Fábio Meurer

Concepção e Análise de Dispositivos Aplicados à Internet das Coisas

Joinville, SC

2015

Fábio Meurer

Concepção e Análise de Dispositivos Aplicados à Internet das Coisas

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica da Universidade Federal de Santa Catarina, Campus de Joinville.

Universidade Federal de Santa Catarina – UFSC

Centro de Joinville

Bacharelado em Engenharia Mecatrônica

Orientador: Prof. Dr. Giovani Gracioli

Coorientador: Prof. Dr. Anderson Wedderhoff Spengler

Joinville, SC

2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Meurer, Fabio
Concepção e Análise de Dispositivos Aplicados à Internet das Coisas / Fabio Meurer ; orientador, Giovani Gracioli ; coorientador, Anderson Wedderhoff Spengler. - Joinville, SC, 2015.
88 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Joinville. Graduação em Engenharia Mecatrônica.

Inclui referências

1. Engenharia Mecatrônica. 2. Internet da Coisas. 3. Sistemas embarcados. 4. Tomada inteligente. 5. Controle de ar condicionado. I. Gracioli, Giovani. II. Spengler, Anderson Wedderhoff. III. Universidade Federal de Santa Catarina. Graduação em Engenharia Mecatrônica. IV. Título.

Dedico aos meus pais.

Agradecimentos

Agradeço a Deus por ter me dado força e fé para superar as dificuldades.

Agradeço aos meus pais, Irineu e Alice, e ao meu irmão Lindomar, por todo o amor, suporte, dedicação e exemplo de perseverança. Agradeço pela educação que me foi dada e pelo esforço despendido ao longo da minha vida.

Agradeço aos meus orientadores, Prof. Giovani e Prof. Anderson, por ter acreditado no meu trabalho, pela paciência e apoio ao longo destes anos. Agradeço pelas discussões técnicas e pelas conversas informais que contribuíram para minha formação.

Agradeço pela oportunidade de fazer parte do Laboratório de Integração Software e Hardware.

Agradeço a todos os meus amigos, pela amizade e pelos inúmeros bate papos que tornaram esses anos muito mais divertidos.

E a todos que diretamente ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

'Só fazemos melhor aquilo que repetidamente insistimos em melhorar. A busca da excelência não deve ser um objetivo, e sim um hábito.'
(Aristóteles)

Resumo

A Internet das Coisas prescreve um mundo de objetos físicos embarcados com sensores e atuadores, conectados por redes sem fio e que se comunicam usando a Internet, moldando uma rede de objetos interligados. Este trabalho abordou o desenvolvimento de uma tomada inteligente e um controle de ar condicionado integrado a um sistema supervisório por comunicação sem fio. A tomada inteligente é integrada a um módulo medidor capaz de medir: potência ativa, corrente eficaz, tensão eficaz, fator de potência e o consumo acumulado. Em experimentos realizados com algumas cargas de teste, o resultado final dos cálculos da incerteza de medição máxima da tensão eficaz de $4,91\%$ e $1,42\%$ para medição de corrente eficaz. Em outro experimento de medição do consumo acumulado, obteve-se um erro de $5,4\%$ do valor teórico esperado. O desenvolvimento do controle do ar condicionado foi capaz de controlar as principais funções de um controle remoto. A classe desenvolvida do ar condicionado para o envio dos comandos, contém um tamanho de 1759 bytes. Segundo os experimentos, o dispositivo é capaz de operar a uma distância máxima de 4 metros do ar condicionado. O sistema supervisório foi criado no ambiente de desenvolvimento gráfico LabVIEW e proporciona uma interface simples ao usuário para controle dos dispositivos e visualização dos dados da tomada inteligente.

Palavras-chaves: Internet das Coisas. Sistemas Embarcados. Tomada Inteligente. Controle de Ar Condicionado. Sistema Supervisório.

Abstract

The Internet of Things prescribes a world of physical objects embedded with sensors and actuators, connected by wireless networks and communicating using the Internet, shaping a network of interconnected objects. This work addressed the development of a smart plug and an air conditioning control integrated into a supervisory system with a wireless communication. The smart plug is integrated into a meter module capable of measuring: active power, effective current, effective voltage, power factor and accumulated consumption. In experiments with some test charges, the final result of the measurement uncertainty calculations of the effective voltage is $4,91\%$ and $1,42\%$ to the measuring of effective current. Another experiment measuring the accumulated consumption gave an error of $5,4\%$ of the expected theoretical value. The development of the air conditioning control was able to control the main functions of a remote control. The air conditioning class designed for sending commands, contains a 1759 byte size. According to experiments, the device is capable of operating at a maximum distance of 4 meters from the air conditioner. The supervisory system was created in LabVIEW graphical development environment and provides a simple user interface for control devices and visualization of smart plug data.

Key-words: Internet of Things. Embedded Systems. Smart Plug. Air Conditioning Control. Supervisory System.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

A/D	Analógico/Digital
ADESD	*Application-Driven Embedded System Design*
ARM7	*Acorn RISC Machine 7*
ASK	*Amplitude Shift Keying*
EPOS	*Embedded Parallel Operating System*
FSK	*Frequency Shift Keying*
IAR	*Ingenjörsfirman Anders Rundgren*
IC	*Interrupt Controller*
ID	Identificador
IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet of Things* (Internet das Coisas)
IR	Infravermelho
LabVIEW	*Laboratory Virtual Instrument Engineering Workbench*
LED	*Light Emitting Diode*
LISHA	Laboratório de Integração Software/Hardware (*Software/Hardware Integration Lab*)
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
SPI	*Serial Peripheral Interface*
TC	Transformador de Corrente
UART	*Universal Synchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
VI	*Virtual Instrument*

Lista de símbolos

Ω	ohm(s)
^oC	grau(s) Celsius
$\frac{\partial V_b}{\partial R_6}$	derivada parcial de V_b em relação à R_6
$\frac{\partial V_b}{\partial R_9}$	derivada parcial de V_b em relação à R_9
$\frac{\partial V_i}{\partial R_b}$	derivada parcial de V_i em relação à R_b
$\frac{\partial V_i}{\partial I_c}$	derivada parcial de V_i em relação à I_c.
Δt_N	tempo de amostragem
σ_{I_c}	incerteza da corrente I_c
σ_{R_6}	incerteza do resistor R_6
σ_{R_9}	incerteza do resistor R_9
σ_{R_b}	incerteza do resistor de base R_b
σ_{V_b}	incerteza da tensão V_b
σ_{V_i}	incerteza da tensão V_i
A	ampère(s)
A_{rms}	ampères(s) eficaz
C_i	consumo acumulado
C_{i-1}	consumo acumulado do ciclo de cálculo anterior
FP	fator de potência
G_i	ganho da corrente
G_v	ganho da tensão
Hz	hertz
I_c	corrente na carga
I_{RMS}	corrente eficaz

I_s	corrente induzida pelo transformador de corrente
$k\Omega$	quilo-ohm(s)
kHz	quilohertz
kS/s	quilosiemens por segundo
kW	quilowatt(s)
kWh	quilowatt(s)-hora
m	metro(s)
mA_{rms}	miliampère(s) eficaz
ms	milissegundo(s)
mV_{pico}	volt(s) de pico
mV_{rms}	milivolt(s) eficaz
MHz	megahertz
N	número de amostras
ppm	parte(s) por milhão
P	potência ativa
R_b	resistor de base do sensor de corrente
S	potência aparente
V	volt(s)
V_a	tensão na carga
V_b	tensão na entrada do microcontrolador do módulo medidor referente à medição da tensão
V_i	tensão na entrada do microcontrolador do módulo medidor referente à medição da corrente
V_{rms}	volt(s) eficaz
V_{RMS}	tensão eficaz
us	microsegundos
W	watt(s)

Wh	watt(s)-hora
Ws	watt(s)-segundo

Sumário

1 Introdução

O conceito da Internet das Coisas (IoT - *Internet of Things*) começou a ser desenvolvido em 1999 nos laboratórios do Massachusetts Institute of Technology, nos Estados Unidos. A proposta é ligar todas as coisas à Internet, desde sofisticados equipamentos até os mais simples objetos, para que estes possam se comunicar entre si e entre os usuários e consumidores, com o fim de gerar informações a serem usadas nas mais diferentes funções (ASHTON, 2009).

Segundo Atzori, Iera e Morabito (2010), a IoT é um paradigma que preconiza um mundo de objetos físicos embarcados com sensores e atuadores, conectados por redes sem fio e que se comunicam usando a Internet, moldando uma rede de objetos inteligentes capazes de realizarem processamento, capturar variáveis ambientais e reagirem a estímulos externos. Esses objetos interconectam-se entre si e com outros recursos e podem ser controlados através da Internet, permitindo o surgimento de uma miríade de aplicações que poderão se beneficiar dos novos tipos de dados, serviços e operações disponíveis.

Vários avanços tecnológicos recentes possibilitaram o surgimento da IoT, tais como redes de sensores sem fio, comunicação móvel e computação ubíqua. No entanto, há ainda uma série de desafios a serem superados para alavancar a ampla disseminação desse paradigma, principalmente com relação ao desenvolvimento de aplicações e a alta diferença decorrente da inerente diversidade de tecnologias de hardware e software desse ambiente.

O funcionamento da IoT pode ser categorizado em seis tópicos (MAYER, 2009):

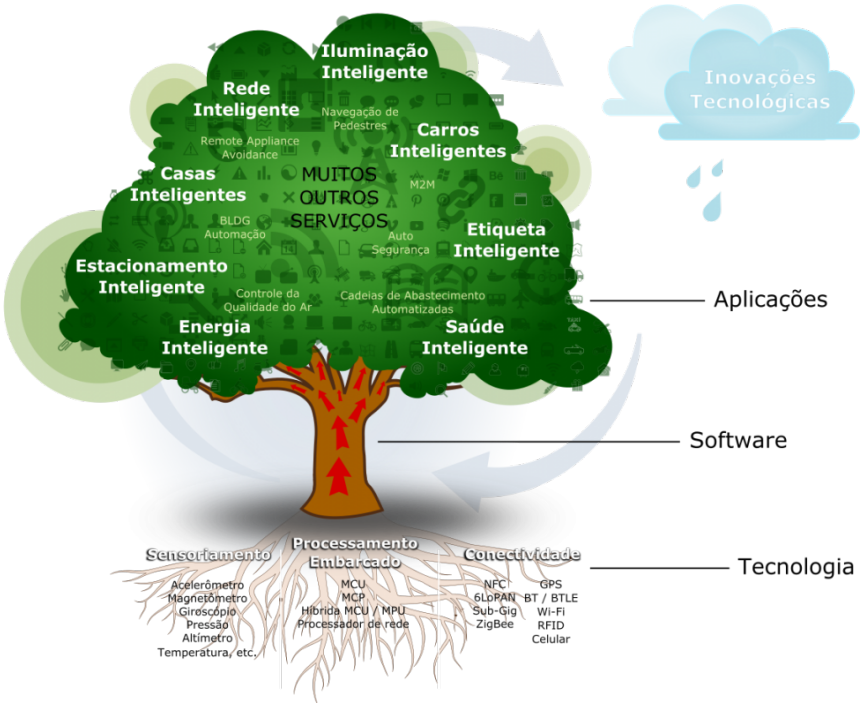
1) Comunicação - É desta forma que os dispositivos trocam dados entre eles;
2) Sensores - São utilizados para capturar e representar o mundo físico no mundo digital;
3) Atuadores - Executam ações no mundo físico desencadeadas no mundo digital;
4) Armazenamento - Guarda os dados recebidos, de todos os dispositivos;
5) Dispositivos - São utilizados pelo usuário na interação com o mundo físico;
6) Processamento - É utilizado como *data mining* e serviços;

A ligação de todos os objetos apresentados na IoT, trás benefícios claros à nossa sociedade, possibilitando maior controle e compreensão de como os sistemas interagem, e em última analise, proporcionando melhor qualidade de vida a todos. A Figura 1 explica as tecnologias necessárias e exemplifica algumas aplicações da IoT.

Umas das aplicações da IoT é no âmbito da Domótica. Com a ajuda de sensores e dispositivos de controle, é possível gerenciar a sua casa a partir da Internet.

A Domótica é uma tecnologia relativamente recente e é responsável pela gestão de

Figura 1 – Infográfico da IoT.



Fonte: Sena (2015).

todos os recursos habitacionais necessários para satisfazer as necessidades de comunicação, segurança e comodidade diária das pessoas (COSTA, 2012). Consiste, basicamente, na automação doméstica das habitações (casa, escritório ou residência).

Bastante difundida nos diversos ramos industriais, a Domótica alcançou as edificações corporativas. A cada dia, novos componentes que agregam tecnologias a ela relacionadas são aplicados às instalações prediais. A aplicação da domótica tem demonstrado que é possível proporcionar ou ampliar benefícios em fatores como: gerenciamento técnico, conforto, economia, prevenção de acidentes e falhas de equipamentos, e também segurança aos usuários.

O gerenciamento do consumo de energia e água, os controles de iluminação, acesso, climatização, comunicação etc., integrados e comandados por um sistema de automação, têm a possibilidade de tornar o ambiente de trabalho do edifício mais produtivo, saudável e eficiente (DIAS, 2004).

Nesta perspectiva, este trabalho visa desenvolver dispositivos ligados à IoT, utilizando o hardware EPOSMote II e o software EPOS como ferramenta principal para concepção do mesmo. Também, será utilizado o LabVIEW para criação de um sistema supervisório para o usuário final. Com os dispositivos construídos, neste trabalho também será analisado o desempenho perante as funcionalidades de cada dispositivo.

1.1 Objetivos

Especificar e desenvolver uma tomada inteligente e um controle de ar condicionado integrado à um sistema supervisório.

Objetivos Específicos

- Projetar e especificar cada dispositivo;
- Construir um protótipo de cada dispositivo;
- Criar um sistema supervisório;
- Testar e avaliar as funcionalidades de cada dispositivo;
- Integrar os dispositivos e avaliar o desempenho.

2 Revisão Teórica

Neste capítulo é apresentado os principais conceitos utilizados neste trabalho de conclusão de curso. Inicialmente, o capítulo trata sobre o sistema operacional EPOS e o hardware EPOSMote II. Em seguida, apresenta-se a linguagem de programação gráfica LabVIEW desenvolvido pela *National Instruments*. Por fim, o capítulo provê uma breve contextualização sobre a comunicação infravermelho dos controles remotos.

2.1 EPOS

O EPOS (*Embedded Parallel Operating System*) é um sistema operacional desenvolvido pelo Laboratório de Integração Software e Hardware (LISHA - *Software/Hardware Integration Lab*) para plataformas embarcadas que implementa muitas das funcionalidades utilizadas em redes de sensores sem-fio. O EPOS visa automatizar o desenvolvimento de sistemas embarcados, de modo que os desenvolvedores possam em suas aplicações (LISHA, 2015a).

O EPOS foi construído um estudo de caso da com metodologia de Projeto de Sistemas Embarcados Guiado pela Aplicação (ADESD - *Application-Driven Embedded System Design*) proposto por Fröhlich (2001) para projetar e implementar os componentes de software e hardware que podem ser automaticamente adaptadas para satisfazer os requisitos de aplicações específicas. Além disso, o sistema EPOS fornece um conjunto de ferramentas para selecionar, adaptar e unir componentes em um *framework* específico à aplicação, possibilitando assim a geração automática de uma instância do sistema orientado à aplicação.

EPOS suporta uma grande variedade de arquiteturas, variando de 8 a 32 bits. Atualmente arquiteturas suportadas são: IA32, AVR8, PPC32, MIPS e ARM7.

No EPOS, os processos são gerenciados pelas abstrações *Thread* e *Task*. Cada *thread* armazena seu contexto em sua própria pilha. A abstração de contexto define o conjunto de dados que precisa ser armazenado para um fluxo de execução e, deste modo, cada arquitetura define seu próprio contexto.

O tempo é tratado pela família de abstrações *Timepiece*. Estas abstrações são suportadas através dos mediadores *Timer*, *Timestamp Counter* (TSC) e *Real-Time Clock* (RTC). A abstração *Clock* é responsável por um controle estrito de tempo e está disponíveis em sistemas que possuam um dispositivo de relógio de tempo real (RTC). A abstração *Alarm* pode ser utilizada para gerar eventos que acordem uma *thread* ou chamem uma função. Alarmes têm ainda um evento mestre de altíssima prioridade que está associado

com um período de tempo pré-definido. Este evento mestre é utilizado para acionar o algoritmo de escalonamento do sistema quando o *quantum* de escalonamento é atingido, nos casos em que uma configuração com um escalonador baseado em tempo é utilizada. A abstração *Chronometer* é utilizada para realizar operações de medição de tempo.

A família de abstrações *Synchronizer* provê mecanismos que garantem a consistência de dados em ambientes com processos concorrentes. O membro *Mutex* implementa um mecanismo de exclusão mútua que entrega duas operações atômicas: *lock* e *unlock*. O membro *Semaphore* implementa, como o próprio nome diz, um semáforo, que é uma variável inteira cujo valor apenas pode ser manipulado indiretamente através das operações atômicas *p* e *v*. O membro *Condition* realiza uma abstração de sistema inspirada no conceito de variável de condição, que permite a uma *thread* esperar que um predicado se torne válido.

Controle de entrada e saída (I/O) de dispositivos periféricos é disponibilizado no EPOS pelo mediador de hardware correspondente. O mediador *Machine* armazena as regiões de I/O e o mediador IC (*Interrupt Controller*) trata a ativação ou desativação de interrupções individuais. Para lidar com as diferentes interrupções existentes em diferentes plataformas e contextos, o EPOS atribui um nome e uma sintaxe independente da plataforma para as interrupções pertinentes ao sistema operacional.

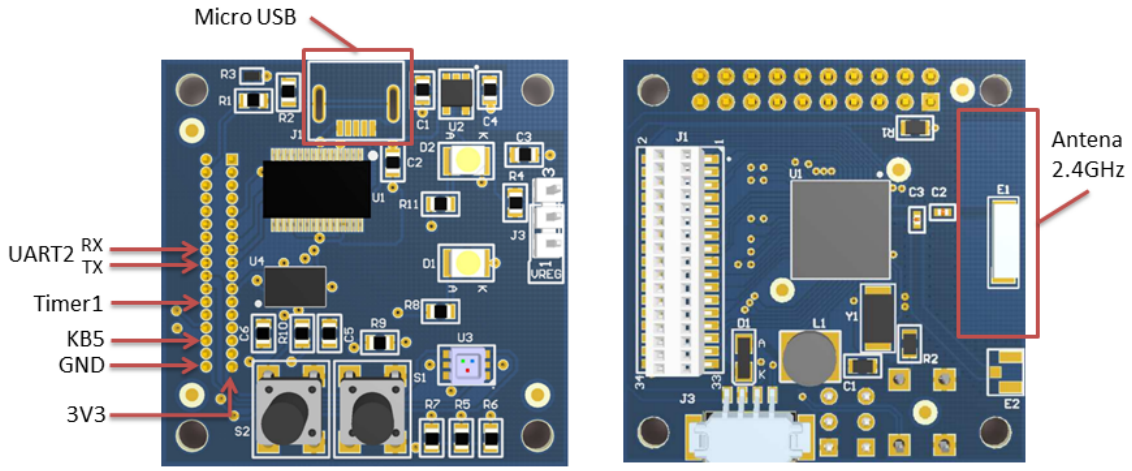
2.2 EPOSMote II

O projeto EPOSMote, do LISHA, inicialmente foi voltado para o desenvolvimento de um módulo de rede de sensores sem fio com base na plataforma ZigBit, e focado em aplicações de Agricultura de Precisão (LISHA, 2015b). A primeira versão, o EPOSMote I, baseia-se no módulo MeshNetics ZigBit MNZB-24-B0, com um controlador AVR de 8-bit, capacidade de comunicação IEEE 802.15.4 e um pequeno conjunto de sensores.

O EPOSMote II, Figura 2, surgiu da necessidade de expandir e da dificuldade de programar o EPOSMote I. Essa plataforma foca em pesquisa e apresenta uma arquitetura modular, permitindo fácil programação e expansão. Seu hardware foi projetado como uma arquitetura de camadas composta por um módulo principal, um módulo de sensoriamento e um módulo de alimentação (LISHA, 2015b).

O módulo principal é responsável pelo processamento, armazenamento de dados e comunicação. Possui um processador ARM7 32-bit, com 128kB de memória flash, 96kB de RAM e um transceptor compatível com o padrão IEEE 802.15.4. O módulo de sensoriamento básico possui um sensor de temperatura e um acelerômetro, LEDs, botões e uma micro USB (que também pode ser utilizada como fonte de alimentação). Ele é ligado ao módulo de processamento através de um conector fêmea. A fonte de alimentação da placa vem de um regulador de tensão que converte $5V$ do barramento USB para $3.3V$.

