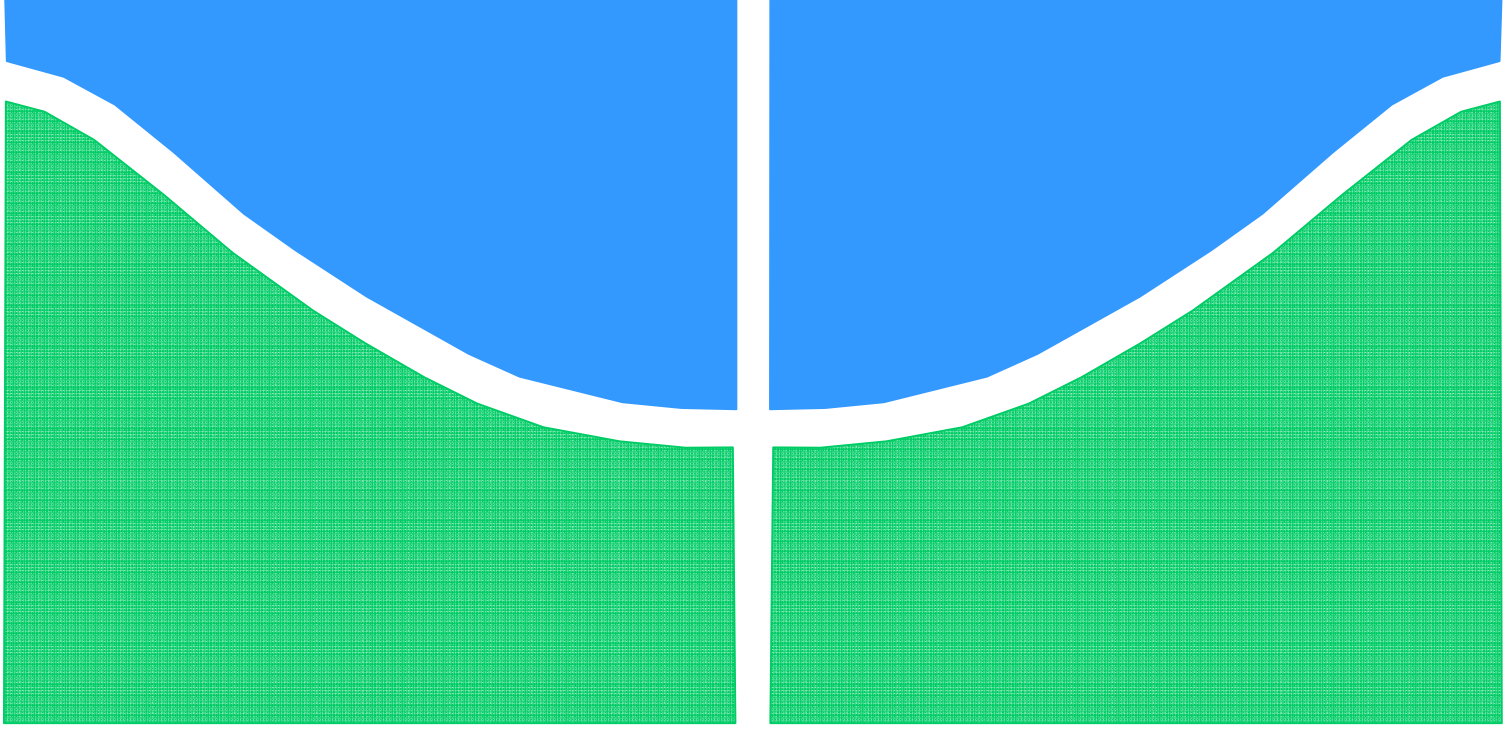

PROJETO DE GRADUAÇÃO

Implementação de Nó Móvel de Conforto Térmico

André Vidal Shinoda
Luiz Thiago Monterei dos Santos

Brasília, Março de 2010

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

Curso de Graduação em Engenharia Elétrica

PROJETO DE GRADUAÇÃO

Implementação de Nó Móvel de Conforto Térmico

André Vidal Shinoda
Luiz Thiago Monterei dos Santos

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Eletricista.

Banca Examinadora

Prof. Adolfo Bauchspiess, UnB/ENE (Orientador) _____

Prof. Ricardo Zelenovsky, UnB/ENE _____

Prof. Lélio Ribeiro Soares Junior, UnB/ENE _____

Brasília, Março de 2010

FICHA CATALOGRÁFICA

SHINODA, ANDRÉ VIDAL &
SANTOS, LUIZ THIAGO

Implementação de Nó Móvel de Conforto Térmico, [Distrito Federal], 2009.

COMPLETAR

REFERÊNCIA BIBLIOGRÁFICA

CESSÃO DE DIREITOS

AUTORES: André Vidal Shinoda, Luiz Thiago Monterei dos Santos.

TÍTULO DO TRABALHO DE GRADUAÇÃO: Implementação de Nó Móvel de Conforto Térmico.

GRAU: Engenheiro ANO: 2010

_____

André Vidal Shinoda
SQSW 102 Bloco 'E' ap 204 – Sudoeste
70670-205 Brasília – DF – Brasil

_____

Luiz Thiago Monterei dos Santos
SQSW 300 Bloco 'P' ap 203 – Sudoeste
70673-054 Brasília – DF – Brasil

RESUMO

Ambient Intelligence é um novo paradigma que leva a uma densa rede de sensores e atuadores que fornecem serviços ao usuário de forma praticamente invisível. Normalmente utilizam-se sensores *wireless* que oferecem uma grande flexibilidade em termos de implementação, em particular no *retrofitting* de prédios já em utilização.

A automação predial vem crescendo em importância e em particular redes sem fio *Zig Bee*, que foram desenvolvidas especificamente para este fim, têm sido empregadas. O Projeto FINEP-LabInov aprovado pela FT/UnB é focado em ambiente inteligente wireless.

Neste projeto deverá ser investigada a medição móvel de conforto térmico, pois é sabido que um mesmo ambiente pode apresentar diferenças significativas de conforto térmico de acordo com a posição neste. Além disso, de acordo com preferências pessoais e o tipo de atividade desenvolvida, pode haver diferentes índices de conforto térmico.

Um módulo ZigBit que contêm um microcontrolador *ATMega1281* deverá ser equipado com bateria, sensor de temperatura e sensor de batimento cardíaco e fornecer assim o índice de conforto térmico ISO7730.

Nesse projeto o enfoque será sobre o sensor de batimento cardíaco. Para tanto utilizaremos o monitor de freqüência cardíaca *S610i* da marca *POLAR*.

Palavras Chave: ZigBee, PMV, Microcontolador, Frequência Cardíaca

ABSTRACT

Ambient Intelligence is a new paradigm that remits to a dense web of sensors and actuators which provide services to users in an almost invisible way. Normally it is used wireless sensors that offer a great flexibility, when talking about implementation, in particular for retrofitting of already in use buildings.

The importance of automation for buildings is increasing every day, specially the wireless networks that have been implemented on this field.

ZigBee, one of these wireless networks, is following this process, once that this technology was developed specifically for this kind of automation. The Project FINEP-LabInov approved by FT/UnB is focused on wireless intelligent ambient.

This project will investigate mobile measures of termic comfort, since different spots of the same ambient can show significant differences of termic comfort. Beyond this, based on personal preferences and kinds of activities practiced, differents termic comfort indices can be calculated.

A ZigBit module powered by a microcontroller ATMega1281 will be equipped with battery sensor, temperature sensor and heartbeat sensor. Then it will provide the Termic Comfort Index ISO7730.

In this conclusion work the focus will be on the heartbeat sensor. It will be used the heartbeat sensor POLAR s610i.

Keywords: ZigBee, PMV, microcontroller, Heart Rate.

ÍNDICE

LISTA DE FIGURAS

NÃO ESQUECER DE COLOCAR TODAS AS REFERENCIAS NAS IMAGENS!!!!

LISTA DE TABELAS

NÃO ESQUECER DE COLOCAR TODAS AS Tabelas!!!!

LISTA DE SIGLAS

A – Amperes

A_{Du} – Área da superfície do corpo

ADC – *Analog-to-Digital Converter*

AES – *Advanced Encryption Standart*

AmI – *Ambient Intelligence*

API – *Application Programming Interface*

APS – *Application Support Sublayer*

BPM – Batimentos por minuto

Bps – Bytes por segundo

BPSK – *Binary Phase-shift Keying*

BSP – *Board Support Package*

C – Calor perdido por convecção

CI – *Circuit Integrated*

DIP – *Duap in-line package*

DSS – *Distribution System Service*

E_d – Calor perdido pela difusão de vapor de água pela pele

E_{re} – Calor perdido por respiração latente

E_{sw} – Calor perdido pela evaporação do suor na superfície da pele

ECG – Sinal Eletrocardiográfico

EE – *Energy Expenditure*

EEPROM – *Electrically Erasable Programmable Read-Only Memory*

F – Faraday

FFD – *Full Function Device*

GPIO – *General Purpose Input/Output*

H – Calor interno

HAL – *Hardware Abstration Layer*

HR – *Heart Rate*

Hz – *Hertz*

I²C – *Inter-Integrated Circuit*

IEEE – *Institute of Electrical and Electronics Engineers*

IR – Infravermelho

IRQ – *Interrupt Request*

ISO – International Organization for Standardization

ISM – *Industrial, Scientific and Medical*

JTAG – *Digital interface for debugging of embedded device*

Kg – Quilograma

L – Calor perdido por respiração seca

LAVSI – Laboratório de Automação, Visão e Sistemas Inteligentes

LED – *Light Emissor Diode*

M – Taxa metabólica do corpo humano

MAC – *Medium Access Control*

MCU – *Microcontroller Unit*

PPD – Predicted Percentage of Dissatisfied

PMV – *Predicted Mean Vote*

QPSK – *Quadrature Phase-shift Keying*

R – Calor perdido por radiação

RF – Radio Frequência

RFD – *Reduced Function Device*

RTOS – *Real Time Operation System*

SPI – *Serial Peripheral Interface*

TTL – *Transistor-Transistor Logic*

UART – *Universal Asynchronous Receiver/Transmiter*

USART – *Universal Synchronous/Asynchronous Receiver/Transmiter*

USB – *Universal Serial Bus*

W – Trabalho mecânico realizado

Wi-fi – *Wireless Fidelity*

ZDO – *ZigBee Device Object*

η – Eficiência mecânica

1 Introdução

1.1 Aspectos Gerais

Com o avanço da tecnologia, as pessoas buscam cada vez mais e mais, certo nível de conforto, sem ter que, para isso, realizar muito esforço. Com esse intuito o conceito de *Ambient Intelligence* foi criado e vem se tornando mais popular.

Dentro desse contexto, a automação predial torna isso mais acessível, provendo uma rede de sensores e atuadores para realizar diversas tarefas, entre elas, oferecer segurança, economia de energia, conforto térmico e auxilio médico.

Um conceito importante nessa área é a rede *wireless,* que vem se tornando mais empregada em vários ramos, por ter se tornado mais acessível, tornando assim a automação predial mais robusta. A implementação de redes sem fio revolucionou o jeito de fazer uma automação predial, pois, hoje em dia, não necessitamos mais realizar tantas obras e causar transtornos em uma edificação, com o uso dessa tecnologia. A essa característica se deu o nome de *retrofitting*, que nada mais é que a instalação de uma rede sem fio de sensores em uma edificação reduzindo custos de uma obra e tempo na sua implementação.

Com essa filosofia, o Laboratório de Automação, Visão e Sistemas Inteligentes – LAVSI – vem trabalhando em um projeto de automação predial. Este projeto conta com vários colaboradores e visa à instalação de uma planta piloto para o desenvolvimento de ferramentas de promoção do conforto térmico com a racionalização eficiente de energia. O estado atual do projeto é bem desenvolvido, com alguns sensores implementados e com uma economia de energia satisfatória.

Esse projeto é focado em um método de análise de conforto térmico, determinado pelo índice PMV (*Predicted Mean Vote*). O índice PMV visa estimar o conforto térmico de um indivíduo e possui sete valores, variando de -3 (que corresponde a frio) a +3 (que corresponde a quente), sendo o mesmo estimado com base na ISO 7730. Para seu cálculo, é necessária a aquisição das seguintes variáveis de conforto térmico, através de sensores: temperatura do ar, temperatura radiante média, velocidade do ar, umidade relativa do ar, tipo de vestimenta do indivíduo e o nível de atividade do indivíduo.

Estas variáveis, junto com suas respectivas implementações e medições, vêm sendo trabalhadas por equipes do projeto, separadamente em primeira instância e depois juntas, num projeto de estágio mais avançado, no controlador do aparelho de ar condicionado.

1.2 Objetivo do Projeto

O objetivo desse projeto é a implementação de um sensor de batimentos cardíacos em um ambiente inteligente com os sensores, de temperatura, umidade do ar e velocidade do ar, já implementados. A comunicação desses sensores para um supervisório se dá pela tecnologia wireless, ZigBee, IEEE 802.15.4, na qual os sensores enviam os dados para um módulo coordenador ligado à um computador com o software do supervisório instalado. Neste projeto especificamente, sinais de batimento cardíaco serão coletados, mensurados por um aparelho de medição da marca Polar, modelo S610i, depois esses dados serão tratados e enviados ao módulo coordenador. Com esses dados implementados no supervisório, será analisada a validade de nosso método no cálculo do PMV, pois o batimento cardíaco está diretamente ligado ao nível de atividade de um indivíduo.

Esse projeto está dividido em cinco capítulos da seguinte maneira: Capítulo 1 trata da introdução do projeto, falando o que será analisado e o que já foi feito previamente; Capítulo 2 trata da fundamentação teórica dos temas abordados nesse projeto e uma descrição sobre cada um destes; Capítulo 3 trata do desenvolvimento do projeto, relatando o método proposto para a realização dos objetivos propostos, a construção física e programação computacional envolvida nesse projeto. Nesse capítulo pode ser visto as imagens dos circuitos montados assim como os seus respectivos esquemas; Capítulo 4 trata da análise dos resultados obtidos por meio de gráficos e tabelas e a comparação com modelos existentes; Capítulo 5 trata da conclusão do projeto e das perspectivas futuras no ramo de atividades do projeto em estudo. Nos anexos estão apresentados códigos referentes à programação do supervisório e do microcontrolador.

1.3 Trabalhos Anteriores

Em projetos anteriores foram implementados módulos *ZigBee*, *MeshNetics* no ambiente inteligente. Atualmente há dois sensores de temperatura no ambiente e dois sensores nos módulos atuadores do ar condicionado, que também pode ser usado para validar a precisão do aparelho de ar condicionado. O desenvolvimento desse projeto utilizando esses módulos foi realizado no trabalho de graduação 'Rede de sensores sem fio para automação predial com módulos *MeshBean*, Queiroz, Azevedo'. O modelo descrito acima está representado na figura 1.3.1.

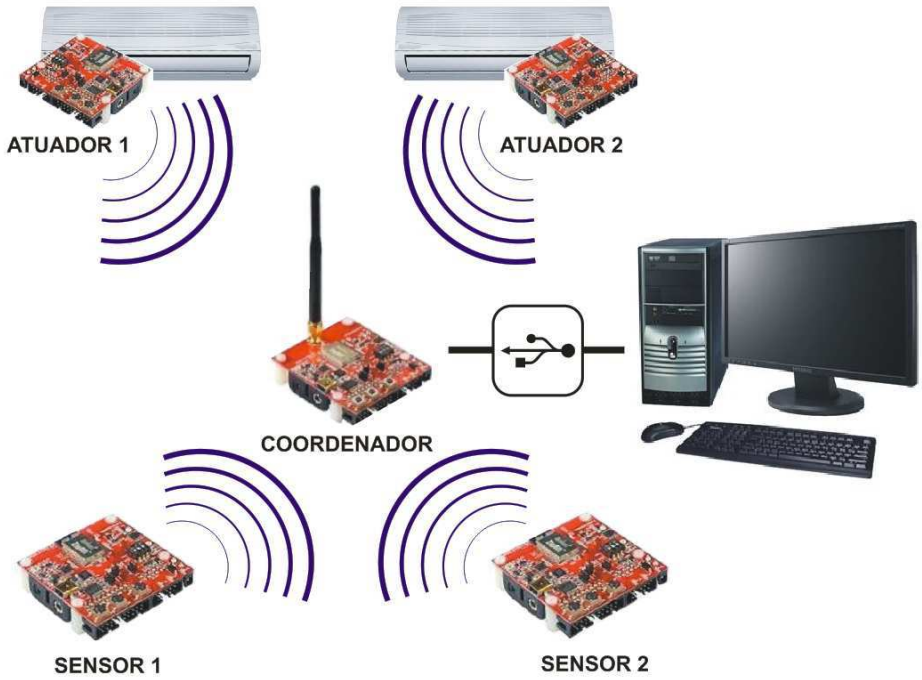

Figura 1.3.1 – Rede de sensores sem fio do projeto anterior.

Para os sensores de umidade e velocidade do vento foram utilizados módulos *ZigBee* em um circuito *Breakout* e enviados a um coordenador. O estudo desse modelo foi proposto no trabalho de graduação 'Instrumentação e controle de um sistema de ar condicionado híbrido'.

Em ambos os projetos o valor da variável nível de atividade do indivíduo, que será estudada nesse projeto, era pré-estipulado, assim como o tipo de vestimenta, no supervisório para uma faixa de valores.

O intuito é que em trabalhos futuros não seja mais necessário a inserção de valores das variáveis feitas pelo usuário no supervisório. A idéia é que o usuário chegue ao ambiente inteligente e não precise interagir com o supervisório, sendo todas as variáveis mensuradas por sensores.

2 Fundamentação Teórica

2.1 Conforto Térmico

O conceito de conforto térmico é bem amplo, não tendo como ter uma definição única. Basicamente, o conforto térmico é o estado quando o indivíduo se sente satisfeito no ambiente térmico em que se encontra. Se sentir confortável é quando não se deseja nenhuma alteração no ambiente, nem para mais frio nem para mais quente ou nem para mais seco, nem para mais úmido.

Assim, para satisfazer as pessoas em certo ambiente, tentou-se desenvolver projetos para controlar esse nível de conforto térmico. Porém, alguns empecilhos dificultaram o projeto. O corpo humano não sente os parâmetros climáticos isolados um do outro. Não é possível sentir a temperatura do ar, a velocidade de ar ou temperatura de radiação separadamente. O que se registra é o efeito de todos os parâmetros juntos, tentando regular a temperatura interna do corpo para se manter sempre constante. Além disso, fatores como o nível de atividade, tipo de roupa e geometria do lugar, interferem no conforto térmico de uma pessoa pra outra.

Para regular o conforto térmico em um ambiente, criaram-se normas que visam estimar uma sensação térmica melhor, dentro do possível, para todos da mesma forma.

Existem algumas normas que tratam desse assunto, sendo a principal e a que tomaremos por base para o projeto a ISO 7730 (*Ambientes térmicos moderados – Determinação dos índices PMV e PPD e especificação das condições de conforto térmico*). Esta norma tem uma grande aceitação acadêmica devido a sua base física, sendo a que melhor aproxima as condições de conforto com as quais trabalhamos.

Seguindo essa norma, podemos prever a sensação térmica e o grau de desconforto das pessoas expostas à ambientes de temperatura moderada, e especificar as condições térmicas aceitáveis para o conforto.

Os parâmetros que influenciam no conforto térmico são: nível de atividade física, tipo de roupa, temperatura do ar, temperatura média radiante, velocidade e umidade do ar.

Quando podemos medir ou estimar todos os parâmetros citados, podemos prever o nível de conforto térmico de uma pessoa no ambiente através do calculo de um índice chamado PMV (*Predicted Mean Vote*).

2.2 Índice PMV

O índice PMV foi criado, numa tentativa de representar a porcentagem esperada de pessoas satisfeitas para diferentes condições térmicas do ambiente. É calculado a partir de quatro variáveis físicas (temperatura do ar, temperatura média radiante, velocidade do vento e umidade do ar) e duas pessoais (nível de atividade e tipo de roupa).

O corpo humano produz calor devido ao metabolismo corpóreo que está diretamente relacionado com o nível de atividade exercida. Como existe a tendência em manter a temperatura interna do corpo constante, o homem realiza trocas de calor com o meio ambiente pelos fenômenos de condução, convecção, radiação e evaporação.

Podemos fazer uma breve análise da influência dessas variáveis no conforto térmico. O tipo de roupa utilizada influencia na troca de calor entre a superfície da pele e o ambiente. A perda de calor do corpo por evaporação depende principalmente da umidade relativa do ar, e da velocidade do vento, assim como a convecção depende da temperatura do ar e do fluxo de ar.

Assim, o PMV é calculado da seguinte forma:

$$PMV = \left(0.303E^{-0.042\left(\frac{M}{A_{Du}}\right)} + 0.028\right)\left(\frac{H}{A_{Du}} - E_d - E_{sw} - E_{re} - L - R - C\right) \quad (1)$$

Onde,

$M \rightarrow$ Taxa metabólica do corpo humano

$A_{Du} \rightarrow$ Área da superfície do corpo

$H \rightarrow$ Calor interno

$E_d \rightarrow$ Calor perdido pela difusão de vapor de água pela pele

E_{sw} → Calor perdido pela evaporação do suor na superfície da pele

E_{re} → Calor perdido por respiração latente

L → Calor perdido por respiração seca

R → Calor perdido por radiação

C → Calor perdido por convecção

Entre essas variáveis, a que interessará nesse trabalho é a medida de nível de atividade $\frac{H}{A_{Du}}$, que é calculada da seguinte maneira:

$$\frac{H}{A_{Du}} = \frac{M}{A_{Du}}(1-\eta) \qquad (2)$$

Onde o calor interno H é a soma da taxa metabólica M e do trabalho mecânico W realizado. E η é a eficiência mecânica (W/M).

Medir o nível de atividade significa medir o gasto de energia durante uma atividade. Todos os processos que acontecem no nosso corpo, tanto externamente quanto internamente, gastam energia, tornando esse tipo de medida muito difícil.

Foram criados alguns métodos para estimar esse gasto de energia. Um deles seria medir o calor que nós liberamos. Porém continua sendo muito complexo dessa maneira, pois isso tem que ser feito num ambiente totalmente fechado sob condições especiais, dificílimas de obter. Esse método denomina-se calorimetria direta. Outra forma é medir a quantidade de oxigênio absorvido pelo corpo, chamada de calorimetria indireta. Para tanto, necessita-se de um instrumento para medir o fluxo de oxigênio respirado, máscara para respiração e outras aparelhagens que se tornam inviáveis para um uso diário, como seria a idéia deste projeto.

Finalmente chegamos a outro método indireto, o qual foi utilizado neste projeto, que seria medir a taxa de batimentos cardíacos. Esse monitoramento do nível de atividade pode ser feito através dessa taxa devido uma relação existente entre consumo de oxigênio e batimentos cardíacos, apresentada num estudo de Hiiloskorpi, 1999.

Em outro estudo de Hiiloskorpi, 2003 formulou-se equações para se estimar o gasto de energia em atividades de pequena e grande intensidade em função da taxa de batimentos cardíacos.

O modelo utilizado aqui é descrito por equações distintas para homens e mulheres, e intensidades de atividade:

- Para mulheres com nível de atividade baixo (abaixo de 3METs):

$$\mathrm{EE} = -4.70 + 0.0449 \times \mathrm{HR} - 0.0019 \times \mathrm{peso} + 0.00052 \times \mathrm{HR} \times \mathrm{peso} \quad (3.7)$$

- Para mulheres com nível de atividade alto:

$$\mathrm{EE} = -5.92 + 0.0577 \times \mathrm{HR} - 0.0167 \times \mathrm{peso} + 0.00052 \times \mathrm{HR} \times \mathrm{peso} \quad (3.7)$$

- Para homens com nível de atividade baixo (abaixo de 3METs):

$$\mathrm{EE} = 4.56 - 0.0265 \times \mathrm{HR} - 0.1506 \times \mathrm{peso} + 0.00189 \times \mathrm{HR} \times \mathrm{peso} \quad (3.7)$$

- Para homens com nível de atividade alto:

$$\mathrm{EE} = 3.56 - 0.0138 \times \mathrm{HR} - 0.1358 \times \mathrm{peso} + 0.00189 \times \mathrm{HR} \times \mathrm{peso} \quad (3.7)$$

Onde,

EE é o gasto de energia em $kcal/min$

Peso em Kg

HR é a taxa de batimentos em bpm(batimentos por minuto)

1 MET = 58,15 $^{W}/_{m^2}$

O índice PMV utiliza a seguinte escala de sensação térmica:

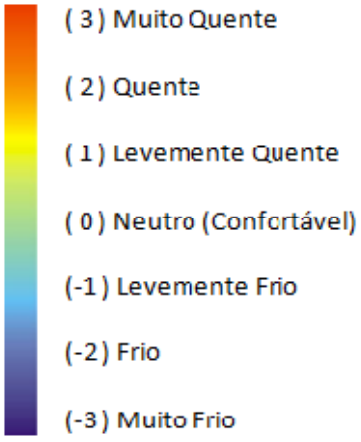

Figura 2.2.1 – Interpretação do índice PMV.

Podemos calcular o índice PPD (porcentagem de pessoas desconfortáveis termicamente) através do índice PMV:

$$\mathrm{PPD}=100\text{-}95.e^{\left(\text{-}0{,}03353.\mathrm{PMV}^4\text{-}0{,}2179.\mathrm{PMV}^2\right)} \qquad (3)$$

O que buscamos nesse projeto de ambiente inteligente é sempre manter o PMV entre -0,5 e +0,5. Nessa faixa estima-se que o número de pessoas insatisfeitas (PPD) seja inferior a 10%.

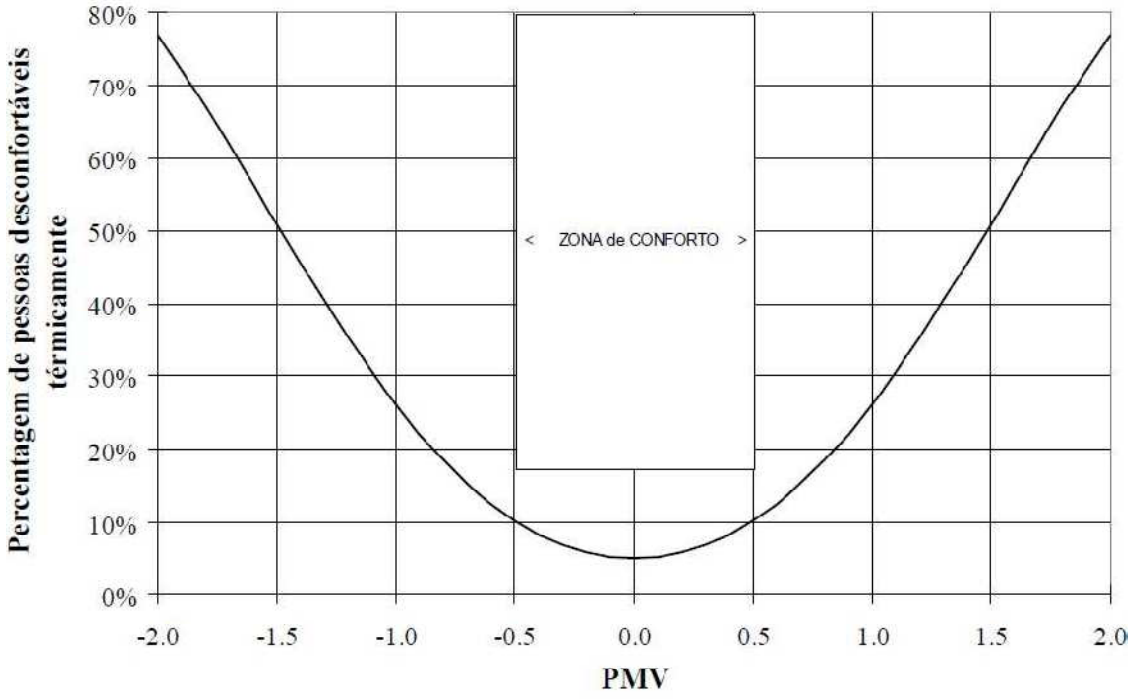

Figura 2.3.1 – Relação do índice PPD com o PMV.

