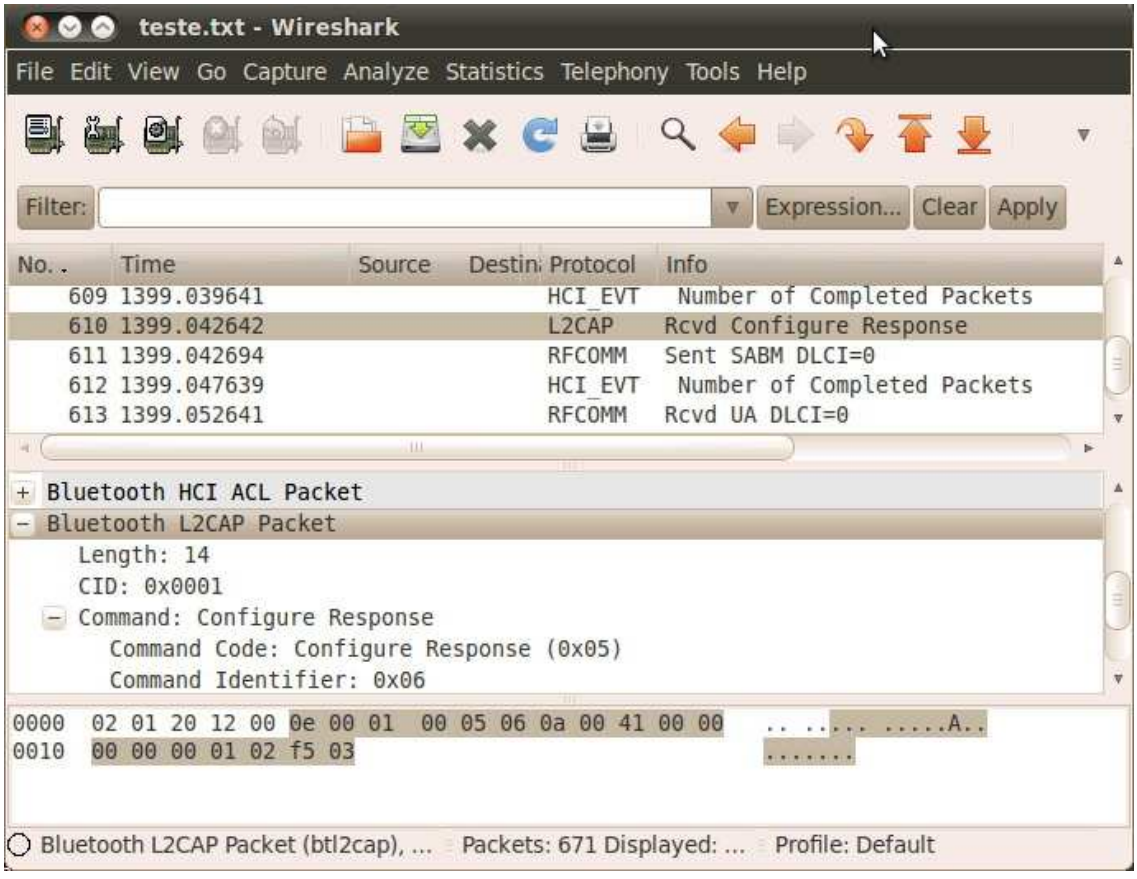


Figura 4.1 Tela do Aplicativo Wireshark – Analisador de Pacotes.

A Figura 4.1 mostra o aplicativo Wireshark abrindo uma parte do pacote L2CAP. Com esse aplicativo é possível analisar os principais pacotes da pilha Bluetooth. Ainda, é possível ver nesta mesma tela, os pacotes RFCOMM e HCI. O arquivo acima analisado, foi criado pelo comando [**# hcidump –R -a –w teste.txt**] , depois, em outro terminal , foram executados alguns comandos com hcitool e para finalizar a gravação do arquivo teclou-se [ctrl+C]. Após isso, iniciamos Wireshark e abrimos o arquivo gerado “teste.txt” para ser analisado.

4.3 *Software* Obexd

O *software* Obexd prove uma pilha de protocolos Bluetooth que pode ser utilizada para a transferência de arquivos entre dispositivos. Inicialmente o protocolo de comunicação *Object Exchange* (OBEX) foi utilizado na transferência de dados binários por dispositivos infravermelhos. Mais tarde a SIG, aproveitando-se da idéia, deu prosseguimento ao seu uso, agora, como um dos protocolos suportados pela pilha Bluetooth. O OBEX funciona sobre as camadas RFCOMM e L2CAP da banda básica e que pode ser visto na Figura 2.9. Já o *software* Obexd especificamente, implementa toda a pilha de protocolos necessária para a transferência de arquivos e foi desenvolvido pelo mesmo grupo do Bluez. Trata-se de um código aberto e está disponível para *download* no endereço: www.bluez.org. Baseado nesse *software*, o nosso sistema envia um arquivo de texto para os dispositivos Bluetooth ao alcance. A instalação dos pacotes Obexd, segue o mesmo procedimento descrito no item 4.1.1. Abaixo é mostrado o comando específico da instalação que deve ser executado no terminal do Linux:

```
# apt-get install obexftp
```

4.3.1 ObexFTP

O ObexFTP é utilizado para acessar arquivos em um equipamento móvel, como um telefone celular, por exemplo. Com um outro comando, o obexftpd, você pode transferir arquivos entre computadores utilizando infravermelho, Bluetooth ou TPC/IP. Esta ferramenta permite acessar a biblioteca ObexFTP por meio de um comando simples no terminal Linux.

As principais configurações do Obexftp podem ser vistas adiante ou digitando [# man obexftp]. Como exemplo em nosso projeto fizemos uma transferência de arquivo com sucesso entre um computador com módulo Bluetooth USB e um celular Sony Ericsson K660i, utilizando os comandos:

```
# hcitool scan
Scanning ...
        00:1F:81:00:02:00       U2
        00:23:45:90:17:D5       W380i
        00:21:9E:21:19:1E       K660i – Douglas
# obexftp –b 00:21:9E:21:19:1E –-nopath –-noconn –-put tempo.txt
Browsing 00:21:9E:21:19:1E ...
Connecting..\done
Tried to connect for 431ms
Sending “tempo.txt”...|done
Disconnecting../done
```

Os dois comandos acima foram utilizados, o primeiro [# hcitool scan] para descobrir os dispositivos Bluetooth ativos na área de cobertura, e, o segundo, para enviar o arquivo “tempo.txt” para o dispositivo 00:21:9E:21:19:1E escolhido. A opção de compatibilidade **--nopath** foi utilizada para que o próprio telefone celular determine onde o arquivo recebido deva ser gravado. A outra opção, também de compatibilidade, **--noconn** retira da conexão a identificação do usuário, não enviando o cabeçalho do pacote contendo essa informação para o móvel. Outras configurações de envio de arquivo, se fazem necessárias quando temos outras marcas de telefones móveis, PDAs entre outros. Em nosso projeto, utilizamos uma transferência com as opções predefinidas de envio. No entanto, precisaríamos desenvolver um estudo mais aprofundado se quiséssemos transmitir um arquivo de texto para qualquer dispositivo Bluetooth, não esquecendo que esse “qualquer dispositivo” precisa implementar o recebimento de arquivos, ou seja, ter o protocolo OBEX devidamente instalado. Para saber quais serviços e protocolos um determinado dispositivo possui utilizamos o comando **sdptool** que será visto em seguida.