Figura 2 – EPOSMote II.



Adaptado de: LISHA (2015b).

2.3 LabVIEW

O LabVIEW (*Laboratory Virtual Instrument Engineering Workbench*) é uma ferramenta de software da *National Instruments*, líder na indústria para o projeto de sistemas de teste, medida e controle. Usando o ambiente integrado do LabVIEW para realizar a interface com sinais do mundo real, analisar os dados para informações importantes e compartilhar resultados. A instrumentação virtual poder ser usada em diferentes tipos de aplicações, desde o projeto até a prototipagem e implementação.

LabVIEW é uma linguagem de programação gráfica que utiliza ícones, em vez de linhas de texto, para criar aplicações. Em contraste às linguagens de programação baseadas em texto, em que instruções determinam a execução do programa, o LabVIEW utiliza programação baseada em fluxo de dados, onde o fluxo dos dados determina a execução.

Os programas criados no LabVIEW são chamados de instrumentos virtuais ou simplesmente VIs (*Virtual Instrument*). Controles são entradas e indicadores são saídas. Cada VI é composto de partes principais:

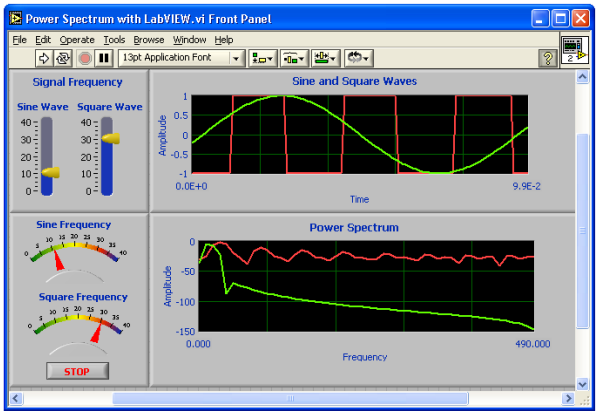
- Painel frontal – Onde o usuário interage com o VI;
- Diagrama de blocos – O código que controla o programa;
- Painel de conector e ícone – Define o modo de conectar o VI a outros VIs.

No LabVIEW, constrói-se uma interface de usuário, utilizando um conjunto de ferramentas e objetos. A interface de usuário é conhecida como Painel Frontal, Figura 3. Adiciona-se o código utilizando representações gráficas de funções para controlar os objetos do painel frontal. O diagrama de bloco (Figura 4) contém esse código, e sob certos aspectos ele se assemelha a um fluxograma.

A interação quando o programa esta em execução é realizada no painel frontal. Pode-se controlar o programa, mudar entradas e visualizar os dados atualizados.

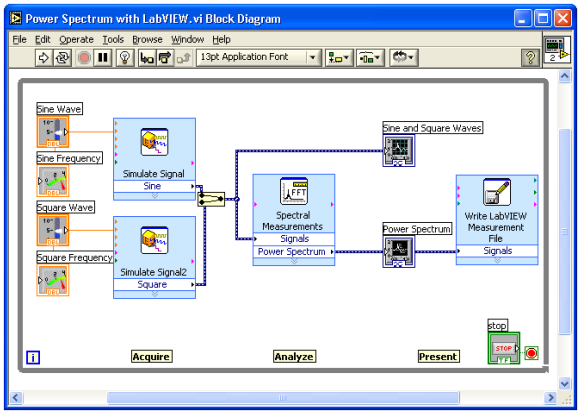
Todo controle ou indicador no painel frontal tem um terminal correspondente no diagrama de blocos. Quando um VI é executado, os valores dos controles fluem pelo diagrama de blocos, onde eles são usados em funções e os resultados são passados para outras funções ou indicadores através de fios.

Figura 3 – Painel frontal.



Fonte: Autor, 2015.

Figura 4 – Diagrama de blocos.



Fonte: Autor, 2015.

O painel frontal do LabVIEW é um meio confortável para construir programas com uma boa interface gráfica. O programador não necessita de escrever qualquer linha de código. A apresentação gráfica dos processos aumenta a facilidade de leitura e de utilização. Uma grande vantagem em relação às linguagens baseadas em texto é a facilidade com que se cria componentes que se executam paralelamente. Em projetos de grande dimensão é muito importante planejar a sua estrutura desde o início (como acontecem nas outras linguagens de programação).

As desvantagens do LabVIEW em relação à programação por texto são:

- Pequenas mudanças podem obrigar a profundas reestruturações do programa, uma vez que sempre que se insere um novo bloco é necessário voltar a ligar os fios e os símbolos para restabelecer o funcionamento;
- Para evitar confusões de linhas é habitual introduzir mais variáveis do que aquelas que são estritamente necessárias, diminuindo-se assim a velocidade de programação e contrariando-se, de algum modo, o modelo de fluxo de dados.

O LabVIEW segue o modelo de fluxo de dados para executar VIs. Um nó no diagrama de blocos executa quando todas as entradas estão disponíveis. Quando um nó completa a execução, ele fornece dados para os terminais de saída e passa o dado para o próximo nó caminho do fluxo de dados. Visual Basic, C++, JAVA e a maioria das

linguagens de programação baseadas em texto seguem o modelo de controle de fluxo na execução do programa. No controle de fluxo, a ordem sequencial dos elementos do programa determina a ordem de execução do programa.

2.3.1 SubVIs

SubVIs são VIs (formados de um painel frontal e diagrama de blocos) que são usados dentro de um outro VI. As Funções são os blocos que formam todos os VIs e elas não possuem painel frontal ou diagrama de blocos.

Um nó de subVI corresponde a uma chamada de sub-rotina em linguagens de programação baseadas em texto. O diagrama de blocos que contém vários subVIs idênticos chama o mesmo subVI diversas vezes.

Os controles e indicadores do subVI recebem e retornam informação para o diagrama de blocos do VI principal.

Criar um subVI em seu código aumenta a legibilidade e a capacidade de reutilização dos VIs. Pode-se transformar qualquer código LabVIEW, ou parte deste, em um subVI que pode ser usado em outros códigos do LabVIEW.

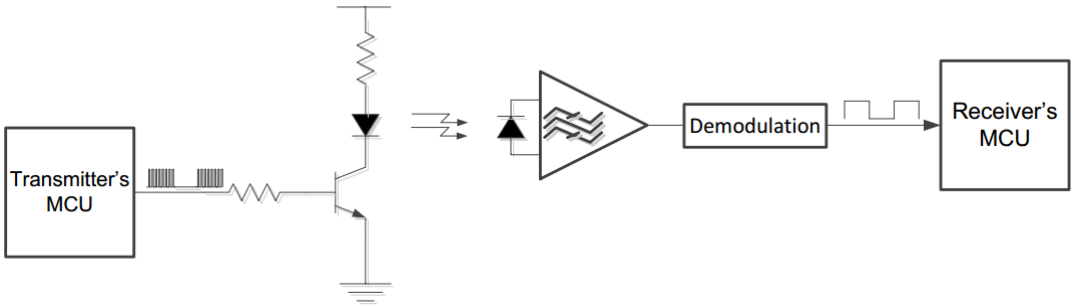
2.4 Comunicação Infravermelho de Controles Remotos

Controles remotos infravermelhos usam luz infravermelha (IR) para transmitir informações. A luz infravermelha emite um sinal modulado a partir de um diodo emissor IR controlado por um microcontrolador transmissor. A modulação pode ajudar o receptor distinguir os sinais desejados de outras fontes de ruído infravermelhos. A modulação é feita a partir do sinal de uma portadora (normalmente uma onda quadrada com a frequência mais elevada) com um sinal de dados que carrega a informação efetiva.

O receptor usa um fotodiodo para converter a luz infravermelha para corrente. Um amplificador operacional é frequentemente usado para converter a corrente em tensão, que passa por um amplificador de ganho e filtro antes da demodulação. O sinal da portadora é retirado durante a demodulação. O sinal demodulado pode ser diretamente ligado a um microcontrolador receptor para decodificação, de acordo com a Figura 5.

Comunicação sem fios por infravermelhos é simples de projetar, barato de fabricar, robusto e eficiente no consumo de energia, e, portanto, é amplamente utilizado hoje em dia em aparelhos eletrônicos.

Figura 5 – Modulação e demodulação IR.



Fonte: TEXAS INSTRUMENTS (2015b).

2.4.1 Modulação e Codificação

Todos os projetos de controle remoto infravermelho modernos usam modulação digital. Duas tecnologias básicas de modulação digital são modulação por mudança em amplitude (*Amplitude Shift Keying* - ASK) e Modulação por mudança em frequência (*Frequency Shift Keying* - FSK). ASK representa lógica 1 e 0 mudando a amplitude da portadora, e FSK representa estes níveis lógicos usando duas frequências portadoras diferentes.

Modulação ASK

ASK é uma das técnicas mais antigas e mais simples. Tornou-se o modo de modulação mais popular por causa de seu bom desempenho (robusto e baixo consumo de energia), simplicidade de projeto e baixo custo. Segue algumas aplicações:

1. Transmissão via fibra ópticas, onde não existe ruído para interferir na recepção do sinal;
2. Transmissão de dados pôr infravermelho, como os usados em algumas calculadoras;
3. Controle remoto pôr meio de raios infravermelhos, como os usados em aparelhos de televisão;
4. Controle remoto pôr meio de radiofrequência, como os usados para ligar e desligar alarmes de carros, residências ou abrir portões.

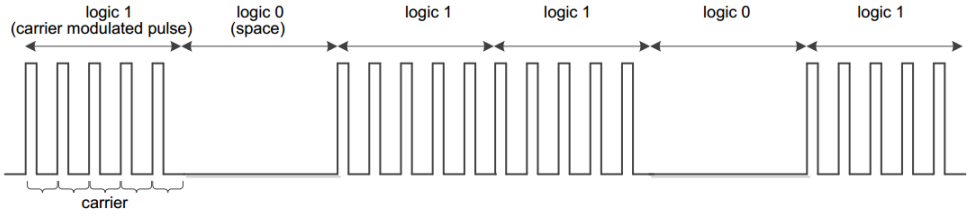
No transmissor, os dados são modulados por uma portadora com frequência na faixa de 30 a 60 kHz. Um espaço é inserido quando nenhum sinal é transmitido.

O receptor está sintonizado na mesma frequência como portadora do transmissor, e todos os outros ruídos são bloqueados pelo filtro de banda do receptor. A frequência da portadora típica é de 30, 33, 36, 38, 40 ou 56 kHz.

Alguns dos principais métodos de codificação em ASK sistema de modulação são descritos a seguir.

Codificação por posição do pulso - É a modulação ASK básica. A largura de cada bit é constante. O pulso portadora modulada significa 1, e o espaço representa 0, como na Figura 6.

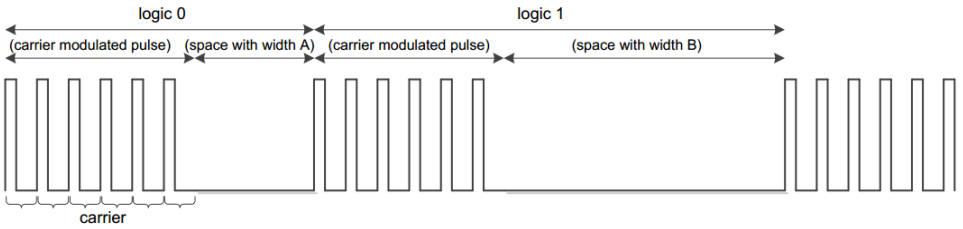
Figura 6 – Codificação por posição do pulso.



Fonte: TEXAS INSTRUMENTS (2015b).

Codificação por Distância de Pulso - Cada bit é composto de um pulso da portadora modulada e um espaço. A largura do espaço distingue 1 e 0, e a largura dos pulsos são constante (Figura 7).

Figura 7 – Codificação por distância de pulso.

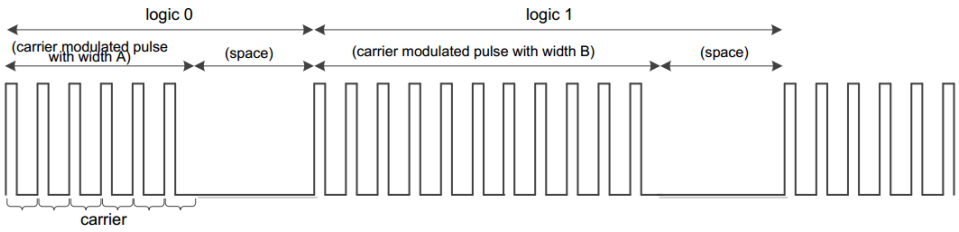


Fonte: TEXAS INSTRUMENTS (2015b).

Codificação por Largura de Pulso - Cada bit é composto de um pulso da portadora modulada e um espaço. A largura de pulso modulada distingue 1 e a lógica 0, e o espaço é constante (Figura 8).

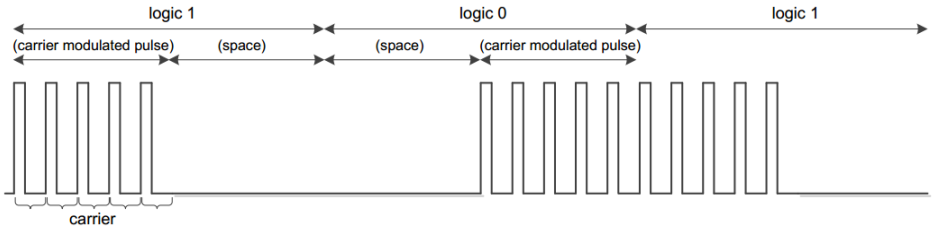
Codificação Manchester - Codificação Manchester é também conhecido como codificação bifásica. Cada bit é composto de um pulso da portadora modulada e um do espaço. A polaridade da transição entre o pulso modulado e o espaço define o nível lógico. Por exemplo, ”do pulso modulado para o espaço” denota 1, e ”do espaço para o pulso modulado” denota 0 (Figura 9).

Figura 8 – Codificação por largura de pulso.



Fonte: TEXAS INSTRUMENTS (2015b).

Figura 9 – Codificação Manchester.

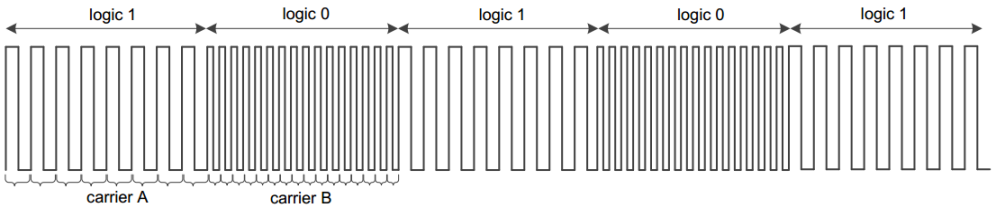


Fonte: TEXAS INSTRUMENTS (2015b).

Modulação FSK

FSK utiliza duas frequências portadoras diferentes para 1 e 0, e não existe espaço entre os pulsos (Figura 10). Esta é uma solução menos atrativa, porque a adoção de duas frequências aumenta a complexidade e custo de demodulação. Assim, FSK não é amplamente usado.

Figura 10 – Modulação FSK.



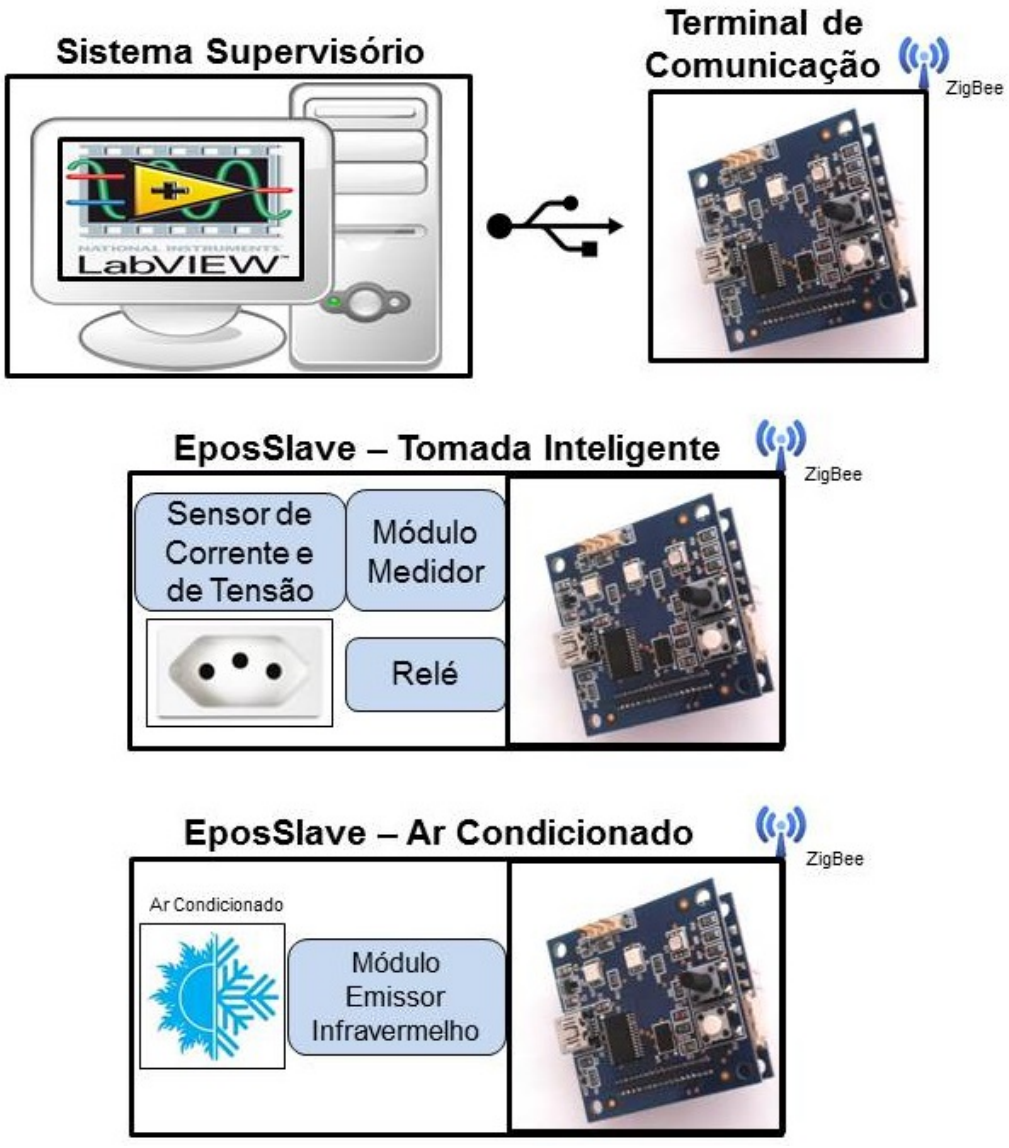
Fonte: TEXAS INSTRUMENTS (2015b).

Neste trabalho foi utilizado os conceitos de comunicação sem fio por infravermelho no desenvolvimento do controle do ar condicionado com o EPOSMote, onde utilizou-se uma modulação ASK com codificação por distância de pulso.

3 Projeto, Especificação e Desenvolvimento

O diagrama de blocos da Figura 11 foi elaborado para representar a interação entre estes elementos do projeto, demostrando suas formas de interação sobre o sistema.

Figura 11 – Diagrama geral do projeto.



Fonte: Autor, 2015.

O projeto constitui de um sistema supervisório que será executado em um computador, atrelado a ele estará conectado um terminal de comunicação, responsável por uma comunicação sem fio com os dispositivos tomada inteligente e controle do ar condicionado.

Portando, o desenvolvimento deste projeto pode ser dividido nos seguintes componentes principais:

- EposSlave – Tomada Inteligente;
- EposSlave – Controle do Ar Condicionado;
- EposMaster – Terminal de Comunicação;
- Sistema Supervisório.

O protótipo da tomada inteligente deve ser capaz de fazer a medição do consumo e o acionamento da carga conectada, além do envio de informações e recebimento de comandos do sistema supervisório. Também, tem como requisito que todo o circuito da tomada deve acomodar-se em uma caixa de tomada padrão ABNT 4x2. A tomada inteligente se aplicará para uso em aparelho elétricos monofásicos, tensão $110/220V$, e com potência máxima delimitada pela corrente máxima no sistema de medição da corrente.

O protótipo do controle do ar condicionado deve ser capaz de controlar as principais funções de um controle remoto: liga/desliga, modo de operação, aumenta/diminui a temperatura, nível da ventilação e turbo.

O terminal de comunicação deve permitir uma comunicação nas duas direções entre o sistema supervisório e os EposSlaves.