O calculo mais aprofundado de cada variável e o respectivo equacionamento do índice PMV pode ser encontrado no anexo I.

2.3 Padrão ZigBee

O padrão ZigBee foi desenvolvido pela associação que conta com mais de 45 empresas denominada ZigBee Alliance. Essas empresas trabalharam em conjunto com o IEEE para se ter um protocolo para redes sem fio padrão que pudesse ter um

controle seguro, de baixo custo e baixo consumo de energia para ser usado em controle de diversos produtos e soluções, incluindo a automação predial. O padrão ZigBee, também conhecido como *HomeRF*, foi homologado em maio de 2003 como o protocolo IEEE 802.15.4.

O ZigBee é uma alternativa para a comunicação de redes que não necessitam de soluções complexas para o controle. É uma tecnologia simples, que utiliza um protocolo de pacotes de dados com certa flexibilidade, podendo assim ser utilizado por vários dispositivos.

O protocolo IEEE 802.15.4 foi desenvolvido para operar na freqüência ISM (*Industrial, Scientific and Medical*), que não requer licença para o funcionamento. No mundo ele opera na faixa de 2.4GHz- utilizando a modulação BPSK (*Binary Phase-shift Keying*), na América ele opera da faixa de 915 MHz e na Europa da faixa de 868MHz, ambas utilizando a modulação QPSK (*Quadrature Phase-shift Keying*). A taxa de transmissão para cada faixa de operação é respectivamente: 250 Kbps, 40 Kbps e 20 Kbps. O alcance de transmissão pode ser de 10m até 100m dependendo da potência dos equipamentos e dos obstáculos físicos. Na figura 2.3.1 podemos comparar vários tipos de protocolos sem fio em termos de alcance e taxa de transmissão.

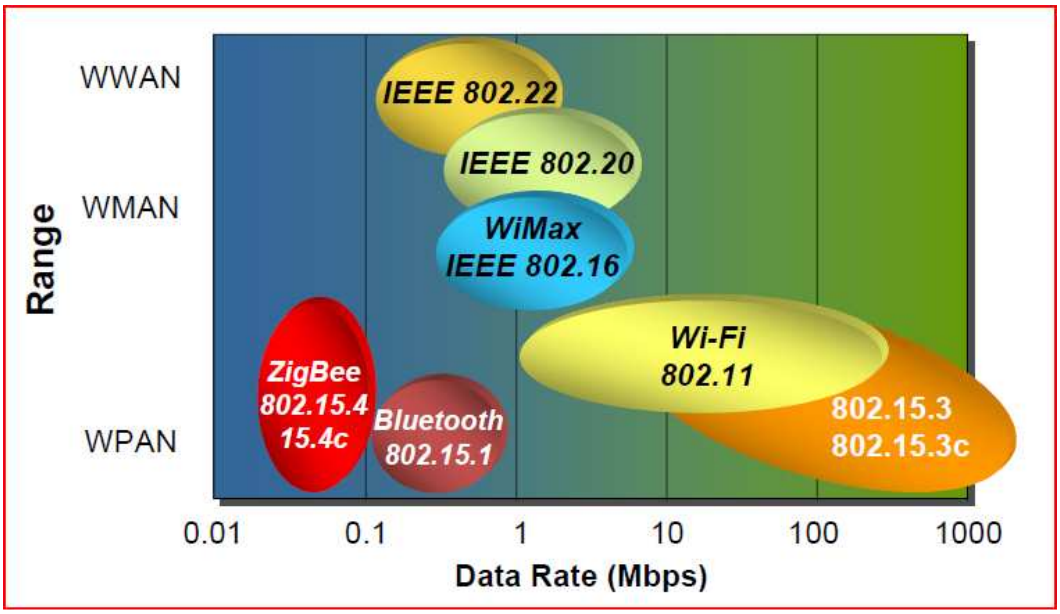

Figura 2.3.1 – Alcance *versus* Taxa de Transmissão de tecnologias *Wireless*.

Na faixa de freqüência de 2.4 GHz podemos ter 16 canais, na faixa de 915 MHz podemos ter 10 canais e na faixa de 868 MHz podemos ter 1 canal.

Outro ponto interessante é o consumo de energia, que o diferencia principalmente das tecnologias atuais de comunicação sem fio, principalmente no modo *Standby*. O ZigBee pode trabalhar no modo ativo, quando está ocorrendo transferência de dados, e no modo *Sleep*, quando não está transmitindo nenhum dado. No modo Ativo o consumo chega a 50 mA, enquanto no modo *Sleep* o consumo chega a 50 µA.

Pelo IEEE, podemos dividir os dispositivos ZigBee em duas categorias:

Full Function Device (FFD): funciona em toda a topologia do padrão, tendo a função de gerenciar a rede e ter acesso a todos os dispositivos. É um dispositivo que possui por sua natureza uma construção mais complexa.

Reduced Function Device (RFD): possui uma limitação em sua topologia, podendo somente ser configurado na topologia estrela, não podendo gerenciar a rede. Pode-se comunicar apenas com um gerenciador de rede. Possui uma construção mais simples.

De acordo com HEILE (2005), um módulo FFD pode coordenar até 254 módulos RFD e a rede inteira pode trabalhar com 65536 módulos, tornando assim um dispositivo de fácil instalação e de uma robustez satisfatória.

Para a topologia em estrela, a rede ZigBee requer pelo menos um dispositivo FDD atuando como gerenciador da rede e pode ter vários dispositivos RFD. A figura 2.3.2 mostra essa topologia:

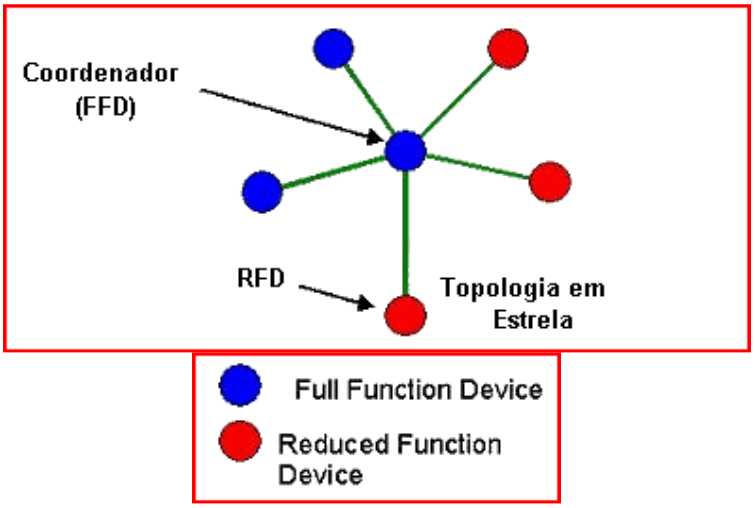

Figura 2.3.2 – Configuração da topologia estrela (PINHEIRO, 2008).

Já para a topologia ponto-a-ponto e em árvore, todos os dispositivos devem ser do tipo FFD, como mostrado na figura 2.3.3.

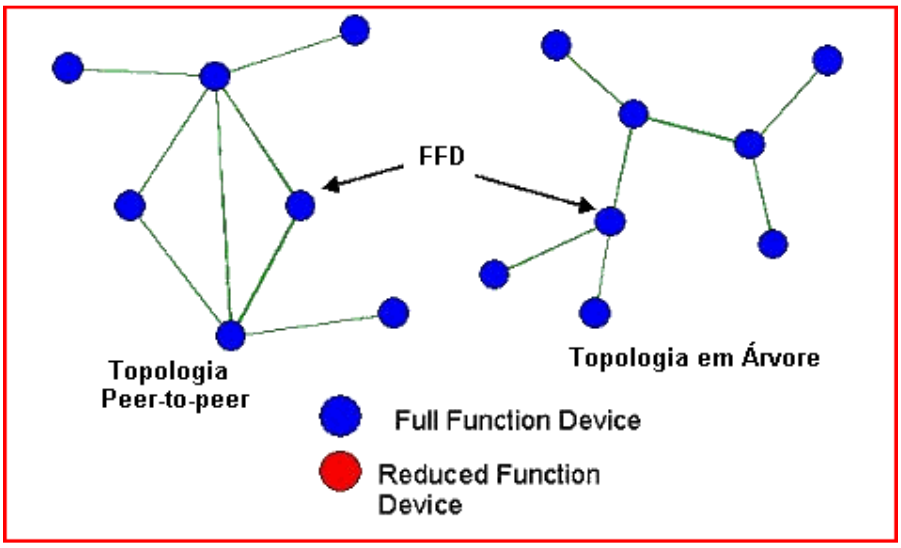

Figura 2.3.3 – Configuração das topologias Ponto-a-Ponto e Árvore (PINHEIRO, 2008).

Os dispositivos ZigBee possuem um rádio transmissor/receptor e um microcontrolador com o firmware embarcado. A arquitetura dos dispositivos ZigBee pode ser visualizada na figura 2.3.4 .

A camada física (PHY) foi projetada para acomodar as necessidades de interfaces de baixo custo, permitindo níveis de integração maiores. Permite que os equipamentos sejam muito simples pelo fato de utilizarem a técnica de transmissão de Seqüência Direta (DSS), tornando assim as implementações mais viáveis.

A camada do *Media Acess Control* (MAC) foi projetada para permitir topologias múltiplas com baixa complexidade. O MAC permite que um dispositivo RFD opere na rede sem a necessidade de grandes quantidades de memória disponíveis, podendo controlar um grande número de dispositivos sem a necessidade de colocá-los em espera. Nessa camada também que se utiliza algum tipo de segurança dos dados, sendo o controle feito em camadas superiores (BAUMANN, 2006).

Nas camadas restantes são feitas os roteamentos de rede, a segmentação de pacotes recebidos e o tratamento desses dados. Nessas camadas também se utiliza um algoritmo que permite a implementação da pilha de protocolos visando balancear os custos das unidades em aplicações e o consumo de baterias, buscando assim a melhor solução em termos de custo-desempenho para a aplicação.

Para a segurança é utilizado o padrão AES (*Advanced Encryption Standard*), realizando uma variedade de rotinas de segurança. Essas rotinas têm a função de prover a confidenciabilidade, a integridade e a autenticidade dos frames da camada MAC. Já o processo de controle é realizado pelas camadas superiores, ajustando as chaves de criptografia e determinando os níveis de segurança que deverão ser usados. Quando a camada MAC transmite ou recebe um frame, verifica-se a fonte ou destino, recupera-se a chave associada com esse destino ou fonte e então utiliza-se essa chave para processar o frame de acordo com a rotina de segurança designada para a chave que está sendo usada (PINHEIRO, 2008).

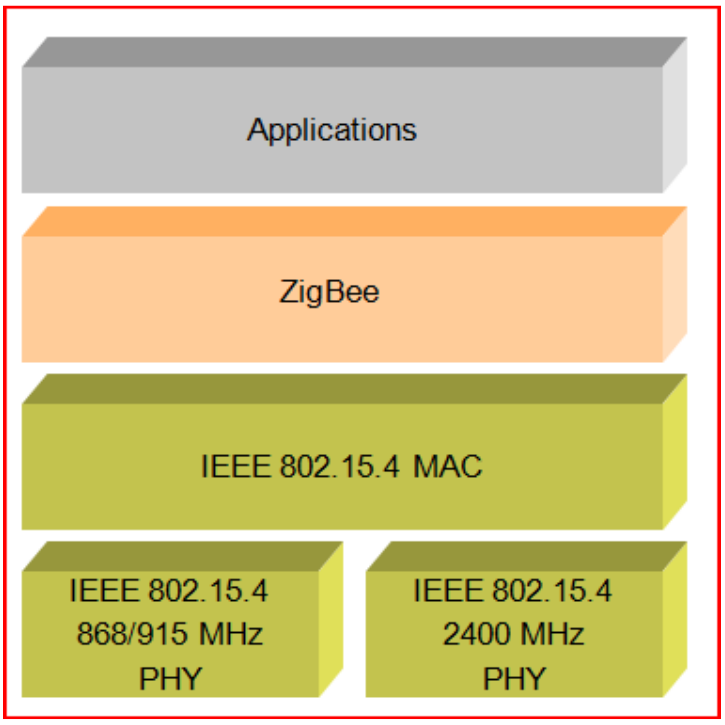

Figura 2.3.4 – Arquitetura de um dispositivo ZigBee (DVORACK, 2005).

Na tabela 2.3.1 podemos ver as especificações dos tipos de tecnologia sem fio mais utilizadas. Vemos que, para nossa aplicação, o melhor custo benefício é o padrão ZigBee, pois não necessitamos de altas taxas de transmissão e estamos interessados principalmente em baixo consumo de energia e de uma grande capacidade de pareamento de dispositivos na rede.

Tecnologia	Taxa Transmissão	Consumo (Tx/Sleep mA)	Alcance Max(m)	Tamanho Pilha	Vantagem	Desvantagem
Wi-fi	54 Mbps	400/20	300	>100KB	Altas taxas de transmissão	Alto consumo
Bluetooth	1 Mbps	50/0.2	100	100 KB	Baixo consumo	Limitação de dispositivos em rede
ZigBee	250 Kbps	50/0.05	100	34 KB/14KB	Baixo custo	Baixas taxas de transmissão

Tabela 2.3.1 – Especificações dos principais tecnologias.

2.4 Módulo ZigBit

A implementação da rede sem fio do projeto em estudo foi feita com módulos ZigBit da ATMEL, antiga MeshNetics.

O módulo ZigBit possui um encapsulamento que contém um microcontrolador ATmega1281 e o *transceiver* de radiofreqüência AT 86RF230. Pode-se, também, escolher a versão com antena integrada ao módulo ou sem antena. A versão escolhida para o projeto foi a com antena integrada, como pode ser visto na figura 2.4.1.

Figura 2.4.1 – Módulo ZigBit com transceiver e o microcontrolador.

Na figura 2.4.2 pode-se observar o modelo esquemático do módulo Zigbit.

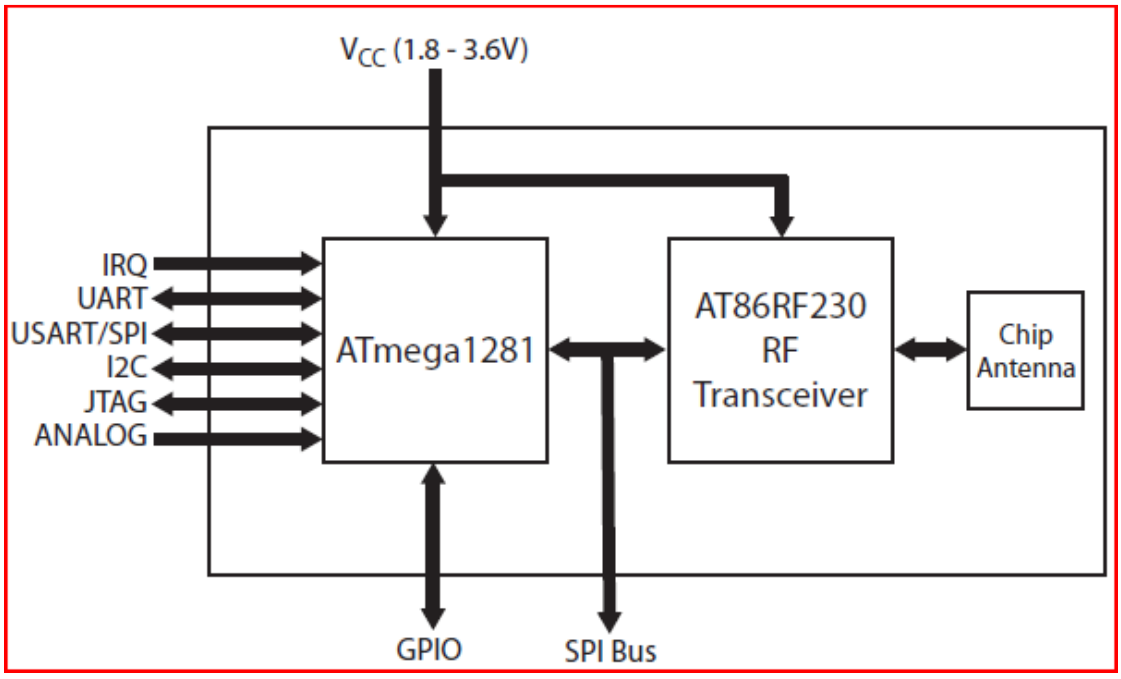

Figura 2.4.2 – Modelo esquemático do módulo ZigBit.

De acordo com o fabricante o módulo pode se comunicar com o computador através do formato de comunicação de dados serial, utilizando o protocolo UART ou USART, sendo o primeiro utilizado em modo assíncrono e o segundo em síncrono e assíncrono. Esse microcontrolador dispõe de 25 terminais GPIO, 4 terminais ADC, e dois terminais IRQ, tornando-o assim muito versátil e robusto.

Com essas características pode-se perceber que o modulo ZigBit se sobressai de outros módulos com microcontroladores integrados a dispositivos sem fio.

Na tabela 2.4.1 pode ser encontradas as especificações relevantes referentes ao módulo ZigBit.

Parâmetros	Valor	Unidade	Observações
Alimentação	1,8-3	V	
Freqüência mínima	2400	GHz	
Freqüência máxima	2483,5	GHz	
Número de canais	16	-	
Largura de Banda por canal	5	MHz	
Potência de saída	3-17	dBm	Ajustável em 16 níveis
Sensibilidade	-101	dBm	
Taxa de transferência	250	Kbps	
Impedância saída (TX)/entrada (RX)	100	Ohms	Para saída balanceada.
Rejeição de canal adjacente	27	dB	
Rejeição de canal alternado	53	dB	
Alimentação	1.8-3	V	
Consumo de corrente RX	19	mA	
Consumo de corrente TX	18	mA	
Consumo de corrente modo *sleep*	6	mA	
Memória flash	128	kB	
Memória EPROM	4	kB	
Memória RAM	8	kB	
Temperatura de operação	40-85	°C	

Tabela 2.4.1 – Especificações do módulo ZigBit.

Na figura 2.4.3 podemos observar os pinos do microcontrolador ATmega1281.

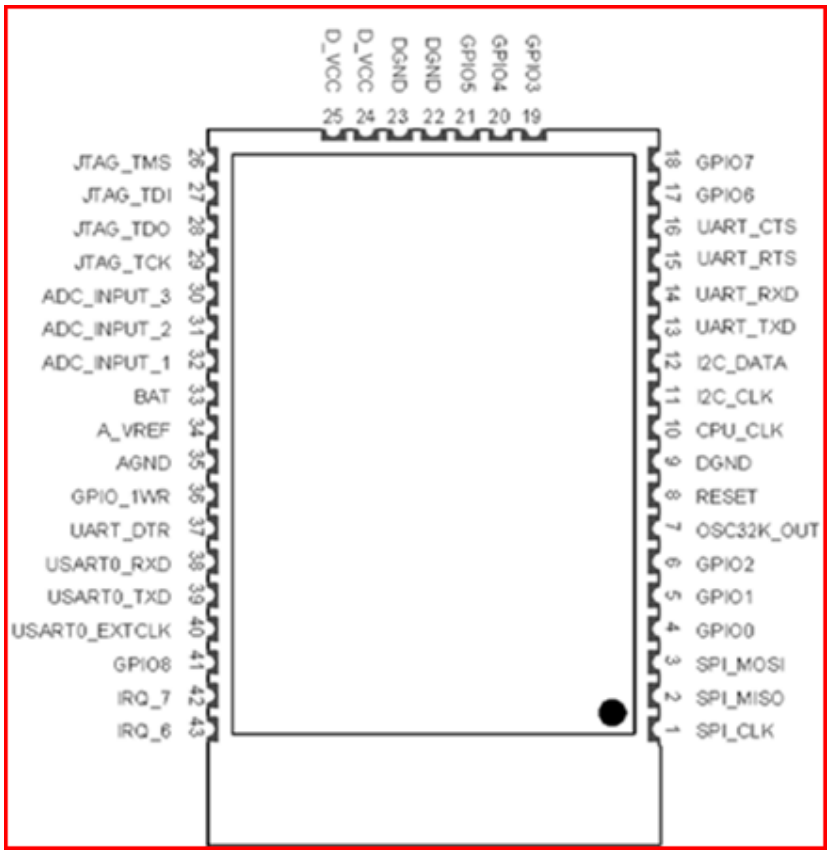

Figura 2.4.3 – Pinos do microcontrolador ATmega1281.

Como descrito no [colocar a referencia do datasheet], a tabela 2.4.2 pode ser visto o nome e a descrição de cada pino do microcontrolador, assim como sua característica (entrada I, saída O, ou ambas I/O).

Pino	Nome	Descrição	I/O
1	SPI_CLK	Reservado para a pilha de operação	
2	SPI_MISO	Reservado para a pilha de operação	
3	SPI_MOSI	Reservado para a pilha de operação	
4	GPIO0	Pino genérico digital	I/O
5	GPIO1	Pino genérico digital	I/O
6	GPIO2	Pino genérico digital	I/O
7	OSC32K_OUT	Clock 32.768 kHz	O
8	RESET	Reset (active low)	I
9,22,23	DGND	Terra Digital	
10	CPU_CLK	Clock RF	O
11	I2C_CLK	Clock Serial I2C	O
12	I2C_DATA	Dados Serial I2C	I/O
13	UART_TXD	Receptor UART	I
14	UART_RXD	Transmissor UART	O
15	UART_RTS	Permissão para enviar para Hardware UART	I
16	UART_CTS	Livre para enviar para Hardware UART	O
17	GPIO6	Pino genérico digital	I/O
18	GPIO7	Pino genérico digital	I/O
19	GPIO3	Pino genérico digital	I/O
20	GPIO4	Pino genérico digital	I/O
21	GPIO5	Pino genérico digital	I/O
24,25	D_VCC	Alimentação digital	
26	JTAG_TMS	Seleção do modo de teste para JTAG	I
27	JTAG_TDI	Teste de dados para JTAG	I
28	JTAG_TDO	Teste de dados para JTAG	O
29	JTAG_TCK	Teste de clock para JTAG	I
30	ADC_INPUT_3	Conversor Analógico/Digital	I
31	ADC_INPUT_2	Conversor Analógico/Digital	I
32	ADC_INPUT_1	Conversor Analógico/Digital	I
33	BAT	Medição de nível de bateria	I
34	A_VREF	Tensão de referencia para o conversor ADC	I/O
35	AGND	Terra analógico	
36	GPIO_1WR	Interface 1-wire	I/O
37	UART_DTR	Terminal de dados pronto para UART	I
38	USART0_RXD	Receptor USART	I
39	USART0_TXD	Transmissor USART	O
40	USART0_EXTCLK	Clock externo para USART	I/O
41	GPIO8	Pino genérico digital	I/O
42	IRQ_7	RF diferencial	I/O
43	IRQ_6	RF diferencial	I/O

Tabela 2.4.2 – Especificações dos pinos do microcontrolador ATmega1281.

Deve-se prestar bastante atenção na ligação correta do microcontrolador com a porta serial, pois o nome do pino 13, UART_TXD, se refere à recepção dos dados provenientes da porta serial. O mesmo ocorre com o pino 14, mas com relação à transmissão. Outro ponto a ser mencionado diz respeito à medição do nível de bateria através do pino 33. Para medir o nível de bateria basta dividir a tensão desse pino por três. Para maior segurança do microcontrolador todas os pinos digitais dispõem de um diodo para a proteção do mesmo.

Paralelamente, foi utilizado no projeto o kit de desenvolvimento MeshBean, que é a integração do módulo ZigBit com periféricos e sensores, como pode ser visto na figura 2.4.4.

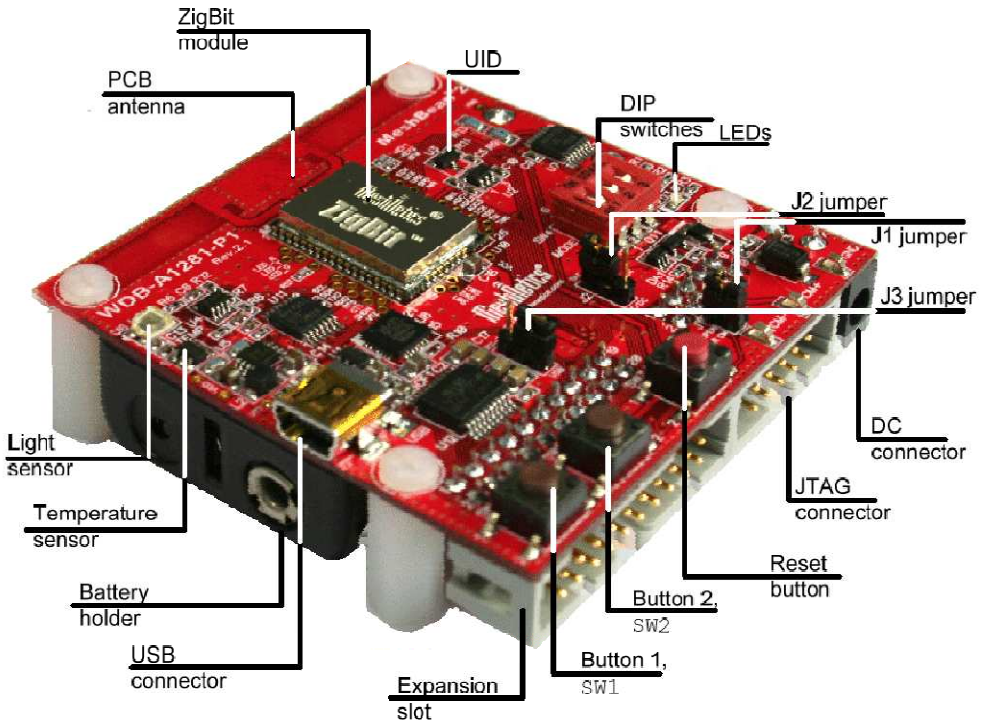

Figura 2.4.4 – Kit de desenvolvimento MeshBean.

Dentre as funcionalidades disponíveis do kit têm-se botões, DIP switches, sensores, LED's, conector para JTAG e conectores para o uso de outros recursos do ZigBit. Por dispor desses periféricos o kit de desenvolvimento é considerado mais

didático, pois permite um melhor aprendizado e aperfeiçoamento na programação do microcontrolador.

Os sensores presentes no kit de desenvolvimento são: Temperatura LM73CIMK do fabricante National Semiconductors, conectado ao barramento I^2C; Luminosidade TSL2550T do fabricante TAOS, conectado ao barramento I^2C; Nível de bateria implementado com um divisor resistivo e conectado à uma entrada ADC.

O kit de desenvolvimento pode ser conectado ao computador através da comunicação USB, utilizando o CP2102, que converte de USB para UART, do fabricante Silicon Labs. Utilizando o driver fornecido pelo fabricante é possível ver a porta de comunicação USB pode ser vista como uma porta COM.

2.5 Plataforma BitCloud

A plataforma BitCloud é um software de desenvolvimentos que gerencia os recursos computacionais com o objetivo de que todos os eventos da rotina sejam atendidos dentro do tempo esperado, tornando assim um sistema em tempo real. Softwares que gerenciam os recursos computacionais também são conhecidos como RTOS (*Real Time Operation System*). Em linhas gerais podemos dizer que o BitCloud tornou mais flexível o desenvolvimento de aplicativos do módulo ZigBee, tornando mais simples e mais prático o desenvolvimento desses aplicativos.