4.3.2 SDPtool

O comando sdptool é utilizado para controlar e interrogar o protocolo SDP. Ele prove uma interface para consultas SDP nos dispositivos Bluetooth e também para administração local do SDP. Abaixo segue um exemplo de uso do sdptool:

```
# sdptool browsing 00:23:45:90:17:D5
Browsing 00:23:45:90:17:D5 ...
Service Description: Sony Ericsson W380
Service RecHandle: 0x10000
Service Class ID List:
  "PnP Information" (0x1200)

Service Name: OBEX SyncML Client
Service RecHandle: 0x10001
Service Class ID List:
  UUID 128: 00000002-0000-1000-8000-0002ee000002
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
  "OBEX" (0x0008)

Service Name: Dial-up Networking
    Channel: 2
...

Service Name: Serial Port
    Channel: 3
...

Service Name: PAN Network Access Point
Service Description: NAP provides access to internet for one connecting
PANu
...

Service Name: Hands-Free Gateway
    Channel: 4
...
```

```
Service Name: Headset Gateway
    Channel: 5
...

Service Name: OBEX Object Push
    Channel: 6
...

Service Name: OBEX File Transfer
Service RecHandle: 0x1000c
Service Class ID List:
  "OBEX File Transfer" (0x1106)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 7
  "OBEX" (0x0008)
Profile Descriptor List:
  "OBEX File Transfer" (0x1106)
    Version: 0x0100

Service Name: OBEX IrMC Sync Server
    Channel: 8
...

Service Name: OBEX Phonebook Server
    Channel: 9
...
```

O comando sdptool na opção *browsing* e com endereço de dispositivo 00:23:45:90:17:D5 traz as informações de serviços disponíveis no dispositivo Bluetooth e são apresentados na tela. Podemos observar, que para o móvel utilizado no exemplo, possui os serviços de OBEX (envio de objetos, transferência de arquivos, agenda) , rede *dial-up*, porta serial, ponto de acesso à Internet, *hands-free* (mãos livres) e *headset gateway* para áudio, entre outros. As descrições de alguns serviços foram resumidas (...) de forma a simplificar o exemplo. É importante observar que quase todos serviços possuem o campo "*Channel*" que pode ser utilizado como parâmetro pelo comando obexftp.

4.4 *Software* Aplicativo

Antes de descrever a solução encontrada para a realização da operação de envio de informações por Bluetooth, vale lembrar que um dos objetivos de nosso trabalho era “o envio de mensagens de texto através da mínima pilha de protocolos para a comunicação ponto a ponto”. Depois de desenvolvido todo um estudo teórico sobre o Bluetooth e sua pilha de protocolos, optou-se por utilizar algo que já existe e atende perfeitamente a condição estabelecida no projeto. Utilizando o protocolo OBEX podemos transmitir a informação, não na forma de mensagem de texto, mas de um arquivo de texto, atendendo perfeitamente a condição de utilização da mínima pilha de protocolos necessária para a tarefa. É importante ressaltar que a utilização do protocolo OBEX através do aplicativo ObexFTP da BlueZ é um software modular de código aberto em **linguagem C** e está disponível para *download* no endereço www.bluez.org. Portanto, a utilização posterior do código fonte, visando à implementação do nosso projeto em um sistema embarcado, por exemplo, é plenamente possível. Na seqüência do nosso trabalho, vamos mostras as soluções de *software* para receber a informação de temperatura que chega pela RS-232 e transmiti-la por meio de Bluetooth.

4.4.1 Minicom

O Minicom é programa que serve como controlador de modem em modo de texto, e também com programa para emulação de terminal para sistemas operacionais Linux. Originalmente escrito por Miquel Van Smoorenburg, foi projetado a partir do programa Telix modelado no popular MS-DOS. O Minicom inclui um diretório de discagem, ANSI e emulação VT100, entre outros recursos. Ainda, possui um menu para a programação dos modos de comunicação como, por exemplo, velocidade de transmissão, quantidade de bits de dados, *start* e *stop* bits. O típico uso do Minicom é a criação de uma comunicação serial entre dois dispositivos utilizando a RS-232. Em nosso projeto, o Minicom foi utilizado para os testes e configurações de recebimento de informações vindas do kit do 8051 para o PC através de uma comunicação serial pela RS-232. O Minicom esta disponível em várias distribuições do Linux e para instalá-lo você pode digitar:

```
# apt-get install minicom
```

4.4.2 Recebendo Informações pela RS-232

O *software* responsável pela leitura das informações da porta serial RS-232, que foram geradas pelo *hardware* coletor, cujo software está especificado no item 3.2.4.1, foi desenvolvido de maneira que as configurações sejam: modo assíncrono de 8 ou 9 bits, mais 1 *stop* bit, *baudrate* 9600 bps, sem controle de fluxo por *software* ou *hardware*. Como um *start* bit normalmente é indicado pelo nível lógico alto 1 e o *stop* bit indicado pelo nível lógico baixo 0, ter 8 ou 9 bits não faz diferença na transmissão da nossa informação.

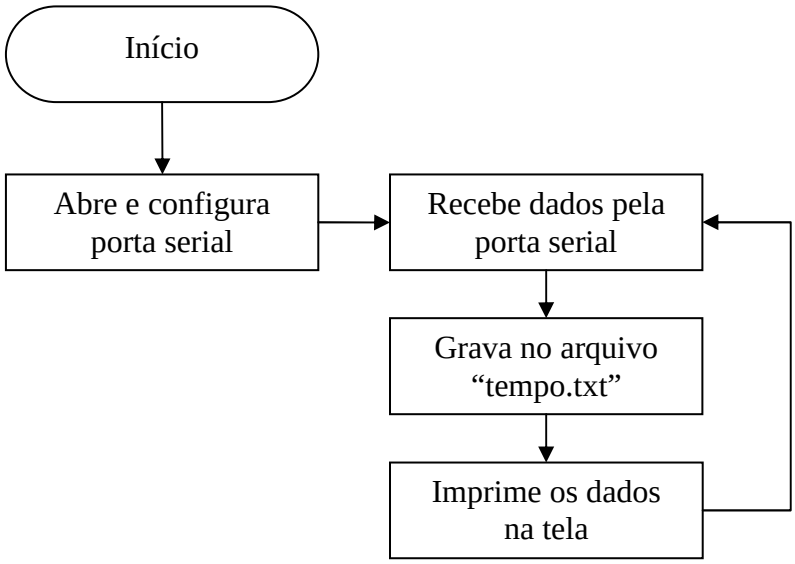


Figura 4.2 Fluxograma Geral da Recepção de Dados pela Porta Serial.

