UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUCAS LONGO

INTERNET DAS COISAS: USO DE SENSORES E ATUADORES NA AUTOMAÇÃO DE UM PROTÓTIPO RESIDENCIAL

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2015

LUCAS LONGO

INTERNET DAS COISAS: USO DE SENSORES E ATUADORES NA AUTOMAÇÃO DE UM PROTÓTIPO RESIDENCIAL

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Engenharia de Computação, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO
2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Às 13 horas e 30 minutos do dia 27 de novembro de 2015, na sala V107, da Universidade Tecnológica Federal dão Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Vinicius Pegorini (orientador), Fábio Favarim e Robison Cris Brito para avaliar o trabalho de conclusão de curso com o título **Internet das coisas: uso de sensores e atuadores na automação de um protótipo residencial**, do aluno **Lucas Longo**, matrícula 01065157, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Vinicius Pegorini
Orientador (UTFPR)

_____
Robison Cris Brito
(UTFPR)

_____
Fábio Favarim
(UTFPR)

_____
Beatriz Terezinha Borsoi
Coordenador de TCC

_____
Marco Antonio de Castro Barbosa
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

LONGO, Lucas. Internet das coisas: uso de sensores e atuadores na automação de um protótipo residencial. 100f. Monografia de Trabalho de Conclusão de Curso – Engenharia de Computação – Universidade Tecnológica Federal do Paraná. Pato Branco, 2015.

Atualmente muito se fala sobre um conceito inovador que basicamente diz que todos os objetos utilizados no cotidiano das pessoas estarão conectados à Internet e terão informações e dados acessados diretamente por meio de um endereço IP único atribuído a esses objetos, cada objeto será capaz de obter informações do ambiente e interagir de acordo com os dados obtidos. Esse conceito leva o nome de Internet das Coisas ou *Internet of Things*, em inglês. São objetos e sistemas que utilizam sensores, microprocessamento e atuadores para realizar interação com o ambiente. Nesse trabalho será abordado um sistema para automação residencial, neste caso a residência é vista como um objeto de Internet das Coisas, a conexão da residência com a Internet será feita por meio de um servidor Web instalado e configurado dentro da sua central de comando em um Raspberry Pi. O usuário é capaz de escolher as ações que deseja realizar ou mesmo colocar em modo automático. Essas ações são enviadas ao microcontrolador, responsável por ativar os atuadores presentes no sistema, por meio de comunicação via radiofrequência. Além de ser responsável pelos atuadores o microcontrolador também faz a leitura dos sensores utilizados para o modo automático dos atuadores, os dados obtidos também são enviados a central por meio de comunicação via radiofrequência, esses dados são exibidos para o usuário para que possa tomar decisões. Além disso, a interface de acesso do usuário é responsiva, ou seja, a interface é capaz de adaptar-se a qualquer dispositivo que tenha acesso à Internet, isso faz com que o usuário possa acessar a interface de controle de sua residência de qualquer lugar desde que esteja conectado à Internet.

Palavras-chave: Internet das coisas. Automação residencial. Radiofrequência.

ABSTRACT

LONGO, Lucas. Internet of Things: use of sensors and actuators in automation of a residential prototype. 100f. Monografia de Trabalho de Conclusão de Curso – Engenharia de Computação – Universidade Tecnológica Federal do Paraná. Pato Branco, 2015.

Nowadays much is said about an innovative concept that basically says that all objects used in daily life will be connected to the Internet and the information provided by these objects will be accessed directly via a single IP, each object will be able to obtain information from the environment and interact according to the obtained data. This concept takes the name Internet of Things. These objects and systems uses sensors and actuators to make interaction with the environment. In this work a system for home automation will be developed, in this study the home is seen as an Internet of Things object. The connection of the Internet will be made using a Web server installed in a Raspberry Pi. The user is able to choose the actions of the actuators or even put in automatic mode. These actions are sent to the microcontroller, responsible for activating the actuators in the system through communication via radio frequency. Besides being responsible for actuators microcontroller also reads the sensors used to automatic actuators, the data is also sent to the central through communication via radio frequency, this data is displayed for the user so he can make decisions. In addition, user access interface is responsive, which means that the interface is able to adapt to any device that have Internet access. The user can access the control interface of his home from anywhere with Internet connection.

Keywords: Internet of Things. Residential automation. Radio frequency.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS E ABREVIATURAS

ACK	*Acknowledge*
ADC	*Analog-Digital Conversor (Conversor analógico digital)*
AJAX	*Asynchronous Javascript and XML*
AmI	*Ambient Intelligence*
CRC	*Cyclic Redundancy Check*
EDI	*Eletronic Data Interchange*
FIFO	*First-In First-Out*
GIS	*Geographic Information System*
GND	*Ground (Terra)*
GPS	*Global Position System*
HTML	*HyperText Markup Language*
IoT	*Internet of Things*
ISTAG	*European Commission's IST Advisory Group*
ITU	*International Telecommunications Union*
JIT	*Just-in-Time*
JSON	*JavaScript Object Notation*
PDA	*Personal Digital Assistant*
PHP	*PHP Hipertext Processor*
PID	*Packet Identifier (Identificador do Pacote)*
PTX	*Primary Transmissor*
PRX	*Primary Receiver*
Ppm	Partes por milhão
PWM	*Pulse Width Modulation*
QR-code	*Quick Response Code*
REST	*Representational state transfer*
RFID	*Radio Frequency Identification*
RX	Receptor
SCLK	*Source Clock*
SSH	*Secure Shell*
SPI	*Serial Peripheral Interface*
TI	Tecnologia de Informação
TX	Transmissor
UART	*Universal Asynchronous Receiver/Transmitter*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais fornecendo uma visão geral do assunto principal de pesquisa, que é Internet das Coisas, e no mesmo inserindo a proposta do trabalho. Em seguida são apresentados os objetivos do trabalho e a justificativa. Por fim está a organização do texto por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

Para uma melhor compreensão do conceito de Internet das Coisas (*Internet of Things* (IoT)), inicialmente é necessário entender o conceito de ambientes inteligentes e de computação pervasiva. Isso porque esses conceitos estão envolvidos na conceituação de Internet das coisas. Ambientes inteligentes e computação pervasiva podem ser tornados possíveis por meio de dispositivos e conceitos relacionadas à Internet das coisas.

O termo Ambiente Inteligente - do inglês, *Ambient Intelligence* (AmI) - foi introduzido em 2001 pela ISTAG (*European Commission's IST Advisory Group*). Esse termo refere-se a um método técnico que visa caracterizar uma situação existente por meio da interação entre informações existentes em um ambiente. Essas informações estão relacionadas a dispositivos presentes no ambiente. Nesse contexto, a expressão da informação e do conhecimento está relacionada à representação do mundo real. E, a aplicação da expressão do conhecimento refere-se a um ambiente computacional inteligente, no sentido de interação do usuário com os elementos que fazem parte desse ambiente.

O conceito de AmI fornece a visão de uma sociedade informatizada, na qual as pessoas estão cercadas de dispositivos inteligentes que estão nos mais diversos tipos de objetos, sendo capazes de reconhecer e responder a diferentes indivíduos de forma transparente, discreta e muitas vezes invisível (KO; RAMOS, 2010; ISTAG, 2010). Dessa maneira, um AmI é um ambiente digital capaz de interagir e auxiliar os indivíduos no seu dia-a-dia. Os ambientes inteligentes estão relacionados aos dispositivos do usuário, como *Personal Digital Assistant* (PDA), *smartphone* e serviços que possam obter informações por meio da rede, em servidores próximos ao usuário.

Os ambientes inteligentes devem ser pró-ativos, agindo de maneira autônoma e antecipando o que o usuário necessita. Em alguns casos podem até mesmo substituir os usuários, ou seja, é um ambiente que por meio de sensores e outras fontes de informação

(*web*, mídias, contatos sociais, notícias, etc.) é capaz de tomar decisões automáticas e atuar, utilizando uma Inteligência Artificial e o histórico de informações, visando tomar a melhor decisão.

Os ambientes inteligentes estão vinculados à computação pervasiva e a Internet das coisas porque eles permitem interação dos usuários com o ambiente e facilitam essa interação por meio de dispositivos conectados à Internet.

Um conceito de computação pervasiva, introduzido em 1991 por Mark Weiser, diz que as tecnologias mais profundas são as que desaparecem e que estão tão entrelaçadas ao cotidiano das pessoas e que passam despercebidas (WEISER, 1993). Para tornar essa visão realidade é necessária a criação de ambientes saturados de capacidade computacional e de comunicação, mas que estejam graciosamente integrados com o usuário (DHINGRA; ARORA, 2008). Com o patamar tecnológico existente atualmente é possível o desenvolvimento de aplicações utilizando esse conceito.

Com esses dois conceitos caracterizados é possível introduzir a ideia de Internet das Coisas. Wu et al. (2010) expressam que não há uma definição amplamente aceita para IoT e que o termo foi usado pela primeira vez por Kevin Ashton em 1998. IoT descreve uma arquitetura de serviço de informação global e emergente baseada na Internet (WEBER, 2009).

A IoT é uma revolução tecnológica que representa o futuro da computação e comunicação e seu desenvolvimento depende de inovação técnica em um número de campos importantes, indo de sensores de rede sem fio a nanotecnologia (ITU, 2005). Tecnicamente a arquitetura da IoT é baseada em ferramentas de comunicação, principalmente itens que possuem *tags Radio Frequency Identification* (RFID) vinculadas. Além disso, são utilizadas tecnologias como *Global Position System* (GPS), *Eletronic Data Interchange* (EDI), *Geographic Information System* (GIS) e *Just-in-Time* (JIT), dentre outras.

A ideia da IoT consiste em objetos que utilizando essas tecnologias e de um endereçamento único podem cooperar uns com os outros a fim de alcançar um objetivo comum. E devem garantir o pleno funcionamento e a interoperabilidade dos dispositivos envolvidos, proporcionando a eles um alto grau de inteligência e autonomia, enquanto asseguram confiabilidade, privacidade e segurança.

Para Wu et al (2010) o objetivo da IoT é facilitar a troca de informação sobre, entre outras coisas, bens em uma cadeia se suprimento global, isto é, a infraestrutura de Tecnologia de Informação (TI) deve prover informações sobre os itens (“coisas”) que circulam por essa cadeia de uma forma segura e confiável. E ampliando o escopo de aplicação inicial, a IoT pode também servir com uma espinha dorsal (*backbone*) para a computação pervasiva,

permitindo que ambientes inteligentes reconheçam e identifiquem objetos e obtenham informações da Internet para facilitar suas funcionalidades adaptativas (WEBER, 2009).

Considerando o contexto apresentado de importância da IoT, neste trabalho é utilizado o conceito de IoT para o desenvolvimento de um protótipo de um controle de dispositivos em uma residência via *web*. Sensores e atuadores ligados a um controle principal podem interagir com o ambiente e tomar medidas autônomas ao mesmo tempo em que permitirão ao usuário visualizar os dados obtidos com os leitores e assim agendar ações ou tomar ações imediatas. Essas ações de interação e a visualização dos dados obtidos dos sensores serão realizadas por meio de uma página *web*. Assim, um servidor *web* será implementado em um Raspberry Pi para que a interação do usuário possa ocorrer por meio de um navegador *web*.

1.2 OBJETIVOS

A seguir são apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 OBJETIVO GERAL

Implementar sensores e atuadores que possam coletar dados que serão usados pelo sistema para tomar decisões de maneira autônoma em um protótipo de residência com base no conceito de Internet das Coisas utilizando Raspberry Pi como ambiente computacional.