A separação das camadas da plataforma BitCloud é regulamentada pela IEEE 802.15.4. Além da pilha do núcleo, que implementa o protocolo, o BitCloud contém camadas adicionais para facilitar o desenvolvimento de aplicações do usuário, como a *Task Manager, Security* e *Power Manager*, camadas de abstração de hardware com *Hardware Abstration Layer* (HAL) e a *Board Support Package* (BSP) [colocar a referencia da ATMEL]. Na figura 2.5.1 pode ser observado detalhadamente essas camadas.

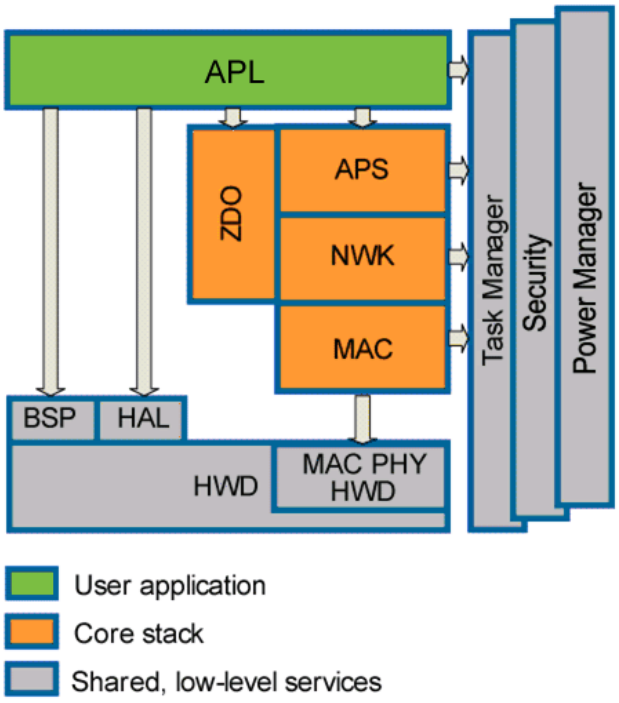

Figura 2.5.1 - Arquitetura da pilha do BitCloud.

De acordo com a BitCloud Stack Documentation [colocar a referencia do Bit Stack Doc], as principais camadas da pilha do BitCloud está descrita abaixo:

- *Application Support Sublayer* (APS): visível para a aplicação proporciona o mais alto nível das APIs relacionadas com a rede.
- *ZigBee Device Object* (ZDO): são APIs que implementam a gestão da rede e a gestão de energia. O ZDO também define o tipo do dispositivo, os comandos *device and services discovery* que permitem o dispositivo responder a requisições remotas e disponibilizam APIs para transmissão e aquisição de dados ponto a ponto, multiponto e por radiodifusão.

Existem três camadas de serviços *Task Manager, Security* e *Power Manager*. Estas funcionalidades estão disponíveis para aplicações do usuário e também podem ser utilizados por camadas de nível mais baixo.

- *Task Manager*: realiza o agendamento de tarefas que intermedeia o uso do MCU entre as camadas internas e a aplicação do usuário. O *Task Manager* utiliza um algoritmo baseado em prioridade de fila (*priority queue-based*),

especificamente desenvolvido para arquitetura de pilha multicamada e demandas de tempo crítico dos protocolos de rede.

- *Power Manager*: é responsável por desligar o dispositivo, salvar o estado do sistema quando prepara o dispositivo para dormir e restaurar o sistema quando o dispositivo acorda.
- *Hardware Abstraction Layer* (HAL): contém APIs para o uso dos recursos de hardware (EEPROM, app, sleep e watchdog timers), bem como os drivers para facilitar a integração com uma série de periféricos externos (IRQ, I2C, SPI, UART, 1-wire).
- *Board Support Package* (BSP): contém um conjunto de drivers para gerenciar os periféricos (sensores, UID chip, chaves e botões) do kit de desenvolvimento MeshBean .
- *Configuration Server*: é o componente da pilha que guarda os parâmetros de configuração chave e permite ao usuário reconfigurar esses parâmetros sem recompilar o núcleo das camadas da pilha. O usuário pode configurar parâmetros sem lidar com diversas versões de bibliotecas da pilha, por exemplo, pode-se configurar topologia de rede, network fan-in, o alcance, PAN ID, máscara de canal, etc.

3 Desenvolvimento

3.1 Sistema Proposto

Para manter um nível de conforto térmico adequado foi proposto o sistema da figura 1.3.1. Esse mesmo sistema foi proposto no trabalho Rede de sensores sem fio para automação predial com módulos MeshBean. Para a adequação a este projeto foram feitas algumas modificações e implementações.

Foi utilizado um módulo coordenador, que recebe todos os valores dos sensores e os respectivos níveis de bateria. Esse módulo foi utilizado com o kit de desenvolvimento MeshBean e se comunica com o software do supervisório no computador, através de uma porta serial.

Inicialmente, os sensores que estão dormindo acordam para realizar medidas de temperatura e bateria feitas a cada 10 segundos. Após a leitura os dados são enviados para o coordenador e os sensores voltam a dormir. A leitura de bateria possibilita ao software supervisório indicar o momento da troca das mesmas.

Os atuadores posicionados acima dos equipamentos de ar condicionado também enviam medidas de temperatura a cada 10 segundos. Essas medidas foram incluídas para comparar o sistema proposto com o sistema de controle do fabricante, uma vez que é essa a temperatura que o ar condicionado considera para acionar o compressor. [Colocar a referencia do trabalho do goiano e do luzi]

O ambiente em estudo foi parte do LAVSI, que possuía dois aparelhos de ar condicionado, dois módulos MeshBean atuando como sensores de tempertatura, dois módulos MeshBean atuando como sensores de temperatura e atuadores dos aparelhos de ar condicionado, o módulo coordenador descrito acima e um medidor de batimentos cardíacos, que não será mostrado na figura pois ele pode estar em qualquer parte do ambiente. A limitação do medidor de batimentos cardíacos está na condição de que o usuário sempre transportará consigo o módulo receptor e não ultrapassará os limites de alcance do módulo ZigBit.

A configuração para esses sensores foi feita de acordo com a figura 3.1.1.

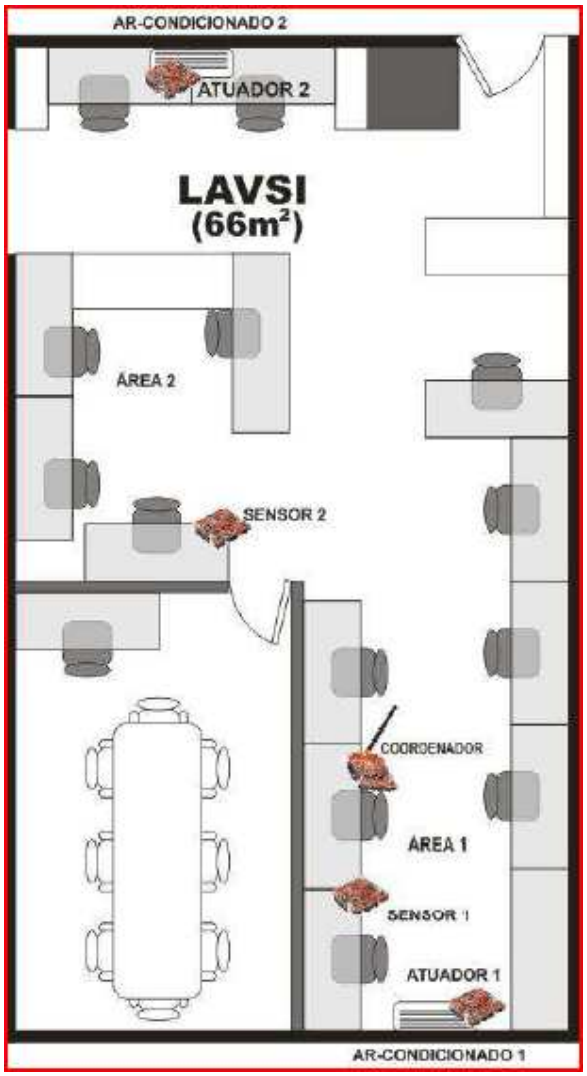

Figura 3.1.1 – Ambiente em estudo do projeto com os sensores e atuadores.

Para cada módulo apresentado na figura 3.1.1 e para o medidor de batimentos cardíacos, há um código instalado no microcontrolador, que para diferentes atribuições (coordenador, sensor, atuador, medidor de batimentos cardíacos) realizará diferentes tarefas.

Para os módulos coordenador, sensor e atuador, que foram desenvolvidos com o kit de desenvolvimento MeshBean o código será o mesmo e a distinção de cada módulo será feita através do posicionamento das chaves do DIP Switches. Já para o medidor de batimentos cardíacos será desenvolvido um código especial, diferente dos demais, mas com o mesmo compromisso (transmissão de dados e economia de energia).

No campo da programação, o programa AVR Studio 4 foi utilizado para desenvolver os algoritmos que estão presentes nos microcontroladores dos

módulos. Esse algoritmo é dividido em três partes: a primeira referente à estrutura do programa, que é denominada *lowpower.c;* a segunda se refere ao módulo coordenador, denominada *coordinator.c*; a última é referente ao demais módulos, denominada *enddevice.c*.

No algoritmo principal, *lowpower.c*, está determinado os estados nos quais os módulos poderão estar na rede. Esses estados são:

- *APP_INITIAL_STATE:* corresponde ao estado inicial da aplicação, que é no momento em que se liga o módulo ou quando se reinicia o mesmo.
- *APP_NETWORK_JOINING_STATE:* corresponde ao momento que ocorre a requisição para que o módulo se conecte à rede.
- *APP_NETWORK_JOINED_STATE:* corresponde ao momento que a rede foi estabelecida e o módulo encontra-se inserido nela, interagindo com os demais módulos.

Após determinar os estados dos módulos, o aplicativo definirá o tipo de aplicação de cada módulo, invocando as funções referentes a cada módulo. O tipo de aplicação tem duas possibilidades:

- *DEVICE_TYPE_COORDINATOR:* momento em que é definido o módulo como coordenador e invocado a função pertinente a esse modulo.
- *DEVICE_TYPE_END_DEVICE:* momento em que é definido o módulo como *end device* e invocado a função pertinente a esse módulo.

Na figura 3.1.2 pode-se observar o diagrama de estados presente no algoritmo *lowpower.c.*

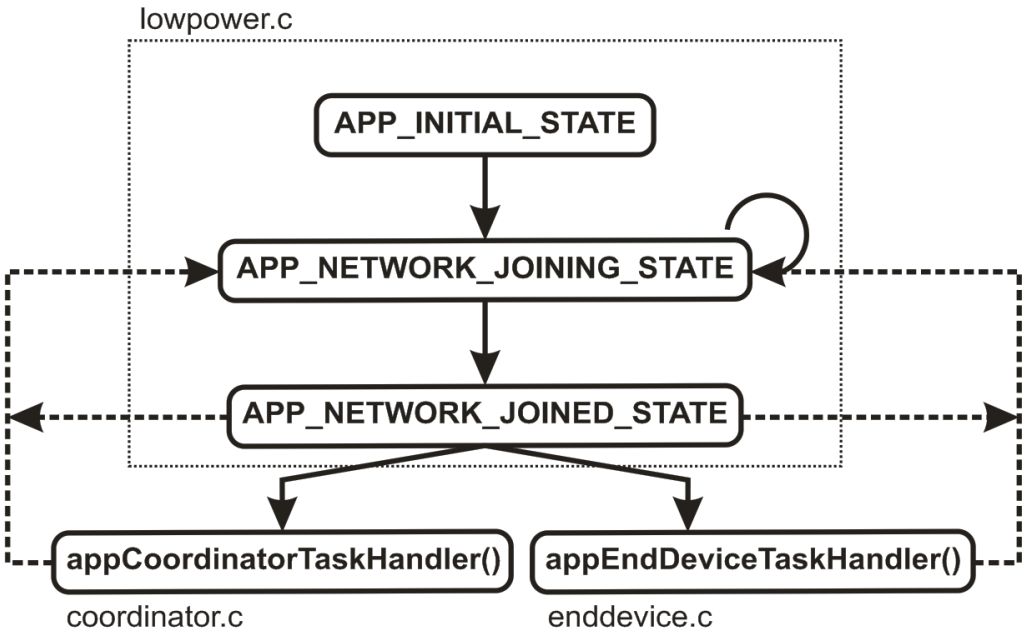

Figura 3.1.2 – Diagrama de estados presente no algoritmo *lowpower.c*.

Para o reconhecimento do tipo do módulo, é feita uma leitura dos DIP Switches, como dito anteriormente. A configuração para cada módulo está apresentada na tabela 3.1.1 .

Função	Endereço dos DIP Switches
Coordenador	0
Sensor 1	1
Sensor 2	2
Atuador 1	3
Atuador 2	4
Medidor de batimentos cardíacos	5

Tabela 3.1.1 – Configuração de funcionamento dos módulos.

Como foi dito anteriormente, o módulo medidor de batimentos cardíacos não possui DIP Switches, sendo assim já estabelecido no algoritmo o seu endereço.

No algoritmo do módulo coordenador, *coordinator.c*, os dados provenientes dos outros módulos são identificados e preparados para serem enviados para o supervisório, através da porta serial. Os dados provenientes dos outros módulos são:

- temp_sensor_1
- temp_sensor_2
- temp_atuador_1
- temp_atuador_2
- bat_sensor_1
- bat_sensor_2
- bpm
- bat_bpm

No trecho abaixo é onde é feita a distinção dos dados e o recebimento dos mesmos para as variáveis citadas acima.

```
void appCoordinatorDataInd(APS_DataInd_t* ind)
{
  uint8_t str[100];
  uint8_t length;

  AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;
  remetAdress = ind->srcAddress.shortAddress;

  if(ind->srcAddress.shortAddress == 1)
  {

        temp_sensor_1 = appMessage->temperature;
        bat_sensor_1 = appMessage->battery;
  }
  if(ind->srcAddress.shortAddress == 2)
  {

        temp_sensor_2 = appMessage->temperature;
        bat_sensor_2 = appMessage->battery;
  }
  if(ind->srcAddress.shortAddress == 3)
  {
        temp_atuador_1 = appMessage->temperature;
```

```
        bat_atuador_1 = appMessage->battery;
        status_atuador_1 = appMessage->porta;
    }
    if(ind->srcAddress.shortAddress == 4)
    {
        temp_atuador_2 = appMessage->temperature;
        bat_atuador_2 = appMessage->battery;
        status_atuador_2 = appMessage->porta;
    }

    // Para o modulo receptor de batimentos cardiacos
    if(ind->srcAddress.shortAddress == 5)
    {
        bpm = appMessage->batimento;
        bat_bpm = appMessage->battery;
    }

}
```

Nos módulos atuador_1 e atuador_2 é também encontrado a variável porta, na qual nos indica o estado das portas de atuação. Essas variáveis pode assumir os valores indicados na tabela 3.1.2.

Atuador_1	Atuador_2	Comando para split 1	Comando para split 2
0	0	Off	Off
1	0	On	Off
0	1	Off	On
1	1	On	On

Tabela 3.1.2 – Valores assumidos pelas portas dos atuadores.

Esses comandos indicam se os atuadores devem ou não ligar os compressores. No capítulo 4 essas atuações poderão ser vistas no supervisório.

Resumindo, o módulo coordenador, quando ligado, tentará formar uma rede com os módulos restantes (sensores, atuadores e medidor de batimentos cardíacos). O modo que ele tentará realizar a sincronização entre os módulos será

de forma automática, não precisando pressionar nenhum botão. Após ser identificado como módulo coordenador é chamada a função referente a esse módulo. Logo após, o módulo recebe as informações dos outros módulos, envia serialmente para o supervisório e, dependendo da situação encontrada, manda o sinal para a atuação dos atuadores. Após esse ciclo começa tudo novamente, funcionando ininterruptamente

O algoritmo *enddevice.c*, é usado para os sensores, atuadores e medidor de freqüência cardíac, sendo diferenciado pela leitura dos DIP Switches, como dito anteriormente. A parte de inicialização é igual para todos os módulos. Ao entrar na rede e ser reconhecido o módulo estará em algum dos estados apresentados abaixo.

- *DEVICE_ACTIVE_IDLE_STATE:* Estado no qual o módulo já está inserido na rede e já está realizando as medições.
- *DEVICE_MESSAGE_SENDING_STATE:* Estado após a aquisição dos dados. Com os dados nas respectivas variáveis é realizado o processo de envio para o módulo coordenador.
- *DEVICE_SLEEP_PREPARE_STATE:* Estado no qual o módulo faz a requisição para poder dormir, economizando energia e esperando o próximo ciclo de aquisição de dados.
- *DEVICE_AWAKENING_STATE:* Estado em que é feita a requisição para que o módulo se torne ativo, realizando o ciclo descrito nos estados acima.

O ciclo é repetido em 10 segundos, período no qual novos dados serão recebidos no módulo coordenador.

Nos sensores as variáveis adquiridas são as temperaturas e o respectivos níveis de bateria. Quando ocorre o envio dessas informações a luz vermelha acende, indicando a transferência desses dados para o coordenador.

Já nos módulos atuadores são adquiridas, além das temperaturas, o estado das portas dos mesmos. O trecho do algoritmo referente ao acionamento dos compressores é encontrado abaixo.

```
void appEndDeviceDataInd(APS_DataInd_t* ind)
{
```

```
  uint8_t control = 0;
  AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;
  ind = ind; // Warning prevention

  control = appMessage->control;

  if(control == 1)
  {
    BSP_OnLed(LED_YELLOW);
        GPIO_ADC_INPUT_1_set();
  }
  if(control == 0)
  {
    BSP_OffLed(LED_YELLOW);
        GPIO_ADC_INPUT_1_clr();
  }

}
```

Quando há o sinal de acionamento dos compressores, a luz amarela acende, indicando o funcionamento do ar condicionado. Da mesma forma, quando é desligado, a luz também desliga. A instrução que realiza a ativação dos compressores é a seguinte:

GPIO_ADC_INPUT_1_set();

Na instrução acima é indicado o tipo de configuração para a porta do microcontrolador, GPIO, a porta específica, *ADC_INPUT_1* e ativação ou desativação. Para a ativação é usado o comando *set()* que coloca a porta em nível alto. Para a desativação, deve-se utilizar o comando *clr(),* que leva a porta para o nível baixo.

O módulo receptor de batimentos cardíacos é o que tem uma função mais particular e deve ter um código somente para realizar tal tarefa. Por não possuir DIP Switches, já foi atribuído o valor de 5 para o endereço do mesmo através da seguinte instrução:

nwkAddr = 5;

Desse modo é atribuído o valor 5 para o endereço da rede, de acordo com a tabela 3.1.1.

Após feita a sincronização desse dispositivo na rede começa o processo de aquisição dos batimentos. Para tanto foi utilizado os conceitos de interrupção no microcontrolador. Falaremos mais a respeito do batimento cardíaco nos tópicos abaixo.

Sabemos que quando o coração encontra-se em funcionamento normal são gerados dois pulsos (sístole e diástole) em um ciclo. Com a utilização de um dispositivo capaz de captar o batimento cardíaco e gerar pulsos elétricos, que será apresentado nos tópicos adiante, pode-se receber esses pulsos. Sabendo o tempo entre os pulsos gerados pelo coração podemos calcular o valor do batimento cardíaco. Para o correto funcionamento devemos receber o primeiro batimento e o terceiro, que é no momento em que começa a repetir o ciclo do coração.

A idéia de utilizar a interrupção serve para o microcontrolador avaliar o estado das portas em estudo constantemente. Quando ocorre o primeiro pulso o microcontrolador realiza a interrupção, que é habilitada quando o pino IRQ_6 vai para zero. Feito a interrupção, o tempo que esse evento ocorreu fica armazenado em uma variável. Quando ocorre o outro batimento, no caso o terceiro batimento, ocorre a interrupção gerada pela porta IRQ_7. Nesse momento é adquirido o tempo e calculado a diferença de tempo entre as duas interrupções ocorridas. Com operações de conversão de unidades chegamos ao valor do batimento.

O trecho do código referente às instruções e o cálculo dos batimentos está apresentado abaixo.

```
/*********************************************************************
 Description:  Interrupcao
 Parameters:   none
 Returns:      none
*********************************************************************/

void funcao2 (void)
{

 HAL_DisableIrq(IRQ_7);
 cont++;
 if (cont == 4)
 {
  HAL_DisableIrq(IRQ_7);
       BSP_OffLed(LED_YELLOW);
  bpm2 = HAL_GetSystemTime();
  bpm22 = bpm2 ;
  bpm3 = (bpm22 - bpm);
  bpm3 = bpm3/1000;
```

```
    bpm3 = bpm3/60;
    batimento = (int) (3 + (1/bpm3));
          cont = 0;
    appMessageBuffer.message.batimento = batimento;
    appDeviceState = DEVICE_MESSAGE_SENDING_STATE; //Switch device state
to application message sending
    SYS_PostTask(APL_TASK_ID); // Application task posting

  }
  else
    HAL_EnableIrq(IRQ_7);

}

void funcao (void)
{

  HAL_DisableIrq(IRQ_6);

  BSP_OnLed(LED_YELLOW);
  bpm1 = (int) HAL_GetSystemTime();
  bpm = bpm1;

}


static void testeInterrupcao(void)
{
  HAL_RegisterIrq (IRQ_6 , IRQ_RISING_EDGE , funcao);
  HAL_EnableIrq(IRQ_6);

}

static void testeInterrupcao2(void)
{
  HAL_RegisterIrq (IRQ_7 , IRQ_RISING_EDGE , funcao2);
  HAL_EnableIrq(IRQ_7);

}
```

Na função funcao ocorre a primeira interrupção e é armazenado o tempo em que ocorreu esse evento através da seguinte instrução:

bpm1 = (int) HAL_GetSystemTime();

Na função funcao2 ocorre a segunda interrupção. Antes de mais nada devemos rejeitar os pulsos indesejados para o cálculo, utilizando um contador e uma instrução condicional, cont e if respectivamente. Quando os pulsos são gerados eles são enviados paras as duas portas do microcontrolador. Para a segunda interrupção deve-se rejeitar o primeiro e o segundo pulsos, capturando apenas o terceiro.

Porém, após vários testes, constatamos que o ideal seria colocar a interrupção para ocorrer no quarto pulso, tornando o nosso sistema mais estável. Também foi necessário somar ao valor final do batimento três unidades pois observamos que assim o valor do batimento ficava praticamente igual ao o do relógio do fabricante, que será citado em tópicos posteriores.

Fazendo as operações matemáticas necessárias, encontramos os valor do batimento.

O algoritmo também fornece informações sobre a bateria. Com posse de todos os dados, o módulo é capaz de enviar ambos os dados para o módulo coordenador e fazer a requisição para poder sair do estado ativo e desativar, economizando energia.

O código completo referente aos módulos pode ser encontrado no anexo AII.

O processo de gravação e o hardware necessário para tal função estão detalhados nos próximos tópicos.

3.2 Batimento Cardíaco

Como explicado previamente, o batimento cardíaco foi utilizado neste projeto como a variável responsável pelo cálculo do nível de atividade de um indivíduo. Entraremos um pouco no campo da biologia para explicar este fenômeno.

O batimento cardíaco é um evento associado com o ciclo cardíaco, ele demarca o início e o fim deste. O ciclo cardíaco é a seqüência de eventos, que ocorrem entre um batimento e outro, responsável pelo fluxo sanguíneo, também conhecido como pressão sanguínea.

O coração bate para circular o sangue pelas células do corpo. Resumidamente ele se contrai para ejetar o sangue oxigenado em direção as artérias, essa fase é chamada de sístole; e se relaxa para receber o sangue proveniente das veias, na fase chamada de diástole.

O ciclo cardíaco é controlado por uma série de impulsos elétricos produzidos por células especiais localizadas no próprio coração.

A freqüência do ciclo cardíaco é o que chamamos de taxa de batimentos (HR – Heart Rate), medida em bpm. Geralmente, sob condições normais, cada ciclo dura

em torno de 1 segundo. Porém a HR pode variar sob influência de hormônios, exercícios físicos e emoções.

A HR pode ser facilmente medida através da pulsação em qualquer ponto do corpo humano no qual existe uma artéria próxima a superfície da pele. O pulso, o pescoço e o lado esquerdo do peito são os lugares mais comuns para se medir esta pulsação.

Existem sensores que se baseiam nesta idéia de medir pulsação. Utilizam materiais bem sensíveis a movimentos mecânicos, detectando-os na superfície da pele. Porém por serem sensíveis, a medição tem que ser feita no músculo totalmente imóvel. Ou seja, o indivíduo deve se manter imóvel durante a medição. Este tipo de medição só serve para medições rápidas, o que não é o caso deste trabalho.

Logo foi utilizado outro tipo de medição para a HR.

A figura 3.2.1 representa a principal componente para o desenvolvimento deste projeto.

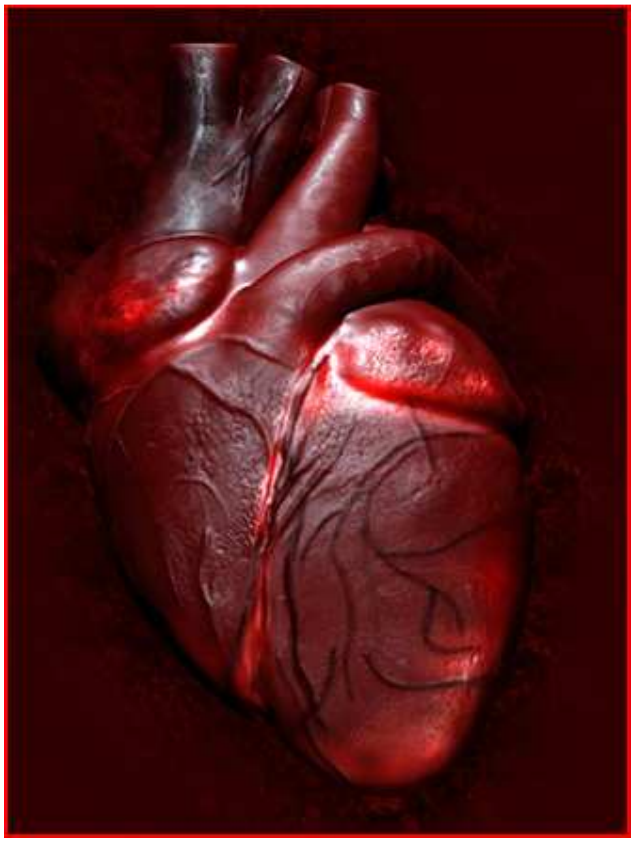

Figura 3.2.1 – Ilustração de um coração humano - indicador do nível de atividade física neste projeto

3.3 Monitor de HR

A medição completa da HR consiste em três partes: transmissão, recepção e um circuito eletrônico que interpreta os dados dessa medição.

Para monitorar a HR, utilizamos o aparelho muito comum entre atletas, o Polar S610i com o transmissor Wearlink 31 Coded, também do fabricante Polar.

Um transmissor envolto por uma cinta fixada no peito do usuário capta os batimentos e os envia diretamente para um relógio receptor no pulso, que recebe a informação. Na figura 3.3.1 pode ser observado o conjunto relógio cinta do fabricante Polar.

Figura 3.3.1 – Conjunto relógio cinta para medir freqüência cardíaca da Polar.