A Figura 4.2 exibe o fluxograma geral utilizado na recepção dos dados que chegam pela porta serial RS-232. Esse programa, (v. Apêndice C, p. 99), abre e configura a porta serial, recebe os dados no formato de 5 caracteres ASC, grava no arquivo "tempo.txt" e imprime na tela a informação que corresponde à temperatura coletada. O programa fica constantemente lendo os dados que chegam e sobrescrevendo o arquivo "tempo.txt" em um *loop* infinito. O tempo entre leituras está definido no programa que roda no 8051, e foi configurado entre 3 e 4 segundos. O código fonte do programa que roda no 8051 pode ser encontrado no Apêndice A, p. 93.

4.4.3 Utilizando o Shell Script

O Shell Script é uma **Linguagem de programação interpretada** usada em vários sistemas operacionais e depende do interpretador de comandos utilizado. Um interpretador de comandos muito conhecido é o ***bash***, encontrado na maioria das distribuições GNU/Linux. Na linha de comandos de um shell, podemos utilizar diversos comandos um após o outro, ou mesmo combiná-los em uma mesma linha. Se colocarmos diversas linhas de comandos em um arquivo texto simples, teremos em mãos um Shell Script, ou um script em shell, já que Script é uma descrição geral de qualquer programa escrito em linguagem interpretada, ou seja, não compilada. Outros exemplos de linguagens para scripts são o php, perl, python, javascript entre outros.

Em nossa aplicação, para facilitar a programação do envio dos dados de temperatura gravados no arquivo "tempo.txt", fizemos o uso de duas rotinas em Shell Script que selecionam os dispositivos Bluetooth na área de cobertura e em seguida dispara a transmissão do arquivo para esses dispositivos. A Figura 4.3 trás a idéia da execução dos shells script.

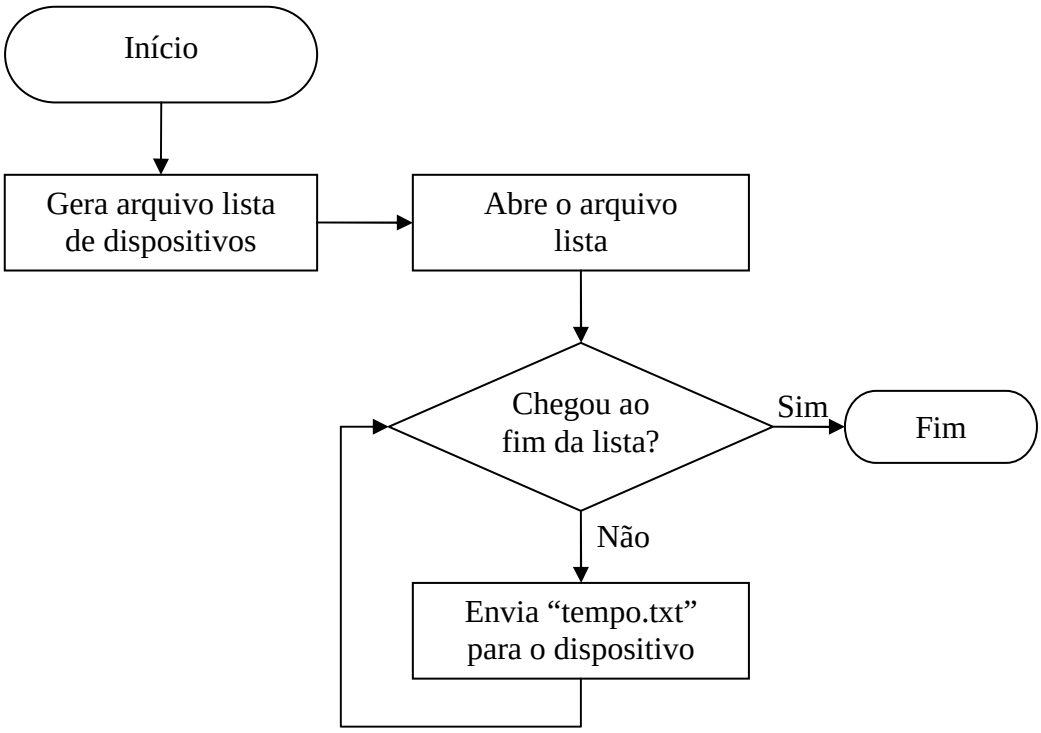


Figura 4.3 Fluxograma Geral do Envio dos Dados por Bluetooth.

O fluxograma apresentado na Figura 4.3, mostra que um arquivo contendo a lista de dispositivos Bluetooth é gerado, na seqüência, o arquivo é aberto, e, enquanto não chegar ao final do arquivo, ou seja, no final da lista, é enviado o arquivo “tempo.txt” para o dispositivo Bluetooth. O próprio Shell Script muda para o próximo da lista. Na seqüência apresentaremos os comandos que compreendem os dois Shell Scripts utilizados em nossa aplicação.

Primeiro Script: buscabt.sh

```
# !/bin/bash
# ARQUIVO: buscabt.sh
# Execução: ./buscabt.sh
#
echo "Procurando por Bluetooth..."
hcitool scan > listabt
./enviabt.sh listabt
echo "FIM."
```

Segundo Script: enviabt.sh

```
#!/bin/bash
# ARQUIVO: enviabt.sh
# Parâmetro: listabt (gerado por: hcitool scan > listabt)
# Execução: ./enviabt.sh listabt
#
nlin=1
while read linha
do
     aux=`echo $nlin $linha | cut -d" " -f2`
     if [ $nlin -ge 2 ]
     then
          echo $aux
          obexftp -b $aux --put tempo.txt
     fi
     nlin=`expr $nlin + 1`
done < $1
```

O primeiro Script se define por separar a lista de dispositivos ativos na área de cobertura. O comando **hcitool** com a opção **scan**, tem sua saída direcionada para um arquivo chamado “listabt” com o resultado da pesquisa. O resultado apresenta o endereço do dispositivo Bluetooth e seu nome. Na seqüência, o próprio Script em execução chama o outro Script “enviabt.sh” e como parâmetro é passado o nome do arquivo gerado pelo primeiro Script “listabt”. O segundo Script recebe como parâmetro o arquivo “listabt”, estabelece um número de linhas para saber a quantidade de dispositivos encontrados, a primeira linha é descartada, e, na seqüência, envia o arquivo “tempo.txt” para cada um dos dispositivos da

lista. Detalhando melhor, da lista gerada é aproveitado somente a coluna que contém o endereço Bluetooth, o Script separa esse endereço e o passa como parâmetro para o comando obexftp, que por sua vez, juntamente com a opção **--put** seguida do nome do arquivo "tempo.txt", que é gerado constantemente pelo programa da serial (item 4.3.1), envia este arquivo para cada dispositivo da lista.