1.2.2 OBJETIVOS ESPECÍFICOS

- Desenvolver um servidor *web* que permita o gerenciamento de sensores e atuadores, com uma interface responsiva podendo ser utilizado em diversos dispositivos com acesso à Internet, facilitando o gerenciamento do sistema.
- Permitir a execução autônoma dos atuadores envolvidos no protótipo e permitir a atuação em ações predefinidas e configuradas.
- Apresentar o uso de Raspberry Pi na implementação do conceito de Internet das Coisas.

1.3 JUSTIFICATIVA

Em um futuro não muito distante, os objetos do dia-a-dia estarão conectados à Internet podendo ser acessados e controlados de qualquer lugar, a qualquer hora, de qualquer dispositivo que seja capaz de conectar-se à Internet. Esses objetos também serão capazes de realizar ações de maneira autônoma com o objetivo de suprir necessidades do usuário sem que ele intervenha.

O uso de sensores como, por exemplo, os de temperatura, umidade, luminosidade, presença e gases juntamente com atuadores como relés, válvulas solenóides, transmissores infravermelhos, *tags* RFID, *speakers* em uma residência tem o objetivo de monitoramento e atuação automática ou comandada pelo usuário.

Uma maneira de facilitar o gerenciamento dos sensores e acionar os atuadores desses ambientes é por meio de um servidor *web* que utiliza *Representational State Transfer* (REST) para o envio e recebimento de mensagens por meio do protocolo HTTP utilizando-se de arquivos no formato *JavaScript Object Notation* (JSON). Facilitando, assim, a troca de mensagens entre cliente e servidor.

REST é um estilo de arquitetura de software para sistemas distribuídos, que trabalha com identificadores de recursos (*Uniform Resource Identifier* (URI)) que são *Uniform Resource Locator* (URL) com identificadores e parâmetros. A arquitetura REST é baseada em recursos e verbos, cada recurso é referenciado como um URI (um endereço *web*, por exemplo), as ações desses recursos são criar, ler, alterar e excluir que são mapeadas para as ações HTTP de POST, GET, PUT e DELETE (ESTELLER-CURTO et al., 2012).

1.3 ORGANIZACÃO DO TEXTO

O restante deste texto está organizado em capítulos. No Capítulo 2 é apresentado o referencial teórico que versa sobre computação pervasiva, ambientes inteligentes e Internet das Coisas. No final do capítulo são apresentados trabalhos semelhantes ao proposto, representados o estado da arte.

No Capítulo 3 são apresentados os materiais e o método utilizados para desenvolver o trabalho. E o Capítulo 4 apresenta os resultados obtidos. O Capítulo 5 apresenta as conclusões obtidas com o desenvolvimento deste trabalho.

2 REFERENCIAL TEÓRICO

A seguir são apresentados conceitos que fornecem suporte a proposta deste trabalho que está baseada em Internet das Coisas. Esses conceitos estão relacionados à computação pervasiva, ambientes inteligentes, sensores, atuadores, sistemas embarcados e Internet das Coisas. O capítulo é finalizado com a apresentação de trabalhos semelhantes ao sendo proposto.

2.1 COMPUTAÇÃO PERVASIVA

Em 1991 Mark Weiser, apresentou sua visão para o que considerou computação ubíqua que agora é chamada de computação pervasiva. Ele escreveu que as tecnologias mais profundas são as que desaparecem. Isso porque elas se entrelaçam no tecido do cotidiano até que se tornam indistinguíveis (WEISER, 1993).

Dhingra e Arora (2008) definem computação pervasiva como: uma pessoa, muitos computadores. Computadores incorporados ao ambiente permitindo à tecnologia residir como *background* para interatividade das pessoas com os objetos e acontecimentos do ambiente. E, ainda, que esses objetos interajam entre si e com atuadores que realizam ações nesses ambientes.

A computação pervasiva é uma tendência crescente no mercado de incorporação de microcontroladores em objetos do cotidiano, a palavra pervasiva remete a “existente em todos os lugares”. Os dispositivos de computação pervasiva são completamente conectados e tem acesso disponível de maneira permanente.

O ambiente de computação pervasiva envolve quatro grandes áreas convergentes (DHINGRA, ARORA, 2008): dispositivos (computação), redes (comunicação), *middleware* (interface de usuário) e aplicações.

Os dispositivos de computação pervasiva podem ter formas e tamanhos diferentes, sendo capazes de se comunicarem uns com os outros e atuarem de maneira inteligente. Os dispositivos inteligentes podem ser separados em três categorias: sensores, processadores e atuadores (DHINGRA; ARORA, 2008). Os sensores são capazes de detectar mudanças no ambiente, de comportamento dos usuários e de capturar comandos humanos. Os processadores interpretam e analisam os dados de entrada. Os atuadores são dispositivos de saída que respondem à informação processada por meios mecânicos ou eletrônicos.

As redes pervasivas realizam a conexão entre dispositivos eletrônicos independentes e redes maiores. Essa conexão pode ser feita por meio de ligação por cabos ou tecnologias de rede sem fio. O desenvolvimento efetivo de sistemas de computação pervasiva depende do nível de interoperabilidade dos dispositivos envolvidos (DHINGRA, ARORA, 2008).

2.2 AMBIENTES INTELIGENTES

Ambiente inteligente é um termo que denota áreas que são capazes de estar cientes da presença de pessoas e como reagir na presença das mesmas (CHLOUBA, 2009). As funções do sistema devem ser independentes dos seres humanos e servir como um guia através da área inteligente.

O conceito de AmI proporciona uma visão de uma sociedade informatizada na qual as pessoas estão cercadas de dispositivos inteligentes que estão integrados nos mais diferentes tipos de objetos do cotidiano. As pessoas estão cercadas por dispositivos de computação, além dos tradicionais de computadores pessoais, *smarthphones*, *tablets*, GPS, há sensores como as *tags* RFID, os infravermelho de movimento e os de identificação por biometria. Esses dispositivos são capazes de reconhecer e responder a presença de diferentes indivíduos e atuar de maneira invisível (KO; RAMOS, 2010).

O paradigma de AmI representa uma visão de futuro da computação inteligente na qual os ambientes fornecem suporte, no sentido de auxílio e apoio, às pessoas que os habitam (AARTS; RUYTER, 2009). Nesse novo paradigma de computação as entradas e saídas de mídia convencionais deixam de existir, no lugar das mesmas surgem sensores e processadores que estarão integrados nos objetos do dia-a-dia, trabalhando juntos em harmonia para fornecer auxílio aos habitantes (SADRI, 2011).

No contexto da Figura 1, os dados e as informações não são apenas coletados de sensores automáticos, eles também podem ser adquiridos de outras fontes como contatos sociais, *web*, notícias, mídias, etc. Além disso as ações automáticas são dirigidas também para os outros atuadores do AmI (outras pessoas ou dispositivos autônomos) e para as outras fontes (por exemplo, a atualização de informações). Os eventos externos são separados em outros atuadores e eventos inesperados (por exemplo, uma tempestade).

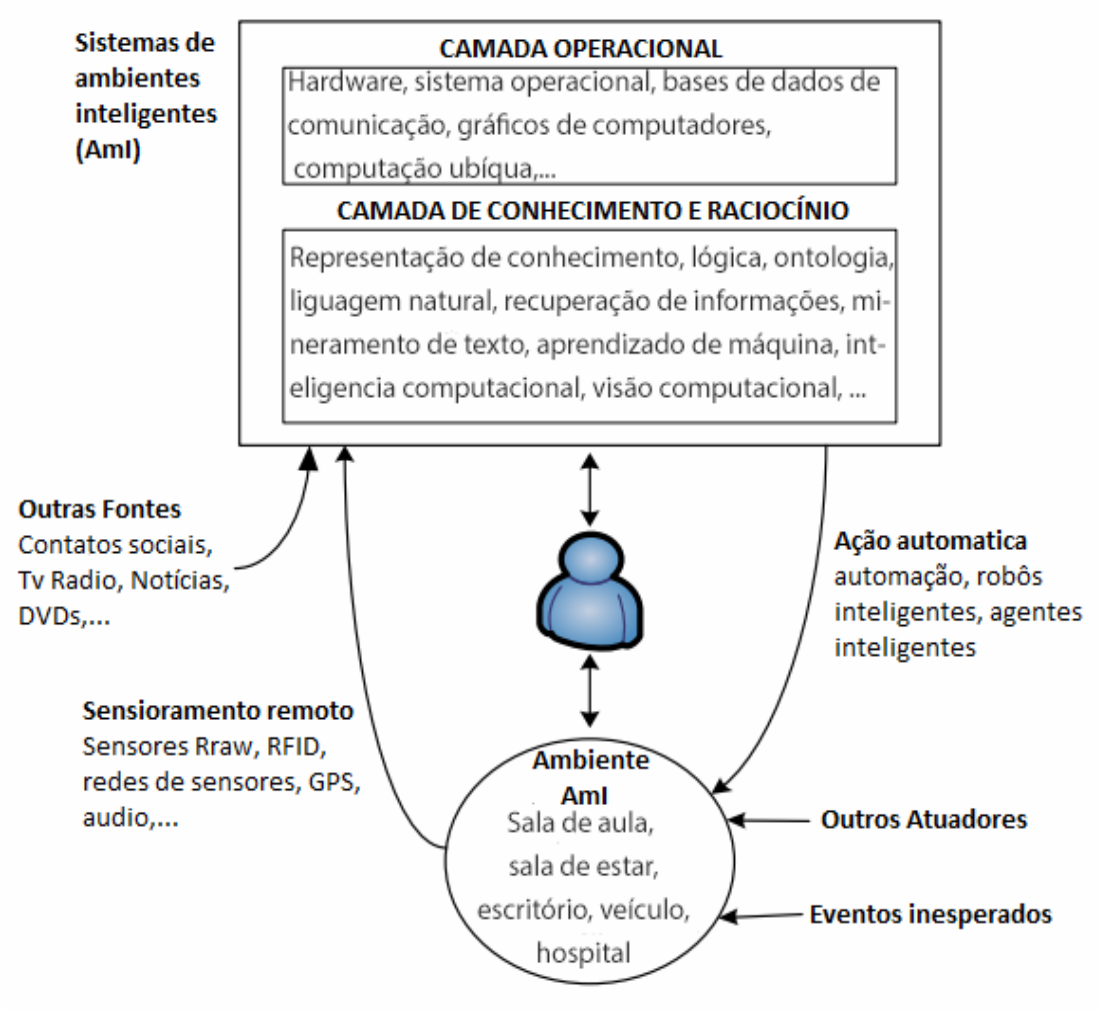

Figura 1 - A visão de Ambientes Inteligentes da perspectiva da Inteligência Artificial
Fonte: traduzido de Ko e Ramos (2010, p. 747).

Considerando o uso de técnicas de Inteligência Artificial, AmI visa fornecer uma interpretação das informações contextuais obtidas de sensores que podem estar incorporados aos do ambiente e adaptar o ambiente para as necessidades do usuário de forma adaptativa e antecipada. Um sistema AmI é identificado por várias características (ACAMPORA et al., 2013), apresentadas a seguir. Apesar da transparência e ubiquidade amplamente atribuída a esses ambientes, uma característica importante é o aspecto de inteligência desses ambientes:

a) Ciência do contexto – explorar informações da situação e do contexto;
b) Personalizado – ambiente personalizado e customizado de acordo com as necessidades de cada indivíduo;
c) Antecipatório – antecipar as necessidades de um indivíduo sem a mediação consciente do mesmo;
d) Adaptativo – adaptável as mudanças de necessidade do indivíduo;

e) Ubíquo – está incorporado e integrado aos ambientes do dia-a-dia;
f) Transparência – situa-se no *background* do dia-a-dia das pessoas de forma não intrusiva

2.3 SENSORES E ATUADORES

Sensores são dispositivos capazes de mensurar grandezas físicas do meio e transformar em um sinal elétrico que podem então ser processados e interpretados por outros equipamentos como por exemplo microcontroladores e computadores, pode se dizer que são componentes capazes de permitir um equipamento eletrônico a interagir e sentir o mundo.