Um batimento do coração produz um sinal elétrico que é transmitido pelo músculo do coração. Essa atividade elétrica é um sinal eletrocardiográfico (ECG), e é captado através da pele pelo transmissor posicionado na área da pele próxima ao coração.

O transmissor envia um sinal contendo os dados da HR para o receptor no relógio de pulso, que mostra em seu display a HR.

Este relógio por sua vez pode transmitir dados por infravermelho (IR) para um computador, e estes dados são interpretados por seu software específico.

Porém outro problema enfrentado, é que estes dados colhidos do relógio de pulso são gráficos dos valores da HR durante uma atividade que durou um tempo especifico, e não dados contendo a HR instantânea. Logo desenvolvemos um receptor que pega os dados emitidos pela unidade transmissora do polar. Após algumas analises concluímos que estes dados são enviados por radiofreqüência (RF) a uma freqüência de 5 KHz.

3.4 Módulo Receptor

3.4.1 RMCM-01

Este componente do circuito receptor é o receptor dos sinais gerados em RF pela cinta transmissora a cada batimento cardíaco detectado. Logo que recebe o sinal, ele gera um pulso digital correspondente a cada batimento. A figura 3.4.1.1 representa o módulo RMCM-01

Figura 3.4.1.1 – Módulo RMCM01.

Para que receba os dados adequadamente a distância entre transmissor e receptor não pode passar de 80 cm. E para que se obtenha o ganho máximo, eles devem ser alinhados em paralelo.

Existem duas saídas no componente. Uma delas é a saída HRM que gera um pulso de 3 V e 1 ms quando detecta um sinal de batimento. A outra saída FPLS mostra todos os sinais recebidos do jeito que eles são. Na figura 3.4.1.2 podemos observar o sinal gerado pelo pino FPLS.

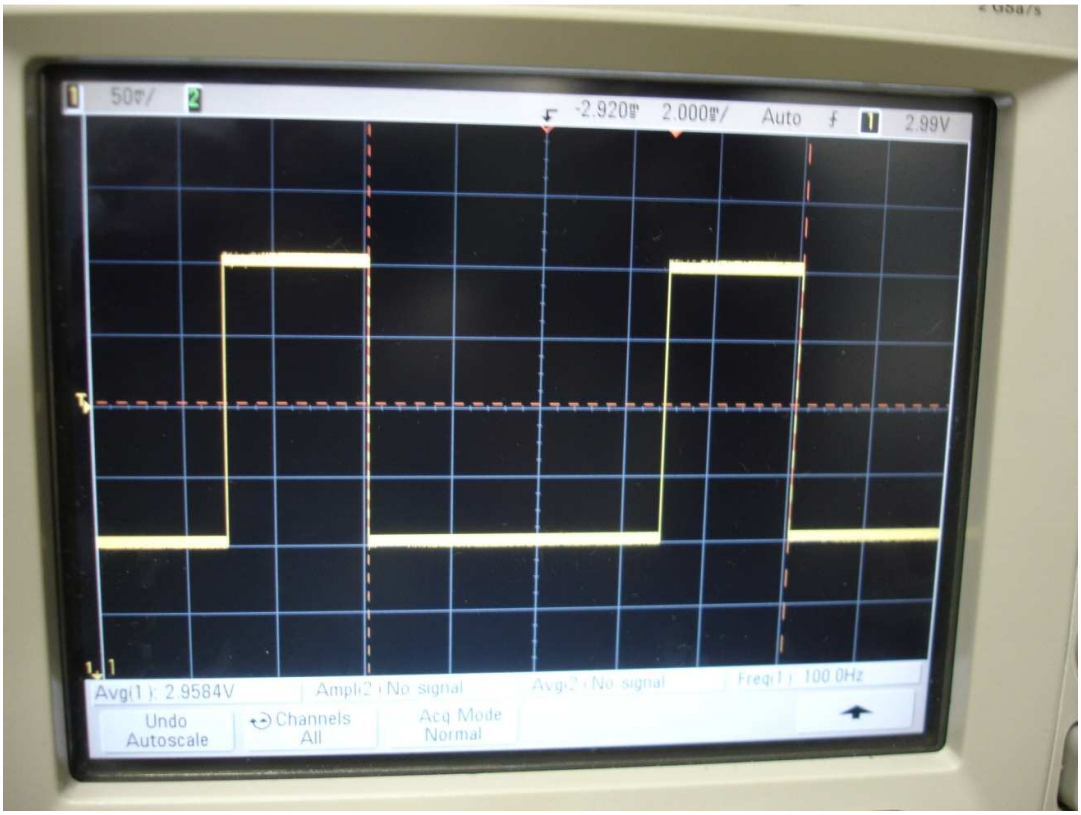

Figura 3.4.1.2 – Sinal Gerado pelo pino FPLS do módulo RMCM01.

As funções de cada pino podem ser vistas na tabela 3.4.1 abaixo:

Pino	Nome	Descrição
1	VCC	Voltagem de Operação
2	OSC_ON	Seleção de *Clock* (Liga/Desliga)
3	F32KIN	Terminal para Cristal ou *Clock* externo
4	OSC	Terminal para Cristal
5	HR	Informação de HR
6	FPLS	Saída do detector de pulsos
7	RESET	Reset (active low)
8	WIDB_DET	Conectar a Vcc
9	LX2	Terminal para Antena
10	LX1	Terminal para Antena
11	GDN	Aterrar

Tabela 3.4.1 – Especificações dos pinos do módulo RMCM01.

Usaremos os dados obtidos na saída FPLS do componente da seguinte maneira: um algoritmo que detectará a quantidade de pulsos lidos nessa saída em certo período de tempo.

Como informação adicional, especialmente para interessados em adquirir este componente, este não é de fácil aquisição. Tivemos que recorrer a empresas no exterior através de seus sites para vendas.

3.4.2 OSCILADOR A CRISTAL

Um oscilador a cristal utiliza a ressonância de um cristal em vibração de um material piezoeletrônico, para criar um sinal elétrico com uma freqüência bastante precisa. Esta freqüência é comumente usada para medir precisamente o tempo. O cristal piezoelétrico mais utilizado é quartzo. Esse cristal contrai quando submetido à tensão elétrica, e o tempo de contração varia conforme a construção do cristal. Quando a contração chega a certo ponto, o circuito libera a tensão e o cristal relaxa, chegando ao ponto de uma nova contração. Assim, os tempos de contração e relaxação desse ciclo determinam uma freqüência de operação, muito mais estável e controlável que circuitos com capacitores.

Na figura 3.4.2.1 é mostrado um oscilador a cristal.

Figura 3.4.2.1 – Oscilador a Cristal.

Para o circuito utilizado, a freqüência de operação é de 32 KHz, logo utilizamos um oscilador a cristal correspondente. Este cristal terá a função de clock no circuito receptor.

O próprio clock do microcontrolador ATmega1281 poderia ter sido usado, porém ele não é tão preciso quanto o cristal.

Agora será explicado como fará parte do circuito receptor, tanto o oscilador a cristal quanto o componente RMCM-01.

3.4.3 CIRCUITO RECEPTOR

Quando foi projetado este circuito receptor era esperado que além de receber os dados dos batimentos enviados pela cinta transmissora, também interpretasse esses dados, calculando um valor de HR que será usado adiante. Também foram considerados os possíveis ruídos e interferências no sinal ECG que deve ser recebido.

A parte referente a recepção e garantia de qualidade dos dados, pode ser visto no circuito da figura 3.4.3.1 e é praticamente garantida pelo RMCM-01.

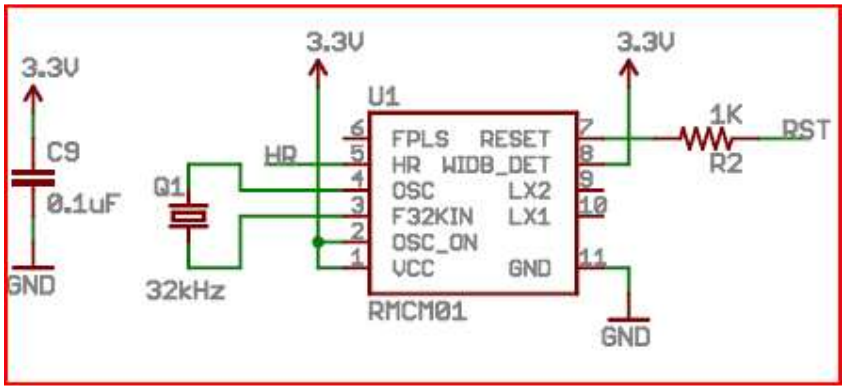

Figura 3.4.3.1 – Receptor de batimentos cardíacos RMCM01.

A saída HRM estará ligada ao microcontrolador ATmega1281 onde será feita a parte de manipulação dos dados, através de software, para se obter um valor de HR que possa efetivamente ser usado. Nas figuras 3.4.3.2 e 3.4.3.3 pode ser

observado o circuito completo do receptor de batimentos cardíacos e o seu esquemático respectivamente.

FOTO DO CIRCUITO COMPLETO

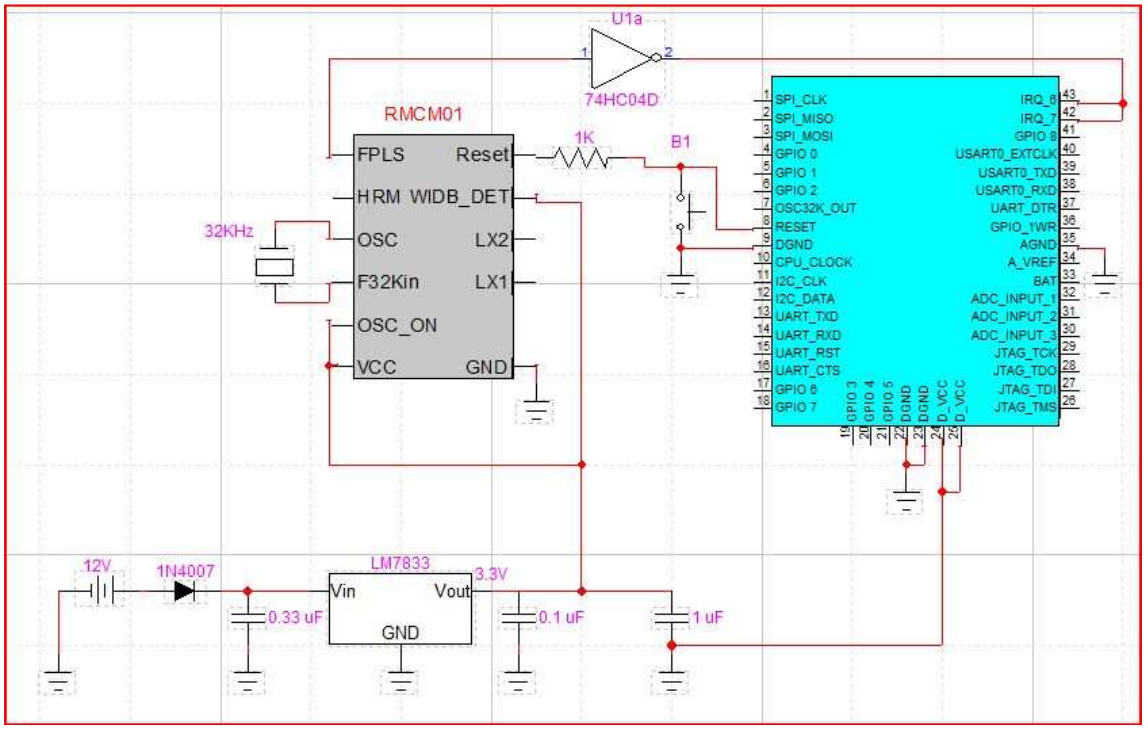

Figura 3.4.3.3 – Esquemático do módulo receptor de batimentos cardíacos.

3.5 Gravadora do microcontrolador

3.5.1 O Hardware

Após conseguir que o receptor operasse adequadamente, foi criado um algoritmo capaz de receber os dados provenientes do módulo RMCM- 01, analisá-los, enviá-los ao módulo coordenador e posteriormente ao supervisório.

Para criar o algoritmo, a ferramenta AVR Studio 4 foi utilizada, incluída no BitCloud, e foram adaptados códigos utilizados em trabalhos anteriores, mas voltado para aplicação deste projeto e com um baixo consumo de energia. O código foi elaborado com base no código *Low Power,* que também acompanha o BitCloud.

Para realizar a gravação desse algoritmo no microcontrolador ATmega1281 foi projetado um circuito capaz de exercer essa função.

O circuito da figura 3.5.1.1 representa a gravadora descrita acima. Esse circuito pode ser dividido em três módulos: Regulador de tensão; Conversor de sinais TTL para RS232; Gravação do microcontrolador ATmega1281.

Nesse capítulo, aprofundaremos mais sobre cada um desses módulos.

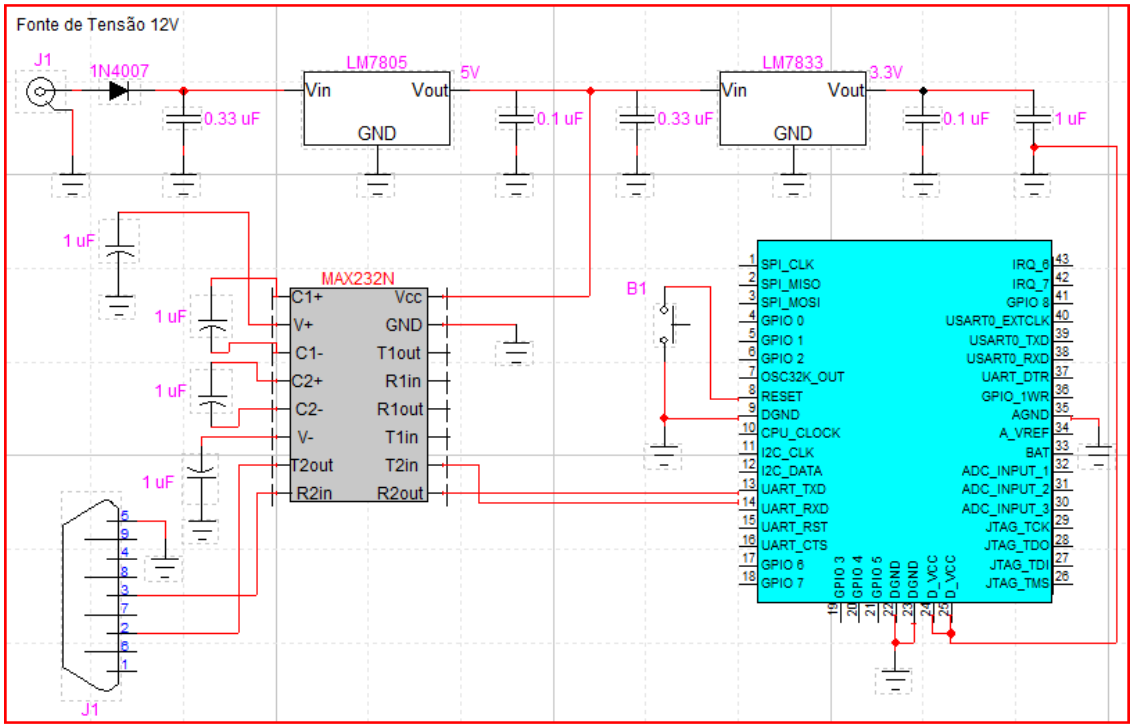

Figura 3.5.1.1 – Esquemático da gravadora do microcontrolador ATmega1281.

O circuito foi confeccionado de modo caseiro, sendo montado em uma placa furada. Nas figuras 3.5.1.2 e 3.5.1.3 pode ser visto o circuito da gravadora com e sem o módulo ZigBit respectivamente.

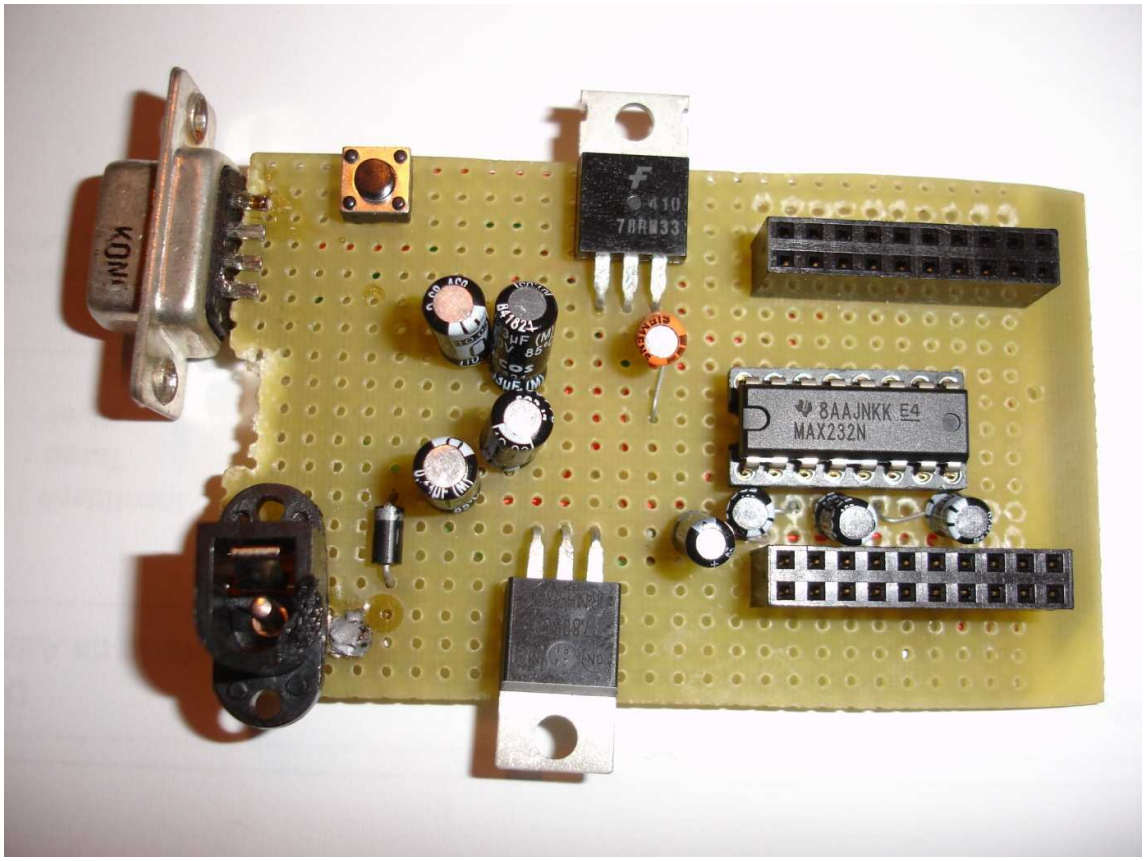

Figura 3.5.1.2 – Gravadora do microcontrolador ATmega1281 sem o módulo ZigBit.

O circuito da figura 3.5.1.2 é composto de:

- 1 conector para alimentação;
- 1 conector fêmea DB-9;
- 1 diodo com a função de proteção do sistema, evitando a inversão de polaridade;
- 2 capacitores eletrolíticos de 0.33 µF;
- 2 capacitores de 0.1 µF;
- 5 capacitores de 1 µF;
- 1 regulador de tensão LM7805;
- 1 regulador de tensão LM7833;
- 1 soquete para CI de 16 pernas;
- 1 conversor MAX232N;
- 2 soquetes para o módulo ZigBit;
- 1 botão;

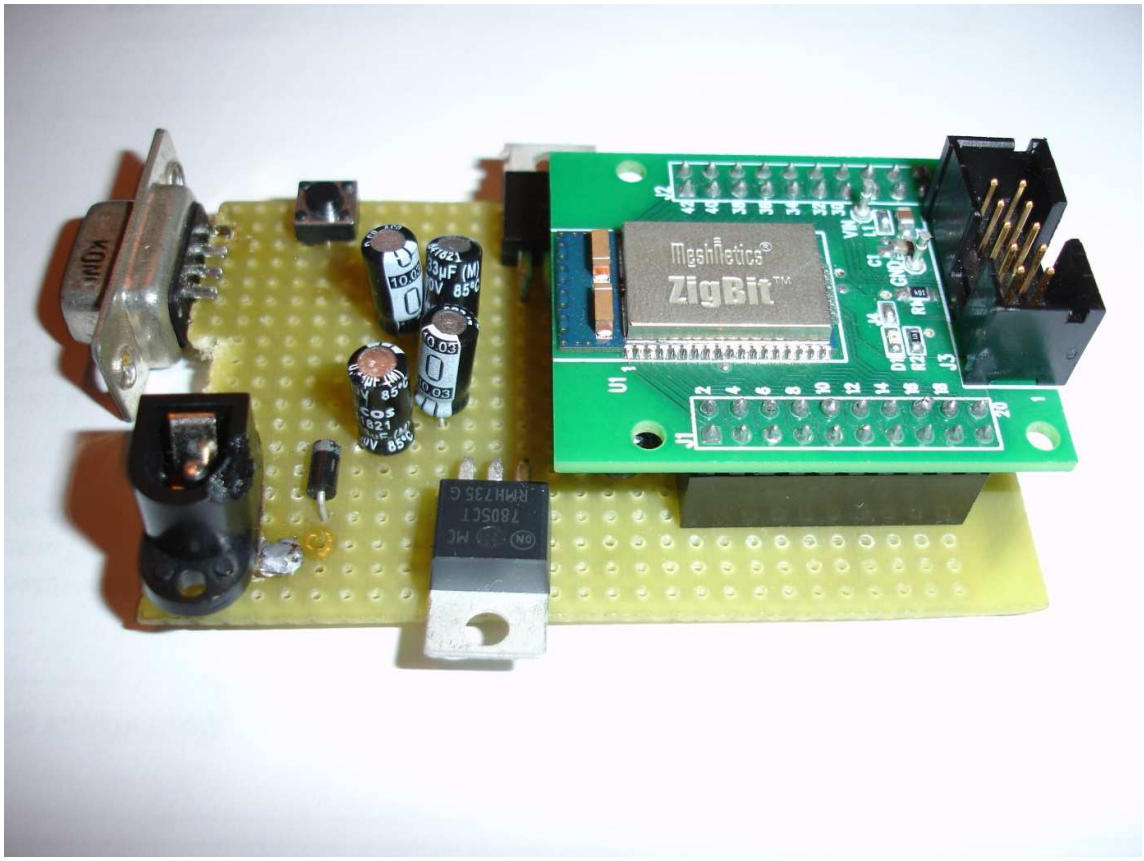

Figura 3.5.1.3 – Gravadora do microcontrolador ATmega1281 com o módulo ZigBit.

A única diferença da figura 3.5.1.2.para a figura 3.5.1.3 é a presença do módulo ZigBit, que é encaixado nos soquetes para o módulo.

Para a alimentação desse circuito foi utilizado uma fonte de 12V DC, alimentado em 110V.

Vale ressaltar que todos os componentes listados acima foram facilmente encontrados em lojas especializadas em eletrônica.

Nos tópicos a seguir serão abordados os três módulos listados acima.

3.5.2 O Regulador de Tensão

O regulador de tensão tem simplesmente a função de diminuir a tensão até um valor desejado de operação. Se, por exemplo, dispõe-se de uma bateria de 12V e deseja-se alimentar um circuito em 5V basta utilizar um regulador de tensão.

Hoje em dia, existem vários modelos de reguladores de tensão e várias tensões de alimentação. A figura 3.5.2.1 mostra o esquemático de um regulador de tensão.

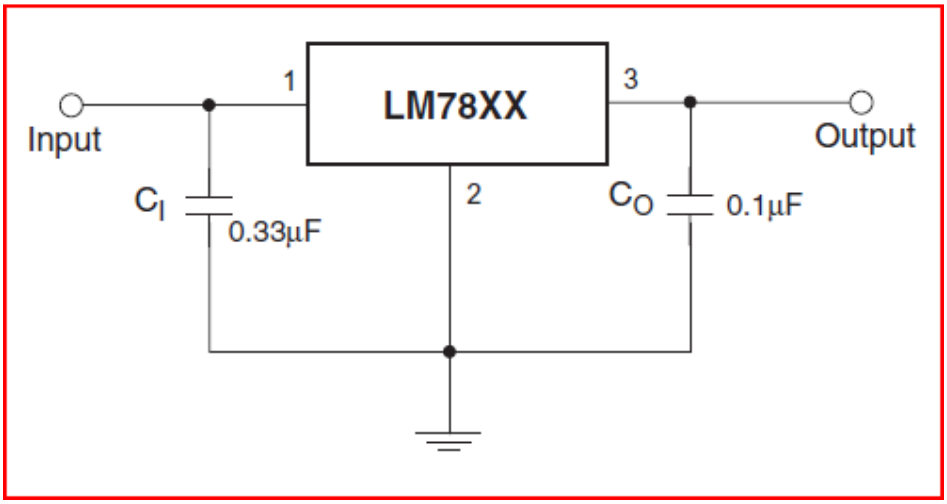

Figura 3.5.2.1 – Regulador de tensão genérico.

Esse dispositivo possui três terminais. O terminal 1 é onde conecta a entrada da alimentação. O valor máximo admissível para essa tensão de entrada tem que ser observada no *datasheet* do respectivo regulador, pois pode variar de um regulador para outro. O terminal 2 é o terra do dispositivo. Ele funciona como uma referência entre a entrada e a saída. O terminal 3 é onde sai a tensão desejada. Os capacitores C_0 e C_1 são usados para melhorar a estabilidade e a resposta em transiente desse dispositivo.

No circuito da gravadora foi utilizado o regulador LM7805 e 7833. O LM7805 foi utilizado para alimentar o conversor de sinais MAX232. O regulador LM7805 usado em nosso projeto recebe uma tensão de 12V de uma fonte externa e coloca na saída uma tensão de 5V. Já o LM7833, que alimenta o módulo ZigBit, recebe uma tensão de 5V e dispõe na saída uma tensão de 3.3V. Com isso toda a parte de

alimentação do circuito está concluída e é mostrada na parte superior da figura 3.5.1.1.

Para o regulador LM7805 a tensão de alimentação está entre 7V e 20V. Para o regulador LM7833 a faixa de alimentação está entre 5V e 18V.

3.5.3 Conversor de sinais TTL para RS232

A gravação do algoritmo presente no computador para o microcontrolador foi feita através da porta serial do computador. A porta serial emprega o protocolo RS232. Esse protocolo é assíncrono, *full-duplex* e especifica dois níveis de tensão para representar os bits 0 e 1. O bit 0 é representado pelo nível de tensão +12V e o bit 1 pelo nível -12V.

Já o microcontrolador trabalha com lógica TTL, que usa +5V e 0V para representar os bits 1 e 0 respectivamente.

Para realizar o interfaceamento da porta serial com o microcontrolador, ou seja, trabalhar com RS232 e TTL, é necessário converter os níveis de tensão. Para converter de TTL para RS232 é necessário um circuito denominado *driver* RS232. Já o processo inverso é denominado *receiver* RS232. [Colocar a Referencia do Zele]

Atualmente podemos encontrar vários modelos de conversores que realizam as duas funções. Há também a possibilidade da confecção de um circuito empregando apenas componentes discretos, mas não foi o caso deste projeto.

O CI empregado na gravadora foi o MAX232N, que possui uma alimentação de 5V e utiliza 5 capacitores de 1 µF, como pode ser visto na figura 3.5.1.1. Para a conexão com a porta serial foi utilizado um conector fêmea DB-9 com apenas os terminais 2 (RX) ,3 (TX) e 5 (GND) utilizados.

Na figura 3.5.3.1 podemos ver o esquemático de um conversor de sinais.

Note que em um mesmo CI há a possibilidade de ligação de dois circuitos distintos, pois o mesmo possui 2 terminais para cada conversão. No caso especifico deste projeto foi utilizado apenas 1 terminal de cada conversor.