É importante nesse momento relatar que o processo ainda não se completou. Ao enviar o arquivo "tempo.txt" com o comando obexftp, o mesmo, tenta abrir uma conexão com o dispositivo Bluetooth, se aceita, envia uma solicitação de consulta ao SDP, perguntando se existe o protocolo OBEX, se existe, envia o arquivo para este dispositivo e encerra a conexão. Claro que esse processo se sucede com inúmeras trocas de mensagens entre os dois dispositivos até chegar ao final com a desconexão do *link*. No Apêndice D, p. 101, podemos encontrar todas as informações trocadas pelo protocolo OBEX que envolvem a transferência de um arquivo entre dispositivos Bluetooth. Estas informações foram geradas pelo hcidump apresentado no item 4.1.2.3 deste trabalho.

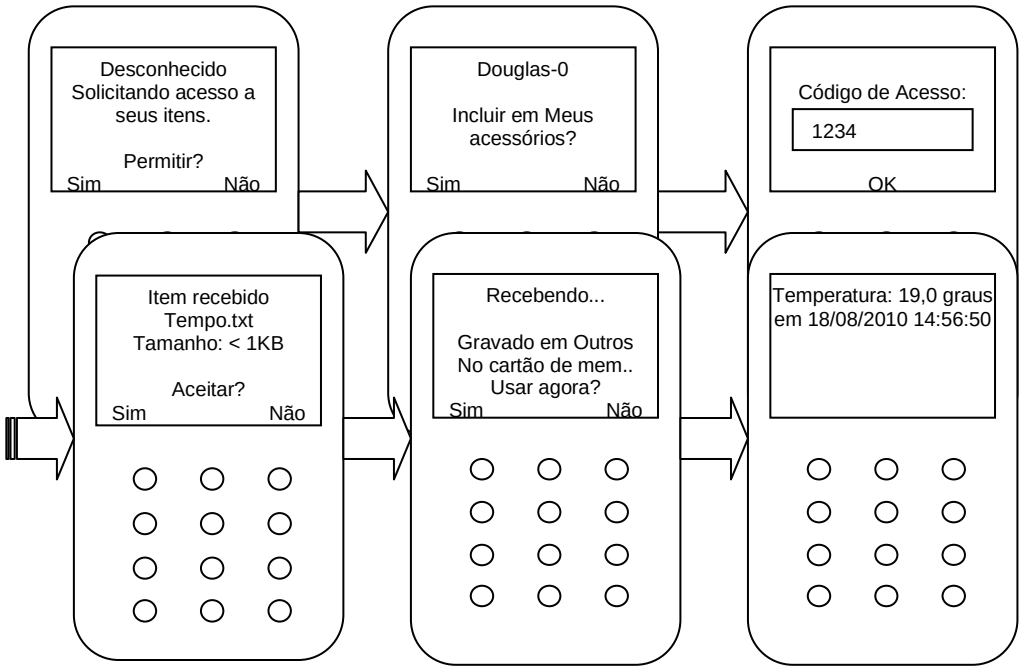


Figura 4.4 Seqüência de Mensagens no Celular.

No outro lado da conexão está o dispositivo Bluetooth que vai receber o arquivo. Utilizamos nos testes, um telefone celular marca Sony Ericsson modelo K660i. Estando o

móvel com o Bluetooth ativo, ao receber o pedido de conexão pergunta se eu devo permitir acesso aos meus itens, se sim, se eu devo incluir o dispositivo solicitante, agora com um nome, como um item de meus acessórios, se sim, solicita o código de acesso (PIN), por exemplo: 1234, depois de digitado o código de acesso, o dispositivo solicitante pergunta se aceito receber o arquivo “tempo.txt”, se aceito, o telefone identifica o lugar onde foi gravado o arquivo, no caso desse aparelho na pasta “Outros” do cartão de memória, e se eu gostaria de usar agora, se sim, é só exibir o arquivo recebido contendo a informação de temperatura coletada há poucos instantes. A Figura 4.4 ilustra esta seqüência de mensagens.

5 *Conclusão*

Este trabalho, entre outras coisas, apresentou um estudo teórico bastante completo sobre o Bluetooth, especialmente voltado para a pilha de protocolos do qual o Bluetooth é formado. O Bluetooth, como um sistema de comunicação de baixo custo, baixo consumo de energia e de fácil utilização, mostrou-se muito consistente para solucionar o problema da transmissão sem fio, das informações de temperatura a pequenas distâncias. Para tanto, o estudo nos levou ao levantamento de dois *softwares* necessários para implementar a pilha de protocolos necessários para o funcionamento do sistema, tendo o computador como *host*. Estes programas, o BlueZ e Obexd, por possuírem código livre, e por serem modulares, podem ser facilmente adaptados para fazer parte de qualquer aplicação Bluetooth, podendo ela, a aplicação, ser "embarcada" a fim de miniaturizar o *hardware*.

Outra etapa nosso trabalho, era o de utilizar um *hardware* pronto para coletar as informações de temperatura e pressão, mas não foi possível consegui-lo dentro de um prazo aceitável para a conclusão de nosso trabalho. Então, decidiu-se pela construção deste *hardware*, baseado num sensor de temperatura e um circuito conversor digital, dispensando o sensor de pressão. O desenvolvimento do *hardware*, proporcionou a aquisição da temperatura e a transmissão para um computador com ajuda do kit didático do 8051. Lembrando, uma das idéias iniciais para a execução do nosso projeto, era a utilização dos recursos disponíveis aqui, na própria instituição IFSC – Campus São José. Por esse motivo, muitas coisas que poderiam ser encontradas "prontas" não foram utilizadas e nós fomos à busca de soluções dentro das condições que nos foram colocadas. Esse trabalho mostrou-se muito "produtivo", dada a diversidade de conhecimento aplicado e deveria servir de exemplo pela multidisciplinariedade a ele empregada.

Durante a execução do projeto, tivemos que superar algumas dificuldades. A primeira delas, que consumiu algumas semanas, foi à escolha e utilização de fonte de alimentação. O circuito do conversor digital precisaria de uma alimentação simétrica de +5V e -5V. A princípio optou-se por utilizar circuitos reguladores (7805 e 7905) e várias fontes externas de tensão. O arranjo se mostrou instável e não produzia as tensões de saída desejadas, nem terra (GND) estável em 0V. Isso atrapalhou em muito a utilização dos níveis lógicos: alto e baixo,

no restante dos circuitos. Dentre as opções, trocamos de fonte várias vezes até encontrar uma Fonte AT, retirada de um computador, que poderia além de fornecer diretamente as tensões necessárias para a alimentação de +5 e -5V os 12V utilizados pelo kit do 8051, resolvendo assim, todos os problemas decorrentes da alimentação dos circuitos. Outra dificuldade apresentada, foi a utilização dos dados da saída do conversor digital que possuía uma característica diferente de outros conversores. A saída era direcionada para um *display* de 7 segmentos, anodo comum, ligados a 5V, que no caso, acendia os seus *leds* com nível lógico baixo, e o mantinha apagado com nível lógico alto. O problema era que ao querer que a informação aparecesse no *display* e ao mesmo tempo fosse enviada para o computador, mostrou-se numa condição anormal e os níveis lógicos não eram coerentes. Foi então, que descobrimos que podíamos utilizar um circuito "*buffer*" para a ligação com os *displays*, e, por possuir entrada de alta impedância, podíamos utilizar, ou dividir o sinal como num barramento, para o envio dessas informações para o computador e para os *displays*, solucionando assim o problema. Outro problema encontrado foi na utilização de um *protoboard*. Quando precisamos de mais espaço para implementação do *hardware*, tivemos que optar por um *protoboard* maior, só que o circuito parou de funcionar corretamente, depois de dias de investigações, mediu-se a resistência na continuidade das trilhas e constatou-se uma diferença de até 1KΩ com poucos centímetros de distância entre os pontos medidos. Optamos então por utilizar 2 *protoboards* da mesma marca, unidos por meio de uma chapa de madeira e fita adesiva.