Segundo Borges e Dores (2010), sensores que operam de forma direta, ou seja, transformando uma forma de energia em outra são chamados de transdutores. Já sensores que atuam de forma indireta alteram suas propriedades sob ação da grandeza medida de forma que essa alteração ocorra de forma proporcional, um bom exemplo são os fotoresistores que alteram sua resistência de acordo com a quantidade de luz do ambiente.

Os atuadores são considerados também transdutores mas fazem o caminho contrário aos sensores, eles transformam sinais elétricos em grandezas físicas sendo capazes de realizar ações que modificam o ambiente em que estão inseridos.

Brugnari e Maestrelli (2010) dizem que atuadores atendem a comandos que podem ser manuais ou automáticos, ou seja, qualquer elemento que realize um comando recebido de outro dispositivo, com base em uma entrada ou critério a ser seguido. Um bom exemplo de atuador é um relé que funciona com pequenas correntes, mas é capaz de ativar ou desativar circuitos externos que possuem correntes elevadas.

2.4 SISTEMAS EMBARCADOS

Os sistemas embarcados podem ser definidos como sistemas de processamento digital que possuem seu software de controle (denominado firmware) armazenado em uma memória presente no circuito eletrônico. São dedicados a uma tarefa específica e interagem com o ambiente por meio de sensores e atuadores, sendo utilizados em aviões, carros, telefones celulares, calculadoras, eletrodomésticos, equipamentos médicos etc. De forma geral, são

empregados na maioria dos equipamentos eletrônicos atuais. (SCHNEIDER DE OLIVEIRA, André; SOUZA DE ANDRADE, Fernando, 2010).

Esses sistemas possuem características que os distinguem dos outros sistemas operacionais convencionais, são elas:

- Funcionalidade única: são desenvolvidos para atender a uma funcionalidade específica enquanto sistemas operacionais convencionais podem ter diversas funcionalidades.

- Restrições de projeto mais rígidas: sistemas embarcados são geralmente desenvolvidos utilizando microcontroladores que possuem recursos muito limitados em comparação com os microprocessadores utilizados por sistemas operacionais convencionais.

- Sistemas reativos em tempo real: sistemas embarcados devem reagir a mudanças no ambiente e fornecer resultados em tempo real.

2.5 INTERNET DAS COISAS

Muitos dos dispositivos existentes nos ambientes de uso das pessoas suportam alguma forma de tecnologia de computação e comunicação. Esses dispositivos são, por exemplo, telefones celulares, sensores, dispositivos de medição, *notebooks, tablets* e *smartphones*. O futuro imediato desses dispositivos é eles interagirem entre si por meio de uma rede como a Internet (ATHREYA; TAGUE, 2013). A interação entre esses dispositivos os permitirá alcançar objetivos comuns (ATZORI; IERA; MORABITO, 2010). Esse ecossistema ou paradigma é chamado Internet das Coisas (*Internet of Things)* e as coisas são os dispositivos que podem realizar processamento, emitir informação para outros dispositivos ou servidores, receber informação de dispositivos ou servidores e atuar a partir das informações recebidas.

Bari, Mani e Berkovich (2013) ressaltam que IoT ainda possui uma definição singular, mas se referem que a ela proverá uma nova dimensão no mundo das Tecnologias de Informação e Comunicação: a conectividade para qualquer coisa. Para esses autores IoT pode ser mais adequadamente referenciada como a Internet das coisas relacionadas. Sendo que coisas são metadados sobre coisas.

A Internet das coisas ocorre quando os objetos passam a fazer parte da Internet, cada objeto é endereçado unicamente e pode ser acessado via Internet e a esses objetos podem ser adicionados serviços e inteligência. E, assim, esses objetos permitem unir o mundo digital

com o mundo real impactando na vida profissional, pessoal e nos relacionamentos sociais da sociedade (COETZEE; EKSTEEN, 2011)

A definição das "coisas" em IoT é muito ampla e inclui uma grande variedade de elementos físicos como *smartphones*, *tablets* e elementos dos ambientes do cotidiano que são equipados com *tags* RFID ou outras que são conectados por um dispositivo de *gateway*. Dentro do conceito de Internet das Coisas a conectividade deve ser a "qualquer hora", em "qualquer lugar" para "qualquer um" em "qualquer hora", em "qualquer lugar" para "qualquer coisa" (ITU, 2005).

No conceito de IoT, coisas possuem identidade e personalidade virtual operando como espaços inteligentes que usam interfaces inteligentes para conectar-se e comunicar-se com o contexto do usuário, do ambiente e social; ou, ainda, como objetos interconectados que tem um papel ativo que pode ser chamada Internet do futuro (INFSO, 2008).

IoT usa uma variedade de dispositivos de identificação de informações sensitivas e equipamentos de processamento de informação, tais como RFID, GPS, GIS, JIT, EDI e outros dispositivos que combinados com a Internet formam uma rede extensa de forma a obter informações e inteligência para a entidade (objetos) (FAN, ZHOU, 2011).

RFID é uma tecnologia que auxilia máquinas ou computadores a identificar objetos, armazenar metadados e controlar dispositivos por meio de ondas de rádio (JIA; FENG; MA, 2010). RFID é um facilitador muito importante para IoT. RFID utiliza ondas de rádio frequência para identificar itens que podem estar conectados (JIA et al., 2012). RFID também provê um meio de rastrear itens em tempo real, fornecendo localização e informações de estado (LEGNER; THIESSE. 2006).

Os dispositivos estão cada vez mais sendo equipados com sensores e atuadores, a combinação de ambos cria um ambiente em que os dispositivos estão conectados à rede e tem a habilidade de "sentir", computar e então agir fazendo parte da Internet. Os objetos físicos estão sendo equipados com RFID ou *Quick Response Code* (QR-codes) os códigos de resposta rápida, que podem ser "sentidos" ou lidos pelos dispositivos. Essa combinação vincula o mundo físico ao mundo digital por meio de dispositivos inteligentes. A *International Telecommunications Union* (ITU) descreve quatro dimensões dentro da Internet das Coisas (COETZEE; EKSTEEN, 2011):

a) Identificação de objetos (*tags* RFID e QR-codes “marcando os objetos”);

b) Sensores e Redes de sensores sem fio ("sentindo as coisas");

c) Sistemas embarcados ("pensando coisas");

d) Nanotecnologia ("diminuindo as coisas");

Exemplos de interações e aplicações de IoT (ATHREYA; TAGUE, 2013):

a) Sensores de temperatura e umidade – para monitoramento de residências;

b) Sensores de monitoramento de sinais vitais – sinais cardíacos, pressão arterial e outros que conectados à pessoa podem capturar dados e enviava-los em tempo real para um especialista, por exemplo, que fará o monitoramento remoto;

c) Sensores de monitoramento de energia – monitoramento de energia em painéis solares visando maximizar a utilização de recursos renováveis de energia;

d) Sensores de monitoração veicular – podem auxiliar departamentos responsáveis por transportes a informar situações de congestionamento, acidentes, perigos por desastres naturais e outros, otimização de rotas e outros;

e) Monitoramento de itens – rastreabilidade de itens em transporte em cadeias de suprimento e indústrias;

Uma arquitetura bastante simples para a IoT é proposta por Wu et al. (2010). Esses autores definem uma arquitetura típica de três camadas: percepção, rede e aplicação. A organização dessa arquitetura é apresentada na Figura 2.

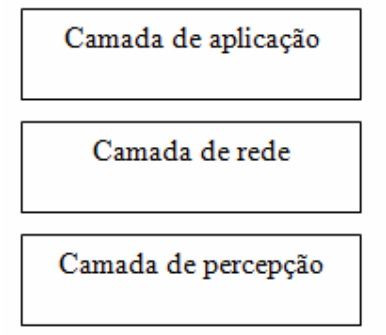

Figura 2 - Arquitetura três camadas da IoT

Fonte: traduzido de Wu et al. (2010, p. 484).

Descrição da Figura 2:

a) Camada de percepção – responsável pela percepção (sensores), identificação dos objetos, obtenção de informações e também responsável pela atuação no ambiente (atuadores) a partir de dados recebidos da camada de aplicação. Pode incluir códigos de barra 2-D e leitores, *tags* RFID e seus dispositivos de leitura e escrita, câmeras, GPS, sensores, terminais e redes de sensores.

b) Camada de rede – tem como função principal a transmissão e o processamento de informações. Essa camada inclui a convergência da rede de comunicação e à internet, centro

de gerenciamento da rede, de informação, de inteligência e outros. Essa camada processa e transmite as informações obtidas da camada de percepção.

c) Camada de aplicação – nesta camada estão as aplicações computacionais que recebem os dados, processam ações de atuação. As ações de atuação são transmitidas pela rede e realizadas pela camada de percepção.

De acordo com Atzori, Iera e Morabito (2010), as definições de Internet das coisas são baseadas em três linhas de visão, como representado na Figura 3. Essas linhas são: a orientada as "coisas", a orientada a "internet" e a orientada a "semântica". E a Internet das coisas é a convergência entre essas três visões, como pode ser visto na Figura 3.

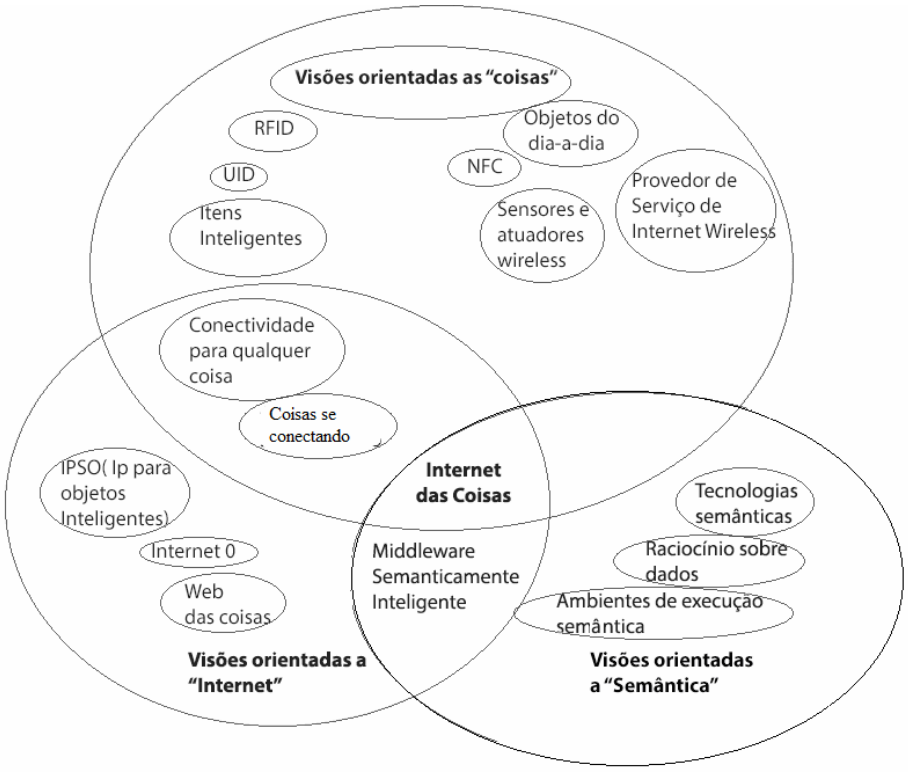

Figura 3 - Internet das coisas como união das 3 visões
Fonte: traduzido de Atzori, Iera e Morabito (2010, p. 3).