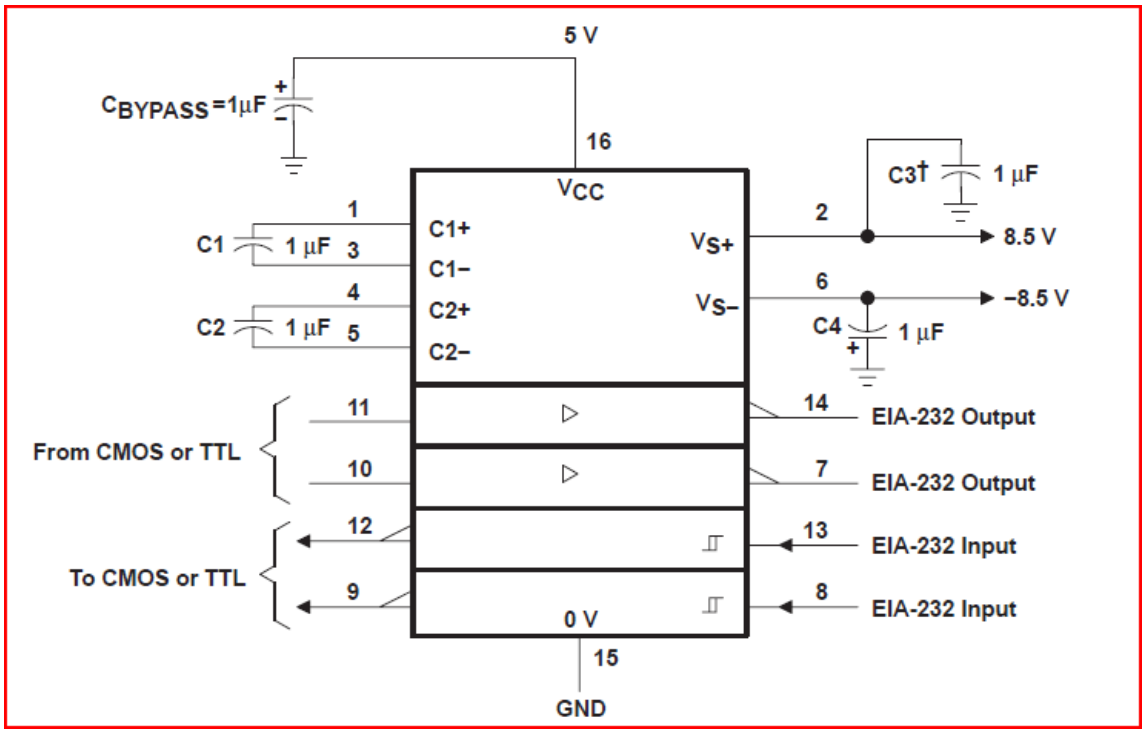

Figura 3.5.3.1 – Esquemático do MAX232N.

Com esse CI podemos comunicar do computador para o microcontrolador fazendo as conexões necessárias dos mesmos com o CI.

Na falta do Ci MAX232N pode ser adquirido outro CI com mesmas características, observando atentamente o valor da alimentação de cada tipo de CI encontrado no mercado.

3.5.4 Gravação no microcontrolador ATmega1281

Com o circuito praticamente pronto, basta conectar o microcontrolador no circuito da gravadora, como mostrado na figura 3.5.1.3. Para a gravação do mesmo, é necessário, além da alimentação e do terra, a conexão dos terminais de transmissão e recepção e também a conexão do pino *Reset* do microcontrolador. Como foi alertado no capitulo 2, o pino referente a transmissão, pino 13, é usado como receptor dos dados e o 14 como transmissor.

Para iniciar a gravação do algoritmo no microcontrolador é necessário um software, que também acompanha o BtCloud, denominando BootLoader. O software BootLoader pode ser visto na figura 3.5.4.1.

Figura 3.5.4.1 – Software BootLoader.

O primeiro passo na hora de gravar um algoritmo é encontrar o local onde ele foi salvo. O formato do arquivo a ser gravado no microcontrolador é o *.srec. Feito isso é necessário especificar a porta a qual o módulo ZigBit está conectado com o computador. Tendo feito esses dois passos é só apertar o botão *Upload* e resetar o módulo. Para resetar o módulo foi utilizado um botão que, quando pressionado, leva o nível de tensão do terminal para o terra, podendo, assim, começar a gravação.

3.6 Software do Supervisório

O software do supervisório, implementado em Visual Basic, tem como principal função, coordenar o fluxo de dados entre os módulos sensores e os módulos de controle, execução da rotina de controle, geração de tabelas e gráficos. Ou seja, monitora e supervisiona as variáveis de dispositivos enviados pelo módulo coordenador, processa e envia respostas pertinentes para o controle do sistema atuador.

Podemos ver a tela principal do software na figura 3.6.1.

Figura 3.6.1 – Tela principal do software supervisório.

Os cinco botões, que podem ser observados na figura 3.6.2, assumem as seguintes funções, respectivamente:

- Configuração da comunicação;
- Conectar;
- Desconectar;
- Comandos e Dados;
- Parâmetros do Conforto Térmico.

Figura 3.6.2 - Botões encontrados no supervisório.

Fazemos a comunicação entre o módulo coordenador e o computador através da comunicação serial RS-232. Através desta comunicação, recebem-se dados com informação da temperatura ambiente, da HR e do nível de bateria dos sensores, onde serão amostrados e analisados.

O primeiro passo é clicar no botão “Configuração de Comunicação” e selecionar o número da porta COM em que está conectado o módulo coordenador, para que se estabeleça a conexão corretamente. A figura referente à configuração de comunicação pode ser vista na figura 3.6.3 abaixo.

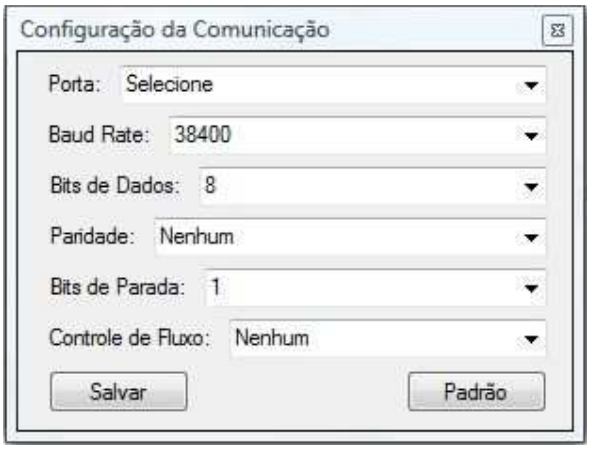

Figura 3.6.3- Janela de Configuração da Comunicação.

Através do botão “Parâmetros do Conforto Térmico” podemos também configurar parâmetros que serão utilizados para o cálculo do conforto térmico, como o tipo de vestimenta, sexo, peso, ou mesmo mudar o set point de temperatura radiante média, velocidade do vento ou da umidade, já que ainda não existiam os sensores para medição destas variáveis, no LAVSI, no início de nosso projeto. Logo apesar de podermos estabelecer valores aproximados da realidade do local, não podemos atualizar estes dados em tempo real.

Figura 3.6.4- Janela de Parâmetros de Conforto Térmico.

Após esses passos, deve-se clicar no botão “Conectar” para iniciar a comunicação entre o coordenador e a porta COM escolhida. Os dados serão enviados em intervalos de 2 segundos. Isto gera uma interrupção no supervisório para o tratamento destes dados. No botão “Comandos e Dados” podemos observar em tempo real um histórico dos dados recebidos e enviados, como pode ser visto na figura 3.6.5.

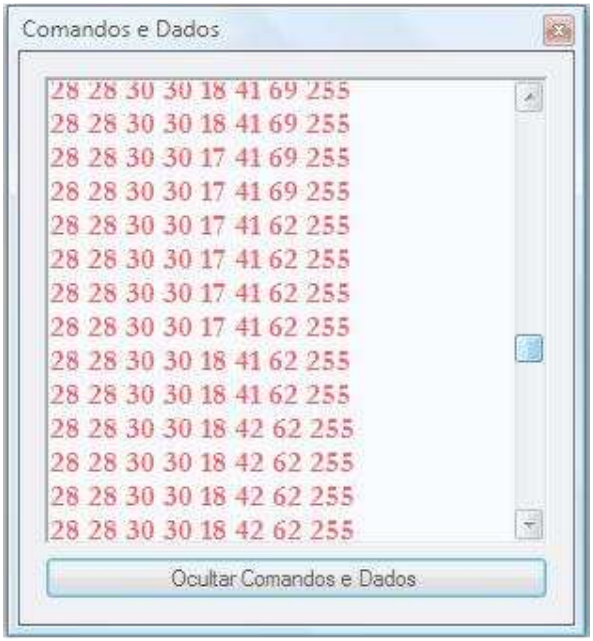

Figura 3.6.5- Janela de Comandos e Dados.

Os valores que aparecem na figura acima correspondem, respectivamente:

- à temperatura do sensor 1;
- à temperatura do sensor 2;
- à temperatura do atuador 1;
- à temperatura do atuador 2;
- ao nível de bateria do sensor 1;
- ao nível de bateria do sensor 2;
- à HR
- ao nível de bateria do medidor de batimentos cardíacos.

Depois de recebido e armazenado esses dados, o supervisório calcula a equação de conforto, os índices PMV e PPD, além dos níveis de bateria, tanto em volts como percentual. Todos esses valores vão, automaticamente, ser armazenados em uma planilha de dados no Excel, visto na figura 3.6.6.

Teste realizado no dia: 26/02/2010																
Teste iniciado em: 1:18								Teste finalizado em: 4:53								
Tempo	Sensor 1	Sensor 2	Atuador 1	Atuador 2	PMV 1	PPD 1	PMV 2	PPD 2	Split 1	Split 2	Bat. S1	Bat. S2	Bat. A1	Bat. A2	Bat. B1	Bpm
01:19:02	27	28	30	28	0,13	100	0,38	100	1	1	9,28%	7,81%	externo	externo	255	74
01:19:11	27	28	29	28	0,13	5,37	0,38	8,06	1	1	9,77%	7,81%	externo	externo	255	64
01:19:21	27	28	29	28	0,13	5,37	0,38	8,06	1	1	21,48%	7,81%	externo	externo	255	83
01:19:31	27	28	30	28	0,13	5,37	0,38	8,06	1	1	9,77%	8,30%	externo	externo	255	83
01:19:41	27	28	30	28	0,46	9,47	0,67	14,34	1	1	9,77%	8,79%	externo	externo	255	90
01:19:51	27	28	30	28	0,13	5,37	0,38	8,06	1	1	9,77%	20,02%	externo	externo	255	79
01:20:01	27	28	30	29	0,13	5,37	0,38	8,06	1	1	9,28%	20,02%	externo	externo	255	74
01:20:11	27	28	30	29	0,13	5,37	0,38	8,06	1	1	9,77%	20,51%	externo	externo	255	74
01:20:21	27	28	30	28	0,13	5,37	0,38	8,06	1	1	9,28%	20,51%	externo	externo	255	83
01:20:31	27	28	30	28	0,13	5,37	0,38	8,06	1	1	9,77%	20,51%	externo	externo	255	83
01:20:41	27	27	30	28	0,13	5,37	0,13	5,37	1	1	9,77%	7,81%	externo	externo	255	76
01:20:51	27	27	30	28	0,13	5,37	0,13	5,37	1	1	9,77%	7,32%	externo	externo	255	83
01:21:01	27	27	30	29	0,46	9,47	0,46	9,47	1	1	9,77%	7,81%	externo	externo	255	86
01:21:11	27	27	30	29	0,13	5,37	0,13	5,37	1	1	9,77%	7,81%	externo	externo	255	79
01:21:21	27	27	30	28	0,46	9,47	0,46	9,47	1	1	9,77%	8,30%	externo	externo	255	86
01:21:31	26	27	30	28	0,26	6,41	0,46	9,47	1	1	9,28%	8,30%	externo	externo	255	86
01:21:41	26	27	30	28	0,26	6,41	0,46	9,47	1	1	9,28%	8,30%	externo	externo	255	86
01:21:51	26	27	30	28	0,26	6,41	0,46	9,47	1	1	9,77%	8,79%	externo	externo	255	86
01:22:01	26	27	30	28	0,26	6,41	0,46	9,47	1	1	9,28%	8,79%	externo	externo	255	86
01:22:11	26	27	30	28	-0,12	5,28	0,13	5,37	0	1	9,77%	8,79%	externo	externo	255	79

Figura 3.6.6- Tabela de valores no Excel.

Além disso, também podemos observar os valores dos parâmetros calculados na própria tela principal do supervisório observado na figura 3.6.7.

Figura 3.6.7- Detalhe dos parâmetros medidos e calculados.

4 Resultados Experimentais e Análise

4.1 Visão Geral dos Experimentos

Foram realizados três experimentos com diferentes parâmetros. As variações foram feitas em cima do período do dia (noite e dia), sexo (masculino e feminino) e localização (ambiente 1 e 2).

Todos tiveram uma mesma configuração, com dois sensores de temperatura, um em cada ambiente; dois atuadores que também medem temperatura, um em cada *split*; um módulo coordenador e um sensor de batimentos cardíacos. Os sensores de temperatura têm uma resolução de 1º C.

As placas foram posicionadas para os testes de forma que o ar frio do ar condicionado e outras fontes de calor não incidissem diretamente no sensor. Evitou-se assim a realização de medidas que não representassem a temperatura média da sala.

A configuração dos dispositivos no ambiente pode ser vista nas figuras 4.1.1 a 4.1.5.

Figura 4.1.1 –Módulo Atuador e Split no Ambiente 1.

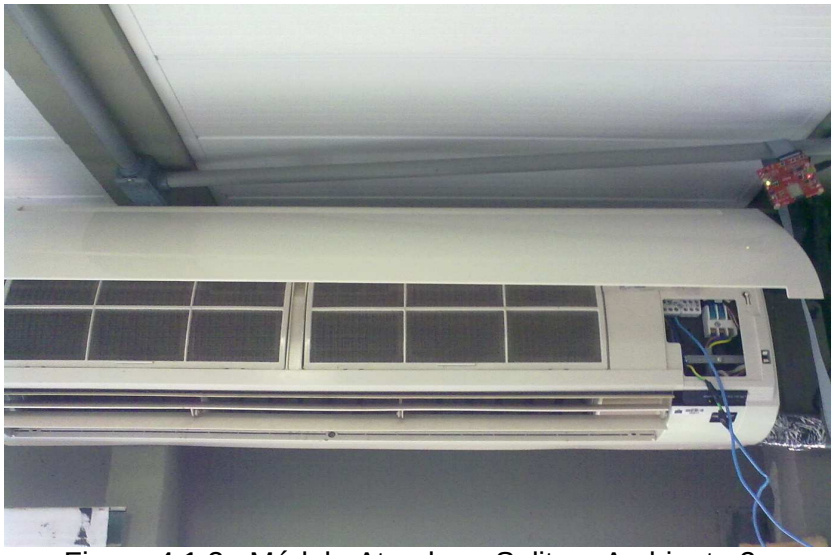

Figura 4.1.2 –Módulo Atuador e Split no Ambiente 2.

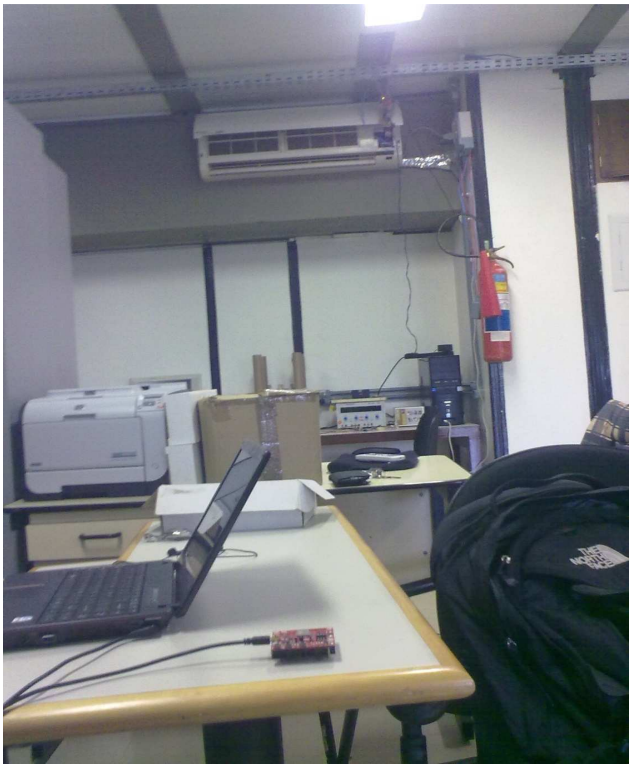

Figura 4.1.3 – Sensor de Temperatura do Ambiente 1.

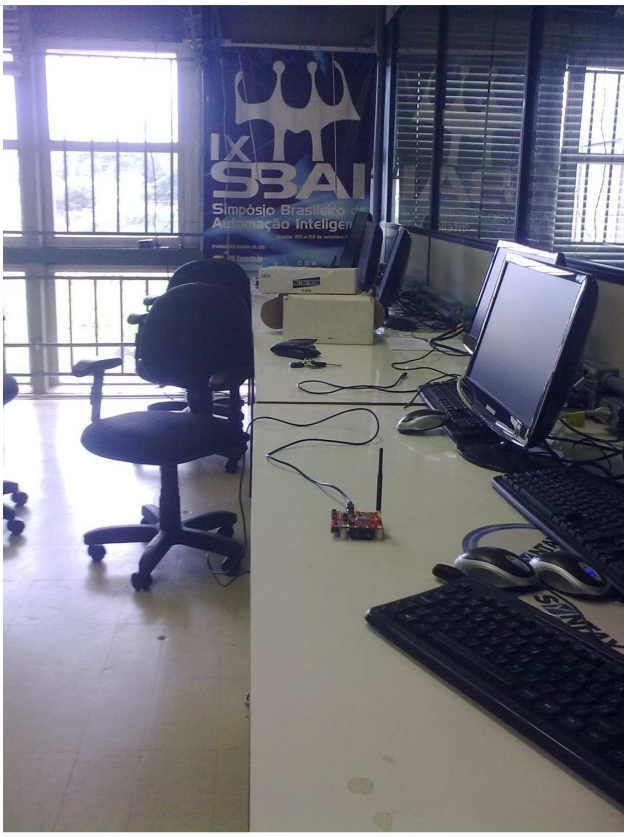

Figura 4.1.4 – Sensor de Temperatura do Ambiente 2.

Figura 4.1.5 – Módulo Coordenador

Os módulos atuadores foram posicionados acima do aparelho, próximo ao teto. Assim, medem a temperatura do ar de retorno, ou seja, a temperatura do ar que retorna ao ar condicionado para ser refrigerado.

A atuação dos aparelhos de ar condicionado foi feita em torno do PMV. Ou seja, visa manter sempre o nível de conforto térmico dos usuários na sala, mantendo o índice PMV entre -0,5 e 0,5. O aparelho é ligado quando este índice é maior que 0,5 e é desligado quando é menor que -0,5.

Desta forma o que se procurou observar nesses experimentos é a influência do nível de atividade medido pelos batimentos cardíacos sobre o índice PMV. Para isso diferentes atividades foram realizadas no ambiente, como ficar sentado e fazer exercícios físicos mais intensos, como correr.

Os resultados primeiramente serão apresentados e analisados de forma individual com seus respectivos gráficos e posteriormente numa análise mais geral dos resultados obtidos.

4.2 Experimento I

Este primeiro experimento foi realizado no dia 26 de fevereiro de 2010 iniciado às 01h18min da manhã e encerrado às 04h53min. A pessoa, sobre a qual foi realizado o teste de medição dos batimentos cardíacos, era do sexo masculino, 89 Kg, vestia roupas de trabalho e ficou situada no ambiente 2. Na hora do experimento a umidade relativa do ar era de 40%, a temperatura média radiante era de 27,5℃ e a velocidade do vento de 1m/s.

Neste teste procuramos simular uma situação real que acontece no ambiente, onde na maior parte do tempo o indivíduo fica sentado realizando atividades no computador. Ao final do teste, pediu-se que o indivíduo aumentasse seu batimento cardíaco, realizando exercícios físicos mais intensos, para se ter uma noção do que a variação na freqüência cardíaca causaria no PMV. Nas figuras abaixo podemos ver os gráficos referentes a este experimento.

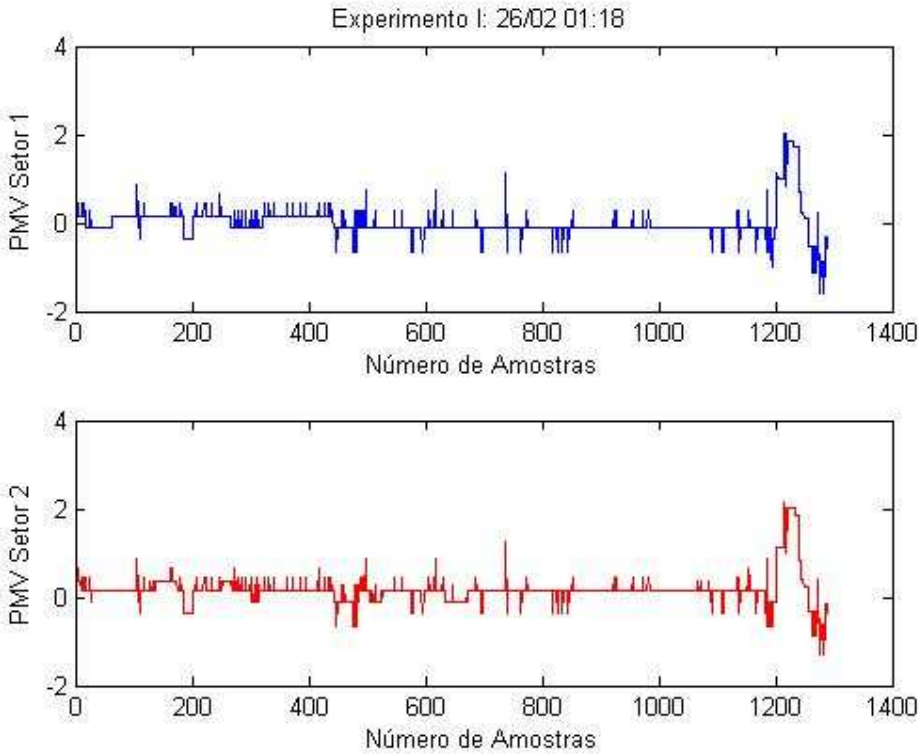

Figura 4.2.1 – Gráficos ‘PMV x Amostras’ no ambiente 1 (azul) e 2 (vermelho).

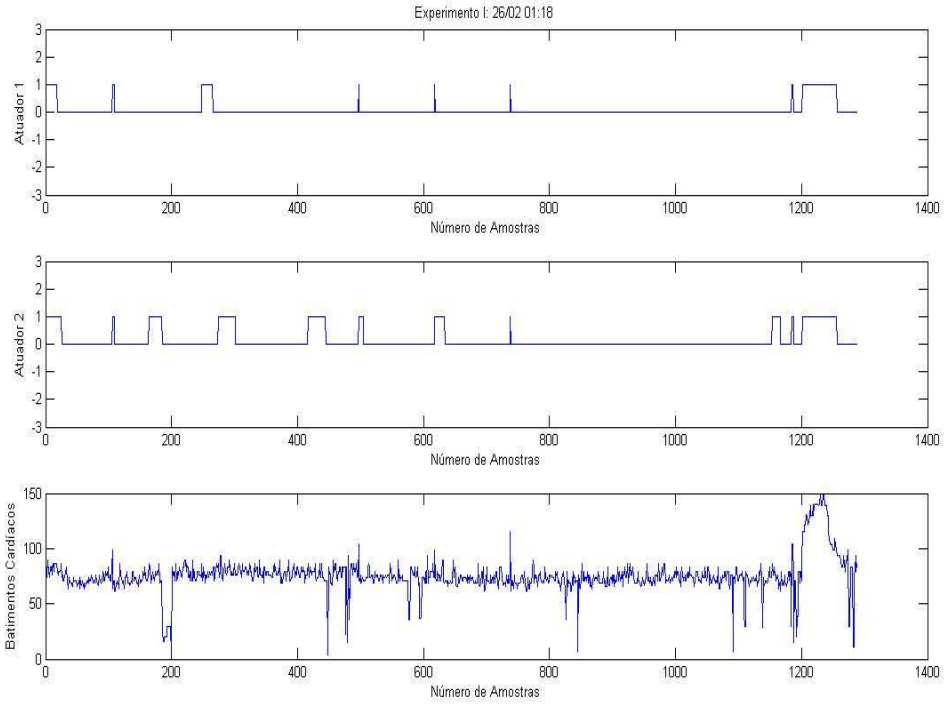

Figura 4.2.2 – Gráficos ‘Atuação dos *Splitters* 1 e 2’ e ‘HR x Tempo’.

Neste experimento em específico, pode-se notar que quando o PMV abaixou de 0 o aparelho foi desligado e quando aumentou mais de 0,5 foi ligado. Este controle foi proposto visando não deixar a temperatura variar muito. Contudo esta escolha foi equivocada, pois acabou ligando e desligando o aparelho mais vezes do que o necessário.

Pode-se notar que algumas vezes temos valores para a HR discrepantes. Isso acontece por causa de falhas na recepção, ou por interferências ou por falta da captação do sinal ECG no momento certo. Isso acarreta num valor errado de PMV, que por duas vezes ligou o aparelho desnecessariamente e algumas vezes, não influenciou, pois jogou o PMV pra baixo, mas o aparelho já estava desligado. Mas foram erros pontuais, nada que influenciou significativamente no resultado geral do experimento.

Também podemos observar nitidamente a atuação do aparelho no final do teste quando a HR fica elevada, aumentando o PMV exigindo que se diminua a temperatura para trazer o índice para zona de conforto.

Outro ponto a se destacar é uma atuação mais presente no ambiente 2, já que é lá que o indivíduo em teste se manteve durante todo o experimento. Sua presença interfere na temperatura do ambiente, já que ele é uma fonte de calor.

4.3 Experimento II

O segundo experimento foi realizado também no dia 26 de fevereiro, porém no período da tarde, inicializando às 17h04min. O experimento teve duração de uma hora, terminando às 18h02min Nesse momento, a temperatura média radiante era de 31ºC e umidade relativa de 43%. O individuo em estudo era do sexo feminino, pesava 55 kg, vestindo roupas leves de verão e situada no ambiente 1.

O intuito desse experimento, assim como o primeiro experimento, era simular condições reais de trabalho, que ocorrem no dia a dia, porém com o indivíduo do sexo feminino. Para tanto, pediu-se que o indivíduo realizasse atividades leves, como exercidas em seu dia de trabalho, e atividades um pouco intensas, a fim de avaliar o funcionamento dos compressores.

Para esse experimento o controle da atuação foi feito da maneira citada no tópico 4.1, acionando os compressores quando o PMV estava acima de 0.5 e desligando abaixo de -0.5.

Para esse experimento o individuo se manteve único e exclusivamente no ambiente 1 ao longo de todo o experimento. Nas figuras 4.3.1 e 4.3.2, pode-se observar variações dos índices PMV para os dois setores e a atuação dos compressores e os batimentos cardíacos respectivamente.