Já na parte de *software*, tivemos dificuldade de estabelecer um PIN dinamicamente ao dispositivo solicitante no estabelecimento de conexões no lado do *host*. O PIN é o código de acesso a um dispositivo Bluetooth e pelo menos, na primeira conexão, ele deve ser passado e corresponder nos dois lados da conexão. Existe um serviço que pode rodar no servidor e fornecer esse PIN dinamicamente, alguns testes foram realizados, mas novos estudos devem ser promovidos.

As perspectivas para a continuação do trabalho são inúmeras. Melhor dizendo, o trabalho só está começando. A idéia é a utilização em um sistema de refrigeração e ar condicionado, portanto, precisamos que *hardware* seja pequeno, que consuma pouca energia e que possa ser "acoplado" ao equipamento de forma a fornecer as informações diretamente aos dispositivos Bluetooth na área de cobertura. Esses dispositivos poderiam ser telefones celulares, ou mesmo, que o controle remoto do aparelho fosse Bluetooth. No mercado

encontramos vários kits de programação PIC / Bluetooth e acreditamos que a saída para o estudo passa por sistemas embarcados onde a pilha de protocolos seria executada a partir do próprio PIC, ou seja, o *software* necessário para executar a pilha de protocolos Bluetooth é processado internamente no próprio PIC. Outra condição, talvez fosse a simplificação do *hardware* coletor com o uso de ferramentas de FPGA.

Ter executado esse projeto e enfrentado todos os tipos de dificuldade nos trouxe grande aprendizado e a visualização de outras soluções que poderiam ser aplicadas em trabalhos futuros. Esperamos dar continuidade ao projeto, muito, por se tratar de algo que não existe, embarcando a aplicação, diminuindo ao máximo o *hardware* necessário para coletar as informações e o adaptando aos sistemas de refrigeração e ar condicionado existente ou em desenvolvimento.

Referências

BLUETOOTH. The Official Bluetooth® Technology Info Site. Disponível no site: “http://www.bluetooth.com/English/Pages/default.aspx”. 2010.

BLUEZ Authors. Official Linux Bluetooth protocol stack. Disponível no site: “http://www.bluez.org/”. 2010.

IEEE Standards Association. IEEE 802.15: *Wireless Personal Area Networks (PANS)*. Disponível no site: “http://standards.ieee.org/getieee802/802.15.html”. 2010.

INFO WESTER. Tecnologia Bluetooth. Disponível no site: “http://www.infowester.com/bluetooth.php”. 2010.

KUROSE, James F.; ROSS, Keith W. *Redes de computadores e a Internet*. 3ª Edição. Pearson Addison Wesley, 2006.

LINUX OPERATION SYSTEM. *How to download files from cellular using Bluetooth* | Linux Operating System - Debian, Ubuntu, Fedora, Gentoo, Arch. Disponível no site: “http://www.go2linux.org/transfer-files-with-bluetooth-Linux”. 2010.

MULLER, N. J. *Bluetooth Demystified*. New York: McGraw-Hill, 2001.

NICOLOSI, D. E. C.; BRONZERI, R. B. *Microcontrolador 8051 com linguagem C*: prático e didático: família AT89S8252 atmel. São Paulo: Érica, 2008.

OLIVEIRA, A. S. de; ANDRADE, F. S. de. *Sistemas embarcados*. *Hardware* e *Firmware* na prática. São Paulo: Érica, 2006.

SWEET, M. R. Serial *Programming Guide for POSIX Operating Systems*. Disponível no site: “http://www.easysw.com/~mike/serial/serial.html”. 2010.

WIKIPÉDIA. *Shell Script*. Disponível em: “http://pt.wikipedia.org/wiki/Shell_script”. 2010.

ZELENOVSKY, R.; MENDONÇA, A. *Microcontroladores e Programação e Projeto com a Família 8051*. Rio de Janeiro: MZ Editora, 2005.

Apêndice A – Código Fonte Software 8051

```
;Projeto   : Monitoramento de Sensores de Pressão e Temperatura
;            Utilizando Redes de Sensores sem Fio - Bluetooth.
;Arquivo   : pcc3.asm
;Compilação: M-IDE Studio for MCS-51 v.0.2.5.10
;Descrição : Software embarcado no 8051 que endereça os MUX;
;            Recebe dado multiplexado e;
;            Converte de 7 segmentos para ASC e envia pela RS-232.
;
      org   0000h
      mov   tmod,#00100001b   ; timer1 modo 2 8-bit e timer0 modo 1 16-bit
      mov   th1,#0FDh         ; 9600 baud
      mov   tl1,#0FDh         ; nao precisa por causa da auto recarga
      setb  tr1               ; dispara timer/counter1
      mov   scon,#11001000b   ; serial port modo 3
inicio:
      clr   a
      mov   50h,a       ; display 1 - fração
      mov   51h,a       ; display 2 - unidade
      mov   52h,a       ; display 3 - decimal
      mov   r0,#50h           ; endereço relativo
      clr   c
      mov   r7,#00h           ; guarda leitura
      setb  p1.4              ; seta porta para leitura
      mov   r4,#00h           ; endereça MUX conta a,b 00 01 10
laco1:      mov   r5,#00h     ;              conta c,d,e 000 -> 110
laco2:      mov   a,r4
      rl    a
      rl    a
      rl    a
      orl   a,r5
      mov   p0,a              ; endereça os multiplex
      call  atraso            ; gera um atraso de 1ms ou 1khz
      mov   a,r7              ; pega conteúdo recebido
      mov   c,p1.4
      rlc   a
      setb  acc.7             ; como não é usado, fica setado em 1
      mov   r7,a              ; guarda conteúdo recebido
      inc   r5
      cjne  r5,#07h,laco2
      inc   r4
      sjmp  dig0              ; call chama como rotina dig0 e sjmp sem
                              ; condições
prox: mov   p2,@r0
      inc   r0                ; incremento o endereço relativo a r0=50h+1
      cjne  r4,#03h,laco1     ; 03 multiplex's -> 00 01 10
      call  envia             ; envia temperatura
      call  delay             ; espera uns 5 segundos para próxima leitura
      sjmp  inicio
;               xabcdefg
dig0: cjne  r7,#10000001b,dig1
      mov   @r0,#"0"    ; ASC zero 0
      sjmp  prox
dig1: cjne  r7,#11001111b,dig2
```

```
      mov   @r0,#"1"               ; vai ser o correspondente ASC
      sjmp  prox
dig2: cjne  r7,#10010010b,dig3
      mov   @r0,#"2"
      sjmp  prox
dig3: cjne  r7,#10000110b,dig4
      mov   @r0,#"3"
      sjmp  prox
dig4: cjne  r7,#11001100b,dig5
      mov   @r0,#"4"
      sjmp  prox
dig5: cjne  r7,#10100100b,dig6
      mov   @r0,#"5"
      sjmp  prox
dig6: cjne  r7,#10100000b,dig7
      mov   @r0,#"6"
      sjmp  prox
dig7: cjne  r7,#10001111b,dig8
      mov   @r0,#"7"
      sjmp  prox
dig8: cjne  r7,#10000000b,dig9
      mov   @r0,#"8"
      sjmp  prox
dig9: cjne  r7,#10000100b,erro
      mov   @r0,#"9"
      sjmp  prox
erro: mov   @r0,#'E'               ; um código para erro
      sjmp  prox
envia:      clr   ti
      mov   sbuf,52h          ; envia digito 1
env1: jnb   ti,env1
      clr   ti
      mov   sbuf,51h          ; envia digito 2
env2: jnb   ti,env2
      clr   ti
      mov   sbuf,#","         ; envia uma virgula
env3: jnb   ti,env3
      clr   ti
      mov   sbuf,50h          ; envia digito 3
env4: jnb   ti,env4
      clr   ti
      mov   sbuf,#" "         ; envia char em branco
env5: jnb   ti,env5
      ret
delay:
        mov r3,#025     ; 255 => 35 segundos, 205 => 3,5s
loop31: mov r2,#255
loop11: mov r1,#255
loop21: djnz r1,loop21
        djnz r2,loop11
        djnz r3,loop31
        ret
atraso:                 ; mov tmod,#01h   ; timer 1 em modo 1 16-bit
      clr   tf0
      mov   th0,#0FCh   ; tempo=1ms ou freq=1khz
      mov   tl0,#066h   ; 64614 ou FC66h
      setb  tr0         ; começa a contar
      jnb   tf0,$       ; espera pelo fim da contagem
      ret
      end
```