A IoT pode ser utilizada para melhorar a vida das pessoas, tanto em caráter pessoal como o profissional, tem como finalidade a interação entre os objetos de maneira inteligente para que executem ações de maneira pró-ativa com o objetivo de auxiliar os usuários.

As aplicações de IoT têm surgido em várias áreas, como, logística, transporte, ambientes inteligentes (casas, edifícios, infra-estrutura), energia, defesa e agricultura. Em essência a Internet das Coisas tem capacidade de influenciar significativamente várias facetas da sociedade.

Um dos desafios para a Internet das coisas está relacionado aos mecanismos de comunicação, pois todos os objetos precisam de um endereçamento que deve ser único. Esse endereçamento é a maneira de acessar os objetos para coletar as informações e realizar o seu controle. Outro desafio diz respeito aos mecanismos de fusão de dados. É necessária uma abordagem capaz de distinguir dados aleatórios e redundantes. Além disso há o interesse em uma avaliação mais aprofundada dos dados processados para melhorar o sistema e torná-lo mais confiável.

2.6 ESTADO DA ARTE

Dentre os vários artigos relacionados a Internet das Coisas sete trabalhos foram identificados por destacar-se em termos de semelhanças com o proposto neste trabalho.

Ko e Ramos (2010) propõem uma arquitetura orientada a Inteligência Artificial para o desenvolvimento de ambientes inteligentes chamada de *Intelligent Systems Research for Ambient Intelligence* (ISyRAmI). Essa arquitetura é composta por quatro módulos: a) a aquisição de dados; b) o armazenamento; c) a conversão e a manipulação desses dados, raciocínio inteligente; d) o apoio decisão/acionamento inteligente. A semelhança com este documento está na utilização dos conceitos de ambientes inteligentes para o desenvolvimento de um protótipo capaz de analisar dados e tomar decisões autônomas para alcançar um objetivo, que no caso deste trabalho é a automação inteligente de uma residência.

Al-Kuwari (2011) descreve um projeto para uma infra-estrutura inteligente para uma casa, chamado pelo autor de Beehouse, que utiliza nós modulares sem fio com a tecnologia ZigBee integrada com o microcontrolador Arduino, o qual é capaz de coletar dados, enviar informação e controlar a maioria dos aspectos da residência, bem como ter a capacidade de acessar esses nós e suas informações por meio de uma interface gráfica. A semelhança deste projeto com o protótipo descrito neste documento está no ambiente em que será aplicado, uma residência, permitindo ao usuário interagir com as informações coletadas nos cômodos da residência.

Hribernik et al. (2011) utilizam como base o paradigma da Internet das coisas para discutir o potencial de criação de aplicações de baixo custo utilizando o microcontrolador Arduino. Eles também mostram o papel humano na fase de concepção e criação desses produtos. O documento “Co-creating the Internet of Things - First Experiences in the

Participatory Design of Intelligent Products with Arduino" usa o mesmo conceito de base que este documento e busca o desenvolvimento de produtos inteligentes.

Michel Vinícius de Melo Euzébio e Emerson Ribeiro de Mello (2011) apresentam um sistema chamado DroidLar que consiste em um sistema de automação residencial completo no qual o usuário utiliza um aparelho celular Android como interface de controle, sendo utilizado o protocolo de comunicação Zigbee para a comunicação dos controladores dos dispositivos. São controlados sistemas eletroeletrônicos por meio de uma rede IP local ou de qualquer lugar do mundo via Internet. Esse trabalho tem como semelhança o acesso e controle a residência de qualquer lugar que possua acesso à Internet.

Brugnari e Maestrelli (2010) apresentam um sistema de automação residencial que se utiliza de microcontroladores e atuadores para proporcionar um controle sobre a residência através da Internet, para isso existe um dispositivo mestre que tem por objetivo manutenção e controle dos dispositivos atuadores e dos demais dispositivos escravos. Esse trabalho assemelha-se na proposta de proporcionar um controle sobre a residência do usuário através da Internet.

Diego Vanderlei Guerreiro de Oliveira e Felipe Joly Petrek (2014) apresentam um sistema para automação residencial capaz de ser controlado por meio da *web*, tem por objetivo fornecer controle sobre equipamentos da residência utilizando-se de uma interface *web* de fácil utilização. O sistema conta com um microcontrolador funcionando como central, o qual trabalha como servidor *web* fornecendo a página de controle ao usuário, essa central envia comandos para um microcontrolador escravo por meio de comunicação cabeada pelo protocolo UART. Esse escravo é responsável por interpretar as mensagens enviadas a ele e realizar as ações relacionadas a mensagem.

Avelino Morganti (2014) apresenta um sistema de automação residencial capaz de controlar e monitorar os dispositivos de uma residência, tais como lâmpadas, cortinas ou a irrigação do jardim, podendo esta ser gerenciada por um painel de controle, uma interface *web* ou mesmo um aplicativo mobile. Tem por objetivo ser sistema simples, robusto e de baixo custo, capaz de controlar o estado dos dispositivos remotamente e proporcionar segurança a uma determinada residência.

3 MATERIAIS E MÉTODO

A seguir estão descritas as ferramentas e as tecnologias que foram utilizados no desenvolvimento do protótipo resultante da realização deste trabalho.

3.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas para implementar o sistema.

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
Win32DiskImager		https://wiki.ubuntu.com/Win32DiskImager	Utilizado para instalar o sistema operacional no cartão SD do Raspberry.
JQuery	2.1.1	http://jquery.com	Biblioteca de JavaScript.
Case Studio 2	2.25	http://www.casestudio.com	Modelagem do diagrama de entidades e relacionamentos do banco de dados.
Linguagem PHP	5.6	http://www.php.net/	Linguagem de programação.
Linguagem C			Linguagem de programação para comunicação com módulo RF e programação de microcontrolador.
Sublime Text	3	http://www.sublimetext.com/	Ambiente de desenvolvimento utilizado para desenvolvimento do código HTML e PHP do sistema.
Code Composer Studio	6.1.0.00104	http://www.ti.com/tool/ccstudio	Ambiente de desenvolvimento para microcontroladores da família Texas Instruments.
MySQL	5	http://www.mysql.com/	Banco de dados.
MySQL WorkBench	5.2 CE	http://www.mysql.com/products/workbench/	Administrador do banco de dados.
Apache	7.0	http://www.apache.org/	Servidor *web* para a aplicação.
Bootstrap	3.2.0	http://getbootstrap.com/	Framework para desenvolvimento da interface responsiva.
Módulo nRF24L01		http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01/nRF24L01-PS	Módulo de radiofrequência responsável pela troca de mensagens do Raspberry Pi com o micro controlador Tiva.
Microcontrolador Tiva C series	TM4C123GH6PM	http://www.ti.com/product/tm4c123gh6pm	Microcontrolador responsável por acionar atuadores e realizar a leitura de sensores.
Raspberry Pi	B	https://www.raspberrypi.org/	Pequeno computador utilizado como servidor web.
Biblioteca WiringPi		http://wiringpi.com/	Biblioteca utilizada para acessar e utilizar os pinos do Raspberry Pi

Quadro 1 - Ferramentas e tecnologias utilizadas

O Quadro 2 apresentar os sensores e atuadores utilizados no desenvolvimento do sistema e a finalidade de cada um deles no projeto.

Sensor/Atuador	Datasheet	Finalidade
Fotoresistor	http://akizukidenshi.com/download/ds/senba/GL55%20Series%20Photoresistor.pdf	Mensurar a quantidade de luz ambiente para então ligar/desligar uma lâmpada, abrir/fechar uma cortina.
Sensor de temperatura LM35	http://www.ti.com/lit/ds/symlink/lm35.pdf	Mensurar a temperatura ambiente para abrir/fechar uma janela.
Sensor de gás e fumaça MQ-2	https://www.seeedstudio.com/depot/datasheet/MQ-2.pdf	Mensurar a quantidade de gás no ar do ambiente para abrir/fechar uma válvula solenóide ligada a rede de água como sistema contra incêndio.
Sensor de umidade do solo - Higrômetro	http://www.seeedstudio.com/wiki/Grove_-_Moisture_Sensor	Mensurar a umidade do solo de um vaso de planta para abrir/fechar uma válvula solenóide ligada a rede de água.
Relés	http://www.datasheet-pdf.com/PDF/SRD-05VDC-SL-C-Datasheet-Songle-720556	Utilizados para acionar atuadores que utilizam corrente alternada para serem ligados.
Lâmpada	--	Utilizada para iluminar o ambiente.
Válvulas Solenóides	--	Utilizadas para controlar uma saída de água
Servo Motores	--	Utilizados para abrir/fechar uma cortina e uma janela

Quadro 2 - Lista de sensores e atuadores

3.1.1 Raspberry Pi

O Raspberry Pi é um "computador miniatura" de baixo custo desenvolvido com o intuito de que todas as pessoas pudessem aprender mais sobre computação e programação utilizando-o em seus estudos. Ele possui as mesmas capacidades de um computador desktop podendo exercer as mesmas funcionalidades. Também tem a capacidade de interagir com o mundo exterior e tem sido amplamente usado em projetos de produtos digitais. O Raspberry Pi B possui as seguintes especificações:

- Processador: Broadcom BCM2835 ARM11;
- Unidade de processamento gráfico (GPU): Videocore 4 Processadores Gráficos integrados Unit (GPU) capaz de reproduzir 1080p em Full-HD;
- Memória: 512 MB de RAM;
- Utiliza GNU/Linux sistema operacional Debian (Raspibian);
- 2 x Portas USB;
- Saída de vídeo HDMI;
- Saída de Vídeo RCA;

- Saída de áudio Jack 3,5 milímetros;
- Porta Ethernet 10/100Mb;
- 5V Micro USB;
- SD, MMC, SDIO flash Slot para cartão de memória;
- 26-pin 2,54 milímetros, Slot de expansão Header;

O Raspberry Pi é utilizado como servidor web, no qual fica hospedada a página de controle do sistema que pode ser acessada pelo usuário. Também é responsável pela configuração e utilização do Módulo nRF24L01 para envio de mensagens.

3.1.2 Microcontrolador Tiva TM4C123GH6PM

Microcontrolador fabricado pela Texas Instruments (www.ti.com), desenvolvido para aplicações industriais incluindo monitoramento remoto, máquinas eletrônicas de ponto-de - venda, teste e medição, dispositivos de rede e switches, automação de fábrica, HVAC e edifício de controle, equipamentos para jogos, controle de movimento, transporte e incêndio e segurança.

Possui as seguintes especificações:

- CPU: ARM Cortex-M4F
- Flash: 256Kb
- Canais DMA (acesso direto a memória): 32
- EEPROM: 2Kb
- Pinos de captura: 24
- Comparadores Digitais: 16
- SRAM: 32
- Timers: 24
- Frequência máxima: 80MHz
- Saídas PWM (Pulse Width Modulation): 16
- GPIOs (Pinos de propósito geral): 43
- SSI/SPI: 4
- UART: 8
- Canais ADC (Conversor analógico digital): 12
- Resolução de ADC: 12bits

O microcontrolador Tiva é utilizado para a obtenção de dados do ambiente por meio da leitura de sensores bem como o processamento desses dados, também é responsável pela configuração e ativação dos atuadores do sistema.

3.1.3 Comunicação

A comunicação entre os componentes do sistema será feita utilizando-se dos protocolos SPI (*Serial Peripheral Interface*) e *Enhanced Shockburst* (Protocolo embarcado no módulo de radio-frequência nRF24L01). O protocolo SPI é utilizado para realizar a comunicação com o módulo RF24L01, e o protocolo *Enhanced Shockburst* é utilizado para troca de mensagens entre os módulos.