O sistema supervisório contém os requisitos de oferecer uma interface de usuário simples e deve conseguir controlar todas as funcionalidades permitidas que serão desenvolvidas

Cada componente será apresentado ao longo desta monografia, a tomada inteligente é apresentado na seção 3.1. O desenvolvimento do controle do ar condicionado é detalhado na seção 3.2. O terminal de comunicação é descrito na seção 3.3 e o sistema supervisório é descrito na seção 3.4. Os testes realizados com os protótipos são apresentados na seção 4.

3.1 Tomada Inteligente

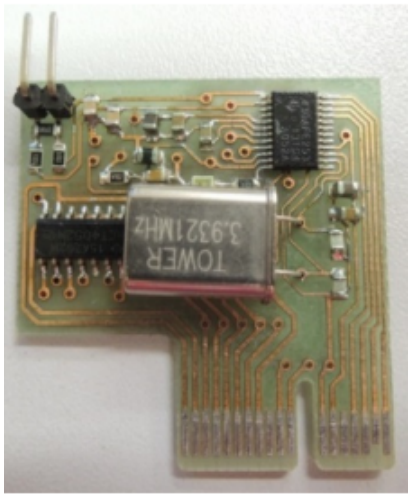
O desenvolvimento da tomada inteligente se divide em um módulo medidor responsável em medir alguns parâmetros elétricos, e um EPOSMotte II com uma aplicação do EPOS responsável pelo acionamento de um relé e a interface do módulo medidor com a comunicação sem-fio. Ao final, é construído um protótipo da tomada inteligente.

3.1.1 Módulo Medidor

O módulo medidor utilizado neste trabalho foi desenvolvido por Bacurau (2014), em sua dissertação de mestrado em Engenharia Elétrica na Universidade Estadual de Campinas, mostrado na Figura 12. Porém, algumas alterações foram realizadas neste módulo medidor para ser compatível com outros subsistemas e atender os objetivos deste trabalho da maneira mais simples.

O objetivo da dissertação de Bacurau (2014) foi o desenvolvimento de um módulo medidor de energia residencial capaz de medir os parâmetros elétricos necessários para a discriminação de consumo de energia por aparelho. Em sua pesquisa, os resultados dos experimentos realizados indicaram a potência ativa, a potência reativa, o fator de potência, a tensão e corrente eficazes e as cinco primeiras harmônicas ímpares do sinal de corrente como os parâmetros mais adequados para uso em sistemas não intrusivos para identificação de consumo por aparelho.

Figura 12 – Módulo medidor.



Fonte: Bacurau (2014).

Segundo Bacurau (2014), os valores de tensão, corrente, fator de potência, potência ativa e potência aparente apresentaram erros inferiores a 0,3% (para cargas resistivas). O erro nas componentes harmônicas do sinal de corrente foi de 0,03% para a componente

fundamental, 0,58% para a 3ª harmônica, 3,69% para a 5ª harmônica, 6,63% para a 7ª harmônica e 11,81% para a 9ª harmônica.

O medidor era capaz de medir seis diferentes combinações de tensão com a utilização de multiplexadores: Neutro – Fase A, Neutro – Fase B, Neutro – Fase C, Fase A – Fase B, Fase B – Fase C e Fase A – Fase C. Os multiplexadores eram controlados pelo microcontrolador através de quatro vias de controle. O microcontrolador do módulo medidor, por sua vez, recebia através de comando SPI a informação de quais fases deveriam ser selecionadas. Porém, essa possibilidade foi eliminada, pois neste projeto cada tomada tem um módulo medidor e a medição da tensão é fixa entre o neutro e a fase.

As alterações foram feitas para que seja calculado o consumo acumulado no módulo medidor, a mudança da comunicação SPI para uma UART e incremento de mais um comando para zerar o consumo acumulado. O módulo medidor é usado para medir a corrente e tensão da carga, calcular a tensão e corrente eficazes, a potência ativa, o fator de potência e o consumo acumulado. Os parâmetros elétricos calculados são enviados para o EPOSMote através da interface UART.

3.1.1.1 Hardware do Módulo Medidor

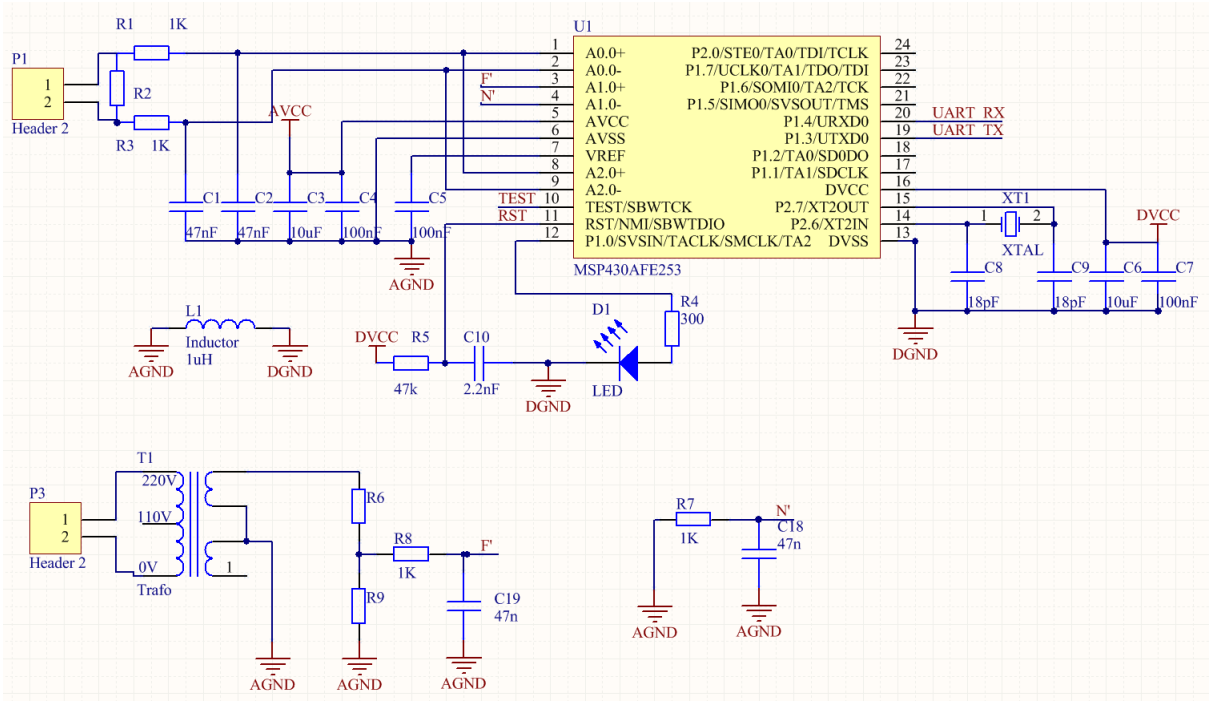
O Módulo Medidor é composto basicamente por circuitos de condicionamento dos sinais de tensão e corrente e um microcontrolador, responsável pelo cálculo e transmissão dos parâmetros elétricos para o EPOSMote.

O microcontrolador da Texas Instruments MSP430AFE253 foi usado para implementar o Módulo Medidor. Este microcontrolador de baixo custo e consumo foi projetado especialmente para a construção de medidores de energia monofásicos. O MSP430AFE253 é equipado com uma unidade de processamento RISC de 16 bits capaz de operar até $12MHz$, $16kB$ de memória de flash, 512 Bytes de memória RAM, interfaces de comunicação UART e SPI, multiplicador em hardware de 16 bits, timer de 16 bits, 11 pinos de entrada e saída digital e três conversores A/D sigma-delta independentes de 24 bits. Estes conversores possuem entradas diferenciais sincronizáveis, permitindo amostragens simultâneas. Além disso, possuem referência de tensão interna de $1,2V$ com variação de $50ppm/^oC$, amplificador de ganho integrado, programável, de até 32 vezes e podem operar com uma taxa de sobreamostragem de até 1024 vezes (TEXAS INSTRUMENTS, 2011). Todas essas características tornam esse controlador adequado para a implementação do Módulo Medidor.

No projeto de Bacurau (2014), dois conversores A/D independentes foram usados para amostragem simultânea dos sinais de tensão e corrente. A amostragem simultânea da tensão e corrente é necessária para o cálculo da potência ativa e fator de potência. Por isso, se faz necessário o uso do MSP430AFE253, que possui três conversores A/D sigma-delta independentes, assim, permitindo amostragens simultâneas.

Na Figura 13 é apresentado o diagrama esquemático do Módulo Medidor desenvolvido por Bacurau (2014). Este circuito é composto por um microcontrolador MSP430AFE253, um cristal oscilador e componentes passivos (resistores, capacitores, indutores e diodos).

Figura 13 – Diagrama esquemático do módulo medidor.



Adaptado de: Bacurau (2014).

3.1.1.2 Sensor de Tensão e Corrente

Os sinais de tensão são isolados, condicionados e repassados para o Módulo Medidor. Um transformador com entrada nominal de $110/220V_{rms}$ e saída $9V_{rms}$ é usado para prover isolação elétrica e proteção do circuito de medição. A amplitude do sinal de tensão de saída do transformador é atenuada por um divisor resistivo composto por resistores de $27k\Omega$ ($\pm 5\%$) e $1k\Omega$ ($\pm 1\%$), resultando em uma tensão nominal de saída de $321mV_{rms}$ ($453mV_{pico}$). Usando um conversor A/D de 16 bits efetivos, com fundo de escala em $\pm 600mV$, será possível medir tensões de até $259V_{rms}$ com resolução de $7,9mV_{rms}$.

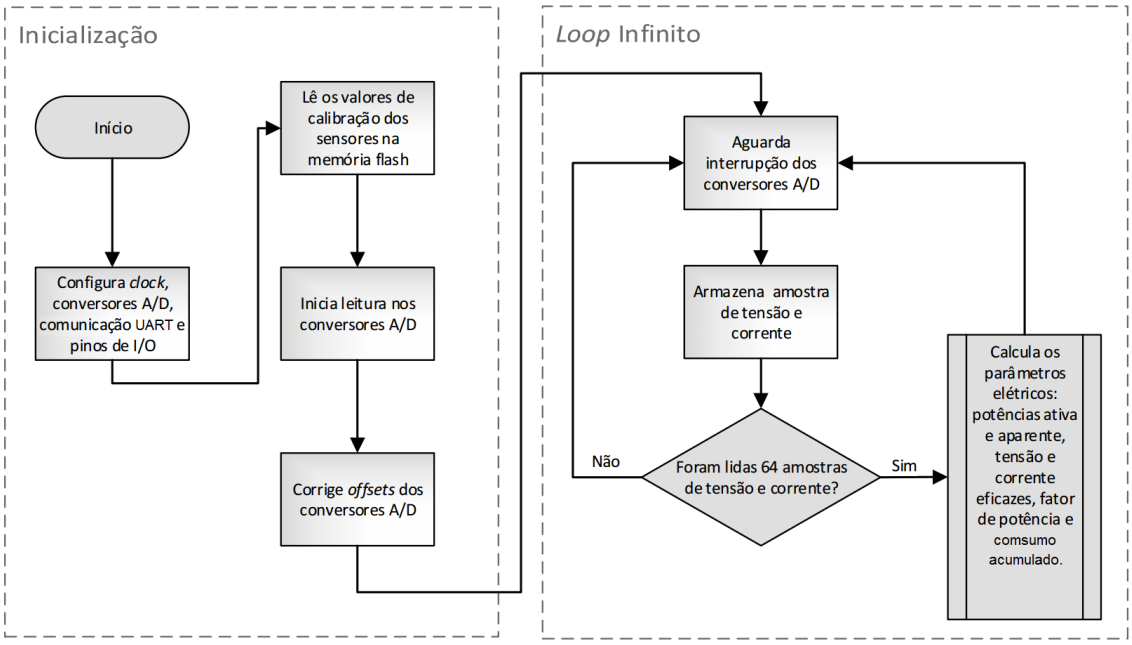
O sinal de corrente é obtido através de um transformador de corrente (TC), modelo JSCT-6, de relação entrada/saída igual a $5A/5mA$, precisão de 1% e corrente máxima de $20A_{rms}$. A corrente induzida no TC será medida através de um resistor de borda de $5,6\Omega$ ligado nos terminais de saída do sensor. Usando um conversor A/D de 16 bits efetivos será possível medir correntes de até $20A_{rms}$ com resolução de $0,61mA_{rms}$.

3.1.1.3 Programa do Módulo Medidor

O programa do Módulo Medidor foi desenvolvido de forma modular, com o objetivo de facilitar futuras atualizações e tornar a leitura e depuração do código mais simples (BACURAU, 2014). O código foi escrito em linguagem C ANSI C99 (ISO/IEC 9899:1999) usando o compilador IAR C/C++ Compiler for MSP430 v6.30.1. Para fazer as alterações no código foi utilizado o mesmo compilador, mas com uma licença de teste de 30 dias.

Na Figura 14 é apresentado um fluxograma que representa o programa do módulo medidor. Este programa será dividido em duas partes: Inicialização e *Loop* Infinito. Os procedimentos presentes na etapa de Inicialização são executados uma única vez, logo após o sistema ser ligado, e só são executados novamente se o microcontrolador for reiniciado. Os procedimentos descritos em *Loop* Infinito são executados periodicamente por tempo indefinido (até que o microcontrolador seja desligado). É nessa etapa que são executados os cálculos dos parâmetros elétricos.

Figura 14 – Fluxograma do programa do Módulo Medidor.



Adaptado de: Bacurau (2014).

Na inicialização do microcontrolador o *clock*, os conversores A/D, a comunicação UART e os pinos de I/O digital são configurados. O *clock* principal do microcontrolador (Master Clock), usado pela unidade de processamento, é configurado para usar o sinal gerado internamente no microcontrolador pelo DCO (do inglês, Oscilador Controlado Digitalmente). O DCO é configurado para operar em $16MHz$. O sinal de clock usado pelos conversores A/D é gerado externamente por um cristal oscilador de $3,93216MHz$.

Após o hardware ser configurado, os valores dos ganhos dos sensores de tensão e corrente são lidos da memória *flash* do microcontrolador. As variáveis de calibração dos

sensores são armazenadas em memória flash para que seja possível a recuperação dessas informações mesmo depois do sistema ser desligado. Desta forma, a calibração dos sensores é necessária uma única vez, antes de serem instalados, e só precisarão ser recalibrados caso sejam substituídos.

Após a leitura dos ganhos dos conversores A/D, os mesmos são iniciados. Os conversores são então calibrados para compensação de *offset*. A calibração é realizada ao executar os seguintes procedimentos:

- As entradas diferenciais de cada um dos dois conversores A/D usados são curto circuitadas internamente, resultando em diferença de potencial de $0V$ nas suas entradas;
- São executadas 64 leituras em cada um dos conversores A/D;
- São calculadas as médias das amostras de cada conversor. Esse valor corresponde ao *offset* de cada um dos conversores.
- As entradas dos conversores A/D são reconectadas aos sensores de tensão e corrente.

As amostras obtidas após a calibração têm seus valores corrigidos através da subtração dos valores de *offset*.

Após a etapa de inicialização, o sistema entra em um *loop* infinito onde são lidas, armazenadas e processadas as amostras de tensão e corrente. As grandezas elétricas são calculadas para cada ciclo de rede. Como os conversores A/D operam com taxa de amostragem de $3,84kS/s$, em redes elétricas de $60Hz$, são necessárias 64 amostras para que se tenha um ciclo completo.

O cálculo das grandezas elétricas só é iniciado após 64 amostras de tensão e corrente serem lidas e armazenadas. Os seguintes parâmetros elétricos são calculados: potência ativa, potência reativa, tensão eficaz, corrente eficaz, fator de potência e consumo acumulado.

O valor eficaz da tensão elétrica é calculado usando a Equação 3.1, em que V_{RMS} é o valor eficaz da tensão, G_v é o ganho das amostras de tensão, n é o índice da amostra, $v[n]$ é a n-ésima amostra do sinal de tensão e N o número total de amostras. O valor eficaz da corrente elétrica é calculada usando uma equação análoga, com ganho das amostras de corrente G_i.

$$V_{RMS} = Gv\sqrt{\sum_{i=1}^{N} \frac{v[n]^2}{N}} \tag{3.1}$$

A potência ativa é calculada usando a Equação 3.2, onde P é o valor da potência ativa, G_i o ganho das amostras de corrente, G_v o ganho das amostras de tensão, n é o índice da amostra, $i[n]$ a n-ésima amostra do sinal de corrente, $v[n]$ a n-ésima amostra do

sinal de tensão e N o número total de amostras.

$$P = G_i.G_v \sum_{i=1}^{N} \frac{i[n].v[n]}{N} \tag{3.2}$$

A potência aparente é calculada a partir dos valores eficazes da tensão e corrente, conforme a Equação 3.3.

$$S = V_{RMS}.I_{RMS} \tag{3.3}$$

O fator de potência corresponde ao quociente da potência ativa pela potência aparente, conforme a Equação 3.4.

$$FP = \frac{P}{S} \tag{3.4}$$

O consumo acumulado é calculado a partir do valor da potência ativa (P) e o tempo de amostragem (Δt_N), conforme a Equação 3.5, onde C_i é o consumo acumulado, C_{i-1} é o consumo do ciclo anterior de cálculos dos parâmetros.

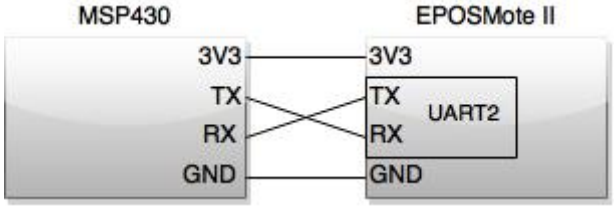
$$C_i = C_{i-1} + P.\Delta t_N \tag{3.5}$$

Como o valor do consumo acumulado em cada ciclo de cálculo é pequeno, portanto, será calculado o consumo em acumulado em Ws e quando esse valor chegar $380Ws$, será convertido em Wh e incrementado na variável de comunicação do consumo acumulado. No Apêndice A é mostrado um fluxograma de cálculo dos parâmetros elétricos no Módulo Medidor.

3.1.1.4 Comunicação (MPS430 - EPOSMote)

O envio dos parâmetros elétricos gerados no módulo medidor para a EPOSMote, bem como a configuração e calibração do módulo medidor, é feita através da troca de mensagens enviadas através da UART. Os microcontroladores são conectados de acordo com a Figura 15.

Figura 15 – Conexões entre o MSP430 e o EPOSMote II.



Fonte: Autor (2015).

Há três tipos básicos de mensagens: *get*, *set* e *parâmetros*. As mensagens *get* e *set* são enviadas exclusivamente do EPOSMote para o Módulo Medidor e iniciam-se com o

caractere sustenido ("#") e terminam com o caractere ponto-e-vírgula (";"). As mensagens parâmetros são enviadas exclusivamente pelos Módulos Medidores para o EPOSMote.

A mensagem *get*, faz a requisição dos parâmetros elétricos gerados no Módulo Medidor pelo EPOSMote. Esta mensagem consiste da seguinte cadeia de três caracteres: "#g;".

As mensagens do tipo *set* são enviadas pelo EPOSMote para configuração do Módulo Medidor. Com este tipo de mensagem é possível fazer as seguintes configurações:

- Definir o valor atual da tensão, permitindo a calibração do ganho do sensor de tensão;
- Definir o valor atual de corrente, permite a calibração do ganho do sensor de corrente;
- Definir quantos ciclos de rede são usados para o cálculo da média das grandezas elétricas.
- Zerar o consumo.

A mensagem do tipo *parâmetros* contém os parâmetros elétricos calculados no módulo medidor, é único tipo de mensagem enviada do Módulo Medidor para o EPOSMote. Essa mensagem possui a seguinte estrutura:

$$<\#><V_{RMS}><I_{RMS}><P><FP><Consumo><;>$$

em que $<V_{RMS}>$ é o valor eficaz da tensão; $<I_{RMS}>$ o valor eficaz da corrente; $<P>$ potência ativa; $<FP>$ fator de potência; $<Consumo>$ consumo acumulado.