Apêndice B – Esquema Elétrico Hardware Coletor

Para o esquema elétrico do circuito coletor de temperatura, utilizamos o programa EAGLE Versão 5.10.0 para Windows (Light Edition) disponível para *download* em www.cadsoft.de/freeware.htm.

Observações: a) O esquema apresenta o circuito do ICL7106 ou invés ICL7107 utilizado na prática, por uma questão de disponibilidade do esquema elétrico somente do modelo ICL7106. O esquema elétrico foi feito como se fosse para o ICL7107. b) Das saída dos circuitos 74244 foram utilizados resistores de 470Ω não apresentados no esquema, para limitar a corrente nos *displays* de 7 segmentos. c) Foi utilizado o esquema elétrico do circuito microcontrolador AT89S51 em substituição ao kit didático do 8051 utilizado. Outras configurações adicionais serão necessárias se houver a necessidade da utilização do microcontrolador e não do kit didático.

Substituir essa página pelo A3 Impresso.

Apêndice C – Código Fonte Software Recepção de Dados pela Porta Serial

```
/*
Projeto: Monitoramento de Sensores de Pressão e Temperatura
         Utilizando Redes de Sensores sem Fio - Bluetooth.
Arquivo: serial2.c
Compilação: gcc -o ser3 serial2.c
Descricao: Recebe dados pela porta /dev/ttyS0 grava no arquivo
           tempo.txt e imprime na tela.
Data: 17/08/2010
*/
#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <time.h>    /* Utilizado pela função data_hora()*/
#include <stdlib.h>  /* Utilizado pela função data_hora()*/

#define DEFAULT_COMM_DEVICE "/dev/ttyS0"
#define DEFAULT_BAUDRATE B9600

int sd;    /* Serial port handle */
struct termios mytty;
void delay(int seg);
int fd; /* File descriptor for the port */

FILE *arq;
char *data_hora();

int open_port(void)
{
            fd = open(DEFAULT_COMM_DEVICE, O_RDWR | O_NOCTTY | O_NDELAY);
            if (fd == -1)
            {
      perror("open_port: Unable to open /dev/ttyS0 - ");
            }
            else
            fcntl(fd, F_SETFL, 0);
            configura_porta();
            return (fd);
}
int configura_porta(void)
{
      bzero(&mytty, sizeof(mytty));
      mytty.c_cflag = DEFAULT_BAUDRATE;
      mytty.c_cflag |= (CS8 | CLOCAL | CREAD);
      mytty.c_iflag = IGNBRK | IGNPAR | ICRNL;
```

```
      mytty.c_oflag = ONLCR;
      mytty.c_lflag = 0;
}
int main()
{
      char letra[5];
      int n;
      open_port();
      do
      {
      *letra=' ';
      n=read(fd, letra, 5);
      if (!(arq=fopen("./tempo.txt", "w")))
      {
            printf("\nERRO arquivo nao foi aberto!");
      } else {
            if (n>0) fprintf(arq,"Temperatura: %sgraus em
%s\n",letra,data_hora());
            fclose(arq);
      }
      if (n>0) printf("\nTemperatura: %sC. %s\n",letra,data_hora());

      } while (1);
close(fd);
return(0);
}

char *data_hora()
{
        int dia,mes,ano,hor,min,seg;
        char aux[100], *pt;
        pt=aux;
        struct tm *local; // vem de time.h
        time_t t;         // vem de time.h
        t=time(NULL);
        local=localtime(&t);
        dia=local->tm_mday;
        mes=local->tm_mon;
        ano=local->tm_year;
        hor=local->tm_hour;
        min=local->tm_min;
        seg=local->tm_sec;
        sprintf(aux,"%02d/%02d/%4d %02d:%02d:%02d",
                    dia,(mes+1),(ano+1900),hor,min,seg);
        // printf("%s",aux);
        return pt;
}
```

Apêndice D – Mensagens Trocadas pelo Protocolo OBEX.