3.1.3.1 Protocolo SPI

O protocolo SPI trabalha de forma *full-duplex*, ou seja, a comunicação ocorre de forma bidirecional sendo possível o envio e o recebimento de dados, essa troca de mensagens ocorre fisicamente por meio dos sinais MOSI (*Master Output Slave Input*), MISO (*Master Input Slave Output*), um sinal de *clock* SCLK (*Souce Clock* – Fonte de clock) e um sinal SS para seleção do periférico.

Neste trabalho o Raspberry Pi e o microcontrolador Tiva atuam como *Masters* e os módulos RF24L01 atuam como *Slaves*, toda a configuração dos módulos será enviada por meio do protocolo SPI (*Serial Peripheral Interface*) bem como as mensagens para serem trocadas entre os módulos nRF24L01. A Figura 4 representa a ligação entre um dispositivo *Master* e seus vários *Slaves*.

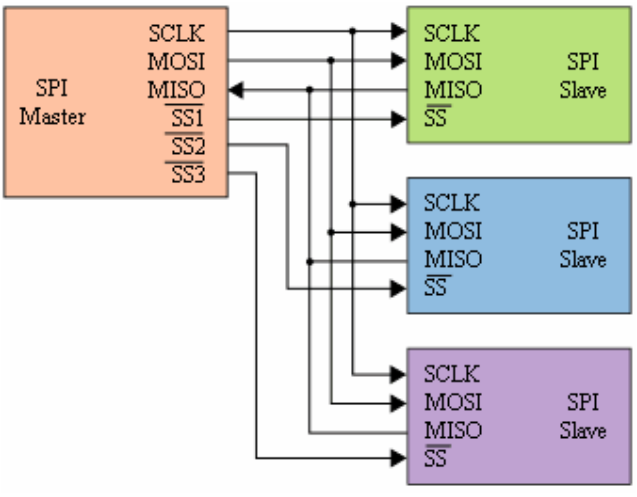

Figura 4 - Ligação para protocolo SPI

A transmissão dos dados ocorre da seguinte maneira: o Master (Raspberry Pi e o microcontrolador Tiva) envia um bit por meio do barramento MOSI (*Master Output Slave Input*) e simultaneamente recebe um bit no barramento MISO (*Master Input Slave Output*). No caso do *Slave*, representados pelos módulos nRF24L01, o bit é recebido no barramento MOSIe um bit é enviado pelo barramento MISO. Esse fluxo de dados é controlado pelo *clock* SCLK do *Master*. Só ocorre troca de dados enquanto houver um sinal de *clock* no barramento SCLK.

3.1.3.2 Protocolo Enhanced Shockburst

O protocolo *Enhanced Shockburst* proporciona o encapsulamento e retransmissão automática de pacotes e pode trabalhar tanto com comunicação unidirecional quanto com comunicação bidirecional, esse protocolo está embarcado no módulo nRF24L01.

Durante a transmissão dos dados o protocolo monta os pacotes e coloca os bits no pacote de dados dentro do transmissor. Durante o recebimento fica constantemente procurando por pacotes com endereços válidos no sinal demodulado. Quando um pacote é encontrado ele é então processado e validado por meio do campo CRC (*Cyclic Redundancy Check*), que será explicado posteriormente. No caso de o pacote ser válido ele então é movido para o *buffer* RX FIFO para posteriormente ser descarregado pelo microcontrolador por meio do protocolo SPI.

Uma transação de pacotes é feita entre dois transceptores. Um tem o papel de PTX (*Primary Transmitter* – transmissor primário) e o outro de PRX (*Primary Receiver* – receptor primário). A transação é sempre iniciada pelo PTX e é encerrada quando o PTX recebe o pacote ACK (*Acknowledge* – pacote utilizado para saber se uma mensagem chega ao destinatário, é enviado ao transmissor) enviado pelo PRX.

O usuário inicia uma transmissão enviando um pacote de PTX para PRX, automaticamente o PTX entra em modo de receptor e aguarda pelo pacote ACK, se o pacote é recebido pelo PRX automaticamente o PRX entra em modo de transmissor e então monta e envia o pacote ACK para o PTX antes de retornar para o modo de receptor. Se o PTX não receber o pacote ACK dentro do tempo pré-determinado automaticamente o PTX retransmite o pacote e entra em modo de receptor ficando na espera do pacote ACK.

Também é possível o envio de dados de PRX ao PTX por meio de uma carga colocada junto ao pacote ACK enviado ao PTX.

O pacote do protocolo *Enhanced ShockBurst* é formado por:

- Preâmbulo – 1 Byte
- Endereço – 3 à 5 Bytes
- Campo de controle de pacote – 9 Bits
- Carga de dados – 0 à 32 Bytes
- CRC – 1 à 2 Bytes

Figura 5 - Pacote do protocolo *Enhanced Shockburst*

O preâmbulo é uma sequência de bits usados para detectar níveis altos e baixos no receptor. Possui o tamanho de um byte podendo ser da forma 01010101 ou 10101010. Se o primeiro bit do endereço é 1 então o preâmbulo é automaticamente configurado para 10101010 e se o primeiro bit for 0 ele é configurado para 01010101. Isto é feito para estabilizar o receptor.

O endereço é usado pelo receptor para verificar se o pacote foi enviado para ele, esse campo pode ter de 3 a 5 Bytes. O campo de controle de pacotes possui 9 bits e é formado por 6 bits do tamanho da carga de dados, 2 bits para o PID (identificador do pacote) e um *bit* para *flag* NO_ACK.

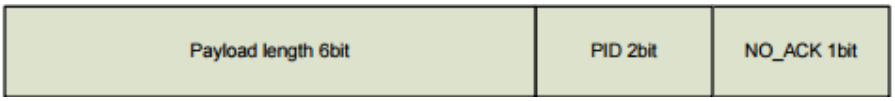

Figura 6 - Formato do campo de controle

O campo PID dentro do campo de controle de pacotes é usado para detectar se o pacote é novo ou se é um pacote retransmitido, o campo PID é incrementado no PTX a cada novo pacote enviado por SPI, e é usado juntamente com o campo CRC para verificar se é um pacote novo ou retransmitido. Se o PID do pacote recebido é o mesmo do anterior o receptor verifica a soma CRC de ambos os pacotes, se forem iguais significa que os pacotes são iguais então o pacote recebido é descartado.

A *flag* NO_ACK quando em estado alto indica ao receptor para não enviar o pacote ACK automaticamente ao transmissor.

O campo CRC é um mecanismo de detecção de erros, e é calculado sobre o endereço. Se o CRC falha o pacote não é aceito.

O protocolo ainda permite uma comunicação bidirecional utilizando o pacote ACK que é enviado ao PTX, é possível colocar até 5 bytes de dados para serem enviados juntos do pacote ACK, para isso os dados devem estar carregados no FIFO TX do PRX quando o pacote é recebido, em seguida os dados são transmitidos juntamente do ACK. Esses dados então ficam armazenados no FIFO RX do PTX prontos para serem descarregados.

3.1.3.3 Módulo nRF24L01

O módulo nRF24L01 é um transceptor de 2.4GHz que possui um protocolo de comunicação embarcado (*Enhanced Shockburst*). É configurado e operado por meio de comandos enviados via SPI. Com essa interface é possível acessar e configurar os registradores do módulo. A taxa de transmissão é de no máximo 2 Mbps. A Figura 7 representa o encapsulamento do módulo nRF24L01 e o Quadro 3 mostra o mapeamento dos seus pinos com suas respectivas funções.

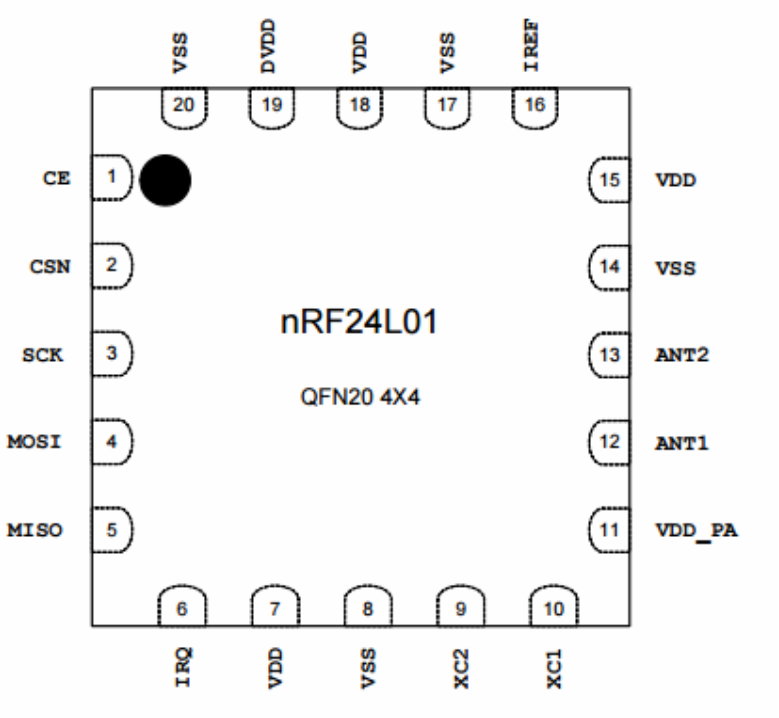

Figura 7 - Encapsulamento módulo nRF24L01

Pino	Nome	Função	Descrição
1	CE	Entrada digital	Ativa os modos RX ou TX
2	CSN	Entrada digital	*Chip Select* SPI
3	SCK	Entrada digital	*Clock* SPI
4	MOSI	Entrada digital	Entrada de dados *Slave* SPI
5	MISO	Saída digital	Saída de dados *Slave* SPI
6	IRQ	Saída digital	Pino de interrupção mascarável. Ativo baixo
7	VDD	Tensão	Tensão de alimentação (1,9V até 3,6V DC)
8	VSS	Tensão	Terra (0V)
9	XC2	Saída analógica	Pino do cristal 2
10	XC1	Entrada analógica	Pino do cristal 1
11	VDD_PA	Saída de Tensão	Saída de tensão de alimentação (1,8V) para o amplificador interno do módulo deve ser conectado em ANT1 e ANT2.
12	ANT1	RF	Antena 1
13	ANT2	RF	Antena 2
14	VSS	Tensão	Terra (0V)
15	VDD	Tensão	Tensão de alimentação (1,9V até 3,6V DC)
16	IREF	Entrada analógica	Corrente de referência. Conectar um resistor de 22 Kohms para o terra.
17	VSS	Tensão	Terra (0V)
18	VDD	Tensão	Tensão de alimentação (1,9V até 3,6V DC)
19	DVDD	Saída de Tensão	Saída de abastecimento interno digital para fins de acoplamento
20	VSS	Tensão	Terra (0V)

Quadro 3 - Mapeamento de pinos do módulo nRF28L01

O modulo nRF24L01 possui 4 modos de operação, são eles:

- *Power Down*: o módulo fica desabilitado com uso mínimo de corrente, nesse modo todos os valores dos registradores estão disponíveis e a comunicação SPI pode ser ativada.

Para entrar nesse modo o bit PWR_UP no registrador CONFIG deve estar em nível baixo. Esse é o modo no qual o módulo é configurado.

- Modos *Standby*: valorizando o bit PWR_UP no registrador CONFIG como nível alto o módulo entra no modo *Standby-I*, o módulo retorna a esse modo quando sai do papel de TX ou RX (quando o pino CE é colocado em nível baixo). O modo *Standby-II* ocorre quando o pino CE está em nível alto no papel de PTX quando o TX FIFO (Fila de envio) está vazio, ele fica no aguardo de um pacote ser carregado no TX FIFO para ser enviado.

- Modo PRX: é o modo de receptor, é ativo quando os bits PWR_UP e PRIM_RX do registrador CONFIG e o pino CE estão em nível alto.