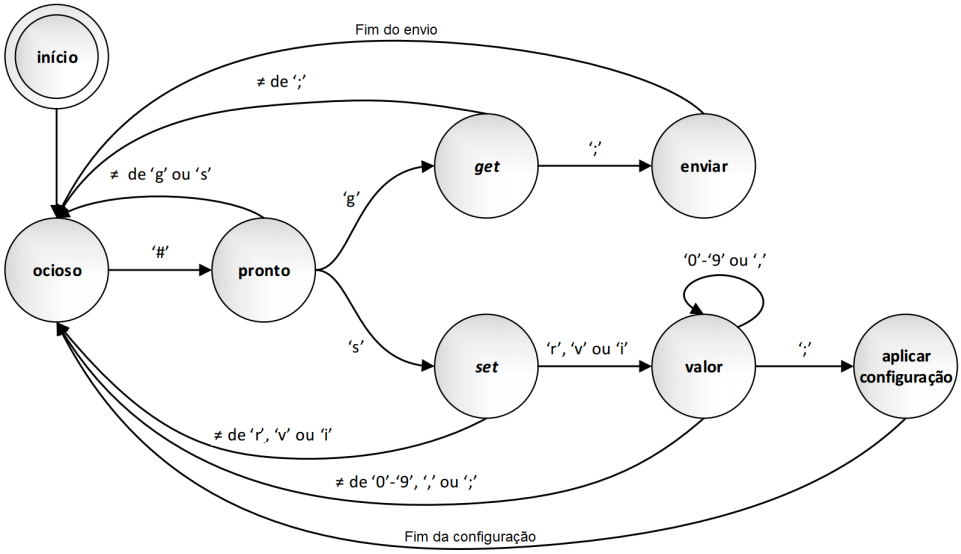
O algoritmo usado para o processamento das mensagens da UART está implementado em forma de máquina de estados. Uma variável global registra o estado atual da máquina, que pode mudar de estado com a chegada de um novo caractere na UART. Na Figura 16 é apresentada uma representação desta máquina de estados.

3.1.2 Programa do EPOS

No programa da tomada inteligente foram criados uma função principal e um alarme. O escalonador é do tipo prioridade, onde as atividades de maior prioridade tem preferência para serem executadas pelo processador.

Na Figura 17 é apresentado um fluxograma de execução da função principal do programa. Primeiramente, o EPOSMote fica tentando receber instruções do terminal de comunicação via *ZigBee*. Quando receber alguma mensagem, verifica-se quem deve obedecer as instruções através de uma comparação do ID do recebedor que está contido na mensagem e um ID que é pré-definido para cada tomada. Posteriormente, verifica qual é o comando para executar uma instrução respectiva para comando. Ao final, é verificado se existe algum dado para enviar através de uma *flag* de controle, caso verdadeiro, é enviado

Figura 16 – Máquina de estados que representa a comunicação UART no Módulo Medidor.



Adaptado de: Bacurau (2014).

para o Terminal de Comunicação. Todas essas atividades são executadas de forma cíclica em um *while* infinito.

No alarme é requisitado os parâmetros elétricos do Módulo Medidor com o objetivo de atualizar o valor no EPOSMote. A requisição é feita através de uma mensagem do tipo get, já explicada na seção 3.1.1.4. Esse alarme está configurado para ocorrer a cada 7 segundos.

3.1.3 Integração dos Subsistemas da Tomada Inteligente

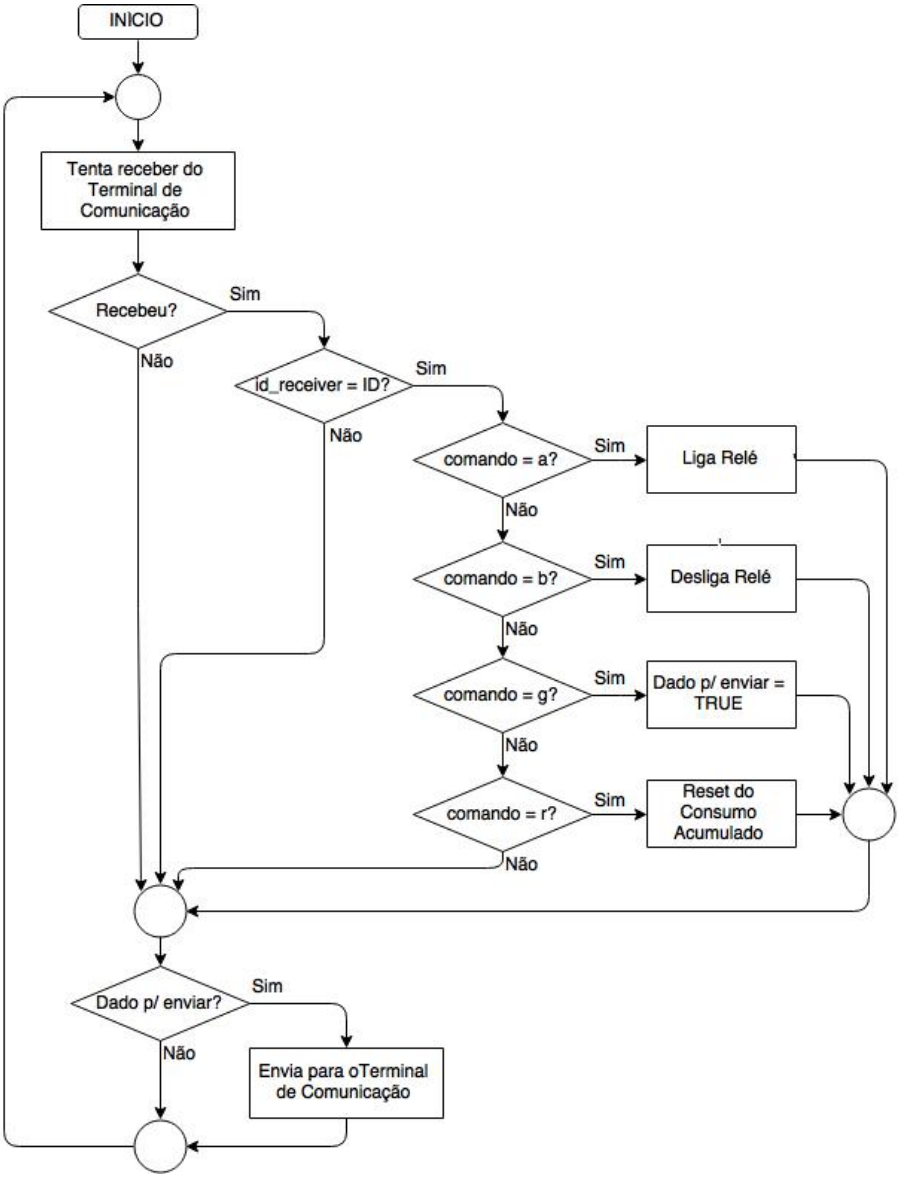
Após todo o estudo, análise, desenvolvimento dos subsistemas e integração dos mesmos, foi construído um protótipo da Tomada Inteligente, onde todos os circuitos devem estar dentro da uma caixa padrão de tomada ABNT 4x2.

Componentes presentes na montagem do protótipo:

- EPOSMote II;
- Retificador $5V$ e $1,5A$;
- Módulo Medidor;
- Shield relé $5V$;
- Módulo Tomada 2P+Terra (NBR) $10A$;
- Transformador de tensão $110/220V_{rms}$ para $9V_{rms}$;
- Transformador de corrente (JSCT-6), relação $5A/5mA$, máx. $20A_{rms}$;
- Resistores: $27k\Omega$ ($\pm 5\%$) e $1k\Omega$ ($\pm 1\%$).

A Figura 18 demostra como foram feitas as conexões entre os circuitos no momento

Figura 17 – Fluxograma da função principal do programa do EPOS para a tomada inteligente.

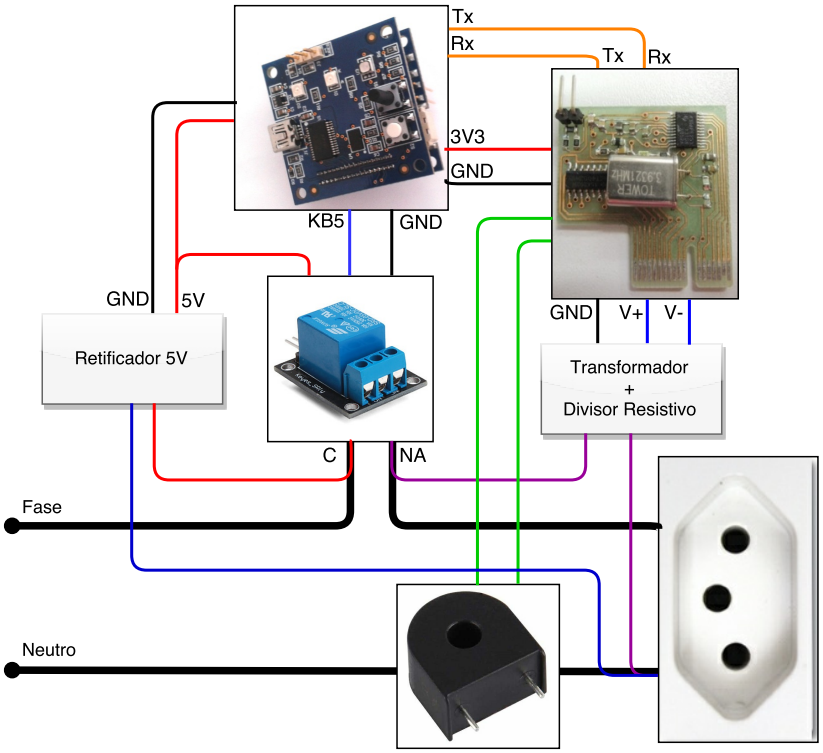


Fonte: Autor (2015).

da montagem do protótipo.

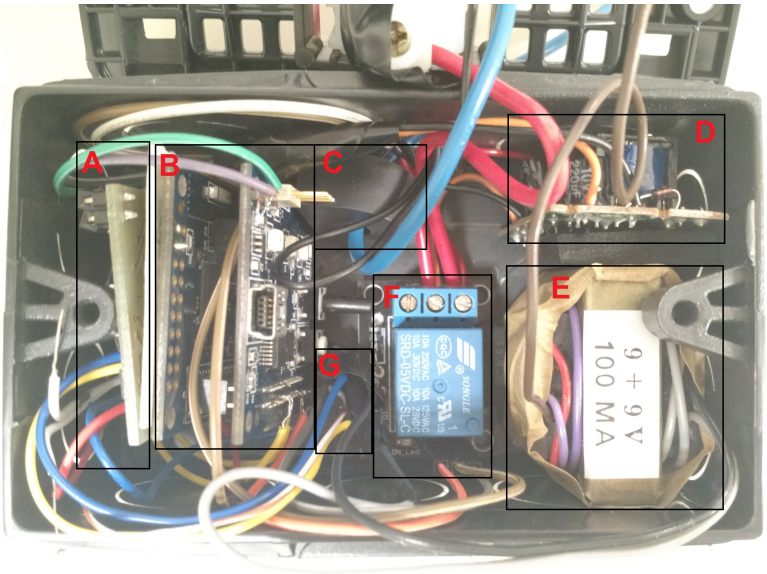
Na Figura 19 podemos ver como os circuitos foram alocados na caixa para que fosse possível o fechamento da mesma.

Figura 18 – Esquemático das conexões elétricas entre os componentes da tomada inteligente.



Fonte: Autor (2015).

Figura 19 – Disposição dos componentes da tomada inteligente na caixa padrão ABNT 4x2.



A - Módulo medidor;
B - EPOSMote II;
C - Sensor de Corrente;
D - Retificador 5V;
E - Transformador do sensor de tensão;
F - Relé;
G - Divisor resistivo do sensor de tensão.

Fonte: Autor (2015).

3.2 Controle do Ar Condicionado

Com o objetivo de integrar o uma solução inteligente para o controle do ar condicionado, optou-se por emular o sinal infravermelho do controle remoto ao invés de desmontar e interferir no circuito interno do ar condicionado. Assim, o sistema é simplificado em um EPOSMote com um módulo emissor infravermelho que emite sinais de acordo com as instruções recebidas do sistema supervisório. Deste modo, o mesmo sistema pode ser usado para controlar outros aparelhos eletrônicos que possuem controles remotos por infravermelho e que estejam ao alcance do sinal.

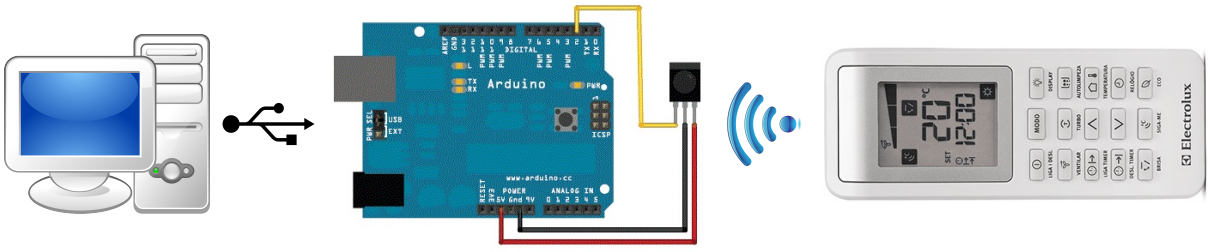
Com base nos conhecimento de comunicação infravermelho de controles remotos, apresentado na seção 2.4, será controlado um ar condicionado Electrolux BI22F, presente no LISHA de Joinville. O desenvolvimento foi dividido na seguintes etapas:

- Decodificação do controle remoto;
- Criação de uma biblioteca do controle remoto para o EPOS;
- Desenvolvimento do código da aplicação do EPOS.

3.2.1 Decodificação

Para a descoberta do protocolo de comunicação infravermelho do controle remoto com o ar condicionado Electrolux iniciou-se um trabalho de engenharia reversa com os sinais de comunicação do controle remotos, com base nos conceitos apresentado na seção 2.4, sobre comunicação infravermelha de controles remotos. O primeiro passo foi a determinação do tempo do pulso modulado e o tempo do espaço dos comandos do controle remoto do ar condicionado. Optou-se em usar a plataforma de prototipagem eletrônica Arduino, pois contém uma gama de usuários que compartilham seus desenvolvimentos em fóruns da internet. Portanto, montou-se o esquema apresentado na Figura 20, com um programa adaptado de Ada (2012), que resultasse no tempo de pulso modulado e o tempo do espaço.

Figura 20 – Esquema para decodificação do controle do ar condicionado.



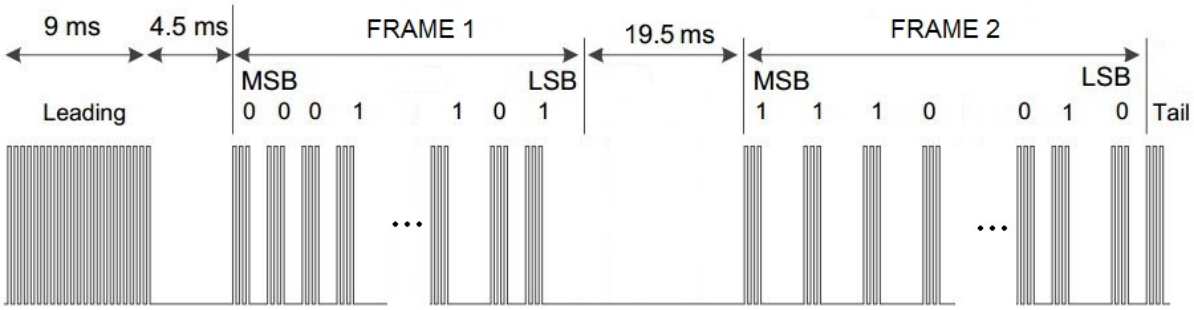
Fonte: Autor (2015).

Com os resultados dos tempos determinou-se que a codificação utilizada na comunicação é por distância de pulso, onde cada bit é um pulso modulado pela portadora e um espaço, em que a largura do pulso modulado é constante, enquanto que a largura do espaço varia para nível lógico 0 e nível lógico 1.

Nível lógico 1 é definido como um período de portadora modulada de $560us$ seguido por um período de espaço de $1690us$. Nível lógico 0 é definido como um período de portadora modulada de $560us$ seguido por um período de espaço $560us$. o protocolo começa com um código *Leading* de $9ms$ período modulados seguido por um período de espaço de $4,5ms$.

Com base nos mesmos resultados, determinou-se que após o código *Leading* é enviado um primeiro *frame* de 36 bits e após um intervalo de aproximadamente $19,5ms$ ocorre o envio do segundo *frame* de 32 bits. Ao final, o protocolo termina com um pulso de cauda (*Tail*) para indicar o fim do pacote. Está cauda é um pulso modulado de $560us$.

Figura 21 – Protocolo da comunicação com o ar condicionado.



Fonte: Autor (2015).

Posteriormente, ouve alteração no código do Arduino de Ada (2012), de tal forma que o conjunto dos valores do pulso modulado e do espaço já fossem traduzidos em bits 1 e 0, o código final do Arduino está disponível no Apêndice D. Com isso, se pode verificar o relacionamento dos sinais do controle remoto com os bits dos *frames* do protocolo de comunicação, e após várias amostragens e análise pode-se montar um quadro do protocolo dos comandos do ar condicionado para os principais botões do controle remoto, mostrado na Figura 22. O cálculo do valor do CRC no segundo *frame* foi obtido e validado com base na tentativa e erro.

Por exemplo, se fosse enviar um comando para o ar condicionado com as seguintes informações: estado ligado (bit *status* = 01, bit W das funções auxiliares = 1); modo resfriar (bits *modo* = 11); nível de ventilação 2 (bits *ventilação* = 01); função eco desligada (bits *eco* = 00); temperatura a 23^oC (bits *temperatura* = 1110); função turbo ligada (bit Y das funções auxiliares = 1); display ligado (bit X das funções auxiliares = 1); função auto limpeza desligada (bit Z das funções auxiliares = 1); função brisa ligada (bits *brisa*

Figura 22 – Quadro dos *frames* do protocolo do controle do ar condicionado Electrolux.

1º Frame							
XX	**0X**	**XX**	**0X**	**XXXX**	**0000 0000**	**Y X W Z**	**0000 1010 0101**
modo	status	velocidade	eco	temperatura	clock	Funções Auxiliares	constante
00 - Automatico 01 - Desumificar 10 - Ventilar 11 - Resfriar	0 - OFF 1 - ON	00 - Auto 10 - nível 1 01 - nível 2 11 - nível 3	0 - OFF 1 - ON	0000 - 16 ºC 1000 - 17 ºC 0100 - 18 ºC 1100 - 19 ºC 0010 - 20 ºC 1010 - 21 ºC 0110 - 22 ºC 1110 - 23 ºC 0001 - 24 ºC 1001 - 25 ºC 0101 - 26 ºC 1101 - 27 ºC 0011 - 28 ºC 1011 - 29 ºC 0111 - 30 ºC	Usado para funções do relógio	Y - Turbo X - Display W - status Z - Auto Limpeza	

2º Frame				
X000	**0000**	**1X00**	**0000 0000 0000 0000**	**XXXX**
Brisa	const.	Compressor	constante	CRC
0 - OFF 1 - ON		0 - OFF 1 - ON		CRC_2 = rev(rev(Temperatura)$_2$ + rev(modo status)$_2$+ 1010_2)

Fonte: Autor (2015).

= 1000). O bit de estado do compressor é definido pelo modo de operação e pelo estado ligado ou desligado do ar condicionado, o bit será 0 (OFF) no modo ventilação ou no estado desligado, em outros casos, o bit será 1 (ON). A Tabela 1 mostra como seria a composição dos bits para o primeiro *frame* do exemplo.

Tabela 1 – Exemplo da composição do Frame 1.

modo	status	velocidade	eco	temperatura		Funções auxiliares.	
11	01	01	00	1110	00000000	1110	000010100101

Fonte: Autor (2015).

Os 4 bits do CRC do segundo *frame*, pode ser calculado após a definição do primeiro *frame*. Seguindo o exemplo anterior, com os bits do primeiro *frame* apresentado na Tabela 1, o cálculo do CRC é demostrado realizado através da Equação 3.6, o único detalhe é a realização do truncamento de 4 bits do passo 3.9 para o 3.10.

$$CRC = reverso\,(reverso(Temperatura) + reverso([modo][status]) + 1010) \quad (3.6)$$
$$CRC = reverso\,(reverso(1110) + reverso(1101) + 1010) \quad (3.7)$$
$$CRC = reverso(0111 + 1011 + 1010) \quad (3.8)$$
$$CRC = reverso(11100) \quad (3.9)$$
$$CRC = reverso(1100) \quad (3.10)$$
$$CRC = 0011 \quad (3.11)$$

Com o cálculo do CRC realizado, o segundo *frame* do exemplo é demostrado na Tabela 2.

Tabela 2 – Exemplo da composição do Frame 2.

brisa		compressor		CRC
1000	0000	1100	0000000000000000	0011

Fonte: Autor (2015).