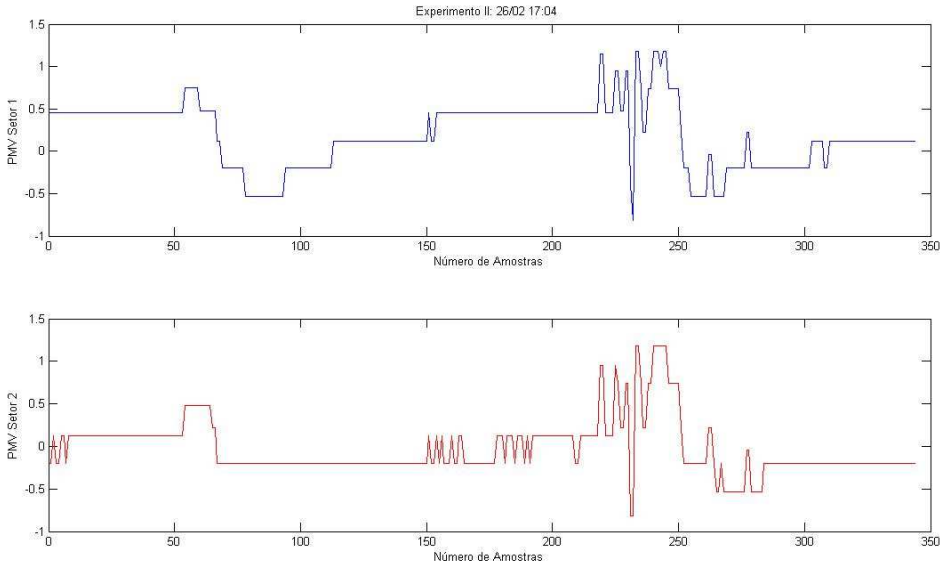

Figura 4.3.1- Gráficos ‘PMV x Amostras’ no ambiente 1(azul) e 2(vermelho).

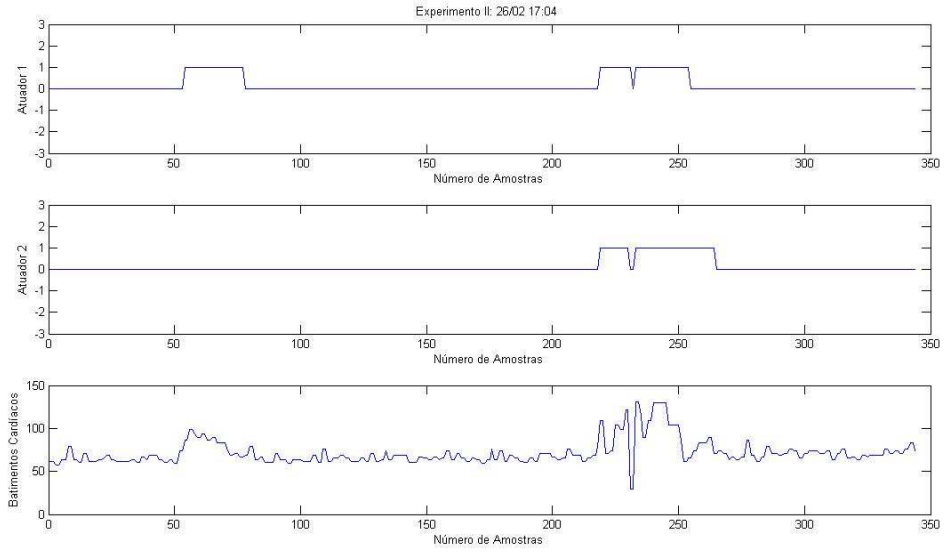

Figura 4.3.2- Gráficos ‘Atuação dos Compressores 1 e 2’ e ‘HR x Tempo’.

No começo do experimento o setor 2 apresentou uma variação no PMV que é explicada pela variação da temperatura daquele ambiente. Nesse experimento não houve variações significativas do PMV.

Analisando detalhadamente os gráficos, percebe-se nitidamente que o acionamento dos compressores estava condicionado ao nível de atividade do indivíduo. Nota-se também um atuação singular do compressor no ambiente 1 devido o fato do indivíduo estar somente nesse ambiente e interferir na temperatura desse ambiente indiretamente.

Observa-se que os índices PMV estão quase sempre nos seus valores de conforto, ou seja, entre -0.5 e 0.5. Nesse experimento não houve grandes erros associados ao funcionamento do receptor de batimentos cardíacos. Porém é possível notar nitidamente o chaveamento do dos compressores por um curto período de tempo. Isso se deve ao fato de que, naquele momento, estava sendo realizado um teste com uma atividade mais intensa que o usual e o indivíduo saiu do raio de cobertura do coordenador, caindo drasticamente o valor de seu batimento cardíaco.

Vale lembrar que a variável que indica o batimento cardíaco do indivíduo integra a equação de conforto térmico dos dois ambientes. Isso explica o fato de o compressor do ambiente 2 também ser acionado no momento que houve a perda de sincronismo entre o módulo coordenador e o módulo receptor de batimentos cardíacos.

4.4 Experimento III

Este experimento foi realizado no dia 28 de fevereiro de 2010 iniciado às 14h08min da tarde e encerrado às 17h01min. A pessoa, sobre a qual foi realizado o teste de medição dos batimentos cardíacos, era do sexo masculino, 71 Kg, vestia roupas leves de verão e ficou situada no ambiente 1. Na hora do experimento a umidade relativa do ar era de 53%, a temperatura média radiante era de 30,5℃ e a velocidade do vento de 1m/s. O controle da atuação foi feito como no Experimento II.

Neste teste demos ênfase em mostrar a influência da HR no PMV. Realizamos atividades em diferentes faixas de freqüência cardíaca, para que se analisassem mais detalhadamente o efeito desta variável. Nas figuras abaixo podemos ver os gráficos referentes a este experimento.

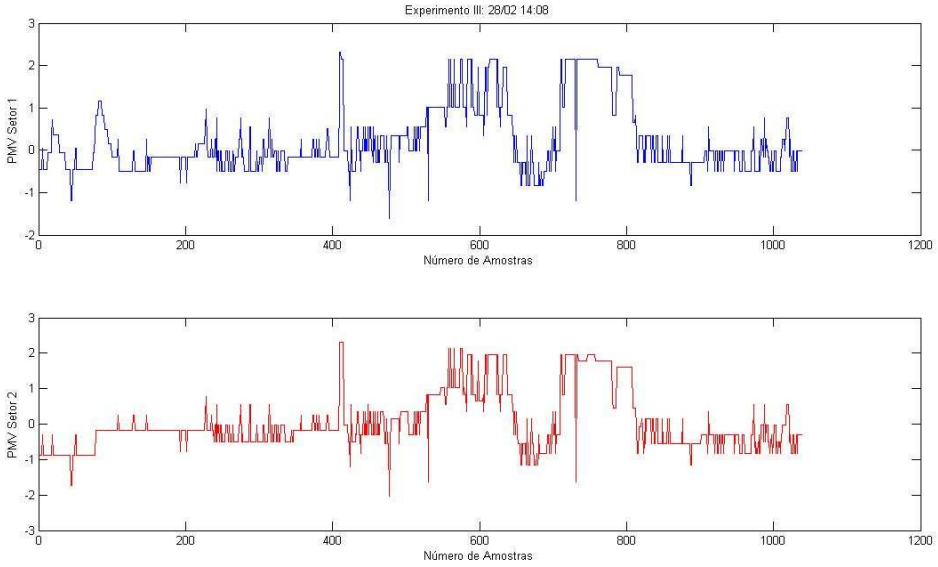

Figura 4.4.1 – Gráficos ‘PMV x Amostras’ no ambiente 1 (azul) e 2 (vermelho).

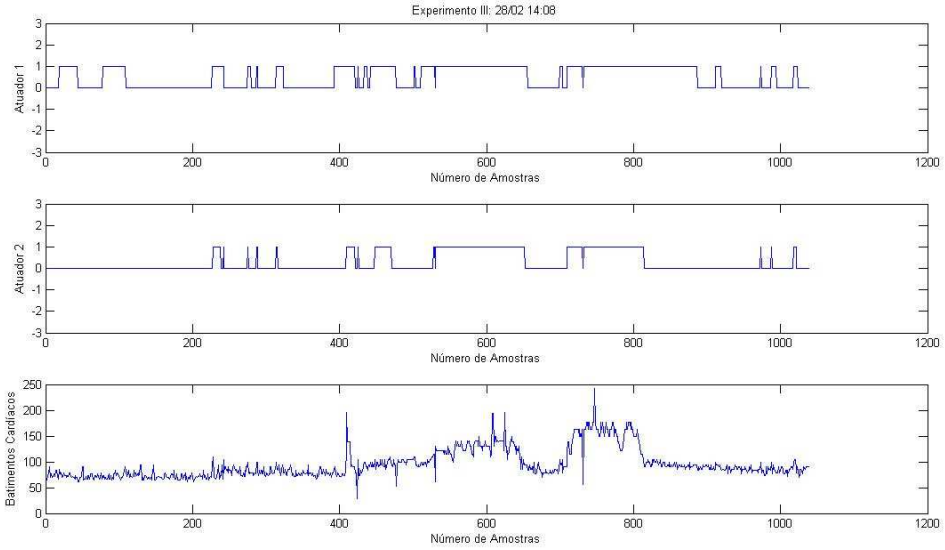

Figura 4.4.2 – Gráficos ‘Atuação dos *Splitters* 1 e 2’ e ‘HR x Tempo’.

Observando a HR, podemos explicar detalhadamente o que aconteceu neste experimento. Iremos explicar este experimento em fases. Lembrando que no gráfico, 300 amostras correspondem a aproximadamente 30 minutos.

A Fase I durou aproximadamente os 30 primeiros minutos. O indivíduo estava sentado descansando, uma atividade que exige um baixo esforço, logo a HR se manteve praticamente constante na faixa de 60 a 80 bpm. Nesta fase, tirando uma

atuação inicial quando o ambiente ainda estava quente, não foi necessária a atuação dos splitters, devido a baixa atividade.

A Fase II durou os próximos trinta minutos. Nela foi realizada uma atividade leve, que se consistiu no individuo sentado, trabalhando no computador. Essa atividade exigiu uma HR na faixa de 70 a 90 bpm. Foi exigido pouca atuação por parte dos aparelhos, já que o nível de atividade também não era alto. Algumas atuações curtas foram registradas devido a erros na variável de batimento cardíaco.

Em seguida tivemos a Fase III. Foi realizada uma atividade leve em pé, que durou cerca de 20 minutos, exigindo uma HR na faixa de 90-120 bpm. Aqui já começamos a ter uma atuação maior dos splitters. Esta faixa de esforço já começa exigir um pouco mais de atuação para que se mantenha o conforto térmico.

Aumentando a intensidade do exercício para uma atividade média em pé, exigindo uma atividade mais intensa do individuo, na faixa de 120-150 bpm. Esta foi a Fase IV e durou, também, cerca de 20 minutos. Podemos notar uma atuação praticamente continua durante toda essa fase, exigida pela emissão de calor do corpo humano devido a atividade mais intensa, o que gera desconforto.

Na Fase V, o indivíduo sentou e descansou por cerca de 10 minutos. Sua HR foi caindo de 150bpm até cerca de 90 bpm. Durante um tempo o aparelho continuo atuando até o PMV voltar a zona de conforto.

Logo após o descanso, a Fase VI, exigiu bastante esforço, tentando trabalhar sempre acima de 150 bpm, ou seja, uma atividade bem intensa. Durou cerca de 20 minutos. Assim como na Fase IV, a atuação foi constante devido ao índice PMV elevado pela HR alta.

Por fim, temos a Fase VII, onde novamente o individuo sentou e descansou por cerca de 30 minutos. Sua HR foi caindo de 180 bpm até por volta de 80 bpm, quando foi encerrado o experimento. Observamos que, principalmente no ambiente 1, os compressores continuaram atuando por um tempo até que após um certo tempo de descanso, e retorno da HR para uma faixa menos intensa, pudesse desligar os aparelhos.

Em algumas amostras do experimento, novamente podemos perceber alguns valores discrepantes, pelos mesmos motivos já explicados nos dois experimentos anteriores. Novamente foram poucas vezes que isso ocorreu. Apesar de ter ligado e desligado os compressores desnecessariamente, não foi algo que interferiu nos resultados como um todo.

Também podemos notar, novamente, que a presença do indivíduo no ambiente 1, fez a atuação neste setor ser maior.

4.5 Análise Geral dos Experimentos

Após a realização de todos os experimentos, pode-se constatar que, para diferentes tipos de cenários e parâmetros, o modulo construído neste projeto se mostrou bastante eficaz em sua função de monitoramento de batimentos cardíacos. Nota-se algumas discrepâncias que podem ser relacionadas com o curto alcance dos módulos e até mesmo medições errôneas de batimentos cardíacos realizados pelo próprio modulo. Notamos também que o próprio relógio do fabricante POLAR apresentava alguns valores errados para os batimentos cardíacos.

No geral acreditamos ter atingido o propósito desse projeto, pois os resultados dos experimentos foram completamente satisfatórios.

Para uma melhor eficiência do sistema, concluímos que pode ser feito um tratamento de erros, no supervisório, para o módulo receptor de batimentos cardíacos. Esse tratamento deve ser feito para que os valores discrepantes que ocorrem de HR não interfiram na atuação dos compressores, evitando os chaveamentos desnecessários, preservando a vida útil do aparelho. Uma sugestão de como isso seria feito é estabelecer um limite para os valores de HR com base na ultima amostra recebida, ou seja, se o novo valor recebido for muito maior ou muito menor, o valor atual será ignorado e será feita uma nova aquisição dos dados. Outra sugestão seria que a atuação só ocorresse após três amostras consecutivas do PMV que acarretassem na mudança de estado dos atuadores.

5 Conclusão

5.1 Considerações finais

A maior parte das edificações hoje em dia apresenta sistemas de ar condicionado centrais, os de maior porte, ou aparelhos de janela ou *split* no caso dos mais antigos ou de menor porte. Porém estes sistemas, na maior parte das vezes, não atuam nem próximos de uma eficiência considerada boa. Ou seja, não se aproximam nem de um conforto térmico razoável para a maioria no ambiente e ficam ligados durante muito mais tempo do que o necessário, consumindo muita energia.

O que propomos neste trabalho é uma alternativa a esses tipos de sistema, que busca um controle dos equipamentos de ar condicionado para se ter uma economia de energia e um conforto térmico adequado aos usuários do ambiente.

O fato de ligar e desligar automaticamente os equipamentos é uma das principais vantagens do sistema proposto. Nos sistemas mais populares, o operador do equipamento dificilmente vai ficar ligando e desligando o aparelho conforme a necessidade, e isso além de um maior gasto energético, muitas vezes tornam o ambiente muito frio, saindo da zona de conforto térmico.

A rede de sensores utilizada neste trabalho apresentou-se flexível e eficaz para automação predial.

A implementação feita sobre a variável “nível de atividade” foi bem interessante, pois muitas vezes apesar da temperatura não estar variando no ambiente, o usuário pode estar fazendo diferentes atividades físicas que, por sua vez, contribuirão de formas diferentes para seu conforto térmico. Pudemos perceber isso nos testes realizados, quando situações desde sentado, até correndo no ambiente, nos proporcionou ver a atuação dos *splitters*, mesmo com o ambiente estando na mesma temperatura.

Alguns pontos negativos podem ser destacados neste projeto, como a baixa resolução dos sensores de temperatura, que variam de 1 em 1º C, resultando em uma resolução também limitada do PMV. Além disso, as perdas de pacotes que acontecem esporadicamente na comunicação da rede ZigBee, quando falta sincronia entre os end devices e coordenador, podem gerar atuações imprecisas dos aparelhos, como ligar ou desligar de forma equivocada. A limitação de alcance dos módulos ZigBee também é algo que pode ser aprimorado.

Em relação ao circuito receptor de HR, esporadicamente percebemos uma interpretação errônea dos batimentos, ora pela perda de um sinal ECG enviada, ora por interferências, ocasionando assim taxas de batimentos cardíacos erradas e atuação indevida dos *splitters*.

Apesar destes pontos negativos e ainda necessitar de refinamento, o projeto proposto se mostrou bem adequado a situações de controle sobre ambientes e situações reais, com uma aplicação direta no mercado e na sociedade. Entrando nessa “onda verde” que vive o planeta na busca de uma diminuição de gastos de recursos naturais, a economia energética aqui proposta é muito bem vinda.

5.2 Perspectivas Futuras

Como forma de aprimorar este sistema de automação predial para conforto térmico e corrigir alguns pontos negativos, algumas sugestões podem ser feitas.

Um aumento no alcance geral da rede de sensores poderia ser feita através de módulos roteadores, criando um enlace de comunicação entre os dispositivos fora do alcance do módulo coordenador, assim a limitação de alcance dos módulos Zigbee seria superada.

Algumas variáveis para o cálculo do PMV (algumas adicionadas neste projeto, relacionadas à HR), como o sexo, peso e tipo de vestimenta, continuam sendo selecionadas no software supervisório, o que não é interessante. Para melhorar a questão destas variáveis, através de um sistema como o de reconhecimento facial, poder-se-ia identificar a pessoa que entrou no ambiente, e depois de cadastrada num banco de dados, dados referentes ao sexo e ao peso seriam atualizados automaticamente para o cálculo do PMV. Nesta mesma idéia, poderia ser feita uma tentativa de identificar o tipo de vestimenta, comparando com diversos trajes cadastrados no banco de dados.

Outra idéia seria rastrear o posicionamento do usuário no ambiente. Quando se tem o dado sobre a localização, muitas coisas podem ser feitas, como o desligamento automático dos equipamentos, quando o usuário sair daquele ambiente.

Em relação aos batimentos cardíacos, além do calculo de PMV, poderia usar este dado como um acompanhamento para a segurança do usuário no caso de um eventual problema de saúde. Caso a taxa estivesse muito alta ou muito baixa, por certo tempo, ativaria um alarme, para alertar médicos ou uma central de emergência.

Além disso, adaptar este projeto para atender a vários usuários simultaneamente, cada um com seu medidor de batimentos cardíacos, e tentar respeitar o nível de conforto térmico de cada um no ambiente seria um desafio para projetos futuros.

6 Referências Bibliográficas

BAUMANN, Chris. Tips *for selecting a Media Access Controller for ZigBee*. ***Industrial Control Designline: TechOnline Community*.** 28 ago. 2006. Disponível em:.<http://www.industrialcontroldesignline.com/192300912;jsessionid=QL0APYY5BZ5A0QSNDLQCKIKCJUNN2JVN?printableArticle=true>.

COUTO, F. L., FIGUEREDO, L. F. C., (2008). Medição Móvel de Conforto Térmico para Rede de Automação Predial *Wireless*. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-nº 011/2008, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF.

DVORAK, Joseph. ***IEEE 802.15.4 and Zigbee Overview*.** Motorola, 2005, 26f. Disponível em: <www.media.mit.edu/resenv/classes/MAS961/readings/802-15-4_Tutorial.ppt>.

FILHO, P.R.M. & DIAS, Y.F.G., (2008). Acionamento de potência para rede de automação wireless. Trabalho de Graduação em Engenharia de Controle e Automação,Publicação FT.TG-nº 012, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF.

GALLO, E. A., RIBEIRO, F. N., (2007). Índice de Conforto Térmico ISO7730 em Automação Predial. Trabalho de Graduação em Engenharia de Controle e Automação, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF.

Queiroz, R. B., Azevedo, R. C A., (2009) Rede de sensores sem fio para automação predial com módulos MeshBean, Trabalho de Graduação em Engenharia Elétrica, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF.

HEILE, Bob. ***ZigBee Alliance Tutorial*.** 2005. Disponível em: <http://www.cs.ucdavis.edu/~aksoy/course/w06/slides/ZigBeeTutorial_05.ppt>.

INDRIA, Y. Design of an individual mobile measurement of thermal comfort. 2006. Tese de mestrado. Universidade de Kaiserslautern, Alemanha.

MWG- ZigBee_Aliance_Overview.pdf

PINHEIRO, José Maurício S. As Redes com ZigBee. **Projeto de Redes**. 27 jul. 2004. Disponível em: <http://www.projetoderedes.com.br/artigos/artigo_zigbee.php>.

AVILA, A. G. , SALOIO, B. H., (2009) Instrumentação e controle de um sistema de ar condicionado híbrido, Trabalho de Graduação em Engenharia de Controle e Automação, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF.

ATMEL CORPORATION, *BitCloud User Guide* 2009, disponível em http://www.atmel.com/dyn/resources/prod_documents/doc8199.pdf .

MESHNETICS, *ZigBit™ Development Kit 2.0 User's Guide*, manual fornecido em CD pelo fabricante.

ANEXO I

Trechos relevantes do código do aplicativo supervisório.

AI.1 Comunicação do software com a porta COM

O trecho de código responsável pela tentativa de estabelecer a comunicação pode ser observado a seguir:

```
'SE A PORTA ESTIVER ABERTA, FECHA A COMUNICAÇÃO
        If COM_port.IsOpen Then

            COM_port.Close()

        End If

        'PASSANDO OS PARÂMETROS PARA A COMUNICAÇÃO
        With COM_port

            'PORTA SELECIONADA
            .PortName = porta_comunicação

            'BAUDRATE
            .BaudRate = Convert.ToInt64(porta_baudrate)

            'BIT DE DADOS
            .DataBits = Convert.ToInt64(porta_bit_dados)

            'BITS DE PARADA
            If porta_bit_parada = "1" Then

                .StopBits = StopBits.One

            End If

            If porta_bit_parada = "1,5" Then

                .StopBits = StopBits.OnePointFive

            End If

            If porta_bit_parada = "2" Then

                .StopBits = StopBits.Two

            End If

            If porta_bit_parada = "Nenhum" Then

                .StopBits = StopBits.None

            End If

            'PARIDADE
            If porta_paridade = "Par" Then

                .Parity = Parity.Even
```

```
            End If

            If porta_paridade = "Ímpar" Then

                .Parity = Parity.Odd

            End If

            If porta_paridade = "Nenhum" Then

                .Parity = Parity.None

            End If

            If porta_paridade = "Marca" Then

                .Parity = Parity.Mark

            End If

            If porta_paridade = "Espaço" Then

                .Parity = Parity.Space

            End If

        End With

        'TENTANDO ABRIR A COMUNICAÇÃO COM A PORTA ESCOLHIDA
        Try

            'ABRINDO A CONEXÃO COM A PORTA
            COM_port.Open()

            'MUDANDO O STATUS DA CONEXÃO E A FONTE
            Label13.Text = "CONECTADO"
            Label13.Font = New Font("Microsoft Sans Serif",
Label13.Font.Size, FontStyle.Bold)
            Label13.ForeColor = Color.Green

            'MUDANDO A IMAGEM DO STATUS DA CONEXÃO
            PictureBox1.Image = Image.FromFile(conectado)
            PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage

        Catch ex As Exception

            MessageBox.Show("Não foi possível iniciar a comunicação com a
porta: " + porta_comunicação + "!", "Problema na comunicação")
            MsgBox(ex.ToString)

        End Try
```

Os parâmetros padrões para a utilização dos módulos MeshBean, foram configurados da seguinte forma:

```
'VARIÁVEIS PARA A COMUNICAÇÃO COM A PORTA
    Public porta_comunicação As String = ""
    Public porta_baudrate As String = "38400"
    Public porta_bit_dados As String = "8"
    Public porta_bit_parada As String = "1"
    Public porta_paridade As String = "Nenhum"
```

```
    Public porta_controle As String = "Nenhum"
```

AI.2 Recebendo os dados

A seguir, pode-se ver como se recebe e se interpreta os dados obtidos do módulo coordenador, que contêm informações como temperatura, HR e nível de bateria.

```
Private Sub COM_port_DataReceived(ByVal sender As System.Object, _
                                  ByVal e As
System.IO.Ports.SerialDataReceivedEventArgs) _
                                  Handles COM_port.DataReceived

        'FUNÇÃO PARA RODAR AUTOMATICAMENTE
        'QUANDO UM DADO FOI ENVIADO PELO MICROCONTROLADOR
        controle_fluxo_dados += 1

        msg_recebida = ""

        Dim str As String = ""

        If e.EventType = SerialData.Chars Then

            Do

                Dim bytecount As Integer = COM_port.BytesToRead

                If bytecount = 0 Then

                    Exit Do

                End If

                Dim byteBuffer(bytecount) As Byte

                COM_port.Read(byteBuffer, 0, bytecount)

                Dim resposta As String = ""

                'teste
                Dim ponteiro As String = 0
                Dim stringteste As String = ""

                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
                                                               ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    temp_sensor_1s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While
                stringteste = ""
```

```
                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
                                                               ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    temp_sensor_2s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While
                stringteste = ""
                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
                                                               ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    temp_atuador_1s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While
                stringteste = ""
                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
                                                               ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    temp_atuador_2s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While
                stringteste = ""
                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
                                                               ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    bat_sensor_1s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While
                stringteste = ""
                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
```

```
                                                                ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    bat_sensor_2s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While

                stringteste = ""

                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
                                                                ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    bpm_s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While

                stringteste = ""

                While (ponteiro < byteBuffer.Length - 3)
                    Dim msg As String = ""
                    msg = System.Text.Encoding.ASCII.GetString(byteBuffer,
_
                                                                ponteiro, 1)
                    stringteste = stringteste & _
                    System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                         ponteiro, 1)
                    ponteiro += 1
                    bat_bpm_s = stringteste
                    If msg = " " Then
                        Exit While
                    End If
                End While

                stringteste = ""

                'fim do teste
                For i As Integer = 0 To byteBuffer.Length - 3

                    str = str &
System.Text.Encoding.ASCII.GetString(byteBuffer, _
                                                                    i, 1)

                Next

                'FAZENDO A COMPARAÇÃO DOS DADOS QUE CHEGARAM
                'PARA ATUALIZAR OU MANTER VALOR ANTERIOR

                'temp sensor1
                Dim temp As Integer = 0
                temp = Convert.ToInt16(temp_sensor_1s)
                If (temp > 10) And (temp < 45) Then
                    temp_sensor_1 = Convert.ToInt16(temp_sensor_1s)
```

```
            Else
                If temp = 0 Then
                    temp_sensor_1 = set_point
                    temp_sensor_1s = set_point.ToString
                End If
            End If

            'temp sensor 2
            temp = 0
            temp = Convert.ToInt16(temp_sensor_2s)
            If (temp > 10) And (temp < 45) Then
                temp_sensor_2 = Convert.ToInt16(temp_sensor_2s)
            Else
                If temp = 0 Then
                    temp_sensor_2 = set_point
                    temp_sensor_2s = set_point.ToString
                End If
            End If

            'temp atuador1
            temp = 0
            temp = Convert.ToInt16(temp_atuador_1s)
            If (temp > 10) And (temp < 45) Then
                temp_atuador_1 = Convert.ToInt16(temp_atuador_1s)
            Else
                If temp = 0 Then
                    temp_atuador_1 = set_point
                    temp_atuador_1s = set_point.ToString
                End If
            End If

            'temp atuador2
            temp = 0
            temp = Convert.ToInt16(temp_atuador_2s)
            If (temp > 10) And (temp < 45) Then
                temp_atuador_2 = Convert.ToInt16(temp_atuador_2s)
            Else
                If temp = 0 Then
                    temp_atuador_2 = set_point
                    temp_atuador_2s = set_point.ToString
                End If
            End If


            bat_sensor_1 = Convert.ToInt16(bat_sensor_1s)

            bat_sensor_2 = Convert.ToInt16(bat_sensor_2s)

            bat_bpm = Convert.ToInt16(bat_bpm_s)

            'End If

            'MessageBox.Show(bat_sensor_1s + " " + bat_sensor_2s)

            bpm = Convert.ToInt16(bpm_s)

            msg_recebida = str

        Loop

    End If

    Me.Invoke(New myDelegate(AddressOf updateTextBox), New Object() {})
```

```
    End Sub
```