- Modo PTX: é o modo de transmissor, é ativo quando o bit PWR_UP está em nível alto e o bit PRIM_RX está em nível baixo, ambos pertencem ao registrador CONFIG, havendo dados carregados em TX FIFO e com um pulso de nível alto em CE maior que 10 microssegundos.

A Figura 8 mostra um fluxograma de como esses modos são configurados bem como os tempos necessários para estar atuando em cada modo.

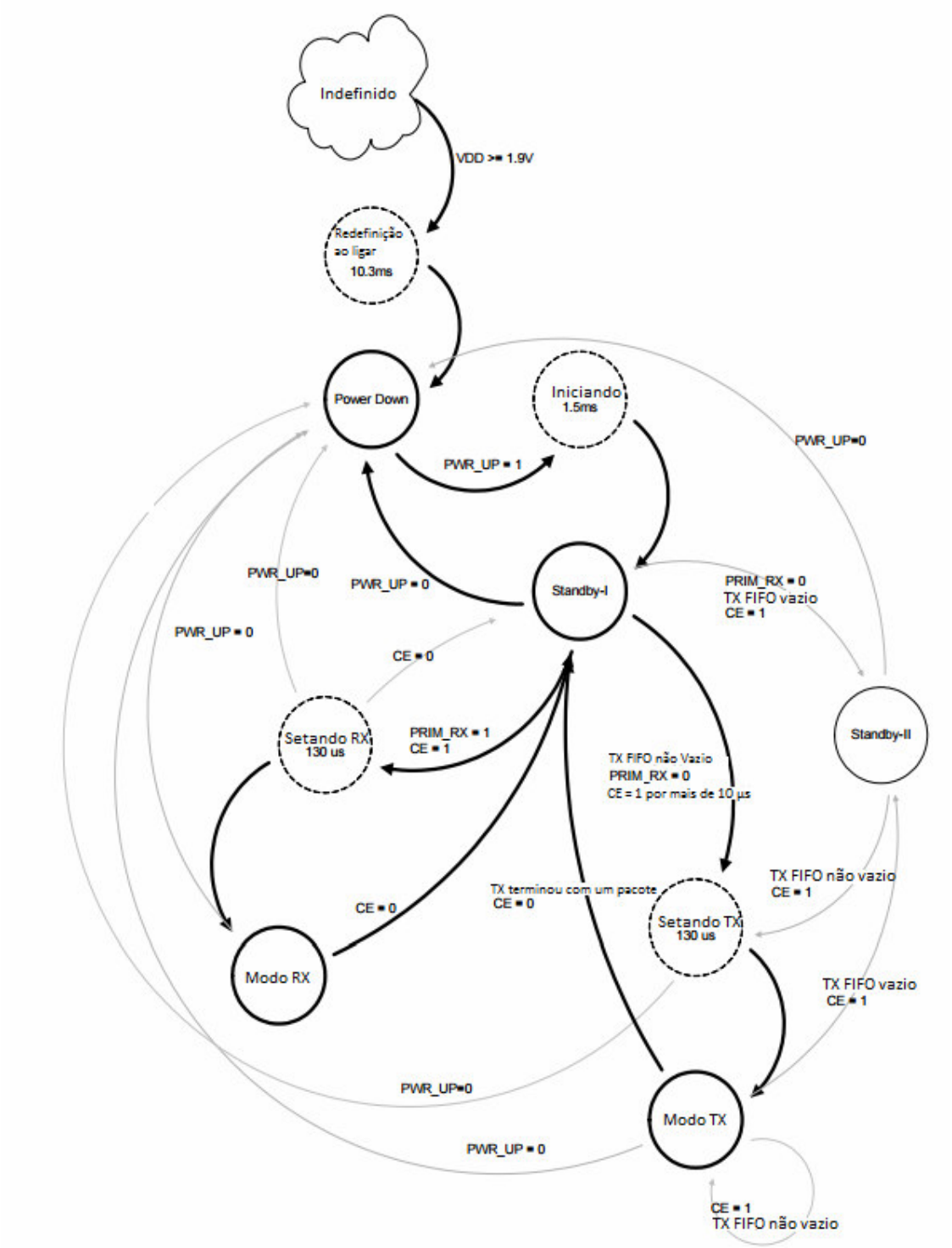

Figura 8 - Fluxograma dos modos de operação do módulo nRF04L01
Fonte: traduzido de Nordic Semiconductor (2007, p. 20).

A comunicação ocorre entre dois módulos quando um está em modo PRX e outro em modo PTX. O envio de dados sempre inicia pelo PTX, o qual envia um pacote ao PRX, se este estiver configurado com o modo autoACK (envio automático de pacote ACK) irá enviar

ao PTX um pacote com o ACK e, se necessário, algum conjunto de dados. A Figura 9 mostra o funcionamento básico do lado PTX antes do envio de um pacote de dados. Na Figura 10 é possível observar o que acontece no lado do PRX ao receber um pacote vindo do PTX.

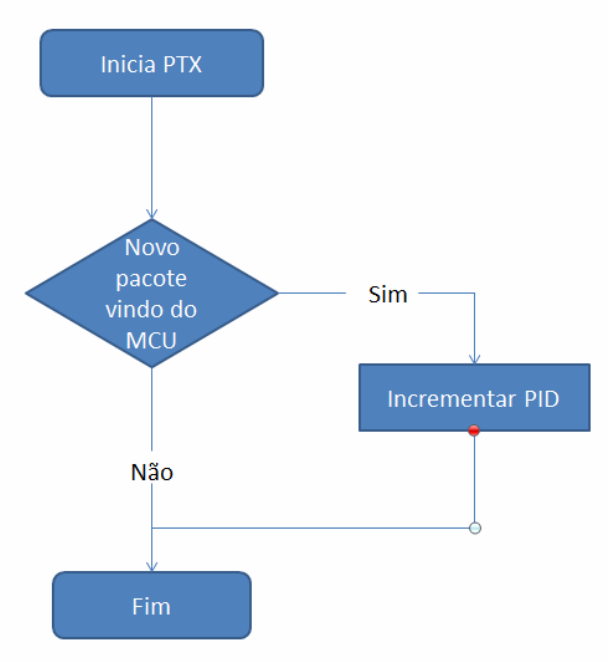

Figura 9 - Funcionamento básico PTX antes de envio de novo pacote

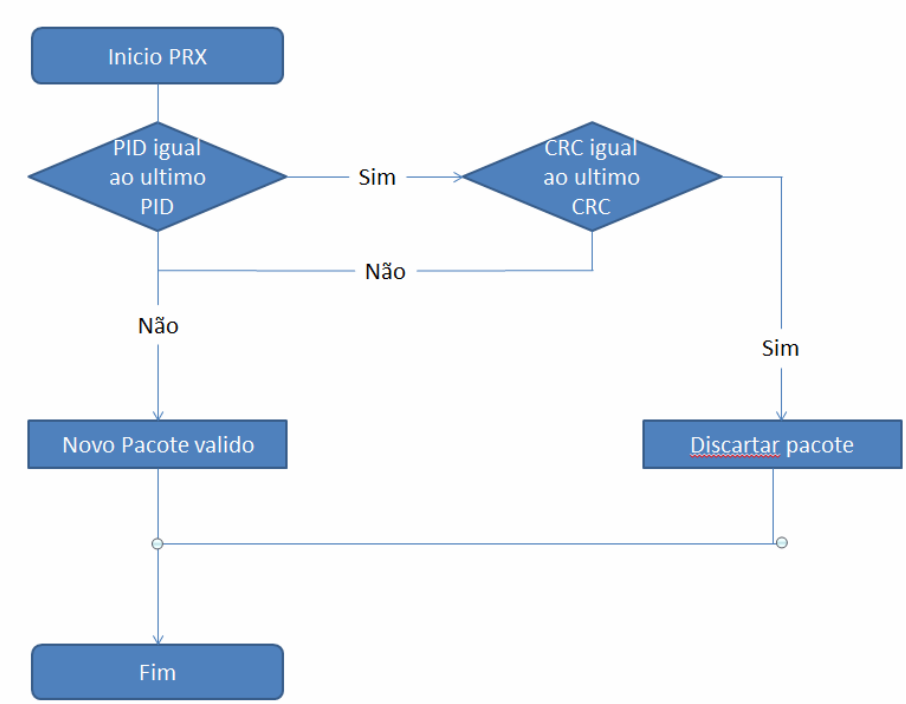

Figura 10 - Funcionamento básico PRX ao receber um novo pacote

3.1.4 Biblioteca WiringPi

A WiringPi é uma biblioteca para acesso aos pinos do Raspberry Pi, escrita na linguagem C e foi desenvolvida para ser familiar para os desenvolvedores que usaram o sistema de pinos do Arduino.

A biblioteca inclui também um programa chamado por GPIO, que foi desenvolvido para ser instalado como um programa *setuid*, podendo ser chamado por um usuário comum mesmo este não possuindo permissões de usuário administrador, que pode ser usado para programar e configurar os pinos. O programa também pode ser utilizado para leitura e escrita dos pinos e para controlá-los. Essa biblioteca também possui módulos adicionais que facilitam o uso das comunicações *Universal Asynchronous Receiver/Transmiter* (UART), I2C (*I-two-C*) e SPI que podem ser usadas com os pinos de propósito geral quando configurados com suas funções especiais.

3.1.5 Sensores

Esta sessão aborda os sensores utilizados no desenvolvimento do sistema. Foram utilizados os sensores fotoresistor, temperatura LM35, sensor de fumaça e gás MQ-2 e um sensor de umidade do solo.

3.1.5.1 Fotoresistor

O fotoresistor é um sensor utilizado para medir o grau de luminosidade de um ambiente. O sensor consiste basicamente de um resistor que altera sua grandeza de acordo com a quantidade de luz que incide no mesmo. Para realizar a medição desse sensor utiliza-se um divisor resistivo para medir a tensão que passa através do sensor. Uma das saídas desse divisor resistivo é ligada a GND (terra) e a outra a entrada do conversor ADC (conversor analógico digital), no nosso caso utilizamos uma alimentação de 5V. A Figura 11 exibe o circuito utilizado para medição do fotoresistor.

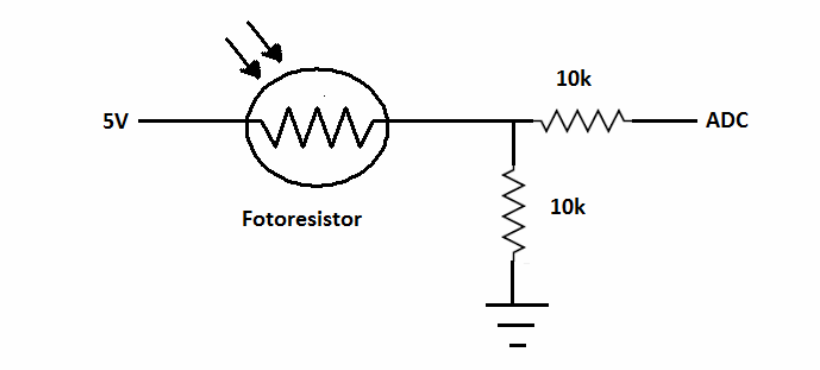

Figura 11 - Circuito do fotoresistor

A variação de resistência de um fotoresistor pode ir de 0 a 1 megaohm.

3.1.5.2 Sensor de temperatura LM35

O sensor LM35 desenvolvido pela Texas Instruments é utilizado para medir a temperatura de um ambiente. O modelo utilizado neste trabalho mede de 0°C a 100°C. A tensão de operação varia de 4V a 30V, a saída é de 10mV por °C. Para realizar a medição desse sensor é utilizada uma alimentação de 5V adiciona-se um resistor ligando os pinos GND e VCout, pino de saída do sensor utilizado para medição do ADC, do sensor e então liga-se o pino VCout a entrada do ADC. A Figura 12 mostra o circuito do sensor LM35.