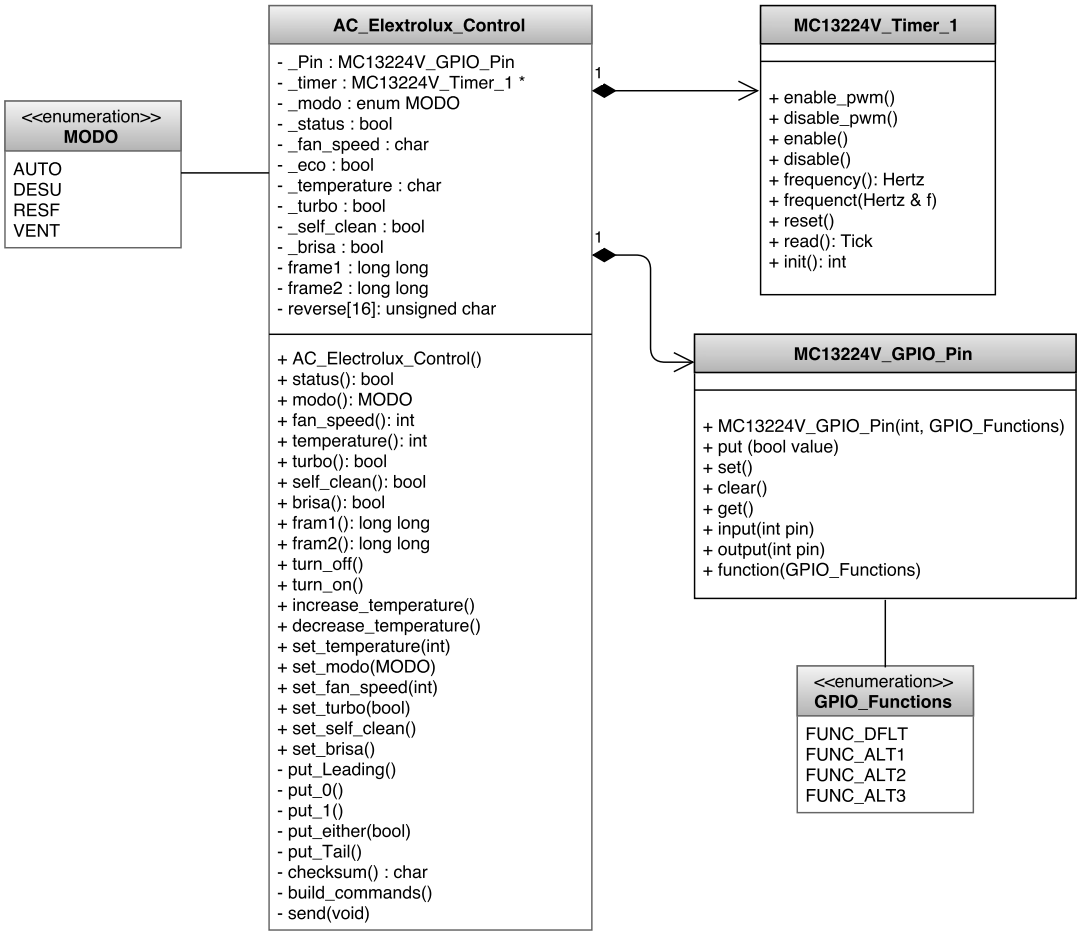
3.2.2 Desenvolvimento com o EPOS

Após a determinação do protocolo de comunicação e do relacionamento dos botões com os bits dos *frames* foi desenvolvido uma classe do controle do ar condicionado Electrolux para o software EPOS e o hardware EPOSMote II, na Figura 23 mostra um representação da classe em diagrama UML.

Para a emissão do sinal infravermelho é utilizado o módulo BOOST-IR Booster-Pack (Figura 24a) da Texas Instruments contectado no EPOSMote. Além dos pinos de alimentação, é conectado o pino *IR-TX* do módulo no pino *Timer1* do EPOSMote. A Figura 24b mostra o circuito emissor de sinal infravermelho do módulo utilizado.

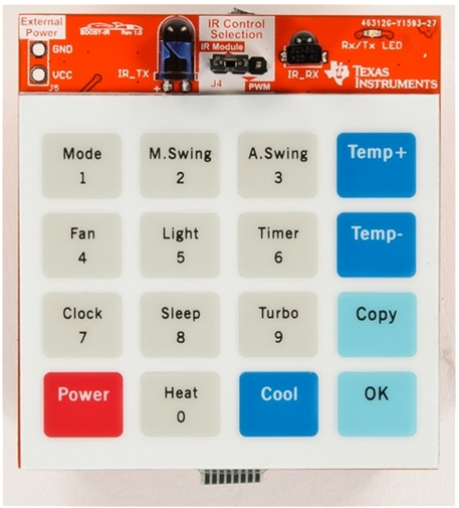
Na Figura 25 é apresentado um fluxograma de execução do programa do controle do ar condicionado no EPOS. Primeiramente, o EPOSMote fica tentando receber instruções do terminal de comunicação via *ZigBee* do Terminal de Comunicação. Quando receber alguma mensagem, verifica se ele é o destinatário da mensagem através de uma comparação do ID do recebedor que está contido na mensagem e um ID que é pré-definido para cada tomada. Posteriormente, verifica qual é o caractere recebido na mensagem para executar uma instrução respectiva para comando, de acordo com a Tabela 3. Ao final, é verificado se existe algum dado para enviar para o Terminal de Comunicação, através de uma *flag* de controle, caso verdadeiro, é enviado para o Terminal de Comunicação. Todas essas atividades são executadas de forma cíclica em um *while* infinito.

Figura 23 – Representação da classe do controle do ar condicionado Electrolux.

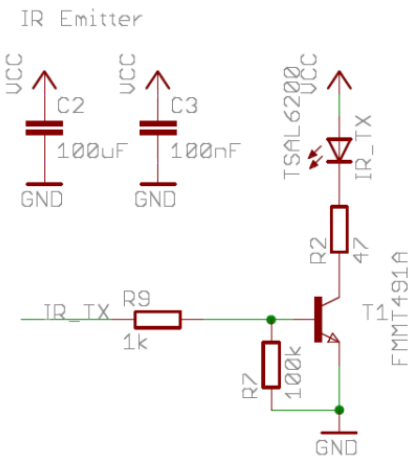


Fonte: Autor (2015).

Figura 24 – (a) Módulo BOOST-IR BoosterPack. (b) Circuito emissor infravermelho.



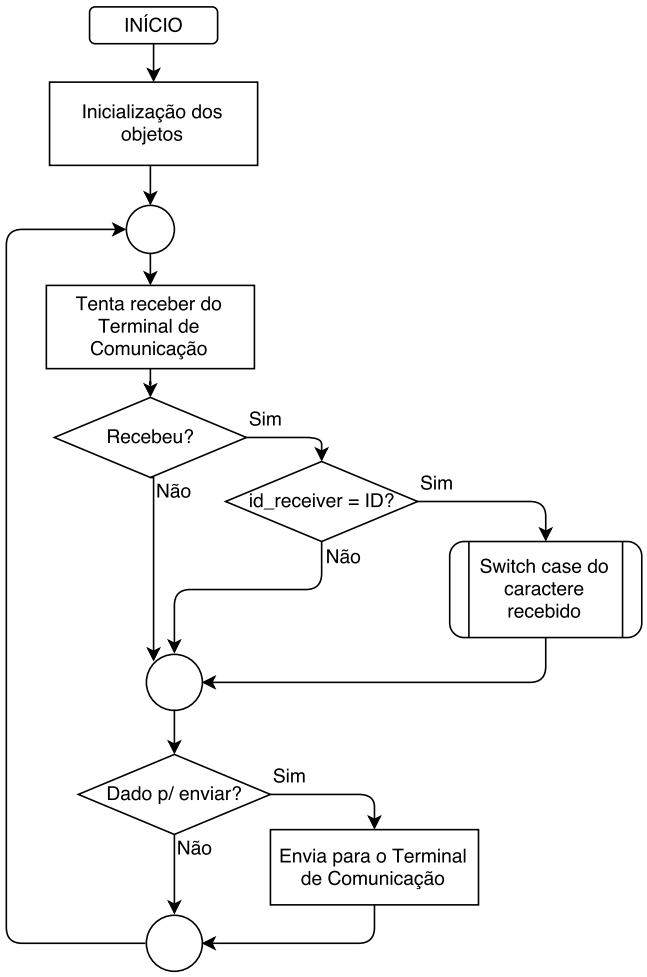
(a)



(b)

Fonte: TEXAS INSTRUMENTS, 2015a.

Figura 25 – Fluxograma do programa de aplicação do EPOS do controle do ar condicionado.



Fonte: Autor (2015).

Tabela 3 – Processamento do comando recebido.

Caractere recebido	**Instrução**
a	Liga
b	Desliga
c	Aumenta a temperatura
d	Diminui a temperatura
e	Alternar a função turbo
f	Configura a velocidade da ventilação
i	Alterna a função de auto limpeza
j	Alterna a função brisa
m	Configura o modo de operação
s	Configura a temperatura

Fonte: Autor (2015).

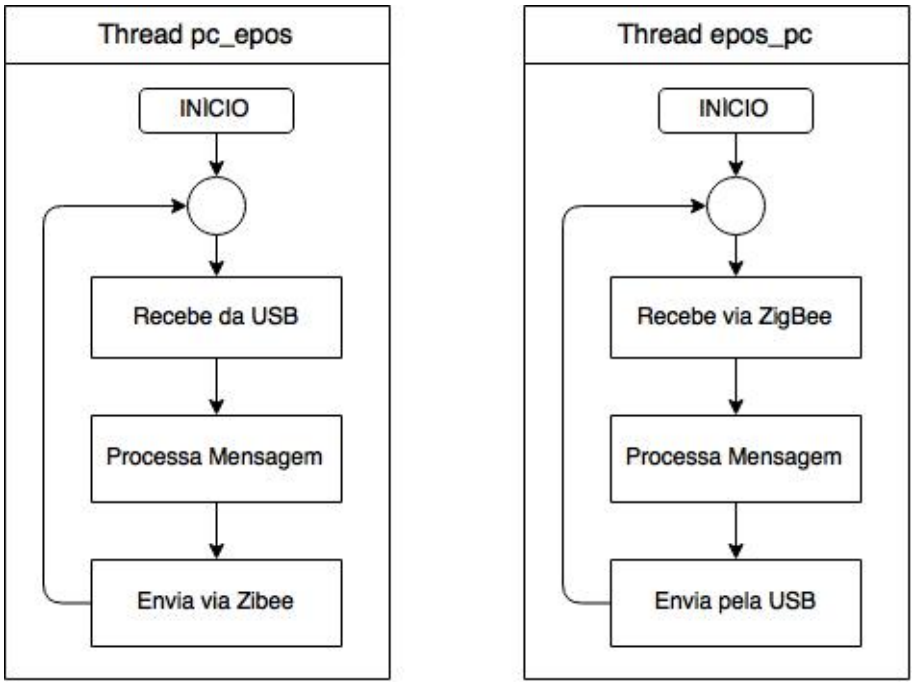
3.3 Terminal de Comunicação

Escalonador Round-Robin é um dos algoritmos utilizados por processos e rede escalonadores em computação. Fatias de tempo são atribuídas a cada processo (*thread*) em porções iguais e de forma circular.

A fim de executar os processos de forma justa, um agendador de Round-Robin geralmente emprega *time-sharing*, dando a cada uma das *thread* o processador por um intervalo de tempo ou *quantum*. O processador é retomado da próxima vez que um intervalo de tempo é atribuído a esse processo. Se o processo termina ou muda seu estado de espera durante o seu *quantum* de tempo atribuído, o agendador seleciona o primeiro processo na fila pronto para executar.

No programa do terminal de comunicação foram criados duas *threads* e utilizado o escalonador Round-Robin para fazer o processamento das *threads*. O *quantum* do escalonador foi alterado de $10ms$ para $50ms$, pois o tempo de $10ms$ não era suficiente para executar a atividade de recebimento via ZigBee. Na Figura 26 é mostrado o fluxograma para cada *thread*.

Figura 26 – Fluxogramas das threads.



Fonte: Autor (2015).

A *thread pc_epos* é responsável pela comunicação entre o sistema supervisório e os EPOS-Slave. Ela fica tentando receber dados da USB, que está conectado com o computador onde está rodando o sistema supervisório. Após receber uma mensagem serial, se faz o processamento de acordo com o protocolo de comunicação de recebimento dos

dados para montar uma estrutura de mensagem (Código 3.1), onde será posteriormente enviado via ZigBee. Todas essas atividades são executadas de forma cíclica em um *while* infinito.

Código 3.1 – Estrutura de mensagem do EPOS-Master para os EPOS-Slave.

```
struct Message1_{
  int id_send;
  int id_receive;
  char cmd;
  int value;
} Message1;
```

A *thread epos_pc* é responsável pela comunicação em os EPOS-Slave e o computador. Primeiramente, ela fica tentando receber dados dos EPOS-Slave via ZigBee. Após o recebimento os dados (Código 3.2), eles são processados para o envio serialmente pela USB para o computador. Todas essas atividades são executadas de forma cíclica em um *while* infinito.

Código 3.2 – Estrutura de mensagem dos EPOS-Slave para o EPOS-Master.

```
union float_char{
      float dfloat;
    unsigned char dchar[4];
};

struct Message2_{
  int id_send;
  int id_receive;
  char state;
  union float_char data[5];
} Message2;
```

3.4 Sistema Supervisório

Sistemas supervisórios são sistemas que utilizam software para monitorar e supervisionar as variáveis e os dispositivos de sistemas de controle conectados através de drives específicos. Neste trabalho é utilizado o LabVIEW como ferramenta chave para a criação do sistema supervisório, detalhada na seção 2.3.

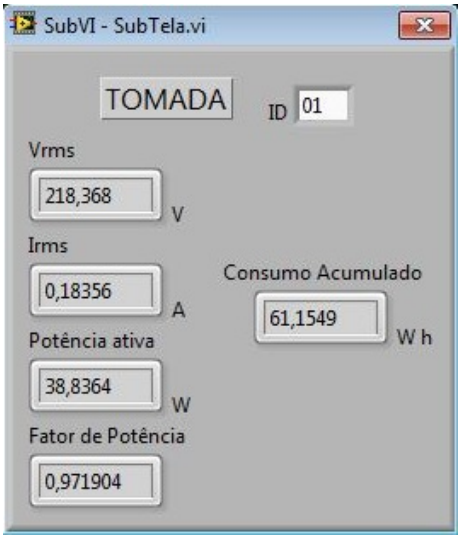
O diagrama de blocos do sistema supervisório foi desenvolvido de forma modular, com o objetivo de facilitar futuras atualizações e tornar a programação e depuração mais simples. Para isso, foram criados os seguintes subVIs:

SubVI Serial Write - Responsável por cuidar do protocolo de comunicação de envio serialmente para o Terminal de Comunicação.

SubVI Serial Read - Responsável pela manipulação dos dados recebidos de acordo com o protocolo de comunicação.

SubVI SubTela - Tem o objetivo de mostrar maiores informações de uma tomada, conforme mostrado na Figura 27.

Figura 27 – SubVI - SubTela.

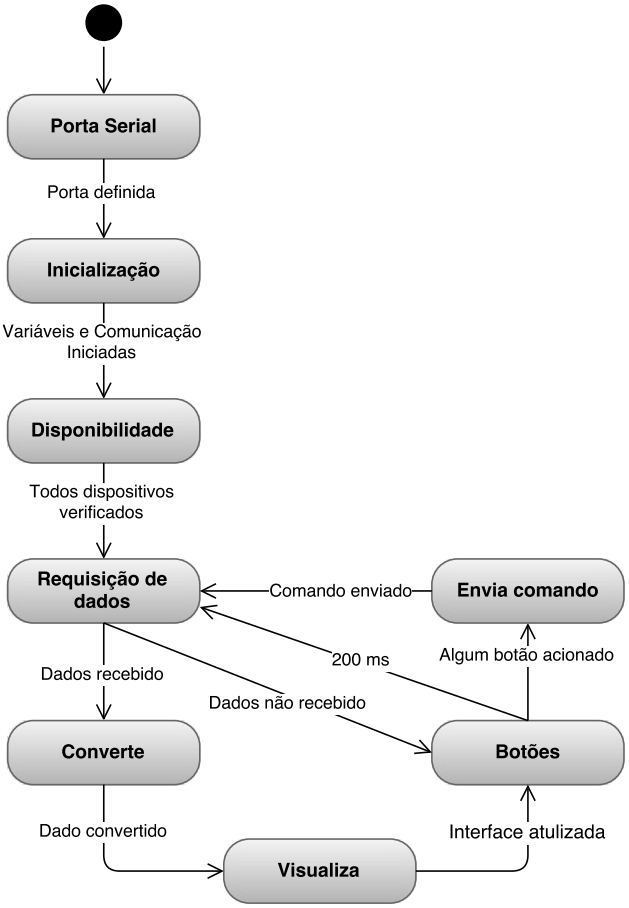


Fonte: Autor (2015).

O programa do LabVIEW está implementado em forma de máquina de estados. Na Figura 28 é apresentada uma representação desta máquina de estados.

Porta Serial - Este estado fica aguardando a definição de qual porta USB está conectado o terminal de comunicação (EposMaster). Ouve a necessidade de criar este

Figura 28 – Máquina de estados da execução do sistema supervisório.



Fonte: Autor (2015).

estado devido a geração de um erro na execução do programa, caso o usuário não definisse antes.

Inicialização - Neste estado ocorre a inicialização de um vetor de *cluster*, onde as informações de cada dispositivos serão salvas. O *cluster* contem os seguintes campos: Tipo, Disponível, Estado, Dado 1, Dado 2, Dado 3, Dado 4 e Dado 5. Também, ocorre neste estado a inicialização da porta serial e suas devidas configurações iniciais. A posição no vetor significa o ID do dispositivo.

Disponibilidade - Neste estado acontece a verificação de quais dispositivos (Epos-Slave) estão disponíveis para serem controlados. O processo ocorre através do envio do comando *get* para o dispositivo e o mesmo deve responder em $500ms$, caso isso não ocorrer, significa que o dispositivo não está disponível. Este processo é feito para cada dispositivo do vetor que foi inicializado no estado anterior. Uma segunda tentativa de verificação de disponibilidade ocorre para os dispositivos que se mostraram não disponíveis na primeira verificação.

Requisição de dados - Neste estado acontece a requisição de dados através do comando *get* para os dispositivos, com objetivo de atualizar as informações do sistema supervisório.

Converte - Responsável pela manipulação dos dados recebidos e o cálculo do consumo acumulado das tomadas.

Visualiza - Este estado faz a atualização dos dados na SubVI SubTela, caso esta sub-tela esteja aberta.

Botões - Neste estado ocorre a verificação se algum botão do painel frontal sofreu alteração de valor. Através de uma ferramenta chamada *Event Structure* é utilizada para associar a mudança de valor dos botões das tomadas e do controle do ar condicionado à um ambiente correspondente desenvolvido para executas tarefas específicas. Essas tarefas são apenas a definição do ID, comando e valor, que deve ser envio para o terminal de comunicação.

Envia comando - Responsável por chamar a SubVI Serial Write, passando o ID do dispositivo que deve receber as instruções, juntamente com o comando e valor.

O painel frontal, Figura 29, foi criado como um exemplo de aplicação do deste Trabalho de Conclusão de Curso. No painel frontal existe quatro conjunto de botões de tomada, cada conjunto contem um botão para ligar e desligar, um botão para reiniciar o consumo e um botão para chamar a SubTela de visualização dos dados. Na interface também contem um indicador do consumo acumulado de todas a quatros tomadas. Na parte do ar controle do ar condicionado, contem os botões para as principais funções do ar condicionado: Liga/Desliga, Temperatura, Modo, Velocidade do Vento, Turbo, Brisa e Auto limpeza.

O botão *Help* do painel frontal tem a função de abrir uma janela com mensagem de instruções de operação do sistema supervisório. O botão *Stop* finaliza a execução do programa. Esses dois botões podem ser clicados à qualquer momento e serão executados ao término da atividade do estado em que o programa se encontra.

Figura 29 – Painel frontal do sistema supervisório.

Fonte: Autor (2015).

4 Avaliação dos Dispositivos

Neste capítulo é apresentado as principais avaliações realizadas com a tomada e com o controle do ar condicionado, desenvolvidos neste projeto de dispositivos aplicados à IoT. O desenvolvimento do projeto com seus devidos testes e experimentos foram realizados no Laboratório de Integração de Software e Hardware (LISHA) da Universidade Federal de Santa Catarina – Campus Joinville.

4.1 Tomada

O primeiro experimento foi a realização de medições da tensão eficaz, corrente eficaz, potência ativa e fator de potência de algumas cargas de teste. Foram utilizados como cargas de teste: duas lâmpadas incandescentes ($40W$ e $60W$), um televisão com especificação de potência de $45W$ e uma sanduicheira com especificação de potencia de $700W$. Com o terminal de comunicação conectado ao computador que está executando o sistema supervisório desenvolvido no LabVIEW, as cargas de teste foram conectadas na tomada inteligente e após alguns segundos os parâmetros elétricos começaram a ser atualizados na subtela do sistema supervisório (Figura 27). A Tabela 4 mostra os parâmetros obtidos para as diferentes cargas de teste.

Tabela 4 – Medições com a tomada inteligente.