AI.3 Configuração dos parâmetros para o Conforto Térmico

Os cálculos referentes a estes parâmetros podem ser observados no trecho de código a seguir:

```
'VALORES PADRÕES PARA O CONFORTO TÉRMICO
  velocidade_vento = 1.0
  umidade_ar = 40
  temp_média_radiante = 27.5


'FUNÇÃO QUE CALCULA O FATOR DE VESTUÁRIO

    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ComboBox1.SelectedIndexChanged

        Select Case ComboBox1.SelectedItem

            Case "Nu"
                i_vest = 0
                i_r_vest = 0

            Case "Calções"
                i_vest = 0.1
                i_r_vest = 0.016

            Case "Tropical"
                i_vest = 0.3
                i_r_vest = 0.047

            Case "Leve de Verão"
                i_vest = 0.5
                i_r_vest = 0.078

            Case "Trabalho"
                i_vest = 0.7
                i_r_vest = 0.124

            Case "Inverno, interior"
                i_vest = 1.0
                i_r_vest = 0.155

            Case "Fato Completo"
                i_vest = 1.5
                i_r_vest = 0.233

        End Select

        calcula_fator_vestuário()

    End Sub
```

```
    'FUNÇÃO QUE CALCULA O FATOR DE VESTUÁRIO
    Private Function calcula_fator_vestuário()


        If (i_r_vest < 0.078) Then

            f_vest = 1.0 + (1.29 * i_r_vest)

        End If

        If (i_r_vest >= 0.078) Then

            f_vest = 1.05 + (0.645 * i_r_vest)

        End If

        ComboBox1.Text = ComboBox1.Text + " (" + i_vest.ToString + ")"

        Return Nothing
    End Function

'CALCULANDO O METABOLISMO

Private Function calc_met()
    If bpm = 0 Then
       bpm = 80
    End If

'Homens
    If gender = 0 Then

       If metabolismo < 3 Then
          metabolismo = 4.56 - 0.0265 * bpm - 0.1506 * weight + 0.00189 *
bpm * weight
       End If

       If metabolismo > 3 Then
       metabolismo = 3.56 - 0.0138 * bpm - 0.1358 * weight + 0.00189 * bpm
* weight
       End If

    End If

'Mulheres
    If gender = 1 Then
       If metabolismo < 3 Then
          metabolismo = -4.7 + 0.0449 * bpm - 0.0019 * weight + 0.00052 *
bpm * weight

       End If

       If testemetab > 3 Then
       metabolismo = -5.92 + 0.0577 * bpm - 0.0167 * weight + 0.00052 * bpm
* weight
       End If

    End If

    metabolismo = (metabolismo * 69.78) / (1.8 * 58.15)
```

AI.4 Cálculo da Equação do Conforto

O trecho de código responsável por calcular a equação do conforto pode ser analisado a seguir.

```
        Dim pmv As Double = 0

        Dim acumulação_de_calor As Double = 0
        Dim convecção As Double = 0
        Dim radiação As Double = 0
        Dim respiração_sensível As Double = 0
        Dim respiração_latente As Double = 0
        Dim transpiração As Double = 0
        Dim difusão_de_vapor As Double = 0
        Dim metabolismo_e_trabalho As Double = 0

        Dim pressão_parcial_vapor As Double = 0
        Dim p_sat As Double = 0
        Dim t_vest As Double = 0

        calc_met()

        'parte referente à metabolismo e trabalho da equação
        metabolismo_e_trabalho = metabolismo - trabalho

        'calculando p_sat para pressão_parcial_vapor
        p_sat = 1000 * Math.Pow(Math.E, (16.6536 - (4030.183 / (temp_área +
235))))

        'calculando pressão_parcial_vapor
        pressão_parcial_vapor = umidade_ar * p_sat

        'parte referente à difusão de vapor da equação
        difusão_de_vapor = -(3.05 * Math.Pow(10, -3) * (5733 - 6.99 *
(metabolismo - trabalho) - pressão_parcial_vapor))

        'parte referente à transpiração
        transpiração = -(0.42 * ((metabolismo - trabalho) - 58.15))

        'parte referente à respiração_latente
        respiração_latente = -(1.7 * Math.Pow(10, -5) * metabolismo * (5867
- pressão_parcial_vapor))

        'parte referente à respiração sensível
        respiração_sensível = -(0.0014 * metabolismo * (34 - temp_área))

    Dim t_vest_interação As Double = temp_área
    Dim teste As String = "interação s/ interação t_pele convecção
radiação" + vbCrLf
    Dim teste1 As Double = 0
    Dim interações As Integer = 0
    While (True)

      interações += 1

      'calculando t_vest
      t_vest = calcular_t_vest(t_vest_interação, temp_área)

      'calcula convecção
      convecção = calcular_convecção(t_vest_interação, temp_área)
```

```
      'calcula radiação
      radiação = calcular_radiação(t_vest_interação)

      teste += t_vest.ToString + " " + t_vest_interação.ToString + " " +
convecção.ToString + " " + radiação.ToString + vbCrLf


      If t_vest < t_vest_interação Then

         t_vest_interação -= 1

         While (True)
            interações += 1
            'calculando t_vest
             t_vest = calcular_t_vest(t_vest_interação, temp_área)

            'calcula convecção
             convecção = calcular_convecção(t_vest_interação, temp_área)

            'calcula radiação
            radiação = calcular_radiação(t_vest_interação)

           teste += t_vest.ToString + " " + t_vest_interação.ToString + " "
+ convecção.ToString + " " + radiação.ToString + vbCrLf

           If t_vest < t_vest_interação Then
              t_vest_interação -= 0.1
                While (True)
                 interações += 1
          'calculando t_vest
          t_vest = calcular_t_vest(t_vest_interação, temp_área)

          'calcula convecção
           convecção = calcular_convecção(t_vest_interação, temp_área)

          'calcula radiação
           radiação = calcular_radiação(t_vest_interação)

          teste += t_vest.ToString + " " + t_vest_interação.ToString + " "
+ convecção.ToString + " " + radiação.ToString + vbCrLf

          If t_vest < t_vest_interação Then
             t_vest_interação -= 0.01
               While (True)
               interações += 1
               'calculando t_vest
               t_vest = calcular_t_vest(t_vest_interação, temp_área)

               'calcula convecção
           convecção = calcular_convecção(t_vest_interação, temp_área)

               'calcula radiação
               radiação = calcular_radiação(t_vest_interação)

               teste += t_vest.ToString + " " + t_vest_interação.ToString +
" " + convecção.ToString + " " + radiação.ToString + vbCrLf
                 If t_vest < t_vest_interação Then
                    t_vest_interação -= 0.001
                    'calculando t_vest
                 t_vest = calcular_t_vest(t_vest_interação, temp_área)

                    'calcula convecção
           convecção = calcular_convecção(t_vest_interação, temp_área)
```

```
                    'calcula radiação
                     radiação = calcular_radiação(t_vest_interação)

                     teste += t_vest.ToString + " " +
t_vest_interação.ToString + " " + convecção.ToString + " " +
radiação.ToString + vbCrLf

                                        Exit While
                                    End If
                                    t_vest_interação += 0.001
                                End While
                                Exit While
                            End If
                            t_vest_interação += 0.01
                        End While
                        Exit While
                    End If

                    t_vest_interação += 0.1

                End While

                Exit While

            End If

            t_vest_interação += 1

        End While

'agora calculando a acumulação de calor

   acumulação_de_calor = metabolismo_e_trabalho + difusão_de_vapor +
transpiração + respiração_latente + respiração_sensível + radiação +
convecção

Private Function calcular_t_vest(ByVal iteração As Double, ByVal temp As
Double)
   Dim t_pele As Double = 0
   Dim resultado As Double = 0

   'calculando t_pele para utilizar em t_vest
   t_pele = (35.7 - (0.0275 * (metabolismo - trabalho)))

   'calculando t_vest
   resultado = (t_pele - (i_r_vest * ((f_vest * (12.1 *
(Math.Sqrt(velocidade_vento))) * (iteração - temp)) + (3.96 * Math.Pow(10,
-8) * f_vest * ((Math.Pow((iteração + 273), 4)) -
(Math.Pow((temp_média_radiante + 273), 4)))))))

    Return resultado

End Function


Private Function calcular_radiação(ByVal iteração As Double)

    Dim resultado As Double = 0

    'parte referente à radiação da equação do pmv
    resultado = -(3.96 * Math.Pow(10, -8) * f_vest * ((Math.Pow((iteração +
273), 4)) - (Math.Pow((temp_média_radiante + 273), 4))))
```

```
    Return resultado

End Function


Private Function calcular_convecção(ByVal iteração As Double, ByVal temp As
Double)

    Dim resultado As Double = 0

    'calculando parte referente a convecção da equação do pmv
    resultado = -(f_vest * (12.1 * (Math.Sqrt(velocidade_vento))) *
(iteração - temp))

    Return resultado

End Function
```

AI.5 Cálculo do PMV

O trecho de código responsável pelo cálculo do índice PMV pode ser observado a seguir:

```
'calculando o pmv
      pmv = acumulação_de_calor * ((0.303 * Math.Pow(Math.E, -(0.042 *
metabolismo))) + 0.028)
```

AI.6 Cálculo do PPD

O trecho de código responsável pelo cálculo do índice PPD pode ser observado a seguir:

```
    Private Function calcular_PPD(ByVal PMV As Double)
        Dim ppd As Double = 0

        ppd = 100 - (95 * Math.Pow(Math.E, ((-0.03353 * Math.Pow(PMV, 4)) -
(0.2179 * Math.Pow(PMV, 2)))))

        Return ppd

    End Function

End Class
```

AI.7 Envio do sinal de controle

O trecho de código responsável por verificar se existe a necessidade de atuação dos compressores dos aparelhos de ar-condicionado, e conseqüente o envio do sinal de controle pode ser observado a seguir:

```
    Private Function verificar_atuação()

        Dim atuar As Boolean = False

        If ((cálculo_PMV(temp_sensor_1) >= 0.5) And (split_1 =
"desligado")) Then

            atuar = True

            split_1 = "ligado"
            PictureBox2.Image = Image.FromFile(ar_ligado)
            PictureBox2.SizeMode = PictureBoxSizeMode.StretchImage
            With Form3.dados

                .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
                .SelectionColor = Color.Blue
                .AppendText("Comando ligar split 1" + vbCrLf)
                .ScrollToCaret()

            End With

        End If

        If (cálculo_PMV(temp_sensor_1) <= 0.5 And (split_1 = "ligado"))
Then

            atuar = True

            split_1 = "desligado"
            PictureBox2.Image = Image.FromFile(ar_desligado)
            PictureBox2.SizeMode = PictureBoxSizeMode.StretchImage
            With Form3.dados

                .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
                .SelectionColor = Color.Blue
                .AppendText("Comando desligar split 1" + vbCrLf)
                .ScrollToCaret()

            End With

        End If

        If ((cálculo_PMV(temp_sensor_2) >= 0.5) = True And (split_2 =
"desligado")) Then

            atuar = True

            split_2 = "ligado"
            PictureBox3.Image = Image.FromFile(ar_ligado)
            PictureBox3.SizeMode = PictureBoxSizeMode.StretchImage
            With Form3.dados

                .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
                .SelectionColor = Color.Blue
                .AppendText("Comando ligar split 2" + vbCrLf)
                .ScrollToCaret()
```

```
            End With

        End If

        If ((cálculo_PMV(temp_sensor_2) <= 0.5) = True And (split_2 =
"ligado")) Then

            atuar = True

            split_2 = "desligado"
            PictureBox3.Image = Image.FromFile(ar_desligado)
            PictureBox3.SizeMode = PictureBoxSizeMode.StretchImage
            With Form3.dados

                .Font = New Font("Garamond", 12.0!, FontStyle.Bold)
                .SelectionColor = Color.Blue
                .AppendText("Comando desligar split 2" + vbCrLf)
                .ScrollToCaret()

            End With

        End If

        If atuar = True Then

            'MessageBox.Show("entrou")
            atuar_split()

        End If

        Return Nothing

    End Function

    Private Function atuar_split()

        auto_save_excel()

        If split_1 = "ligado" And split_2 = "ligado" Then

            COM_port.Write("1")
            COM_port.Write("1")

        End If

        If split_1 = "ligado" And split_2 = "desligado" Then

            COM_port.Write("1")
            COM_port.Write("0")

        End If

        If split_1 = "desligado" And split_2 = "ligado" Then

            COM_port.Write("0")
            COM_port.Write("1")

        End If


        If split_1 = "desligado" And split_2 = "desligado" Then

            COM_port.Write("0")
            COM_port.Write("0")
```

```
        End If

        Return Nothing

    End Function
```

ANEXO II

Trechos relevantes do código do algoritmo gravado nos microcontroladores.

AII.1 Low Power

```
/*****************************************************************************//**
  \file lowpower.c

  \brief Low-Power application implementation.

  \author
    Luiz Thiago Santos
   André Vidal Shinoda

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
    History:
*****************************************************************************/

#include <lowpower.h>

/******************************************************************************
  Global variables
******************************************************************************/
AppState_t appState = APP_INITIAL_STATE;                    // Current application
state
AppDeviceState_t appDeviceState = DEVICE_ACTIVE_IDLE_STATE;    // Current
device state
DeviceType_t appDeviceType;
// Endpoint simple descriptor (ZDO endpoint descriptor)
SimpleDescriptor_t simpleDescriptor = {APP_ENDPOINT, APP_PROFILE_ID, 1, 1,
0, 0 , NULL, 0, NULL};


/******************************************************************************
  Local variables
******************************************************************************/
static HAL_AppTimer_t networkTimer;                        // Timer indicating network
start

static APS_RegisterEndpointReq_t apsRegisterEndpointReq;       // APS Register
Endpoint Request primitive (APS endpoint descriptor)

// ZDO primitives
```

```
static ZDO_StartNetworkReq_t  zdoStartNetworkReq;           // Request parameters
for network start


/**************************************************************************************
  Static functions
**************************************************************************************/
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *conf);
static void initApp(void);                              // Common application initial function
static void startNetwork(void);                          // Start Network
static void startingNetworkTimerFired(void);
static void APS_DataInd(APS_DataInd_t *ind);


/**************************************************************************************
  Implementation
**************************************************************************************/

/*************************************************************************
  Description: Application task handler
  Parameters: none.
  Returns:    none
*************************************************************************/
void APL_TaskHandler()
{
  switch (appState)
  {
    // node is in initial state
    case APP_INITIAL_STATE:               // Initial (after RESET) state
      initApp();                        // Init application
      break;

    case APP_NETWORK_JOINING_STATE:        // Network is in the joining stage
      startNetwork();                    // Start/joing network
      break;

    case APP_NETWORK_JOINED_STATE:         // Network was successfully started
#ifdef _COORDINATOR_
      if (DEVICE_TYPE_COORDINATOR == appDeviceType)
        appCoordinatorTaskHandler();
#endif // _COORDINATOR_
#ifdef _ENDDEVICE_
      if (DEVICE_TYPE_END_DEVICE == appDeviceType)
        appEndDeviceTaskHandler();
#endif // _ENDDEVICE_
      break;

    default:
      break;
  }
}
```

```
/**************************************************************************
  Description:  application and stack parameters init
  Parameters:   none
  Returns:      none
**************************************************************************/
static void initApp(void)
{
  ShortAddr_t nwkAddr;
  bool rxOnWhenIdle;

  appState = APP_INITIAL_STATE;

  // Read NWK address as dipswitch's state.
  nwkAddr = appReadSliders();

  if (0 == nwkAddr)
  {
#ifdef _COORDINATOR_
    appDeviceType = DEVICE_TYPE_COORDINATOR;
    rxOnWhenIdle = true;

    appCoordinatorInit();
#else
    return; // This device can not be coordinator
#endif // _COORDINATOR_
  }
  else
  {
#ifdef _ENDDEVICE_
    appDeviceType = DEVICE_TYPE_END_DEVICE;
    rxOnWhenIdle = false;

    appEndDeviceInit();
#else
    return; // This device can not be end device
#endif // _ENDDEVICE_
  }

  // Set parameters to config server
  CS_WriteParameter(CS_NWK_ADDR_ID, &nwkAddr);
  CS_WriteParameter(CS_DEVICE_TYPE_ID, &appDeviceType);
  CS_WriteParameter(CS_RX_ON_WHEN_IDLE_ID, &rxOnWhenIdle);

  appOpenLeds();
  appState = APP_NETWORK_JOINING_STATE;

  SYS_PostTask(APL_TASK_ID);
}

/**************************************************************************
  Description: ZDO_StartNetwork primitive confirmation was received.
  Parameters: confirmInfo - confirmation information
```

```
  Returns:    none
*****************************************************************/
void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confInfo)
{
  HAL_StopAppTimer(&networkTimer);              // Network join state indication timer
stopping

  // Joined network successfully
  if (ZDO_SUCCESS_STATUS == confInfo->status)     // Network was started
successfully
  {
    appState = APP_NETWORK_JOINED_STATE;          // Application state switching
    appDeviceState = DEVICE_ACTIVE_IDLE_STATE;    // Device state setting

    // Turn network indication on
    appOnLed(APP_NETWORK_STATUS_LED);

    // Set application endpoint properties and register endpoint
    apsRegisterEndpointReq.simpleDescriptor = &simpleDescriptor;
    apsRegisterEndpointReq.APS_DataInd = APS_DataInd;
    APS_RegisterEndpointReq(&apsRegisterEndpointReq);
  }

  SYS_PostTask(APL_TASK_ID);
}

/*****************************************************************
  Description: start network
  Parameters: none.
  Returns:    none
*****************************************************************/
static void startNetwork(void)
{
  // Configure timer for LED blinking during network start
  networkTimer.interval = APP_JOINING_INDICATION_PERIOD;
  networkTimer.mode     = TIMER_REPEAT_MODE;
  networkTimer.callback = startingNetworkTimerFired;
  HAL_StartAppTimer(&networkTimer);

  zdoStartNetworkReq.ZDO_StartNetworkConf = ZDO_StartNetworkConf; // Network
started confirm handler
  ZDO_StartNetworkReq(&zdoStartNetworkReq); // start network

}

/*****************************************************************
  Description: Starting network timer has fired. Toggle LED for blink
  Parameters: none.
  Returns:    none
*****************************************************************/
void startingNetworkTimerFired(void)
{
```

```
  appToggleLed(APP_NETWORK_STATUS_LED);
}

/*************************************************************************
  Description:  Application endpoint indication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************************/
void APS_DataInd(APS_DataInd_t* ind)
{
#ifdef _COORDINATOR_
  if (DEVICE_TYPE_COORDINATOR == appDeviceType)
    appCoordinatorDataInd(ind);
#endif // _COORDINATOR_

#ifdef _ENDDEVICE_
  if (DEVICE_TYPE_END_DEVICE == appDeviceType)
    appEndDeviceDataInd(ind);
#endif // _ENDDEVICE_
}

/*************************************************************************
  Description: Network update notification

  Parameters: ZDO_MgmtNwkUpdateNotf_t *nwkParams - update notification

  Returns: nothing.
*************************************************************************/
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams)
{
  ZDO_StartNetworkConf_t conf;

  if (ZDO_NETWORK_STARTED_STATUS == nwkParams->status)
  {
    conf.status = ZDO_SUCCESS_STATUS;
    ZDO_StartNetworkConf(&conf);
  }
  else if (ZDO_NETWORK_LEFT_STATUS == nwkParams->status)
  {
    appState = APP_NETWORK_JOINING_STATE;
    SYS_PostTask(APL_TASK_ID);
  }
}

// eof lowpower.c
```

AII.2 Coordinator

```
/******************************************************************************//**
  \file coordinator.c

  \brief Lowpower application: Coordinator part of application implementation.

  \author
    Luiz Thiago Santos
    André Vidal Shinoda
  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
    History:
******************************************************************************/

#ifdef _COORDINATOR_

#include <lowpower.h>
#include <util/delay.h>
#include <halInterrupt.h>

/******************************************************************************
  Global variables
 ******************************************************************************/
/******************************************************************************
  Local variables
 ******************************************************************************/

// Temporary data received via network buffer
static uint8_t tmpDataBuffer[APP_TMP_DATA_BUFFER_SIZE];
static uint8_t tmpDataBufferActualLength = 0;

// USART related variables
static HAL_UsartDescriptor_t appUsartDescriptor;       // USART descriptor (required
by stack)
static bool usartTxBusyFlag = false;                   // USART transmission transaction
status
static uint8_t usartTxBuffer[APP_USART_TX_BUFFER_SIZE]];// USART Tx buffer

static APS_DataReq_t apsDataReq;                       // APS Data Request primitive
(for application message sending)
static AppMessageBuffer_t appMessageBuffer;            // Application message
buffer
static AppDataTransmissionState_t appDataTtransmissionState =
APP_DATA_TRANSMISSION_IDLE_STATE;

static HAL_AppTimer_t sendMessageTimer;

static uint16_t temp_sensor_1;
static uint16_t temp_sensor_2;
static uint16_t temp_atuador_1;
static uint16_t temp_atuador_2;
```

```
static uint16_t bat_sensor_1;
static uint16_t bat_sensor_2;
static uint16_t bat_atuador_1;
static uint16_t bat_atuador_2;

//Criando variáveis para o receptor de batimentos cardiacos
static uint16_t bpm;
static uint16_t bat_bpm;

static uint8_t rxBuffer[USART_RX_BUFFER_LENGTH];// read buffer
static uint8_t read_msg[10] = "";
static uint16_t readBytesCount=0;

static uint8_t atuador_1;
static uint8_t atuador_2;
static ShortAddr_t destAdress;

static ShortAddr_t remetAdress;

static bool controle = false;
static uint8_t status_atuador_1;
static uint8_t status_atuador_2;



/*****************************************************************************
  Local functions
 *****************************************************************************/
static void usartInit(void);
static void sendDataToUsart(uint8_t* data, uint8_t length);
static void usartWriteConf(void);

static void messageInit(void);
static void APS_DataConf(APS_DataConf_t *conf);

//static void sendControlMessage(void);

static void timerInit(void);
static void sendUsartMessage(void);

static void appReadByteEvent(uint8_t readBytesLen);
//static void appRead(void);
static void sendControlMessageAt1(void);
static void sendControlMessageAt2(void);



/*****************************************************************
  Description:  Coordinator initialization routine
  Parameters:   none
  Returns:      none
*****************************************************************/
```

```
void appCoordinatorInit(void)
{

usartInit(); // Open USART
  messageInit();

}



/*************************************************************************
  Description:  Device common task handler
  Parameters:   none
  Returns:      none
*************************************************************************/
void appCoordinatorTaskHandler(void)
{
  switch (appDeviceState)                 // Actual device state when one joined network
  {
          case DEVICE_ACTIVE_IDLE_STATE:
     {

            timerInit();



            break;

                    }
    default:
      break;
  }
}

/*************************************************************************
  Description:  Data intication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************************/

void timerInit(void)
{

  // Configure timer for send message to usart
  sendMessageTimer.interval = 2000;
  sendMessageTimer.mode     = TIMER_REPEAT_MODE;
  sendMessageTimer.callback = sendUsartMessage;
  HAL_StartAppTimer(&sendMessageTimer);


}
```

```
/*************************************************************************
  Description:  Data intication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************************/

void sendUsartMessage(void)
{

  HAL_DisableInterrupts();

  uint8_t str[100];
  uint8_t length;

  if (DEVICE_TYPE_COORDINATOR == appDeviceType)
  {
    length = sprintf((char *) str, "%d %d %d %d %d %d %d %d\r\n",
                      temp_sensor_1 , temp_sensor_2 , temp_atuador_1 ,
                                      temp_atuador_2, bat_sensor_1 , bat_sensor_2 ,
                                      bpm , bat_bpm);


    sendDataToUsart(str, length); // write received data to USART
  }

  // Data received indication
  appToggleLed(APP_RECEIVING_STATUS_LED);



        //PARTE PARA ENVIAR OS DADOS PARA OS ATUADORES
        //DE 2 EM 2 SEGUNDOS

        switch (controle)
        {
                case false:
                {
                        controle = true;

                        if(status_atuador_1 != atuador_1)
                        {
                                appMessageBuffer.message.control = atuador_1;
                                sendControlMessageAt1();
                        }
                        break;
                }

                case true:
                {
                        controle = false;
                        if(status_atuador_2 != atuador_2)
                        {
```

```
                        appMessageBuffer.message.control = atuador_2;
                        sendControlMessageAt2();
                    }
                    break;
                }
            }

            HAL_EnableInterrupts();

}

/*************************************************************************
  Description:  Data intication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************************/

void appCoordinatorDataInd(APS_DataInd_t* ind)
{
  uint8_t str[100];
  uint8_t length;

  AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;
  remetAdress = ind->srcAddress.shortAddress;

  if(ind->srcAddress.shortAddress == 1)
  {

        temp_sensor_1 = appMessage->temperature;
        bat_sensor_1 = appMessage->battery;
  }
  if(ind->srcAddress.shortAddress == 2)
  {

        temp_sensor_2 = appMessage->temperature;
        bat_sensor_2 = appMessage->battery;
  }
  if(ind->srcAddress.shortAddress == 3)
  {
        temp_atuador_1 = appMessage->temperature;
        bat_atuador_1 = appMessage->battery;
        status_atuador_1 = appMessage->porta;
  }
  if(ind->srcAddress.shortAddress == 4)
  {
        temp_atuador_2 = appMessage->temperature;
        bat_atuador_2 = appMessage->battery;
        status_atuador_2 = appMessage->porta;
  }

  // Para o modulo receptor de batimentos cardiacos
  if(ind->srcAddress.shortAddress == 5)
```

```
  {
        bpm = appMessage->batimento;
        bat_bpm = appMessage->battery;
  }

}

/*****************************************************************
  Description: Init USART, register USART callbacks.

  Parameters: none.

  Returns: nothing.
*****************************************************************/
void usartInit(void)
{
  usartTxBusyFlag = false;

  appUsartDescriptor.tty            = APP_USART_CHANNEL;
  appUsartDescriptor.mode           = USART_MODE_ASYNC;
  appUsartDescriptor.baudrate       = USART_BAUDRATE_38400;
  appUsartDescriptor.dataLength     = USART_DATA8;
  appUsartDescriptor.parity         = USART_PARITY_NONE;
  appUsartDescriptor.stopbits       = USART_STOPBIT_1;
  appUsartDescriptor.rxBuffer       = rxBuffer;
  appUsartDescriptor.rxBufferLength  = USART_RX_BUFFER_LENGTH;
  appUsartDescriptor.txBuffer       = NULL; // use callback mode
  appUsartDescriptor.txBufferLength  = 0;
  appUsartDescriptor.rxCallback     = appReadByteEvent;
  appUsartDescriptor.txCallback     = usartWriteConf;
  appUsartDescriptor.flowControl    = USART_FLOW_CONTROL_NONE;

  HAL_OpenUsart(&appUsartDescriptor);

}

/*****************************************************************
  Description: Init USART, register USART callbacks.

  Parameters: none.

  Returns: nothing.
*****************************************************************/
static void messageInit(void)
{

  apsDataReq.dstAddrMode           = APS_SHORT_ADDRESS;                 // Short
addressing mode
  apsDataReq.dstAddress.shortAddress = destAdress;                      // Destination
node short address       destAdress
  apsDataReq.dstEndpoint           = APP_ENDPOINT;                     // Destination
endpoint
```

```
  apsDataReq.profileId              = APP_PROFILE_ID;                    // Profile ID
  apsDataReq.clusterId              = APP_CLUSTER_ID;                    // Destination
cluster ID
  apsDataReq.srcEndpoint            = APP_ENDPOINT;                      // Source
endpoint
  apsDataReq.asduLength             = sizeof (AppMessage_t);             // ASDU size
  apsDataReq.asdu                   = (uint8_t *) &appMessageBuffer.message; // ASDU
pointer as an application message
  apsDataReq.txOptions.acknowledgedTransmission = 1;                     //
Acknowledged transmission enabled
  apsDataReq.radius                 = 0;                                 // Default radius
  apsDataReq.APS_DataConf           = APS_DataConf;                      // Confirm
handler


}

/*************************************************************************
  Description: Init USART, register USART callbacks.

  Parameters: none.

  Returns: nothing.
*************************************************************************/

static void APS_DataConf(APS_DataConf_t *conf)
{

  if (APS_SUCCESS_STATUS == conf->status) // Data transmission was
successfully performed
  {
        appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; //
Data transmission entity is idle
  }
  else
  {
    appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
  }

  BSP_OffLed(LED_RED);

}


/*************************************************************************
  Description: Init USART, register USART callbacks.

  Parameters: none.

  Returns: nothing.
*************************************************************************/
static void sendControlMessageAt1(void)
```

```
{
//appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
destAdress = 3;
messageInit();
//appMessageBuffer.message.control = atuador_1;
if(appDataTtransmissionState == APP_DATA_TRANSMISSION_IDLE_STATE)
{
appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;    //
Data transmission entity is busy while sending not finished
APS_DataReq(&apsDataReq);
}


}
/*****************************************************************
  Description: Init USART, register USART callbacks.

  Parameters: none.

  Returns: nothing.
*****************************************************************/
static void sendControlMessageAt2(void)
{
//appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
destAdress = 4;
messageInit();
//appMessageBuffer.message.control = atuador_2;
if(appDataTtransmissionState == APP_DATA_TRANSMISSION_IDLE_STATE)
{
appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;    //
Data transmission entity is busy while sending not finished
APS_DataReq(&apsDataReq);
}
}
/*****************************************************************
  Description: Send Data To Usart

  Parameters: data - pointer to frame being sent to USART
              length - length of the frame in bytes

  Returns: nothing.
*****************************************************************/
void sendDataToUsart(uint8_t* data, uint8_t length)
{
  if (APP_TMP_DATA_BUFFER_SIZE > tmpDataBufferActualLength + length)
  {
    memcpy(&tmpDataBuffer[tmpDataBufferActualLength], data, length);
    tmpDataBufferActualLength += length;
  }

  if (false == usartTxBusyFlag)
  {
```

```
    usartWriteConf();
  }
}

/*****************************************************************************
  Description: Read each byte and store in array.

  Parameters: are not used.

  Returns: nothing.
*****************************************************************************/
static void appReadByteEvent(uint8_t readBytesLen)
{

 BSP_OnLed(LED_RED);

 readBytesLen = readBytesLen;

 HAL_ReadUsart(&appUsartDescriptor,&read_msg,1);

 readBytesCount++;

if(readBytesCount == 1)
 {
     atuador_1 = (uint8_t)atoi(read_msg);
          read_msg[0] = "";

 }
 if(readBytesCount == 2)
 {

    atuador_2 = (uint8_t)atoi(read_msg);
    readBytesCount=0;

          read_msg[0] = "";

    BSP_OffLed(LED_RED);

 }

}

/*****************************************************************
  Description: Writing confirmation has been received. New message can be sent.

  Parameters: none.

  Returns: nothing.
*****************************************************************/
void usartWriteConf(void)
{
  int bytesWritten;
```

```
  if (0 < tmpDataBufferActualLength)  // data waiting to be written to USART
  {
    memcpy(usartTxBuffer, tmpDataBuffer, tmpDataBufferActualLength);
    bytesWritten = HAL_WriteUsart(&appUsartDescriptor, usartTxBuffer,
tmpDataBufferActualLength);
    if (0 < bytesWritten)
    {
      tmpDataBufferActualLength -= bytesWritten;
      usartTxBusyFlag = true;
    }
  }
  else
  {
    usartTxBusyFlag = false;
  }
}

#endif // _COORDINATOR_

// eof coordinator.c
```