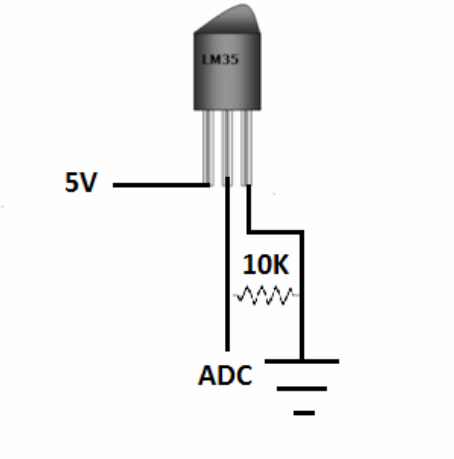

Figura 12 - Circuito LM35

3.1.5.3 Sensor de fumaça e gás MQ-2

O sensor MQ-2 é capaz de detectar gás inflamável e fumaça, medindo entre 300 e 10000 ppm (partes por milhão) de partículas de gás e fumaça no ar. O módulo do sensor é alimentado com 5V e possui duas saídas, uma digital representada pelo pino D0 e uma saída analógica no pino A0. O pino D0 tem como saída 5V ou 0V de acordo com o nível de fumaça ou gás medido. A sensibilidade dessa saída é controlada por um *trimpot*[1] no módulo. Para o sistema desenvolvido neste trabalho será utilizada apenas a saída analógica ligada a entrada do conversor ADC. A Figura 13 mostra o módulo do sensor MQ-2.

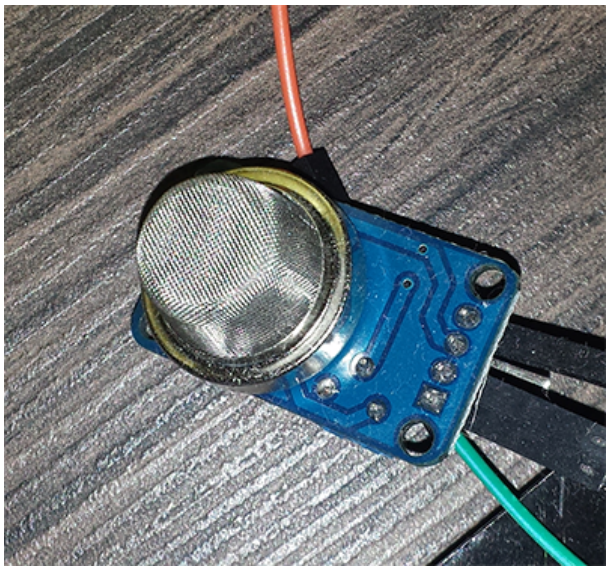

Figura 13 - Módulo do sensor MQ-2

3.1.5.4 Sensor de umidade do solo

Chamado de higrômetro, o sensor de umidade do solo, é utilizado para medir a variação de umidade presente no solo por meio de uma sonda. O módulo possui 2 saídas, uma digital no pino D0 que fica em nível alto (saída em 3,3V) a partir de um nível de umidade detectado de acordo com a sensibilidade ajustada no potenciômetro presente no módulo, e uma saída analógica A0 que pode ser ligada a entrada do ADC. Neste trabalho será utilizada apenas a saída analógica ligada ao ADC para realizar a medição de umidade no solo.

[1] Um *trimpot*, do inglês *trimmer potentiometer*, é um potenciômetro miniatura ajustável.

A alimentação pode ser de 3,3V até 5V, a saída fornecida pelo pino A0 é proporcional a porcentagem de umidade do solo de 0 (saída em 3,3V) a 100% (saída em 0V). A Figura 14 e 15 mostram o módulo do sensor e a sonda utilizada para medição.

Error! Hyperlink reference not valid.

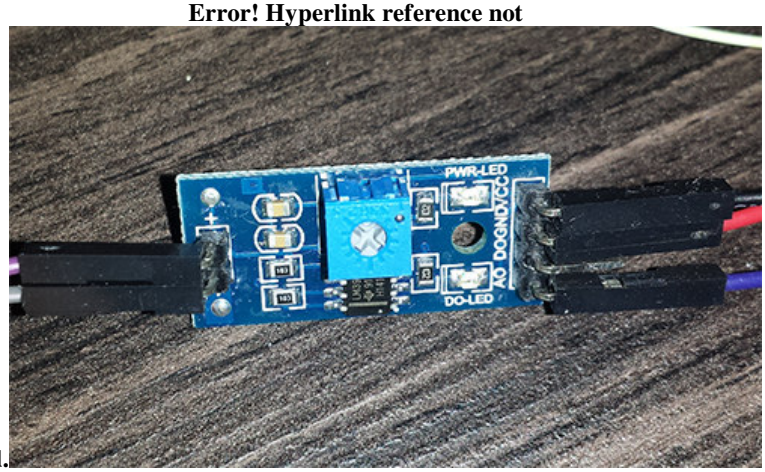

Figura 14 - Módulo do sensor de umidade

Figura 15 - Sonda para medição de umidade do solo

3.1.6 Atuadores

Está sessão aborda os atuadores utilizados no desenvolvimento do sistema. Os atuadores foram: relés, válvulas solenóides e servo motores.

3.1.6.1 Circuito acionador de rele

Este circuito acionador é utilizado para acionar relés de 5V com um pino do micro controlador de 3,3V. Os relés serão utilizados para o acionamento de lâmpadas e solenóides no sistema. Os relés são dispositivos eletromecânicos utilizados para acionamento de equipamentos que utilizam tensões elevadas, o acionamento é feito com uma tensão baixa e os

pinos de acionamento e alimentação dos aparelhos são isolados, não oferecendo riscos para o circuito que aciona o relé.

O circuito para acionamento do relé utiliza os seguintes componentes:

- 1 Diodo 1N4007
- 1 Transistor BC548C
- 2 Resistores de 10K
- 1 relé de 5V.

A Figura 16 mostra a montagem do circuito acionador do relé.

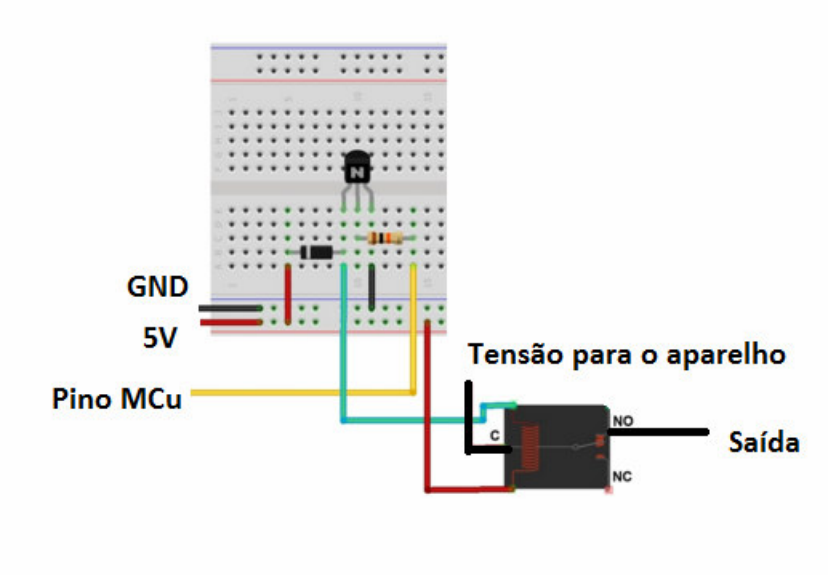

Figura 16 - Circuito acionador de Relé
Fonte: http://www.arduinoecia.com.br/2013/05/ligando-uma-lampada-com-rele.html

Quando o pino MCu está em um nível baixo (0V) o pino da esquerda do transistor não conduz o que faz com que o diodo fique inversamente polarizado e não permite a passagem de 5V para acionamento do Relé. Quando o pino MCu está em nível alto (3.3V) o pino da esquerda do transistor conduz polarizando o diodo diretamente, o que faz com que o diodo conduza a tensão de 5V acionando o Relé.

3.1.6.2 Válvula Solenóide

O funcionamento básico de uma válvula solenóide é de que quando existe uma tensão aplicada ao componente a válvula é aberta, se a tensão é nula então a válvula fecha. Neste trabalho serão utilizadas válvulas solenóides alimentadas por 12V. Para o acionamento das

válvulas será utilizado o circuito do relé da sessão 3.1.6.2. A Figura 17 mostra o modelo da válvula solenóide utilizada.

Figura 17 - Modelo de válvula solenóide

3.1.6.3 Servo Motor de 180 graus

O servo motor é um pequeno motor que muda o seu posicionamento de acordo com o sinal enviado a ele. O sinal é um PWM e o seu posicionamento se ocorre de acordo com a largura do pulso do PWM enviado. Possui 3 pinos para ligação, o pino de alimentação que deve ser ligado a 5V de tensão, pino GND, e o pino de entrada do sinal PWM. Para se posicionar em 0 graus a PWM deve ter uma largura de banda correspondente a 1ms, para 90 graus deve ser 1,5ms e para 180 graus deve ser de 2ms, o período da PWM deve corresponder a 18ms. A Figura 18 mostra o modelo do servo motor utilizado e a Figura 19 mostra os sinais PWM usados para o acionamento do servo motor.

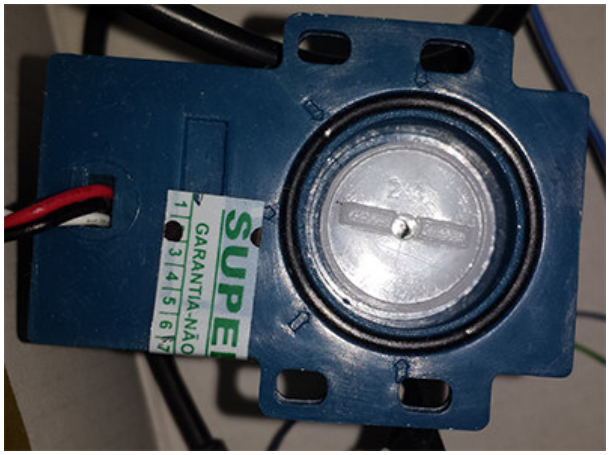

Figura 18 - Modelo de servo motor utilizado

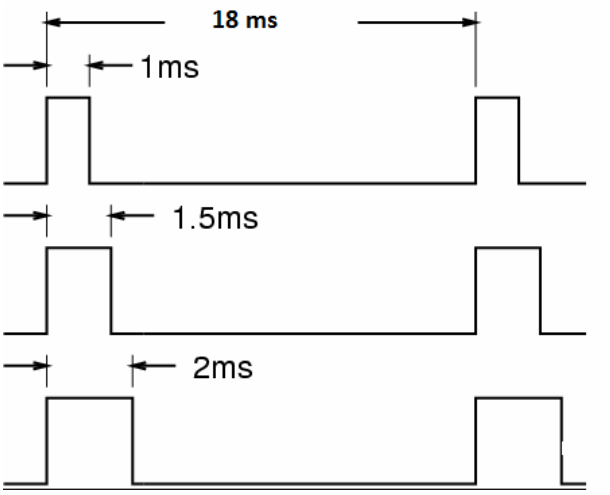

Figura 19 - PWM para uso do servo motor

3.2 MÉTODO

A seguir estão descritas as etapas definidas para o desenvolvimento do protótipo.

a) Levantamento de requisitos

Inicialmente foram definidos os sensores e atuadores existentes no protótipo e que serão implementados. Essa definição é feita a partir de uma pesquisa de mercado para identificar o que há de disponível para compra e que possa ser adquirido em tempo hábil para o desenvolvimento do projeto. Também foram considerados sensores existentes no Laboratório de Hardware e Software do DAINF e que podem ser utilizados por alunos no desenvolvimento de projetos.