Carga de Teste	**Tensão Eficaz (V_{rms})**	**Corrente Eficaz (A_{rms})**	**Potência Ativa (W)**	**Fator de Potência**
Lâmpada 40W	218,36	0,183	38,83	0,971
Lâmpada 60W	217,52	0,272	58,41	0,972
Televisão 45W	211,82	0,217	30,95	0,68
Sanduicheira 700W	211,85	2,953	615	0,98

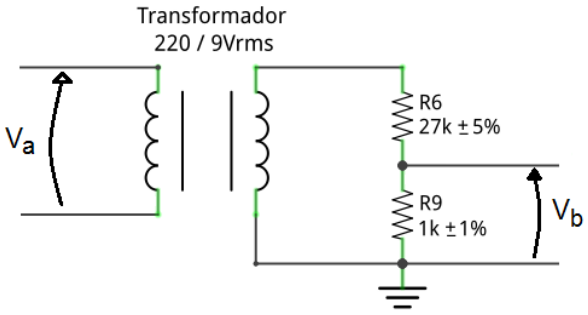
Fonte: Autor (2015).

Nas medições da tomada inteligente, a medição de uma grandeza Y de interesse é feita de maneira indireta, sendo esta grandeza obtida a partir de medidas de medidas de n grandezas primárias $(x_1, x_2, ..., x_n)$. Na Equação 4.1, temos uma expressão para o cálculo da incerteza padrão da grandeza de interesse (TOGINHO FILHO; ANDRELLO, 2009).

$$\sigma_Y = \sqrt{\left(\frac{\partial Y}{\partial x_1}\right)^2 (\sigma_{x_1})^2 + \left(\frac{\partial Y}{\partial x_2}\right)^2 (\sigma_{x_2})^2 + \cdots + \left(\frac{\partial Y}{\partial x_n}\right)^2 (\sigma_{x_n})^2} \tag{4.1}$$

A medição da tensão pelo módulo medidor da tomada passa por um transformador de tensão ($220V_{rms}/9V_{rms}$) e um divisor resistivo, conforme a Figura 30, a tensão (V_b) obtida na entrada do módulo medidor pode ser modelada pela Equação 4.2.

Figura 30 – Medição indireta da tensão.



Fonte: Autor (2015).

$$V_b = \frac{R_9}{R_6 + R_9}\left(\frac{9}{220}\right)V_a \tag{4.2}$$

Assumindo que o transformador é ideal e que as incertezas de medições estão atreladas apenas aos resistores, R_6 e R_9 contém uma incerteza de 5% e 1%, respectivamente. Portanto, a incerteza do valor de tensão V_b, lidos pelo módulo medidor é modelada pela Equação 4.3, onde σ_{R_9} é a incerteza do resistor R_9, σ_{R_6} é a incerteza do resistor R_6, $\frac{\partial V_b}{\partial R_9}$ é a derivada parcial V_b em relação à R_9 e $\frac{\partial V_b}{\partial R_6}$ é a derivada parcial V_b em relação à R_6.

$$\sigma_{V_b} = \sqrt{\left(\frac{\partial V_b}{\partial R_9}\right)^2 (\sigma_{R_9})^2 + \left(\frac{\partial V_b}{\partial R_6}\right)^2 (\sigma_{R_6})^2} \tag{4.3}$$

Para a primeira medição da Tabela 4, com uma lâmpada incandescente de $40W$, obteve-se uma medição de $218,36V_{rms}$ de tensão eficaz, a sua incerteza de medição é obtida através do seguinte cálculo:

$$\begin{aligned}
\sigma_{V_b} &= \sqrt{\left(\frac{R_6}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2 (\sigma_{R_9})^2 + \left(\frac{R_9}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2 (\sigma_{R_6})^2} \\
&= \sqrt{\left(\frac{27k}{(27k+1k)^2}\frac{9}{220}218,36\right)^2 (0,01.1k)^2 + \left(\frac{1k}{(27k+1k)^2}\frac{9}{220}218,36\right)^2 (0,05.27k)^2} \\
&= 0,0156 \\
&= 15,6mV_{rms}
\end{aligned} \tag{4.4}$$

Para uma tensão de $218,36V_{rms}$ na carga, a tensão V_b na entrada do módulo medidor deverá ser $319mV_{rms}$, segundo o cálculo a seguir:

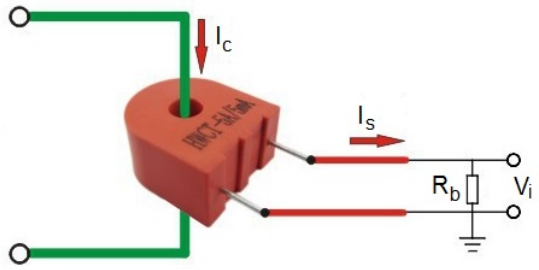
$$V_b = \frac{R_9}{R_6 + R_9}\left(\frac{9}{220}\right)V_a = \frac{1k}{27k + 1k}\left(\frac{9}{220}\right)218,36 = 319mV_{rms} \tag{4.5}$$

Portanto, a medição da tensão eficaz do módulo medidor contém uma incerteza de $4,89\%$, de acordo com o calculo seguinte, para a primeira carga de teste. As incertezas de medições da tensão eficaz para as outras cargas da Tabela 4 encontram-se no Apêndice E.

$$\frac{\sigma_{V_b}}{V_b} = \frac{15,6mV_{rms}}{319mV_{rms}} = 0,0489 = 4,89\% \tag{4.6}$$

A obtenção do valor de corrente pelo módulo medidor é feita através da medição de uma tensão V_i gerada sobre um resistor R_b a partir de uma corrente I_s induzida pela corrente consumida pela carga I_c, exemplificado na Figura 31.

Figura 31 – Medição indireta da corrente.



Fonte: Autor (2015).

A corrente I_s está relacionada com a corrente I_c através da relação de ganho do transformador de corrente, $\frac{I_c}{I_s} = 1000$. Então, a tensão V_i pode ser obtida a partida seguinte equação:

$$V_i = \frac{R_b.I_c}{1000} \tag{4.7}$$

A incerteza do valor de tensão V_i obtidos pelo módulo medidor é modelado pela Equação 4.8, onde σ_{R_b} é a incerteza do resistor R_b, σ_{Ic} é a incerteza do transformador de corrente, $\frac{\partial V_i}{\partial R_b}$ é a derivada parcial de V_i em relação à R_b e $\frac{\partial V_i}{\partial I_c}$ é a derivada parcial de V_i em relação à I_c.

$$\sigma_{V_I} = \sqrt{\left(\frac{\partial V_i}{\partial R_b}\right)^2 (\sigma_{R_b})^2 + \left(\frac{\partial V_i}{\partial I_c}\right)^2 (\sigma_{I_c})^2} \tag{4.8}$$

O transformador de corrente e o resistor de base (R_b) utilizados para aquisição do sinal de corrente, contém uma incerteza de 1%. Portanto, para a primeira medição da Tabela 4, com uma lâmpada incandescente de $40W$ obteve-se uma medição de $0,183A_{rms}$

de corrente eficaz, a incerteza de medição da tensão V_i é obtida através do seguinte cálculo:

$$\begin{aligned}\sigma_{V_I} &= \sqrt{\left(\frac{I_c}{1000}\right)^2 (\sigma_{R_b})^2 + \left(\frac{R_b}{1000}\right)^2 (\sigma_{I_c})^2} \\ &= \sqrt{\left(\frac{0,183}{1000}\right)^2 (0,01.5,6)^2 + \left(\frac{5,6}{1000}\right)^2 (0,01.0,183)^2} \\ &= 0,0000144 \\ &= 0,0144 mV_{rms}\end{aligned} \tag{4.9}$$

Para uma corrente de $0,183 A_{rms}$ na carga, a tensão V_i na entrada do módulo medidor será $1,024 mV_{rms}$, segundo o cálculo a seguir:

$$V_i = \frac{R_b.I_c}{1000} = \frac{5,6.0,183}{1000} = 1,024 mV_{rms} \tag{4.10}$$

Portanto, a medição da corrente eficaz do módulo medidor contém uma incerteza de $1,4\%$, de acordo com o calculo seguinte, para a primeira carga de teste. As incertezas de medições da corrente eficaz para as outras cargas da Tabela 4 encontram-se no Apêndice E.

$$\frac{\sigma_{V_I}}{V_i} = \frac{0,0144 mV_{rms}}{1,024 mV_{rms}} = 0,014 = 1,4\% \tag{4.11}$$

A Tabela 5 mostra o resultado final dos cálculos da incerteza de medição da tensão e corrente eficazes para os diferentes tipos de cargas de teste, com valor máximo de $4,91\%$ para as medições de tensão eficaz e $1,42\%$ para medições de corrente eficaz.

Tabela 5 – Incertezas de medições da tensão e corrente eficaz.

	Tensão Eficaz (V_{rms})		**Corrente Eficaz (A_{rms})**	
Carga de Teste	**Valor medido**	**Incerteza**	**Valor medido**	**Incerteza**
Lâmpada 40W	218,36	4,89 %	0,183	1,4 %
Lâmpada 60W	217,52	4,90 %	0,272	1,41 %
Televisão 45W	211,82	4,91 %	0,217	1,40 %
Sanduicheira 700W	211,85	4,91 %	2,953	1,42 %

Fonte: Autor (2015).

O cálculo da incerteza do fator de potência não foi possível calcular, pois se faz necessário ter os valores da potência aparente para cada carga de teste. As incertezas de medições da potência ativa e do fator de potência estão atrelados as incerteza da tensão e corrente eficazes.

Outro experimento realizado foi a medição do consumo acumulado, com uma lâmpada incandescentes de $60W$. Foi conectado a lâmpada através de uma adaptador e

mensurado o consumo acumulado 30 minutos após acionar a carga, através do sistema supervisório. Obteve-se uma leitura de $28,38Wh$ de consumo acumulado, considerando a lâmpada incandescente ideal com fator de potência igual à 1, temos que a medição do consumo acumulado deveria ser $30Wh$. Portanto, para este experimento o consumo acumulado teve um erro de $5,4\%$ do valor teórico esperado, segundo o seguinte cálculo:

$$Erro(\%) = \frac{30 - 28,38}{30} = 0,054 = 5,4\% \tag{4.12}$$

Possíveis erros sistemáticos presentes nas medições da tomada inteligente podem ser minimizados ou eliminados com uma melhor calibração dos ganhos de tensão e corrente, com testes em bancada com instrumentos que normalmente só são encontrados em laboratórios de homologia e certificação. O equipamento de medição de tensão e corrente alternada, mais preciso que se tinha disponível era um multímetro da marca Minipa, modelo ET-2082C. Portanto, toda a calibração dos ganhos de corrente e tensão do módulo medidor foram realizadas com medições simultâneas com o multímetro. Porém, este multímetro da Minipa possui uma incerteza de medição de $1,2\%$ para medições de tensão alternada no *range* de $750V$ e uma incerteza de medição de 3% para medições de corrente alternada no *range* de $20A$.

4.2 Controle do Ar Condicionado

Um dos primeiros teste de avaliação foi a verificação da distância máxima de operação do controle do ar condicionado desenvolvido neste trabalho. Foi testado com o ar condicionado do mesmo modelo para o controle foi desenvolvido (Electrolux BI22F), posicionou-se o controle na mesma altura do ar condicionado com o emissor IR direcionado para o mesmo, e à uma distância $2m$. Então, foi enviado comandos de mudar a temperatura a partir do sistema supervisório e distanciando horizontalmente o controle. Percebeu-se que os comandos começaram a falhar com distância maiores de $4,2m$. Realizando vários teste em torno do limite, constata que a distância máxima de operação do controle do ar condicionado é de aproximadamente 4 metros.

Outra análise realizada no desenvolvimento do controle do ar condicionado foi a verificação do tamanho da classe desenvolvida para o controle do ar condicionado no EPOS. Utilizando a ferramenta *objdump* (versão 2.2) disponível no compilador do GCC (versão 4.4.4) para a arquitetura ARM, é uma ferramenta que pode ser utilizada para extrair várias informações do arquivo objeto. Através do terminal do Linux, entrou na pasta de aplicação do controle do EPOS e executou o seguinte comando:

/usr/local/arm/gcc/bin/arm-objdump -dS controle

A opção *-dS* da ferramenta *arm-objdump*, faz a exibição misturada do código-fonte com uma exibição dos mnemônico Assembler da desmontagem do arquivo objeto. Assim, pode-se extrai a informação do tamanho da implementação, em bytes, para vários métodos da classe do ar condicionado, mostrado na Tabela 6. Os métodos da classe que não aparecem na Tabela 6, não foram utilizados na aplicação ou o código está implementado *inline*, ou seja, o código do método está embutido no código Assembler de outro método. Após, com um código de teste utilizando o operador *sizeof*, obtém-se o tamanho de cada atributo da classe (Tabela 6). Assim, conclui-se que para o código de aplicação e o código da classe desenvolvida neste trabalho para o controle do ar condicionado (contido no Apêndice D), a classe do controle do ar condicionado está com um tamanho de 1759 bytes.

Em uma última avaliação do controle do ar condicionado foram verificados os limites aceitáveis do controle do programa do EPOSMote. Foram enviados a partir do sistema supervisório comandos que estão fora dos limites, como, por exemplo, nível de ventilação 4 ou temperatura 40^oC. Nesses casos, o programa do EPOSMote simplesmente faz a limitação do valor nas instruções recebidas e envia o sinal infravermelho para o ar condicionado com o limite máximo ou mínimo aceitável. Para o nível de ventilação o programa faz a limitação entre 0 e 4, e para a temperatura o programa faz a limitação entre 16^oC e 30^oC.

Tabela 6 – Tamanho da classe do controle do ar condicionado.

Tipo	**Descrição**	**Tamanho (Bytes)**
Atributo	GPIO_Pin _pin	4
	MC13224V_Timer_1 _timer	8
	enum MODO _modo	4
	bool _status	1
	char _fan_speed	1
	bool _eco	1
	char _temperature	1
	bool _turbo	1
	bool _self_clean	1
	bool _brisa	1
	long long _frame1	8
	long long _frame2	8
	unsigned char reverse[16]	16
Método	AC_Elextrolux_Control	324
	void turn_off	12
	void turn_on	12
	void increase_temperature	12
	void decrease_temperature	12
	void set_temperature	44
	void set_modo	8
	void set_fan_speed	40
	void toggle_turbo	16
	void toggle_self_clean	16
	void toggle_brisa	16
	void put_Leading	120
	void put_0	112
	void put_1	116
	void put_either	16
	void put_Tail	104
	void build_commands	480
	void send	244
TOTAL		**1759**

Fonte: Autor (2015).

5 Considerações finais

Este projeto abordou o desenvolvimento de uma tomada inteligente e um controle de ar condicionado, que se comunica com um sistema supervisório através de comunicação sem fio.

A tomada inteligente é integrada a um módulo medidor de baixo custo, capaz de medir: potência ativa, corrente eficaz, tensão eficaz, fator de potência e o consumo acumulado. Em experimentos realizados com diferentes tipos de cargas de teste, a Tabela 5 mostra o resultado final dos cálculos da incerteza de medição da tensão e corrente eficazes, com valor máximo de $4,91\%$ para as medições de tensão eficaz e $1,42\%$ para medições de corrente eficaz. Em outro experimento de medição do consumo acumulado, para uma lâmpada incandescente de $60W$ durante 30 minutos, obteve-se um erro de $5,4\%$ do valor teórico esperado. No entanto, o sistema de medição pode ser aprimorado através do uso de componentes com melhor precisão e, principalmente, a utilização de instrumentos de medição de alta precisão para fazer a calibração dos ganhos do sinal amostrado no módulo medidor.

O desenvolvimento do controle do ar condicionado foi capaz de controlar as principais funções de um controle remoto: liga/desliga, modo de operação, aumenta/diminui a temperatura, nível da ventilação, turbo, brisa (oscilação) e auto limpeza. Segundo os experimentos, o dispositivo é capaz de operar à uma distância máxima de 4 metros do ar condicionado. A classe desenvolvida do ar condicionado para o envio dos comandos, contém um tamanho de 1759 bytes.

O sistema supervisório foi criado no ambiente de desenvolvimento gráfico LabVIEW, o painel frontal foi criado como um exemplo de aplicação deste projeto como forma de proporcionar uma interface simples ao usuário. A interface contém um conjunto de botões e indicadores que representam quatro tomadas inteligentes e um controle de ar condicionado. No sistema supervisório é possível chamar uma subtela para a visualização de todos os parâmetros medidos da respectiva tomada.

No sistema supervisório apresentado surge como exemplo das possibilidades envolvidas no conceito de Internet das Coisas. Quaisquer que sejam os equipamentos elétricos de uma residência, estes podem ser integrados e monitorados pelo sistema proposto por este trabalho. É possível ainda que tal plataforma seja disponibilizada através da internet, permitindo o acesso de usuários em qualquer lugar do mundo, desde que tenha conectividade com a internet, assim, atingindo o paradigma de Internet das Coisas.

5.1 Trabalhos futuros

Esta monografia constitui uma base vasta para novas ideias, permitindo diferentes rumos de trabalhos futuros: desenvolver novos dispositivos inteligentes, aperfeiçoar os dispositivos desenvolvidos neste trabalho e disponibilizar um sistema supervisório remoto.

Sugestões para desenvolver novos dispositivos:

- Desenvolver uma lâmpada inteligente com sensor de luminosidade, sensor de presença e um controlador de potência;
- Desenvolver um sistema de controle acesso de salas, laboratórios e restaurante universitário com tecnologia RFID presente nas carteira de estudante;
- Desenvolver um sistema de controle de abertura de janelas;
- Desenvolver um sistema de controle e medição do fluxo de água, com identificação de um possível rompimento nas instalações.

Sugestões para aperfeiçoar os dispositivos desenvolvidos neste trabalho:

- Redução do hardware da tomada com a redução do transformador do sinal da tensão da rede e a construção de uma única placa;
- Desenvolver funcionalidades liga/desliga automaticamente do ar condicionado e controle automático de temperatura para o controle de ar condicionado desenvolvido neste trabalho;
- Realizar a decodificação de outros aparelhos eletrônicos que tenho comunicação infravermelho para integrar no dispositivo de controle do ar condicionado.

Sugestões para disponibilizar um acesso remoto do sistema supervisório deste trabalho:

- Utilizar a ferramenta LabVIEW Web Service para controlar o sistema supervisório pela internet;
- Criar um aplicativo de celular e tablet para usar como sistema supervisório dos dispositivos inteligentes.

Outra sugestão de trabalho futuro é o estudo de viabilidade financeira da implantação de módulos de controle e sistemas computacionais em residências. Também, um estudo de viabilidade financeira de tornar os dispositivos e sistema desenvolvidos neste trabalho em produtos para o consumidor final.

Referências

ADA, L. *Using an IR Sensor*. 2012. Acessado em: 12 de agosto de 2015. Disponível em: <https://learn.adafruit.com/ir-sensor/using-an-ir-sensor>.

ASHTON, K. That 'internet of things' thing. *RFID Journal*, 2009.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, 2010.

BACURAU, R. M. *Medidor de Energia Inteligente para Discriminação de Consumo por Aparelho Através de Assinatura de Cargas*. Dissertação (Mestrado) — Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação, 2014.

COSTA, J. C. B. *Ferramenta de apoio ao projeto, configuração e gestão de instalações domóticas*. Dissertação (Mestrado) — Instituto Superior de Engenharia do Porto, 2012.

DIAS, C. L. de A. *Domótica: aplicabilidade às edificações residenciais*. Dissertação (Mestrado) — Universidade Federal Fluminense, 2004.

FRöHLICH, A. A. M. Application-oriented operating systems. In: *GMD - Forschungszentrum Informationstechnik*. [S.l.: s.n.], 2001.

LISHA. *EPOS User Guide*. 2015. Acessado em: 28 de agosto de 2015. Disponível em: <http://epos.lisha.ufsc.br/EPOS+User+Guide>.

LISHA. *EPOSMote II*. 2015. Acessado em: 10 de setembro de 2015. Disponível em: <http://epos.lisha.ufsc.br/EPOSMote+II>.

MAYER, C. P. Security and privacy challenges in the internet of things. In: *Proceedings of KiVS Workshop on Global Sensor Networks (GSN09)*. [s.n.], 2009. Disponível em: <http://doc.tm.uka.de/2009/security-gsn-camera-ready.pdf>.

SENA, H. *Automação Residencial e Internet das Coisas*. 2015. Acessado em: 20 de setembro de 2015. Disponível em: <http://devidaasuaideia.com.br/hangout-automacao-residencial-e-internet-das-coisas/>.

TEXAS INSTRUMENTS. *MSP430AFE2x3, MSP430AFE2x2, MSP430AFE2x1*: Mixed signal microcontroller. [S.l.], 2011. Disponível em: <http://www.ti.com/lit/ds/slas701a/slas701a.pdf>.