As informações abaixo são o resultado da saída do comando hcidump para o envio do arquivo “tempo.txt” pelo comando obexftp e corresponde ao conjunto de mensagens trocadas entre os dispositivos desde o estabelecimento da conexão até seu encerramento.

```
HCI sniffer - Bluetooth packet analyzer ver 1.42
device: hci0 snap_len: 1028 filter: 0xffffffff
< HCI Command: Create Connection (0x01|0x0005) plen 13
  . . ! . ! . . . . . . . .
> HCI Event: Command Status (0x0f) plen 4
  . . . .
> HCI Event: Connect Complete (0x03) plen 11
  . * . . . ! . ! . . .
< HCI Command: Read Remote Supported Features (0x01|0x001b) plen 2
  * .
> HCI Event: Command Status (0x0f) plen 4
  . . . .
> HCI Event: Max Slots Change (0x1b) plen 3
  * . .
> HCI Event: Command Status (0x0f) plen 4
  . . . .
> HCI Event: Read Remote Supported Features (0x0b) plen 11
  . * . . . . . ) . .
< ACL data: handle 42 flags 0x02 dlen 10
    L2CAP(s): Info req: type 2
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Info rsp: type 2 result 0
      Extended feature mask 0x0000
< ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(s): Connect req: psm 1 scid 0x0040
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
< HCI Command: Remote Name Request (0x01|0x0019) plen 10
  . . ! . ! . . . . .
> ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Connect rsp: dcid 0x0000 scid 0x0040 result 1 status 2
      Connection pending - Authorization pending
> HCI Event: Command Status (0x0f) plen 4
  . . . .
> ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Connect rsp: dcid 0x0072 scid 0x0040 result 0 status 0
      Connection successful
< ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(s): Config req: dcid 0x0072 flags 0x00 clen 0
```

```
> ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Config req: dcid 0x0040 flags 0x00 clen 4
      MTU 1024
< ACL data: handle 42 flags 0x02 dlen 18
    L2CAP(s): Config rsp: scid 0x0072 flags 0x00 result 0 clen 4
      MTU 1024
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 14
    L2CAP(s): Config rsp: scid 0x0040 flags 0x00 result 0 clen 0
      Success
< ACL data: handle 42 flags 0x02 dlen 36
    L2CAP(d): cid 0x0072 len 32 [psm 1]
        SDP SSA Req: tid 0x0 len 0x1b
          pat uuid-128 00005005-0000-1000-8000-0002ee000001
          max 65535
          aid(s) 0x0004 (ProtocolDescList)
          cont 00
> HCI Event: Remote Name Req Complete (0x07) plen 255
  . . . ! . ! . K 6 6 0 i   -   D o u g l
  a s . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . .
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 14
    L2CAP(d): cid 0x0040 len 10 [psm 1]
        SDP SSA Rsp: tid 0x0 len 0x5
          count 2
          cont 00
< ACL data: handle 42 flags 0x02 dlen 22
    L2CAP(d): cid 0x0072 len 18 [psm 1]
        SDP SSA Req: tid 0x1 len 0xd
          pat uuid-16 0x1106 (OBEXObjTrnsf)
          max 65535
          aid(s) 0x0004 (ProtocolDescList)
          cont 00
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 38
    L2CAP(d): cid 0x0040 len 34 [psm 1]
        SDP SSA Rsp: tid 0x1 len 0x1d
          count 26
          record #0
              aid 0x0004 (ProtocolDescList)
                 < < uuid-16 0x0100 (L2CAP) > <
                 uuid-16 0x0003 (RFCOMM) uint 0x7 > <
                 uuid-16 0x0008 (OBEX) > >
          cont 00
```

```
< ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(s): Disconn req: dcid 0x0072 scid 0x0040
< ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(s): Connect req: psm 3 scid 0x0041
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(s): Disconn rsp: dcid 0x0072 scid 0x0040
> ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Connect rsp: dcid 0x0000 scid 0x0041 result 1 status 2
      Connection pending - Authorization pending
> ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Connect rsp: dcid 0x0073 scid 0x0041 result 0 status 0
      Connection successful
< ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Config req: dcid 0x0073 flags 0x00 clen 4
      MTU 1013
> ACL data: handle 42 flags 0x02 dlen 16
    L2CAP(s): Config req: dcid 0x0041 flags 0x00 clen 4
      MTU 1024
< ACL data: handle 42 flags 0x02 dlen 18
    L2CAP(s): Config rsp: scid 0x0073 flags 0x00 result 0 clen 4
      MTU 1024
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 18
    L2CAP(s): Config rsp: scid 0x0041 flags 0x00 result 0 clen 4
      MTU 1013
< ACL data: handle 42 flags 0x02 dlen 8
    L2CAP(d): cid 0x0073 len 4 [psm 3]
      RFCOMM(s): SABM: cr 1 dlci 0 pf 1 ilen 0 fcs 0x1c
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 8
    L2CAP(d): cid 0x0041 len 4 [psm 3]
      RFCOMM(s): UA: cr 1 dlci 0 pf 1 ilen 0 fcs 0xd7
< ACL data: handle 42 flags 0x02 dlen 18
    L2CAP(d): cid 0x0073 len 14 [psm 3]
      RFCOMM(s): PN CMD: cr 1 dlci 0 pf 0 ilen 10 fcs 0x70 mcc_len 8
      dlci 14 frame_type 0 credit_flow 15 pri 7 ack_timer 0
      frame_size 1008 max_retrans 0 credits 7
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 19
    L2CAP(d): cid 0x0041 len 15 [psm 3]
      RFCOMM(s): PN RSP: cr 0 dlci 0 pf 0 ilen 10 fcs 0xaa mcc_len 8
      dlci 14 frame_type 0 credit_flow 14 pri 7 ack_timer 0
      frame_size 1007 max_retrans 0 credits 0
< ACL data: handle 42 flags 0x02 dlen 8
    L2CAP(d): cid 0x0073 len 4 [psm 3]
      RFCOMM(s): SABM: cr 1 dlci 14 pf 1 ilen 0 fcs 0x6
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> HCI Event: Link Key Request (0x17) plen 6
  . . ! . ! .
< HCI Command: Link Key Request Reply (0x01|0x000b) plen 22
```

```
  . . ! . ! . . V V # ] p . . . . ( . > .
  . .
> HCI Event: Command Complete (0x0e) plen 10
  . . . . . . ! . ! .
> HCI Event: Encrypt Change (0x08) plen 4
  . * . .