AII.3 End Device (Sensores e Atuadores)

```
/**************************************************************************//**
  \file enddevice.c

  \brief Lowpower application: end device part of application implementation.

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).
```

```
  \internal
    History:
******************************************************************************/

#ifdef _ENDDEVICE_
#include <lowpower.h>
#include <avr/io.h>
#include <gpio.h>
#include <util/delay.h>

/******************************************************************************
  Global variables
 ******************************************************************************/
/******************************************************************************
  Local variables
 ******************************************************************************/
static AppDataTransmissionState_t appDataTtransmissionState =
APP_DATA_TRANSMISSION_IDLE_STATE;
static APS_DataReq_t apsDataReq;                    // APS Data Request primitive
(for application message sending)
static ZDO_SleepReq_t zdoSleepReq;                  // Request parameters for
stack sleep
static ZDO_WakeUpReq_t zdoWakeUpReq;                // Request parameters for
stack awakening
static AppMessageBuffer_t appMessageBuffer;         // Application message
buffer
static uint8_t sliders;

static bool atuador;
static HAL_AppTimer_t sendDataTimer;

static uint8_t status_porta;
//static int teste;
static uint16_t batimento;

/******************************************************************************
  Static functions
******************************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf);       // Sleep confirmation
handler
static void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf);     // Wake up
confirmation handler
static void APS_DataConf(APS_DataConf_t *conf);         // Data transmission
confirmation handler

static void temperaturesSensorHandler(bool result, int16_t temperature);
static void openPeriphery(void);                    // Open LEDs and Temperature
Sensor
static void closePeriphery(void);                   // Close LEDs and Temperature
Sensor
static void sendMessage(void);                      // Send the application message
```

```
static void prepareToSleep(void);

static void batterySensorHandler(int16_t battery);
static void batimentoSensorHandler(int16_t batimento);

static void timerInit(void);
static void sendData(void);

//void appAtuadorDataInd(APS_DataInd_t* ind);


/************************************************************************
  Description:  End device initialization routine
  Parameters:   none
  Returns:      none
************************************************************************/
void appEndDeviceInit(void)
{

  GPIO_ADC_INPUT_1_make_out();

  //Analisa para ver se é atuador
    sliders = appReadSliders();
        if (sliders > 2)
        {
                atuador = true;
        }
        if((sliders > 0)&(sliders < 3)||(sliders == 5))
        {
                atuador = false;
        }

    // Prepare APS Data Request
  apsDataReq.dstAddrMode           = APS_SHORT_ADDRESS;                  // Short
addressing mode
  apsDataReq.dstAddress.shortAddress = 0;                              // Destination
node short address
  apsDataReq.dstEndpoint           = APP_ENDPOINT;                     // Destination
endpoint
  apsDataReq.profileId             = APP_PROFILE_ID;                   // Profile ID
  apsDataReq.clusterId             = APP_CLUSTER_ID;                     // Destination
cluster ID
  apsDataReq.srcEndpoint           = APP_ENDPOINT;                     // Source
endpoint
  apsDataReq.asduLength             = sizeof (AppMessage_t);             // ASDU size
  apsDataReq.asdu                  = (uint8_t *) &appMessageBuffer.message; // ASDU
pointer as an application message
  apsDataReq.txOptions.acknowledgedTransmission = 1;                    //
Acknowledged transmission enabled
  apsDataReq.radius                = 0;                                // Default radius
  apsDataReq.APS_DataConf          = APS_DataConf;                     // Confirm
handler
```

```
  BSP_OpenTemperatureSensor();
  BSP_OpenBatterySensor();

}

/*************************************************************************
  Description:  Device common task handler
  Parameters:   none
  Returns:      none
*************************************************************************/
void appEndDeviceTaskHandler(void)
{
  switch (appDeviceState)                  // Actual device state when one joined network
  {
    case DEVICE_ACTIVE_IDLE_STATE:           // Device ready to temperature
measuring
        {
#ifdef _TEMPERATURE_SENSOR_
     BSP_ReadTemperatureData(temperaturesSensorHandler); // Temperature
measuring
#else
     temperaturesSensorHandler(false, 0);
#endif // _TEMPERATURE_SENSOR_

          batimentoSensorHandler(batimento);
          BSP_ReadBatteryData(batterySensorHandler);


          break;
          }

    case DEVICE_MESSAGE_SENDING_STATE:       // Message sending state
     sendMessage();                       // Application message sending
     break;

    case DEVICE_SLEEP_PREPARE_STATE:         // Prepare to sleep state
     {
            prepareToSleep();                 // Prepare to sleep
          break;
          }

    case DEVICE_AWAKENING_STATE:                          // Awakening state
        {
          BSP_OpenBatterySensor();
     zdoWakeUpReq.ZDO_WakeUpConf = ZDO_WakeUpConf; // ZDO WakeUp
confirm handler defining
     ZDO_WakeUpReq(&zdoWakeUpReq);             // ZDO WakeUp Request
sending
     break;
        }
    default:
```

```
      break;
  }
}

/*****************************************************************
  Description:  Data intication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*****************************************************************/
void appEndDeviceDataInd(APS_DataInd_t* ind)
{

  uint8_t control = 0;
  AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;
  ind = ind; // Warning prevention

  control = appMessage->control;

  if(control == 1)
  {
    BSP_OnLed(LED_YELLOW);
        GPIO_ADC_INPUT_1_set();
  }
  if(control == 0)
  {
    BSP_OffLed(LED_YELLOW);
        GPIO_ADC_INPUT_1_clr();
  }

}

/*****************************************************************
  Description:  Open LEDs and Sensor
  Parameters:   none
  Returns:      none
*****************************************************************/
static void openPeriphery(void)
{
  appOpenLeds();

  BSP_OpenBatterySensor();


#ifdef _TEMPERATURE_SENSOR_
  BSP_OpenTemperatureSensor();
#endif // _TEMPERATURE_SENSOR_
}

/*****************************************************************
  Description:  Close LEDs and Sensor
  Parameters:   none
  Returns:      none
```

```
*****************************************************************/
static void closePeriphery(void)
{
  appOffLed(APP_NETWORK_STATUS_LED);
  appOffLed(APP_SENDING_STATUS_LED);
  appOffLed(APP_RECEIVING_STATUS_LED);
  appCloseLeds();

  BSP_CloseTemperatureSensor();
  BSP_CloseBatterySensor();
}

/*****************************************************************
  Description:  Data sent handler
  Parameters:   conf - APS Data Confirm primitive
  Returns:      none
*****************************************************************/
static void APS_DataConf(APS_DataConf_t *conf)
{
  appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; // Data
transmission entity is idle

  appOffLed(APP_SENDING_STATUS_LED);

  if (APS_SUCCESS_STATUS == conf->status) // Data transmission was
successfully performed
  {
     appDeviceState = DEVICE_SLEEP_PREPARE_STATE; // Switch device state to
prepare for sleep
  }
  else
  {
     appDeviceState = DEVICE_ACTIVE_IDLE_STATE; // Data transmission wasn't
successfully finished. Retry.
  }


  SYS_PostTask(APL_TASK_ID);
}


/*****************************************************************
  Description:  Temperature measured handler
  Parameters:   result - measurement status (true - success, 0 - fail)
                temperature - value measured
  Returns:      none
*****************************************************************/
static void temperaturesSensorHandler(bool result, int16_t temperature)
{
  // Temperature measured will be sent as an application message
  if (true == result)
    appMessageBuffer.message.temperature = temperature;
```

```
  else
    appMessageBuffer.message.temperature = 0;

}

/*************************************************************************
  Description:  Temperature measured handler
  Parameters:   result - measurement status (true - success, 0 - fail)
                temperature - value measured
  Returns:      none
*************************************************************************/
static void batterySensorHandler(int16_t battery)
{
        appMessageBuffer.message.battery = battery;

        appDeviceState = DEVICE_MESSAGE_SENDING_STATE; //Switch device
state to application message sending
    SYS_PostTask(APL_TASK_ID); // Application task posting

}

/*************************************************************************
  Description:  Send the application message
  Parameters:   none
  Returns:      none
*************************************************************************/
static void sendMessage(void)
{

  //Leitura da porta
  if (atuador == true)
  {
        status_porta = GPIO_ADC_INPUT_1_read();
        appMessageBuffer.message.porta = status_porta;
  }

  if (APP_DATA_TRANSMISSION_IDLE_STATE == appDataTtransmissionState)  //
If previous data transmission was finished
  {
    appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;     //
Data transmission entity is busy while sending not finished
    APS_DataReq(&apsDataReq);
    appOnLed(APP_SENDING_STATUS_LED);
  }

}


/*************************************************************************
  Description:  ZDO Sleep Confirm handler
  Parameters:   conf - ZDO Sleep Confirm primitive
  Returns:      none
```

```
*************************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf)
{

  if (ZDO_SUCCESS_STATUS == conf->status)   // Stack was slept successfully
  {

          if(atuador == false)
          {
            closePeriphery();                    // LEDs and Temperature Sensor closing
          }
    appDeviceState = DEVICE_SLEEP_STATE;    // Device actually slept
  }
  else
    SYS_PostTask(APL_TASK_ID);              // Still in current state.
                                    // Application task posting for attempt repeat.
}

/*************************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
*************************************************************************/
static void prepareToSleep(void)
{
  if(atuador == false)
  {
  zdoSleepReq.ZDO_SleepConf = ZDO_SleepConf;  // Sleep Confirm handler
defining
  ZDO_SleepReq(&zdoSleepReq);               // Sleep Request sending
  }
  else
  {
  appDeviceState = DEVICE_SLEEP_STATE;
  timerInit();
  }
}

/*************************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
*************************************************************************/
static void timerInit(void)
{

  // Configure timer for send message to usart
  sendDataTimer.interval = 20000;
  sendDataTimer.mode     = TIMER_REPEAT_MODE;
  sendDataTimer.callback = sendData;
  HAL_StartAppTimer(&sendDataTimer);
```

```
}

/*************************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
*************************************************************************/

static void sendData(void)
{
  HAL_StopAppTimer(&sendDataTimer);
  appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
  SYS_PostTask(APL_TASK_ID);

}

/*************************************************************************
  Description: Device wakeup handler.
  Parameters: none
  Returns:    none
*************************************************************************/
static void wakeUpHandler(void)
{

  appDeviceState = DEVICE_ACTIVE_IDLE_STATE;

  openPeriphery();

  // Turn network indication on
  appOnLed(APP_NETWORK_STATUS_LED);

  SYS_PostTask(APL_TASK_ID);
}

/*************************************************************************
  Description: End device wake up indication

  Parameters: none.

  Returns: nothing.
*************************************************************************/
void ZDO_WakeUpInd(void)
{

  if (DEVICE_SLEEP_STATE == appDeviceState)
  wakeUpHandler();

}

/*************************************************************************
  Description: Wake up confirmation handler
```

```
  Parameters: conf - confirmation parameters

  Returns: nothing.
******************************************************************************/
void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf)
{

  if (ZDO_SUCCESS_STATUS == conf->status)
  wakeUpHandler();
  else
  SYS_PostTask(APL_TASK_ID);

}

#endif // _ENDDEVICE_

// eof enddevice.c
```

AII.4 End Device (Medidor de batimentos cardíacos)

```
/**************************************************************************//**
  \file enddevice.c

  \brief Lowpower application: end device part of application implementation.

  \author
    Luiz Thiago Santos
  André Vidal Shinoda

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
    History:
******************************************************************************/

#ifdef _ENDDEVICE_
#include <lowpower.h>
#include <avr/io.h>
#include <gpio.h>
#include <util/delay.h>
#include <irq.h>
#include <halInterrupt.h>
```

```
/**********************************************************************************
  Global variables
 **********************************************************************************/
/**********************************************************************************
  Local variables
 **********************************************************************************/
static AppDataTransmissionState_t appDataTtransmissionState =
APP_DATA_TRANSMISSION_IDLE_STATE;
static APS_DataReq_t apsDataReq;                    // APS Data Request primitive
(for application message sending)
static ZDO_SleepReq_t zdoSleepReq;                  // Request parameters for
stack sleep
static ZDO_WakeUpReq_t zdoWakeUpReq;                // Request parameters for
stack awakening
static AppMessageBuffer_t appMessageBuffer;         // Application message
buffer
static uint8_t sliders;

static bool atuador;
static HAL_AppTimer_t sendDataTimer;

static uint8_t status_porta;
static uint16_t cont = 0;
static uint16_t batimento;
static uint16_t bpm1;
static uint16_t bpm;
static uint16_t virada = -32768;
static uint16_t bpm2;
static uint16_t bpm22;
static double bpm3;

/**********************************************************************************
  Static functions
**********************************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf);       // Sleep confirmation
handler
static void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf);     // Wake up
confirmation handler
static void APS_DataConf(APS_DataConf_t *conf);         // Data transmission
confirmation handler

static void openPeriphery(void);                    // Open LEDs and Temperature
Sensor
static void closePeriphery(void);                   // Close LEDs and Temperature
Sensor
static void sendMessage(void);                      // Send the application message
static void prepareToSleep(void);

static void batterySensorHandler(int16_t battery);

static void timerInit(void);
static void sendData(void);
```

```
static void testeInterrupcao2(void);

/*****************************************************************
  Description:  Interrupcao
  Parameters:   none
  Returns:      none
*****************************************************************/

void funcao2 (void)
{

  HAL_DisableIrq(IRQ_7);
  cont++;
  if (cont == 4)
  {
    HAL_DisableIrq(IRQ_7);
          BSP_OffLed(LED_YELLOW);
    bpm2 = HAL_GetSystemTime();
    bpm22 = bpm2 ;
    bpm3 = (bpm22 - bpm);
    bpm3 = bpm3/1000;
    bpm3 = bpm3/60;
    batimento = (int) (3 + (1/bpm3));
          cont = 0;
    appMessageBuffer.message.batimento = batimento;
    appDeviceState = DEVICE_MESSAGE_SENDING_STATE; //Switch device state
to application message sending
    SYS_PostTask(APL_TASK_ID); // Application task posting

  }
  else
    HAL_EnableIrq(IRQ_7);

}

void funcao (void)
{

  HAL_DisableIrq(IRQ_6);

  BSP_OnLed(LED_YELLOW);
  bpm1 = (int) HAL_GetSystemTime();
  bpm = bpm1;

}


static void testeInterrupcao(void)
{
  HAL_RegisterIrq (IRQ_6 , IRQ_RISING_EDGE , funcao);
  HAL_EnableIrq(IRQ_6);
```

```
}

static void testeInterrupcao2(void)
{
  HAL_RegisterIrq (IRQ_7 , IRQ_RISING_EDGE , funcao2);
  HAL_EnableIrq(IRQ_7);

}

/*************************************************************************
  Description:  End device initialization routine
  Parameters:   none
  Returns:      none
*************************************************************************/
void appEndDeviceInit(void)
{

  GPIO_ADC_INPUT_1_make_out();

  //Analisa para ver se é atuador
    sliders = appReadSliders();
        if (sliders > 2)
        {
                atuador = true;
        }
        if((sliders > 0)&(sliders < 3)||(sliders == 5))
        {
                atuador = false;
        }

    // Prepare APS Data Request
  apsDataReq.dstAddrMode           = APS_SHORT_ADDRESS;                  // Short
addressing mode
  apsDataReq.dstAddress.shortAddress = 0;                              // Destination
node short address
  apsDataReq.dstEndpoint           = APP_ENDPOINT;                      // Destination
endpoint
  apsDataReq.profileId             = APP_PROFILE_ID;                    // Profile ID
  apsDataReq.clusterId              = APP_CLUSTER_ID;                    // Destination
cluster ID
  apsDataReq.srcEndpoint           = APP_ENDPOINT;                      // Source
endpoint
  apsDataReq.asduLength             = sizeof (AppMessage_t);             // ASDU size
  apsDataReq.asdu                  = (uint8_t *) &appMessageBuffer.message; // ASDU
pointer as an application message
  apsDataReq.txOptions.acknowledgedTransmission = 1;                    //
Acknowledged transmission enabled
  apsDataReq.radius                = 0;                                 // Default radius
  apsDataReq.APS_DataConf          = APS_DataConf;                      // Confirm
handler

  BSP_OpenTemperatureSensor();
```

```
  BSP_OpenBatterySensor();


}

/************************************************************************
  Description:  Device common task handler
  Parameters:   none
  Returns:      none
************************************************************************/
void appEndDeviceTaskHandler(void)
{



  switch (appDeviceState)                 // Actual device state when one joined network
  {
    case DEVICE_ACTIVE_IDLE_STATE:          // Device ready to temperature
measuring
          {
#ifdef _TEMPERATURE_SENSOR_
      BSP_ReadTemperatureData(temperaturesSensorHandler); // Temperature
measuring
#else
      temperaturesSensorHandler(false, 0);
#endif // _TEMPERATURE_SENSOR_

            BSP_ReadBatteryData(batterySensorHandler);
            testeInterrupcao();
           testeInterrupcao2();

            break;
            }

    case DEVICE_MESSAGE_SENDING_STATE:       // Message sending state
     {
            HAL_UnregisterIrq(IRQ_6);
            HAL_UnregisterIrq(IRQ_7);
            sendMessage();                      // Application message sending
      break;
            }

    case DEVICE_SLEEP_PREPARE_STATE:         // Prepare to sleep state
     {
              prepareToSleep();                 // Prepare to sleep
            break;
            }

    case DEVICE_AWAKENING_STATE:                          // Awakening state
          {
            BSP_OpenBatterySensor();
```

```
    zdoWakeUpReq.ZDO_WakeUpConf = ZDO_WakeUpConf; // ZDO WakeUp
confirm handler defining
    ZDO_WakeUpReq(&zdoWakeUpReq);              // ZDO WakeUp Request
sending
    break;
      }
   default:
    break;
 }
}

/************************************************************************
  Description:  Data intication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
************************************************************************/
void appEndDeviceDataInd(APS_DataInd_t* ind)
{

  uint8_t control = 0;
  AppMessage_t *appMessage = (AppMessage_t *) ind->asdu;
  ind = ind; // Warning prevention

  control = appMessage->control;

  if(control == 1)
  {
    BSP_OnLed(LED_YELLOW);
        GPIO_ADC_INPUT_1_set();
  }
  if(control == 0)
  {
    BSP_OffLed(LED_YELLOW);
        GPIO_ADC_INPUT_1_clr();
  }

}

/************************************************************************
  Description:  Open LEDs and Sensor
  Parameters:   none
  Returns:      none
************************************************************************/
static void openPeriphery(void)
{
  appOpenLeds();

  BSP_OpenBatterySensor();


#ifdef _TEMPERATURE_SENSOR_
  BSP_OpenTemperatureSensor();
```

```
#endif // _TEMPERATURE_SENSOR_
}

/************************************************************************
  Description:  Close LEDs and Sensor
  Parameters:   none
  Returns:      none
************************************************************************/
static void closePeriphery(void)
{
  appOffLed(APP_NETWORK_STATUS_LED);
  appOffLed(APP_SENDING_STATUS_LED);
  appOffLed(APP_RECEIVING_STATUS_LED);
  appCloseLeds();

  BSP_CloseTemperatureSensor();
  BSP_CloseBatterySensor();
}

/************************************************************************
  Description:  Data sent handler
  Parameters:   conf - APS Data Confirm primitive
  Returns:      none
************************************************************************/
static void APS_DataConf(APS_DataConf_t *conf)
{
  appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; // Data
transmission entity is idle

  appOffLed(APP_SENDING_STATUS_LED);

  if (APS_SUCCESS_STATUS == conf->status) // Data transmission was
successfully performed
  {
    appDeviceState = DEVICE_SLEEP_PREPARE_STATE; // Switch device state to
prepare for sleep
  }
  else
  {
    appDeviceState = DEVICE_ACTIVE_IDLE_STATE; // Data transmission wasn't
successfully finished. Retry.
  }


  SYS_PostTask(APL_TASK_ID);
}


/************************************************************************
  Description:  Temperature measured handler
  Parameters:   result - measurement status (true - success, 0 - fail)
                temperature - value measured
```

```
  Returns:      none
*****************************************************************/
static void temperaturesSensorHandler(bool result, int16_t temperature)
{
  // Temperature measured will be sent as an application message
  if (true == result)
    appMessageBuffer.message.temperature = temperature;
  else
    appMessageBuffer.message.temperature = 0;

}


/*****************************************************************
  Description:  Temperature measured handler
  Parameters:   result - measurement status (true - success, 0 - fail)
                temperature - value measured
  Returns:      none
*****************************************************************/
static void batterySensorHandler(int16_t battery)
{
        appMessageBuffer.message.battery = battery;


}

/*****************************************************************
  Description:  Send the application message
  Parameters:   none
  Returns:      none
*****************************************************************/
static void sendMessage(void)
{

  //Leitura da porta
  if (atuador == true)
  {
        status_porta = GPIO_ADC_INPUT_1_read();
        appMessageBuffer.message.porta = status_porta;
  }

  if (APP_DATA_TRANSMISSION_IDLE_STATE == appDataTtransmissionState)  //
If previous data transmission was finished
  {
    appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;     //
Data transmission entity is busy while sending not finished
    APS_DataReq(&apsDataReq);
    appOnLed(APP_SENDING_STATUS_LED);
  }

}
```

```
/**************************************************************************
  Description:  ZDO Sleep Confirm handler
  Parameters:   conf - ZDO Sleep Confirm primitive
  Returns:      none
**************************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf)
{

  if (ZDO_SUCCESS_STATUS == conf->status)   // Stack was slept successfully
  {

        if(atuador == false)
        {
          closePeriphery();                 // LEDs and Temperature Sensor closing
        }
    appDeviceState = DEVICE_SLEEP_STATE;    // Device actually slept
  }
  else
    SYS_PostTask(APL_TASK_ID);              // Still in current state.
                                // Application task posting for attempt repeat.
}

/**************************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
**************************************************************************/
static void prepareToSleep(void)
{
  if(atuador == false)
  {
  zdoSleepReq.ZDO_SleepConf = ZDO_SleepConf;  // Sleep Confirm handler
defining
  ZDO_SleepReq(&zdoSleepReq);               // Sleep Request sending
  }
  else
  {
  appDeviceState = DEVICE_SLEEP_STATE;
  timerInit();
  }
}

/**************************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
**************************************************************************/
static void timerInit(void)
{

  // Configure timer for send message to usart
```

```
  sendDataTimer.interval = 20000;
  sendDataTimer.mode     = TIMER_REPEAT_MODE;
  sendDataTimer.callback = sendData;
  HAL_StartAppTimer(&sendDataTimer);


}

/*************************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
*************************************************************************/

static void sendData(void)
{
  HAL_StopAppTimer(&sendDataTimer);
  appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
  SYS_PostTask(APL_TASK_ID);

}

/*************************************************************************
  Description: Device wakeup handler.
  Parameters: none
  Returns:    none
*************************************************************************/
static void wakeUpHandler(void)
{

  appDeviceState = DEVICE_ACTIVE_IDLE_STATE;

  openPeriphery();

  // Turn network indication on
  appOnLed(APP_NETWORK_STATUS_LED);

  SYS_PostTask(APL_TASK_ID);
}

/*************************************************************************
  Description: End device wake up indication

  Parameters: none.

  Returns: nothing.
*************************************************************************/
void ZDO_WakeUpInd(void)
{

  if (DEVICE_SLEEP_STATE == appDeviceState)
  wakeUpHandler();
```

```
}

/**************************************************************************
  Description: Wake up confirmation handler

  Parameters: conf - confirmation parameters

  Returns: nothing.
**************************************************************************/
void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf)
{

  if (ZDO_SUCCESS_STATUS == conf->status)
  wakeUpHandler();
  else
  SYS_PostTask(APL_TASK_ID);

}

#endif // _ENDDEVICE_

// eof enddevice.c
```