TEXAS INSTRUMENTS. User's Guide, *BOOST-IR Infrared (IR) BoosterPack^{TM} Plug-in Module*. 2015. Disponível em: <http://www.ti.com/lit/ug/slau598a/slau598a.pdf>.

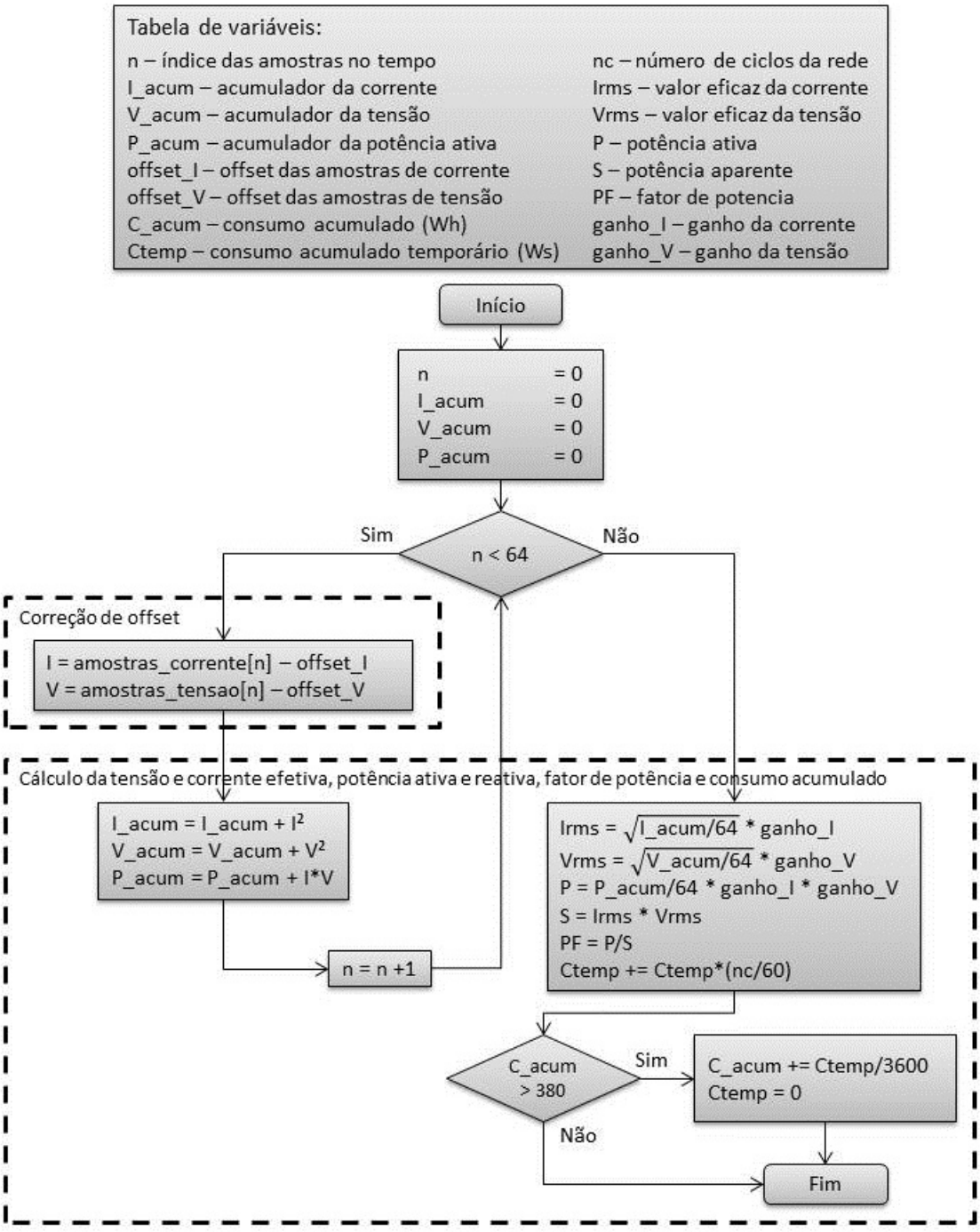
TEXAS INSTRUMENTS. Application Report, *Infrared Remote Control Implementation With MSP430FR4xx*. 2015. Disponível em: <http://www.ti.com/lit/an/slaa644b/slaa644b.pdf>.

TOGINHO FILHO, D. O.; ANDRELLO, A. C. *Medição e propagação de erros*. [S.l.], 2009.

Apêndices

APÊNDICE A – Cálculo dos parâmetros elétricos do Módulo Medidor

Figura 32 – Fluxograma do cálculo dos parâmetros elétricos.



APÊNDICE B – Códigos do EPOS para o Terminal de Comunicação

Código B.1 – Código do Terminal de Comunicação.

```
#include <machine.h>
#include <alarm.h>
#include <mach/mc13224v/emote2_startup.h>
#include <system/config.h>
#include <utility/ostream.h>
#include <utility/string.h>
#include <thread.h>

#define ID_SLAVE 1
#define ID_MASTER 99
#define SIZE 20
//#define MSG

__USING_SYS

union float_char{
  float dfloat;
  unsigned char dchar[4];
};
//struct pc->epos
struct Message1_{
  int id_send;
  int id_receive;
  char command;
  int value;
} Message1;

//struct epos->pc
struct Message2_{
  int id_send;
  int id_receive;
  char state;
  union float_char data[5];
} Message2;

NIC * nic;
NIC::Protocol prot;
NIC::Address src;

OStream cout;
EMote2_Startup * startup;

int func_pc_epos(void);
int func_epos_pc(void);

Thread * pc_epos;
Thread * epos_pc;

Semaphore sem_usb;
```

```
///-----MAIN-----//
int main() {
  nic = new NIC();
  startup = new EMote2_Startup();

  cout << "\nEPOS Master do TCC ID:" << ID_MASTER << endl;

  pc_epos = new Thread(&func_pc_epos);
  cout << "Thread PC->EPOS criada" << endl;

  epos_pc = new Thread(&func_epos_pc);
  cout << "Thread EPOS->PC criada" << endl;

  pc_epos->join();
  epos_pc->join();

  return 0;
}

int func_pc_epos(void)
{
  char next_byte;
  int count;
  char usb_message[SIZE];
  char aux_string[SIZE];

  while (true) {
    while(startup->get_char() != '#');
    sem_usb.p();
    count = 0;
    while((next_byte = startup->get_char()) != ';') {
      usb_message[count] = next_byte;
      count++;
    }
    sem_usb.v();
    usb_message[count--] = '\0';

    Message1.id_send = ID_MASTER;

    aux_string[0] = usb_message[0];
    aux_string[1] = usb_message[1];
    aux_string[2] = '\0';
    Message1.id_receive = atoi(aux_string);

    Message1.command = usb_message[2];

    aux_string[0] = usb_message[3];
    aux_string[1] = usb_message[4];
    aux_string[2] = usb_message[5];
    aux_string[3] = '\0';
    Message1.value = atoi(aux_string);

    while ((nic->send(NIC::BROADCAST, (NIC::Protocol) 1, &(Message1), sizeof(
        Message1))) != 11){
      #ifdef MSG
        cout << "Failed to send!" << endl;
      #endif // MSG
    }
  }
```

```
  return 0;
}

int func_epos_pc(void)
{
  char aux_string2[SIZE];
  int i,j;
  while(true){
    while(!(nic->receive(&src, &prot, &(Message2), sizeof(Message2)) > 0)){
      Thread::yield();
    }

    if(Message2.id_receive == ID_MASTER){
      sem_usb.p();
      startup->put_char('#');                    ///start caracter
      startup->put_char(Message2.id_send);
      startup->put_char(Message2.state);        ///put state of slave
      for( i=0; i<5; i++)
        for( j=0; j<4; j++)
          startup->put_char(Message2.data[i].dchar[3-j]); ///put data
      startup->put_char(';');                    ///end caracter
      sem_usb.v();
    }
  }
  return 0;
}
```

Código B.2 – Alteração das configurações no arquivo *trais.h*.

```
template <> struct Traits<Thread>: public Traits<void>
{
  typedef Scheduling_Criteria::Round_Robin Criterion;
  static const bool smp = false;
  static const bool trace_idle = false;
  static const unsigned int QUANTUM = 50000; // 10000 us
};
```

APÊNDICE C – Códigos do EPOS para a Tomada Inteligente

Código C.1 – Código da Aplicação do EPOS para a Tomada Inteligente.

```
#include <machine.h>
#include <alarm.h>
#include <mach/mc13224v/emote2_startup.h>
#include <system/config.h>
#include <utility/ostream.h>
#include <gpio_pin.h>
#include <semaphore.h>

#define ID_SLAVE 1
#define ID_MASTER 99

//#define MSG

__USING_SYS

const long period = 7; // segundos

union float_char{
  float dfloat;
  unsigned char dchar[4];
};

struct Message1_{
  int id_send;
  int id_receive;
  char cmd;
  int value;
} Message1;

struct Message2_{
  int id_send;
  int id_receive;
  char state;
  union float_char data[5];    // dados do modulo medidor [Vrms,Irms,P,fp,C]
} Message2;

NIC * nic;
NIC::Protocol prot;
NIC::Address src;

OStream cout;
EMote2_Startup * startup;
MC13224V_UART * uart2;

Semaphore sem_uart2;

int func_a(void);
void func_b(void);
void func_c(void);
```

```
bool data_for_send = false;

///-----MAIN----- //
int main() {
  nic = new NIC();
  startup = new EMote2_Startup();
  cout << "\nEposSlave - Tomada ID: " << ID_SLAVE << endl;

  uart2 = new MC13224V_UART(1);
  uart2->config(115200,8,0,1);
  cout << "Portas Uart Iniciadas" << endl;

  cout << "Criando Alarm" << endl;
  Function_Handler handler_b(&func_b);
  Alarm alarm_b(1000000*period, &handler_b, Alarm::INFINITE);

  startup->set_green();
  startup->set_rele();
  Message2.state = 'l';
  Message2.id_send = ID_SLAVE;
  Message2.id_receive = ID_MASTER;

  cout << "OK!" << endl;
  while(true){
    int s = -1, c1 = 0;
    while(s<0){
      #ifdef MSG
        cout << "Tentando receber do Master" <<endl;
      #endif // MSG
      s = nic->receive(&src, &prot, &(Message1), sizeof(Message1));
      if(++c1 > 3)
        break;
    }

    if(s>=0){
      if(Message1.id_receive == ID_SLAVE){
        switch (Message1.inf){
          case 'g':
            data_for_send = true;
            Alarm::delay(50000);
            break;

          case 'a':   //liga
            startup->set_rele();
            startup->set_green();
            Message2.state = 'l';
            #ifdef MSG
              cout<<"Ligado!" << endl;
            #endif // MSG
            break;

          case 'b':   //desliga
            startup->clear_rele();
            startup->clear_green();
            Message2.state = 'd';
            #ifdef MSG
              cout << "Desligado!" <<endl;
            #endif // MSG
            break;
```

```
            case 'r':   // reset no consumo
              sem_uart2.p();
              uart2->put('#');
              uart2->put('s');
              uart2->put('z');
              uart2->put(';');
              sem_uart2.v();
              #ifdef MSG
                cout << "Consumo zerado" << endl;
              #endif
              break;
          }
        }
      }

      if(data_for_send){
        int r = 0;
        #ifdef MSG
          cout << " Vrms: " << Message2.data[0].dfloat << endl;
          cout << " Irms: " << Message2.data[1].dfloat << endl;
          cout << " Pot ativa: "<< Message2.data[2].dfloat << endl;
          cout << " FP: " << Message2.data[3].dfloat << endl;
          cout << " Consumo: " << Message2.data[4].dfloat << " W h" << endl;
        #endif // MSG
        while (r != 11){
          r = (nic->send(NIC::BROADCAST, (NIC::Protocol) 1, &(Message2), sizeof(
              Message2)));
        }

        if(r==11){
          cout << "send" << endl;
          data_for_send = false;
        }
      }
    }
    cout << "The End" <<endl;
    return 0;
}

void func_b(void){
    int i,j;
    char next_byte, error;
    int count, checksum;
    char uart_message[50];

    //limpeza do buffer da uart
    while(uart2->has_data())
    uart2->get();

    error = 0;
    sem_uart2.p();
    uart2->put('#');
    uart2->put('g');
    uart2->put(';');
    sem_uart2.v();

    //Espera pelo bit start of frame
    i=0;
    while(next_byte != '#') {
```

```
      if(uart2->has_data()){
        next_byte = uart2->get();
      }else{
        if (++i > 300) {
          error++;
          #ifdef MSG
            cout << "Timeout Error!" << endl;
          #endif // MSG
          break;
        }
        Alarm::delay(100);
      }
    }

    if(!error){
      count = 0;
      checksum = 0;
    //Recebe dados ate o bit end of frame
    while((next_byte = uart2->get()) != ';') {
      uart_message[count++] = next_byte;
      checksum ^= checksum;
    }

    if((!checksum) && (count > 20)) {
      for(i=0; i<5; i++){
        for(j=0; j<4; j++){
          Message2.data[i].dchar[3-j] = uart_message[i*4+j];
        }
      }
    } else {
      error++;
      #ifdef MSG
        cout << "Checksum Error!" << endl;
      #endif // MSG
      }
    }
  }
```

Código C.2 – Código do *emote2 startup.h* com as alterações.

```
// EPOSMote2 Startup Board Mediators

#ifndef __emote2_startup_h
#define __emote2_startup_h

#include <cpu.h>
#include <machine.h>
#include <sensor.h>
#include <uart.h>
#include <gpio_pin.h>

__BEGIN_SYS

class EMote2_Startup
{
public:
    EMote2_Startup()
  : _led1_r((Traits<MC13224V>::emote_version == 0x2f) ? 10 : 23),
    _led1_g((Traits<MC13224V>::emote_version == 0x2f) ?  9 : 24),
    _led1_b((Traits<MC13224V>::emote_version == 0x2f) ? 11 : 25),
```

```
    _led2  ((Traits<MC13224V>::emote_version == 0x2f) ?  8 : 26),
    _led3  ((Traits<MC13224V>::emote_version == 0x2f) ? 23 : 27),
    _but1  ((Traits<MC13224V>::emote_version == 0x2f) ? 24 : 11),
    _but2  ((Traits<MC13224V>::emote_version == 0x2f) ? 22 : 10),
    _rele (26)          //KBI_4 of MC13224V
  {
      reset();
  }

    ~EMote2_Startup() {}

    void reset() {
//      _termistor.power(OFF);
//        _accel.power(OFF);
//        _uart.power(OFF);
        clear_green();
        clear_red();
        clear_rgb();
        clear_rele();
    }

    // RELE
    void set_rele() { _rele.set(); }
    void clear_rele() { _rele.clear(); }

    // Temperature Sensor
    int temperature() { return _termistor.sample(); }
    Temperature_Sensor & temperature_sensor() { return _termistor; }

    // TODO: Accelerometer
    int x_acceleration() { return _accel.sample_x(); }
    int y_acceleration() { return _accel.sample_y(); }
    int z_acceleration() { return _accel.sample_z(); }
    Accelerometer & accelerometer() { return _accel; }

    // Host/USB UART
    char get_char() { return _uart.get(); }
    void put_char(char c) { _uart.put(c); }
    UART & uart() { return _uart; }

    // LED1 (RGB)
    void set_rgb(char red, char green, char blue)
    {
        //TODO: use PWMs
        if(red) _led1_r.set();
        else _led1_r.clear();
        if(green) _led1_g.set();
        else _led1_g.clear();
        if(blue) _led1_b.set();
        else _led1_b.clear();
    }
    void clear_rgb()
    {
        _led1_r.clear();
        _led1_g.clear();
        _led1_b.clear();
    }

    // LED2
    void set_green() { _led2.set(); }
```

```
        void clear_green() { _led2.clear(); }

        // LED3
        void set_red() { _led3.set(); }
        void clear_red() { _led3.clear(); }

        // BUT1
        // TODO: interrupts
        bool get_button1() { return _but1.get(); }

        // BUT2
        // TODO: interrupts
        bool get_button2() { return _but2.get(); }

    private:
        Temperature_Sensor _termistor;
        Accelerometer _accel;
        UART _uart;
        GPIO_Pin _led1_r;
        GPIO_Pin _led1_g;
        GPIO_Pin _led1_b;
        GPIO_Pin _led2;
        GPIO_Pin _led3;
        GPIO_Pin _but1;
        GPIO_Pin _but2;
        GPIO_Pin _rele;
    };

    __END_SYS

    #endif
```

APÊNDICE D – Códigos Utilizados no Desenvolvimento do Controle do Ar Condicionado

Código D.1 – Código utilizado no processo de decodificação pelo Arduino. Adaptado de: Ada (2012).

```
// IR DEFINITIONS
#define IRpin_PIN       PIND
#define IRpin           3

#define MAXPULSE        65000
#define MAX_PULSE_PAIRS 70

uint16_t pulses[MAX_PULSE_PAIRS][2];
uint8_t currentPulse = 0;
uint32_t codigo = 0;

unsigned long currMicros, lastMicros, diffMicros;

boolean newCodeToRead = false;
void readIR(){
  if(newCodeToRead)
    return;

  currMicros = micros();
  if(currentPulse < MAX_PULSE_PAIRS){
    diffMicros = currMicros - lastMicros;
    if(IRpin_PIN & (1 << IRpin)){
      pulses[currentPulse++][1] = diffMicros;
      lastMicros = currMicros;
    }
    else{
      pulses[currentPulse][0] = diffMicros;
      lastMicros = currMicros;
    }
  }
  else
    newCodeToRead = true;
}

int decodeIR(){
  unsigned long lastTime = lastMicros;
  unsigned long nowTime = micros();

  if(nowTime > lastTime){
    unsigned long diffTime = (nowTime - lastTime);

    if(newCodeToRead || (diffTime > MAXPULSE && currentPulse != 0)){
      detachInterrupt(0);
      printpulses();
```

```
        newCodeToRead = false;
        currentPulse = 0;
        attachInterrupt(0, readIR, CHANGE);
        return 0;
      }
    }
    return -1;
}

void printpulses(void) {
    Serial.println("\n\nReceived: \nOFF    ON");
    for (uint8_t i = 0; i < currentPulse; i++) {
      Serial.print(pulses[i][0], DEC);
      Serial.print(" usec, ");
      Serial.print(pulses[i][1], DEC);
      Serial.println(" usec");
    }

    // print it in a 'array' format
    Serial.println("int IRsignal[] = {");
    Serial.println("ON, OFF");
    for (uint8_t i = 0; i < currentPulse-1; i++) {
      Serial.print(pulses[i][1], DEC);
      Serial.print(", ");
      Serial.print(pulses[i+1][0], DEC);
      Serial.println(",");
    }
    Serial.print(pulses[currentPulse-1][1], DEC);
    Serial.println(", 0};");


    int y =0;
    for (uint8_t i = 0; i < currentPulse-1; i++) {
      if(abs(int(pulses[i][1])-560) < 150){
        if(abs(int(pulses[i][1])-560) < 150){
          if(abs(int(pulses[i+1][0])-560) < 150){      // 0
            Serial.print("0");
          } else if(abs(int(pulses[i+1][0])-1650) < 150){ // 1
            Serial.print("1");
          }else{                        //?? 1 ou 0
            Serial.print("z");
          }
          if(++y > 3){
            Serial.print(".");
            y=0;
          }
          //Serial.println();
        }else{
          Serial.print("...");
        }
      }
    }
    Serial.println();
}

void setup(void){
    lastMicros = micros();
    attachInterrupt(0, readIR, CHANGE);

    Serial.begin(9600);
```

```
    Serial.println("Ready to decode IR!!!");
}

void loop(void){
  int decoded = decodeIR();
}
```

Código D.2 – Código do Arquivo-Cabeçalho da Classe do Controle do Ar Condicionado.

```
#ifndef __ac_control2_h
#define __ac_control2_h

#include <alarm.h>
#include <system/config.h>
#include <gpio_pin.h>
#include <timer.h>

__BEGIN_SYS

class AC_Elextrolux_Control {
public:
  enum MODO {
    AUTO  = 0b00, //Automatico
    DESU  = 0b01, //Desumificacao
    RESF  = 0b10, //Resfiar
    VENT  = 0b11  //Ventilar
  };

private:
//MEMBERS
  GPIO_Pin _pin;
  MC13224V_Timer_1 _timer;

  enum MODO _modo;
  bool _status;
  char _fan_speed;
  bool _eco;
  char _temperature;
  bool _turbo;
  bool _self_clean;
  bool _brisa;

  long long _frame1;
  long long _frame2;

  unsigned char reverse[16];

public:
  AC_Elextrolux_Control();

  ~AC_Elextrolux_Control() {}

//FUNCTION
  bool status()    { return _status; }
  MODO modo()      { return _modo; }
  int fan_speed()    { return int(_fan_speed); }
  int temperature()    { return (_temperature+16); }
  bool turbo()     { return _turbo; }
  bool self_clean() { return _self_clean; }
  bool brisa()     { return _brisa; }
```

```
    long long frame1(){ return _frame1; }
    long long frame2(){ return _frame2; }

  //FUNCTION OF CONTROL
    void turn_off();
    void turn_on();

    void increase_temperature();
    void decrease_temperature();
    void set_temperature(int temp = 23);

    void set_modo(MODO mod);

    void set_fan_speed(int level_speed);

    void set_turbo(bool t = true);
    void toggle_turbo();

    void set_self_clean(bool t);
    void toggle_self_clean();

    void set_brisa(bool b);
    void toggle_brisa();

  //FUNCTION TO SEND
  private:
    static void func() {}

    void put_Leading(void);

    void put_0(void);

    void put_1(void);

    void put_either(bool either);

    void put_Tail(void);

    char checksum(void);

    void build_commands();

    void send(void);
  };

  __END_SYS

  #endif
```