> ACL data: handle 42 flags 0x02 dlen 8
    L2CAP(d): cid 0x0041 len 4 [psm 3]
      RFCOMM(s): UA: cr 1 dlci 14 pf 1 ilen 0 fcs 0xcd
< ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(d): cid 0x0073 len 8 [psm 3]
      RFCOMM(s): MSC CMD: cr 1 dlci 0 pf 0 ilen 4 fcs 0x70 mcc_len 2
      dlci 14 fc 0 rtc 1 rtr 1 ic 0 dv 1 b1 0 b2 1 b3 1 len 8
> ACL data: handle 42 flags 0x02 dlen 9
    L2CAP(d): cid 0x0041 len 5 [psm 3]
      RFCOMM(d): UIH: cr 0 dlci 14 pf 1 ilen 0 fcs 0x63 credits 20
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 13
    L2CAP(d): cid 0x0041 len 9 [psm 3]
      RFCOMM(s): MSC CMD: cr 0 dlci 0 pf 0 ilen 4 fcs 0xaa mcc_len 2
      dlci 14 fc 0 rtc 1 rtr 1 ic 0 dv 1 b1 1 b2 1 b3 1 len 3
< ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(d): cid 0x0073 len 8 [psm 3]
      RFCOMM(s): MSC RSP: cr 1 dlci 0 pf 0 ilen 4 fcs 0x70 mcc_len 2
      dlci 14 fc 0 rtc 1 rtr 1 ic 0 dv 1 b1 0 b2 1 b3 1 len 8
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 13
    L2CAP(d): cid 0x0041 len 9 [psm 3]
      RFCOMM(s): MSC RSP: cr 0 dlci 0 pf 0 ilen 4 fcs 0xaa mcc_len 2
      dlci 14 fc 0 rtc 1 rtr 1 ic 0 dv 1 b1 1 b2 1 b3 1 len 3
< ACL data: handle 42 flags 0x02 dlen 9
    L2CAP(d): cid 0x0073 len 5 [psm 3]
      RFCOMM(d): UIH: cr 1 dlci 14 pf 1 ilen 0 fcs 0xb9 credits 33
< ACL data: handle 42 flags 0x02 dlen 34
    L2CAP(d): cid 0x0073 len 30 [psm 3]
      RFCOMM(d): UIH: cr 1 dlci 14 pf 0 ilen 26 fcs 0xa5
        OBEX: Connect cmd(f): len 26 version 1.0 flags 0 mtu 1024
        Target (0x46) = Sequence length 16
        . . { . . < . . . N R T . . . .
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(d): cid 0x0041 len 8 [psm 3]
      RFCOMM(s): RPN CMD: cr 0 dlci 0 pf 0 ilen 3 fcs 0xaa mcc_len 1
      dlci 14 br 170 db 0 sb 1 p 0 pt 0 xi 0 xo 0
      rtri 0 rtro 0 rtci 0 rtco 0 xon 70 xoff 0 pm 0xf913
< ACL data: handle 42 flags 0x02 dlen 18
    L2CAP(d): cid 0x0073 len 14 [psm 3]
      RFCOMM(s): RPN RSP: cr 1 dlci 0 pf 0 ilen 10 fcs 0x70 mcc_len 8
      dlci 14 br 7 db 3 sb 0 p 0 pt 0 xi 0 xo 0
      rtri 0 rtro 0 rtci 0 rtco 0 xon 17 xoff 19 pm 0x3f7f
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(d): cid 0x0041 len 8 [psm 3]
      RFCOMM(s): RPN CMD: cr 0 dlci 0 pf 0 ilen 3 fcs 0xaa mcc_len 1
```

```
          dlci 14 br 170 db 0 sb 0 p 0 pt 0 xi 1 xo 0
          rtri 0 rtro 0 rtci 1 rtco 0 xon 19 xoff 127 pm 0x703f
      < ACL data: handle 42 flags 0x02 dlen 18
          L2CAP(d): cid 0x0073 len 14 [psm 3]
            RFCOMM(s): RPN RSP: cr 1 dlci 0 pf 0 ilen 10 fcs 0x70 mcc_len 8
            dlci 14 br 7 db 3 sb 0 p 0 pt 0 xi 0 xo 0
            rtri 0 rtro 0 rtci 0 rtco 0 xon 17 xoff 19 pm 0x3f7f
      > ACL data: handle 42 flags 0x02 dlen 40
          L2CAP(d): cid 0x0041 len 36 [psm 3]
            RFCOMM(d): UIH: cr 0 dlci 14 pf 1 ilen 31 fcs 0x63 credits 1
              OBEX: Connect rsp(f): status 200 len 31 version 1.0 flags 0 mtu
12288
              Connection ID (0xcb) = 14
              Who (0x4a) = Sequence length 16
              . . { . . < . . . N R T . . . .
      < ACL data: handle 42 flags 0x02 dlen 95
          L2CAP(d): cid 0x0073 len 91 [psm 3]
            RFCOMM(d): UIH: cr 1 dlci 14 pf 0 ilen 87 fcs 0xa5
              OBEX: Put cmd(f): len 87
              Connection ID (0xcb) = 14
              Name (0x01) = Unicode length 20
              . t . e . m . p . o . . . t . x . t . .
              Length (0xc3) = 48
              End of Body (0x49) = Sequence length 48
              T e m p e r a t u r a :   1 8 , 6   . g
              r a u s   e m   1 8 / 0 8 / 2 0 1 0   1
              3 : 5 0 : 4 7 .
      > HCI Event: Number of Completed Packets (0x13) plen 5
        . * . . .
      > HCI Event: Number of Completed Packets (0x13) plen 5
        . * . . .
      > ACL data: handle 42 flags 0x02 dlen 12
          L2CAP(d): cid 0x0041 len 8 [psm 3]
            RFCOMM(d): UIH: cr 0 dlci 14 pf 1 ilen 3 fcs 0x63 credits 1
              OBEX: Put rsp(f): status 200 len 3
      < ACL data: handle 42 flags 0x02 dlen 16
          L2CAP(d): cid 0x0073 len 12 [psm 3]
            RFCOMM(d): UIH: cr 1 dlci 14 pf 0 ilen 8 fcs 0xa5
              OBEX: Disconnect cmd(f): len 8
              Connection ID (0xcb) = 14
      > HCI Event: Number of Completed Packets (0x13) plen 5
        . * . . .
      > ACL data: handle 42 flags 0x02 dlen 12
          L2CAP(d): cid 0x0041 len 8 [psm 3]
            RFCOMM(d): UIH: cr 0 dlci 14 pf 1 ilen 3 fcs 0x63 credits 1
              OBEX: Disconnect rsp(f): status 200 len 3
      < ACL data: handle 42 flags 0x02 dlen 8
          L2CAP(d): cid 0x0073 len 4 [psm 3]
            RFCOMM(s): DISC: cr 1 dlci 14 pf 1 ilen 0 fcs 0xe7
      > HCI Event: Number of Completed Packets (0x13) plen 5
        . * . . .
      > ACL data: handle 42 flags 0x02 dlen 8
          L2CAP(d): cid 0x0041 len 4 [psm 3]
            RFCOMM(s): UA: cr 1 dlci 14 pf 1 ilen 0 fcs 0xcd
      < ACL data: handle 42 flags 0x02 dlen 8
          L2CAP(d): cid 0x0073 len 4 [psm 3]
            RFCOMM(s): DISC: cr 1 dlci 0 pf 1 ilen 0 fcs 0xfd
      > HCI Event: Number of Completed Packets (0x13) plen 5
        . * . . .
      > ACL data: handle 42 flags 0x02 dlen 8
```

```
    L2CAP(d): cid 0x0041 len 4 [psm 3]
      RFCOMM(s): UA: cr 1 dlci 0 pf 1 ilen 0 fcs 0xd7
< ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(s): Disconn req: dcid 0x0073 scid 0x0041
> HCI Event: Number of Completed Packets (0x13) plen 5
  . * . . .
> ACL data: handle 42 flags 0x02 dlen 12
    L2CAP(s): Disconn rsp: dcid 0x0073 scid 0x0041
< HCI Command: Disconnect (0x01|0x0006) plen 3
  * . .
> HCI Event: Command Status (0x0f) plen 4
  . . . .
> HCI Event: Disconn Complete (0x05) plen 4
  . * . .
```