Código D.3 – Código da Classe do Controle do Ar Condicionado para o EPOS.

```
#include <alarm.h>
#include <mach/mc13224v/ac_electrolux_control.h>

__BEGIN_SYS

AC_Elextrolux_Control::AC_Elextrolux_Control()
  : _pin(9, MC13224V_GPIO_Pin::FUNC_ALT1),
  _timer(func, MC13224V_Timer::PWM)
```

```
{
  _timer.disable_pwm();

  _modo = RESF;
  _fan_speed=0x0; // level auto
  _eco = false;
  _temperature = 0b0111;  //23 C = 0b0111 + 16
  _turbo = false;
  _self_clean = false;
  _status = true;
  _brisa = false;

  reverse[0]=0x0;
  reverse[1]=0x8;
  reverse[2]= 0x4;
  reverse[3]= 0xc;
  reverse[4]= 0x2;
  reverse[5]= 0xa;
  reverse[6]= 0x6;
  reverse[7]= 0xe;
  reverse[8]= 0x1;
  reverse[9]= 0x9;
  reverse[10]= 0x5;
  reverse[11]= 0xd;
  reverse[12]= 0x3;
  reverse[13]= 0xb;
  reverse[14]= 0x7;
  reverse[15]= 0xf;

  send();
}

void AC_Elextrolux_Control::turn_off() {
  _status = false;
  send();
}

void AC_Elextrolux_Control::turn_on() {
  _status = true;
  send();
}

void AC_Elextrolux_Control::increase_temperature() {
  set_temperature(temperature()+1);
}

void AC_Elextrolux_Control::decrease_temperature() {
  set_temperature(temperature()-1);
}

void AC_Elextrolux_Control::set_temperature(int temp){
  if(temp < 16) temp = 16;
  if(temp > 30) temp = 30;
  _temperature = temp - 16;
  send();
}

void AC_Elextrolux_Control::set_modo(MODO mod){
  _modo = mod;
  send();
```

```
}

void AC_Elextrolux_Control::set_fan_speed(int level_speed){
  if(level_speed < 0) level_speed = 0;
  if(level_speed > 3) level_speed = 3;
  _fan_speed = level_speed;
  send();
}

void AC_Elextrolux_Control::set_turbo(bool t) {
  _turbo = t;
  send();
}

void AC_Elextrolux_Control::toggle_turbo() {
  _turbo = !turbo();
  send();
}

void AC_Elextrolux_Control::set_self_clean(bool t) {
  _self_clean = t;
  send();
}

void AC_Elextrolux_Control::toggle_self_clean() {
  _self_clean = !self_clean();
  send();
}

void AC_Elextrolux_Control::set_brisa(bool b){
  _brisa = b;
  send();
}

void AC_Elextrolux_Control::toggle_brisa() {
  _brisa = !brisa();
  send();
}

void AC_Elextrolux_Control::put_Leading(void) {
  _timer.enable_pwm();
  Alarm::delay(9000);//9ms
  _timer.disable_pwm();
  Alarm::delay(4500);//4,5ms
}

void AC_Elextrolux_Control::put_0(void) {
  _timer.enable_pwm();
  Alarm::delay(520);//560-30
  _timer.disable_pwm();
  Alarm::delay(560);
}

void AC_Elextrolux_Control::put_1(void) {
  _timer.enable_pwm();
  Alarm::delay(520);//560-30
  _timer.disable_pwm();
  Alarm::delay(1690);//1690
}
```

```
void AC_Elextrolux_Control::put_either(bool either) {
  if (either)
    put_1();
  else
    put_0();
}

void AC_Elextrolux_Control::put_Tail(void){
  _timer.enable_pwm();
  Alarm::delay(560);
  _timer.disable_pwm();
  _pin.clear();
}

char AC_Elextrolux_Control::checksum(void){
  return (char(reverse[0xF & (0b1010 + reverse[(_frame1 >> (32-8)) & 0xF] + reverse
      [(_frame1 >> (24-8)) & 0xF])]));
}

void AC_Elextrolux_Control::build_commands(){
  //Frame 1
  _frame1 = 0; //reset
  //modo
  _frame1 = _modo;

  //status
  _frame1 = _frame1 << 2;
  _frame1 |= _status;

  //_fan_speed
  _frame1 = _frame1 << 4;
  _frame1 |= 0b1100 & reverse[ 0b0011 & _fan_speed];

  _frame1 |= _eco;

  //temperature
  _frame1 = _frame1 << 4;
  _frame1 |= 0xF & reverse[0xF &  _temperature];

  _frame1 = _frame1 << 8;   // configuracao do relogio, nao usado

  //sexto conj. de bits
  _frame1 = _frame1 << 1;
  _frame1 |= _turbo;

  _frame1 = _frame1 << 1;
  _frame1 |= 1;

  _frame1 = _frame1 << 1;
  _frame1 |= _status;

  _frame1 = _frame1 << 1;
  _frame1 |= _self_clean;

  _frame1 = _frame1 << 4;

  //_frame1 |= 0b10100101;  // constante

//CMD 2
  _frame2 = 0;
```

```
    _frame2 = _brisa << 11;
    // terceiro conj.
    _frame2 |= 0b1000;
    _frame2 |= ((_status||(_modo==VENT))?0:1) << 2;

    _frame2 = _frame2 << 20;

    _frame2 |= (0xF & checksum());
  }

  void AC_Elextrolux_Control::send(void) {
    build_commands();

    put_Leading();

  //put_frame_1
    for (int i = 27; i >= 0; i--) {
      put_either(_frame1 & (1 << i));
    }
    //constant
    for (int i = 7; i >= 0; i--) {
      put_either(0b10100101 & (1 << i));
    }

    //Alarm::delay(19500);
    Alarm::delay(17600);

  //put_frame_2
    for (int i = 31; i >= 0; i--) {
      put_either(_frame2 & (1 << i));
    }

    put_Tail();
    //Delay 19,5 ms
    //Alarm::delay(19500);
    Alarm::delay(17600);
  }

__END_SYS
```

Código D.4 – Alteração no *timer.h* para gerar um pwm de 38kHz.

```
      MC13224V_Timer(const Hertz& f, Handler* handler, const INSTANCE instance, const
          Mode m):
      instance(instance), offset(instance * 0x20) {
      if (m==NORMAL){ //Rotina padrao
        ctrl(0x20); // untill CNTR reach COMP1
        sctrl(0x0); // reset SCTRL
        load(0x0); // reset LOAD
        frequency(f);
        csctrl(0x41);
        ctrl( (D128<<9) | ctrl() );
        cntr(0x0);
        ctrl( (0x1<<13) | ctrl() );
        handlers[instance] = handler;
        IC::int_vector(IC::IRQ_TIMER, handler_wrapper);
        enable();
      } else if(m==PWM){     //rotina PWM
        load(0x0); // reset LOAD
        cntr(0x0); //reset contador
```

```
        sctrl(0x5); //habilitar saida
        csctrl(0x70);
        comp1(316); // 24 GHz / 38 kHz = 316
        cmpld1(316);
        comp2(315);
        cmpld2(315);
        ctrl(0x3024); //configura timer
        handlers[instance] = 0;
        disable_pwm();
    } else if(m==EXT){     //rotina externa
        ctrl(0x2420); /* Set up mode */ //timer 2, timer 1 = 0x2220
        sctrl(0x00);
        cntr(0x00); /* Reset counter register */
        load(0x00); /* Reset load register */
        comp1(0x01); /*Contar ate 1*/
        sctrl(0x20); /*Apenas bit 6 = = 1*/
        ctrl(ctrl()|0x01<<13); /* Run counter */
        handlers[instance] = handler;
        IC::int_vector(IC::IRQ_TIMER, handler_wrapper);
        enable();
    }
  }

    void enable_pwm(){
    enbl(enbl() | 0x1<<instance);
  }

  void disable_pwm(){
    enbl(enbl() & ~(0x1<<instance));
  }
```

Código D.5 – Código da Aplicação do EPOS para o Controle do Ar Condicionado.

```
#include <machine.h>
#include <mach/mc13224v/emote2_startup.h>
#include <system/config.h>
#include <utility/ostream.h>
#include <mach/mc13224v/ac_electrolux_control.h>

#define ID_SLAVE 6
#define ID_MASTER 99

#define MSG

__USING_SYS

union float_char{
  float dfloat;
  unsigned char dchar[4];
};

//struct pc->epos
struct Message1_{
  int id_send;
  int id_receive;
  char command;
  int value;
} Message1;

//struct epos->pc
```

```
struct Message2_{
  int id_send;
  int id_receive;
  char state;
  union float_char data[5];
} Message2;

NIC * nic;
NIC::Protocol prot;
NIC::Address src;

OStream cout;
EMote2_Startup * startup;
AC_Elextrolux_Control * ac;

volatile int temperature = 23;

volatile bool data_for_send = false;

///-----MAIN----- //Scheduling_Criteria::Priority
int main() {
  nic = new NIC();
  startup = new EMote2_Startup();

  cout << "\nSlave - Controle Ar condicionado  ID: " << ID_SLAVE << endl;

  ac = new AC_Elextrolux_Control();

  Message2.id_send = ID_SLAVE;
  Message2.id_receive = ID_MASTER;

  cout << "OK" << endl;

  while(true){
    int s = -1, c1 = 0;
    while(s<0){
      s = nic->receive(&src, &prot, &(Message1), sizeof(Message1));
      if(++c1 > 3)
        break;
    }

    if(s>=0){ // s>=0 significa que recebeu alguma mensagem

      #ifdef MSG
        cout << "___\n";
        cout << "IDsend: "   << (int) Message1.id_send << endl;
        cout << "IDrec: "   << (int) Message1.id_receive << endl;
        cout << "Command: "  << Message1.command << endl;
        cout << "Value: " << Message1.value << endl;
      #endif // MSG

      if(Message1.id_receive == ID_SLAVE){
        switch (Message1.command){
          case 'g':
            #ifdef MSG
              cout << "\nstatus:" << ac->status() << endl;
              cout << "modo:" << ac->modo() << endl;
              cout << "fan_speed: " << ac->fan_speed() << endl;
              cout << "temp: " << ac->temperature() << endl;
              cout << "turbo: " << ac->turbo() << endl;
```

```
      cout << "self_clean " << ac->self_clean() << endl;
      cout << "brisa: " << ac->brisa() << endl;

      cout << "frame1: " << bin << ac->frame1() << endl;
      cout << "frame2: " << bin << ac->frame2() << endl;
    #endif //MSG

    Message2.state = (ac->status())?'l':'d';
    Message2.data[0].dfloat = ac->modo();
    Message2.data[1].dfloat = ac->fan_speed();
    Message2.data[2].dfloat = ac->temperature();
    Message2.data[3].dchar[0] = ac->turbo();
    Message2.data[3].dchar[1] = ac->self_clean();
    Message2.data[3].dchar[2] = ac->brisa();

    data_for_send = true;
    break;
  case 'a':
    ac->turn_on();
    cout << "Turn ON" << endl;
    break;
  case 'b':
    ac->turn_off();
    cout << "Turn OFF" << endl;
    break;
  case 'c':
    ac->increase_temperature();
    cout << "Increase Temp " << ac->temperature() << endl;
    break;
  case 'd':
    ac->decrease_temperature();
    cout << "Decrease Temp " << ac->temperature() << endl;
    break;
  case 'e':
    ac->toggle_turbo();
    cout << "Turbo " << ac->turbo() << endl;
    break;
  case 'f':
    ac->set_fan_speed(Message1.value);
    cout << "Set Speed " <<
    ac->fan_speed() << endl;
    break;
  case 's':
    ac->set_temperature(Message1.value);
    cout << "Set Temp " << ac->temperature() << endl;
    break;
  case 'i':
    ac->toggle_self_clean();
    cout << "Self Clean " << ac->self_clean() << endl;
    break;
  case 'j':
    ac->toggle_brisa();
    cout << "Brisa " << ac->brisa() ;
    break;
  case 'm':
    switch (Message1.value){
      case 1:
        ac->set_modo(AC_Elextrolux_Control::AUTO);
        break;
      case 2:
```

```
                    ac->set_modo(AC_Elextrolux_Control::DESU);
                    break;
                  case 3:
                    ac->set_modo(AC_Elextrolux_Control::RESF);
                    break;
                  case 4:
                    ac->set_modo(AC_Elextrolux_Control::VENT);
                    break;
                }
                cout << "Modo " << ac->modo() << endl;
                break;
            }
          }
        }

        if(data_for_send){
          int r = 0;
          while (r != 11){
            r = (nic->send(NIC::BROADCAST, (NIC::Protocol) 1, &(Message2), sizeof(
                Message2)));
          }

          if(r==11)
            data_for_send = false;
        }
      }
      return 0;
    }
```

APÊNDICE E – Cálculos de incertezas de medição da tensão e corrente eficazes

Incertezas da Tensão Eficaz

Para a segunda medição da Tabela 4, uma lâmpada incandescente de $60W$, obteve-se uma medição de $217,52V_{rms}$ de tensão eficaz, a sua incerteza de medição é obtida através do seguinte cálculo:

$$\begin{aligned}\sigma_{V_b} &= \sqrt{\left(\frac{R_6}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2(\sigma_{R_9})^2+\left(\frac{R_9}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2(\sigma_{R_6})^2} \\ &= \sqrt{\left(\frac{27k}{(27k+1k)^2}\frac{9}{220}217,52\right)^2(0,01.1k)^2+\left(\frac{1k}{(27k+1k)^2}\frac{9}{220}217,52\right)^2(0,05.27k)^2} \\ &= 0,0156 \\ &= 15,6mV_{rms}\end{aligned} \tag{E.1}$$

$$V_b = \frac{R_9}{R_6+R_9}\left(\frac{9}{220}\right)V_a = \frac{1k}{27k+1k}\left(\frac{9}{220}\right)217,52 = 317,8mV_{rms} \tag{E.2}$$

$$\frac{\sigma_{V_b}}{V_b} = \frac{15,6mV_{rms}}{317,8mV_{rms}} = 0,0490 = 4,9\% \tag{E.3}$$

Para a terceira medição da Tabela 4, uma televisão de $45W$, obteve-se uma medição de $211,82V_{rms}$ de tensão eficaz, a sua incerteza de medição é obtida através do seguinte cálculo:

$$\begin{aligned}\sigma_{V_b} &= \sqrt{\left(\frac{R_6}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2(\sigma_{R_9})^2+\left(\frac{R_9}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2(\sigma_{R_6})^2} \\ &= \sqrt{\left(\frac{27k}{(27k+1k)^2}\frac{9}{220}211,82\right)^2(0,01.1k)^2+\left(\frac{1k}{(27k+1k)^2}\frac{9}{220}211,82\right)^2(0,05.27k)^2} \\ &= 0,0152 \\ &= 15,2mV_{rms}\end{aligned} \tag{E.4}$$

$$V_b = \frac{R_9}{R_6+R_9}\left(\frac{9}{220}\right)V_a = \frac{1k}{27k+1k}\left(\frac{9}{220}\right)211,82 = 309,4mV_{rms} \tag{E.5}$$

$$\frac{\sigma_{V_b}}{V_b} = \frac{15,2mV_{rms}}{309,4mV_{rms}} = 0,0491 = 4,91\% \tag{E.6}$$

Para a quarta medição da Tabela 4, uma sanduicheira de $700W$, obteve-se uma medição de $211,85V_{rms}$ de tensão eficaz, a sua incerteza de medição é obtida através do seguinte cálculo:

$$\begin{aligned}
\sigma_{V_b} &= \sqrt{\left(\frac{R_6}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2(\sigma_{R_9})^2 + \left(\frac{R_9}{(R_6+R_9)^2}\left(\frac{9}{220}\right)V_a\right)^2(\sigma_{R_6})^2} \\
&= \sqrt{\left(\frac{27k}{(27k+1k)^2}\frac{9}{220}211,85\right)^2(0,01.1k)^2 + \left(\frac{1k}{(27k+1k)^2}\frac{9}{220}211,85\right)^2(0,05.27k)^2} \\
&= 0,0152 \\
&= 15,2mV_{rms}
\end{aligned} \tag{E.7}$$

$$V_b = \frac{R_9}{R_6+R_9}\left(\frac{9}{220}\right)V_a = \frac{1k}{27k+1k}\left(\frac{9}{220}\right)218,36 = 309,5mV_{rms} \tag{E.8}$$

$$\frac{\sigma_{V_b}}{V_b} = \frac{15,2mV_{rms}}{309,5mV_{rms}} = 0,0491 = 4,91\% \tag{E.9}$$

Incertezas da Corrente Eficaz

Portanto, para a segunda medição da Tabela 4, uma lâmpada incandescente de $60W$ obteve-se uma medição de $0,272A_{rms}$ de corrente eficaz, a incerteza de medição da tensão V_i é obtida através do seguinte cálculo:

$$\begin{aligned}
\sigma_{V_I} &= \sqrt{\left(\frac{I_c}{1000}\right)^2(\sigma_{R_b})^2 + \left(\frac{R_b}{1000}\right)^2(\sigma_{I_c})^2} \\
&= \sqrt{\left(\frac{0,272}{1000}\right)^2(0,01.5,6)^2 + \left(\frac{5,6}{1000}\right)^2(0,01.0,272)^2} \\
&= 0,0000215 \\
&= 0,0215mV_{rms}
\end{aligned} \tag{E.10}$$

$$V_i = \frac{R_b.I_c}{1000} = \frac{5,6.0,183}{1000} = 1,523mV_{rms} \tag{E.11}$$

$$\frac{\sigma_{V_I}}{V_i} = \frac{0,0215mV_{rms}}{1,523mV_{rms}} = 0,0141 = 1,41\% \tag{E.12}$$

Portanto, para a terceira medição da Tabela 4, com uma televisão de $45W$, obteve-se uma medição de $0,217A_{rms}$ de corrente eficaz, a incerteza de medição da tensão V_i é obtida através do seguinte cálculo:

$$\begin{aligned}\sigma_{V_I} &= \sqrt{\left(\frac{I_c}{1000}\right)^2(\sigma_{R_b})^2 + \left(\frac{R_b}{1000}\right)^2(\sigma_{I_c})^2} \\ &= \sqrt{\left(\frac{0,217}{1000}\right)^2(0,01.5,6)^2 + \left(\frac{5,6}{1000}\right)^2(0,01.0,217)^2} \\ &= 0,0000171 \\ &= 0,0171mV_{rms}\end{aligned} \tag{E.13}$$

$$V_i = \frac{R_b.I_c}{1000} = \frac{5,6.0,217}{1000} = 1,215mV_{rms} \tag{E.14}$$

$$\frac{\sigma_{V_I}}{V_i} = \frac{0,0171mV_{rms}}{1,215mV_{rms}} = 0,014 = 1,40\% \tag{E.15}$$

Portanto, para a quarta medição da Tabela 4, uma sanduicheira de $700W$, obteve-se uma medição de $2,953A_{rms}$ de corrente eficaz, a incerteza de medição da tensão V_i é obtida através do seguinte cálculo:

$$\begin{aligned}\sigma_{V_I} &= \sqrt{\left(\frac{I_c}{1000}\right)^2(\sigma_{R_b})^2 + \left(\frac{R_b}{1000}\right)^2(\sigma_{I_c})^2} \\ &= \sqrt{\left(\frac{2,953}{1000}\right)^2(0,01.5,6)^2 + \left(\frac{5,6}{1000}\right)^2(0,01.2,953)^2} \\ &= 0,0002349 \\ &= 0,2338mV_{rms}\end{aligned} \tag{E.16}$$

$$V_i = \frac{R_b.I_c}{1000} = \frac{5,6.2,953}{1000} = 16,536mV_{rms} \tag{E.17}$$

$$\frac{\sigma_{V_I}}{V_i} = \frac{0,2349mV_{rms}}{16,536mV_{rms}} = 0,0142 = 1,42\% \tag{E.18}$$