Após definidos os sensores e atuadores foram elaborados os requisitos do projeto. Esses requisitos foram utilizados como base para a modelagem do sistema.

b) Análise e projeto do sistema

Nesta etapa os requisitos foram modelados e documentados por meio de casos de uso, diagrama de classes, de entidades e relacionamentos e outros. A definição dos diagramas e complementos necessários será realizada a partir da definição dos requisitos.

c) Implementação

Na fase de implementação, o servidor, interface com o cliente e os códigos para a leitura de sensores e para a ativação dos atuadores foram desenvolvidos.

d) Protótipo

Um protótipo de uma residência foi construído nessa etapa do desenvolvimento para a colocação dos sensores e atuadores e a interação com o sistema.

e) Testes

Os testes foram unitários e realizados pelo autor deste trabalho visando identificar erros de codificação. Os testes no protótipo foram realizados para identificar erros de comunicação e de implementação e para verificar se os requisitos definidos foram efetivamente implementados.

4 RESULTADOS

Nesse capítulo são apresentados os resultados obtidos durante o desenvolvimento deste trabalho. Inicialmente é abordado a instalação e configuração do servidor *web* no Raspberry Pi.

4.1 Abordagem Proposta

Neste trabalho foi desenvolvido um protótipo para automação residencial com base no conceito de Internet das Coisas, uma residência é classificada como uma "coisa" de Internet das Coisas.

O sistema utiliza um Raspberry Pi, com o Sistema Operacional Linux juntamente com o servidor *web* Apache e a linguagem de programação PHP. Esse conjunto de tecnologias atua como servidor *web* conectado à rede por cabos ou via *wireless*. O servidor atua no processo de requisição e resposta fornecendo uma página *web* em que o usuário possa selecionar a ação que deseja ser executada, então o Raspberry Pi envia o comando ao Módulo RF nRF24L01 por meio do protocolo SPI, o módulo por sua vez envia ao outro módulo que está ligado ao micro-controlador Tiva por meio do protocolo SPI, o microcontrolador então recebe a mensagem e a processa de acordo com a ação que foi selecionada pelo usuário. A Figura 20 mostra uma visão geral do sistema, a comunicação SPI entre o Raspberry Pi e o módulo nRF24L01 é representado pela letra **A** na figura, a comunicação entre os módulos é representado pela letra **B** e a letra **C** representa a comunicação SPI entre Tiva e o módulo nRF24L01.

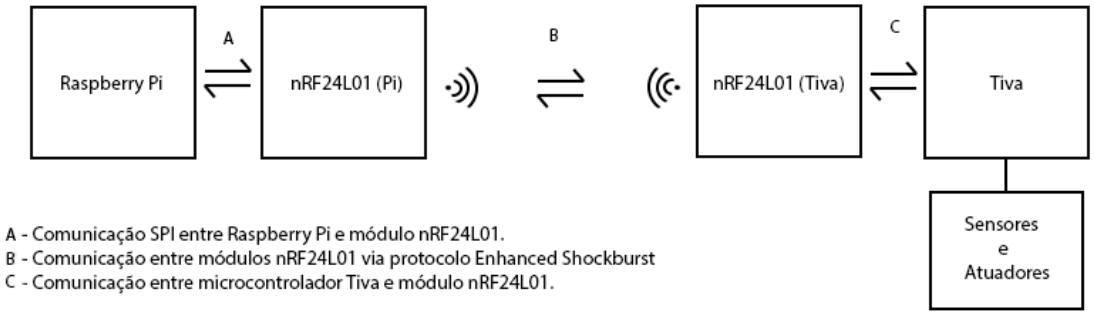

Figura 20 - Visão geral do sistema

Para facilitar a integração do sistema com a *web,* o servidor utiliza a arquitetura REST, que é baseada em recursos e verbos, cada recurso é referenciado como uma URI (um endereço *web*, por exemplo), as ações desses recursos são os verbos criar, ler, alterar e excluir que são mapeadas para as ações HTTP de POST, GET, PUT e DELETE (ESTELLER-CURTO et al., 2012).

Os dados coletados pelos sensores são processados pelo microcontrolador Tiva e são utilizados para tomadas de decisão de acordo com parâmetros informados pelo usuário e também serem utilizados para tomada de decisão pelo servidor, por exemplo, em uma situação em que a umidade está muito alta e uma janela é fechada, ou a temperatura está elevada e o aparelho de ar-condicionado é ligado, o usuário também poderá definir alguns parâmetros para a atuação autônoma do sistema. Os parâmetros escolhidos pelo usuário são armazenados em um banco de dados MySQL, para a configuração inicial em caso de reinício do sistema.

Para a comunicação entre o Raspberry Pi e o microcontrolador Tiva foi implementado o protocolo de comunicação *enhanced shockburst* conforme descrito nos nas seções 4.3, 4.4, 4.6 e 4.7.

O Quadro 4 apresenta os sensores, respectivos atuadores, definidos para sem utilizados no projeto.

Sensores	Atuadores
Temperatura	Abrir/Fechar janelas (servo motor)
Luminosidade	Abrir/Fechar cortinas (servo motor) Ligar/Desligar lâmpadas (relé)
Umidade	Regar jardim (válvula solenóide)
Gás	Sistema anti-incêndio (válvula solenóide)

Quadro 4 - Sensores e atuadores utilizados no projeto

4.2 Instalação e Configuração do Servidor Web

Foram instalados o servidor Apache, o PHP e o banco de dados MySQL. Os materiais necessários são:

- Computador com entrada para cartão SD
- Mouse e teclado USB.
- Adaptador HDMI/VGA.

- Cabo de rede.
- Imagem do SO Raspibian (disponível em: http://downloads.raspberrypi.org/raspbian_latest).
- Software Win32DiskImager.
- Software Putty.

Com o cartão SD no computador é realizada a formatação do mesmo, em seguida por meio do Software Win32DiskImager é montada a imagem do sistema operacional Raspbian no cartão, no software deve ser selecionada a imagem do SO e o cartão SD e clicar no botão Write.

Em seguida é possível ligar o Raspberry Pi ao monitor utilizando o adaptador HDMI/VGA juntamente com o teclado e o mouse e, assim, executar a instalação do SO. Concluída a instalação é possível acessar o sistema operacional utilizando o login **pi** e senha **raspberry**, então com primeira etapa, o acesso SSH é habilitado.

sudo –i

cd /boot

mv boot_enable_ssh.rc boot.rc

reboot

Com essa instalação é possível acessar o Raspberry Pi por uma conexão SSH. Com o sistema ligado à rede utilizando o software Putty entrando com o IP e clicando Open para estabelecer conexão. Nesse momento não é mais necessário o monitor. Por questões de segurança a senha padrão é alterada.

sudo –i

passwdpi

novasenha

A próxima etapa é a instalação do servidor Apache (servidor HTTP para sistemas operacionais mais atuais) com PHP. Inicialmente, os aplicativos do SO são atualizados.

sudo apt-get update

Em seguida é feito o download e executada a instalação

sudo apt-get install apache2 php5 libapache2-mod-php5

Concluída a instalação o serviço do apache deve ser reiniciado.

sudo servisse apache2 restart

A instalação do banco de dados mySQL pode ser realizada.

sudo apt-get install mysql-server mysql-client php5-mysql

Durante a instalação é requisitada a alteração da senha padrão do banco de dados. Concluída essa etapa é necessário instalar um servidor FTP para acessar o sistema de arquivos do sistema operacional. Para servidor FTP foi instalado o vsftpd.

sudo chown –R pi /var/www

sudo apt-get instal vsftpd

Feita a instalação, os arquivos de configuração do sistema FTP são acessados por:

Sudo nano /etc/vsftpd.conf

E são alteradas as seguintes linhas.

anonymous_enable=YES para **anonymous_enable=NO**

#local_enable=YES para **local_enable=YES**

#write_enable=YES para **write_enable=YES**

É adicionada uma nova linha ao final do arquivo.

force_dot_files=YES

Após o arquivo é salvo o serviço de FTP é reiniciado.

sudo servisse vsftpd restart

A última etapa é a instalação do phpMyAdmin para auxiliar na administração do banco de dados. Os arquivos no servidor são extraídos pela conexão ftp. A partir dessas instalações é possível acessar o painel administrativo do phpMyAdmin pelo endereço http://IPdoservidor/phpmyadmin, utilizando o usuário root e a senha previamente definida durante a instalação do banco de dados mysql.

Com a instalação e configuração do servidor concluída é necessário habilitar o comando *sudo* para o usuário utilizado pelo PHP (usuário www-data), para que seja possível a execução de arquivos por meio de comandos enviados pela página web utilizando a função PHP *system* ou outras similares.

Para isso é necessário acessar o sistema operacional utilizando o usuário **root** e alterar o arquivo **/etc/sudoers**.

nano /etc/sudoers

E ao final do arquivo é adicionada a linha

www-data ALL=(ALL) ALL

Assim o usuário www-data utilizado pelo PHP tem permissão para executar o comando sudo e assim executar os arquivos que serão responsáveis pela comunicação com o módulo nRF24L01.

4.3 Configuração do Módulo nRF24L01 com Raspberry Pi

Para a configuração do módulo nRF24L01 utilizando comunicação SPI foi desenvolvido um código em linguagem C que é executado no carregamento do sistema operacional. Para a execução ocorrer quando o sistema operacional é iniciado é necessário colocar o comando que executa o programa no arquivo de script em **/etc/rc.local**. Como mostra a Figura 21.

```
/var/www/init_RF2

/usr/bin/php5 /var/www/recover.php

exit 0
```

Figura 21 - Código do arquivo rc.local

O arquivo chamado para iniciar a configuração do módulo é o **init** que tem como função inicializar os pinos para a comunicação SPI e os pinos necessários para a configuração do módulo (Pinos 2 para CE e 0 para CSN). A Figura 22 mostra o código do arquivo **init.c**.

```
system("/var/www/wiringPi/gpio/gpio load spi"); //Configura pinos para comunicação SPI

system("/var/www/wiringPi/gpio/gpio mode 0 out"); //Habilita pino 0 como saída
system("/var/www/wiringPi/gpio/gpio mode 2 out"); //Habilita pino 2 como saída
system("/var/www/wiringPi/gpio/gpio write 0 1"); //Coloca o pino 0 com saída em nível alto
system("/var/www/wiringPi/gpio/gpio write 2 0"); //Coloca o pino 2 com saída em nível baixo

delay(3000);

system("sudo ./writeReg_RF reg_config 0 A"); //Escreve no registrador config do módulo

delay(5);

system("./configTX_RF"); //Configura o módulo
```

Figura 22 - Código do arquivo init

O arquivo **init** faz chamada à outros dois arquivos em seu código o arquivo **writeReg_RF** utilizado para escrever em um registrador do módulo por meio da comunicação SPI, e o arquivo **configTX_RF** que realiza a configuração do módulo para o modo de TX (transmissão de dados). O uso do arquivo **writeReg_RF** necessita a passagem de alguns parâmetros que são o registrador que se deseja escrever e os 8 bits que serão escritos. A Figura 23 mostra o código do arquivo **writeReg_RF.c**.

```
wiringPiSetup(); //habilita o uso das funções da biblioteca wiringPi
int ret = wiringPiSPISetup(0,5000000);// função para configuração da comunicação SPI

system("gpio write 0 0");// pino CSN colocado em estado baixo

unsigned char retSend = wiringPiSPIDataRW(0,data,2); //função para envio de dados via SPI

system("gpio write 0 1");// pino CSN colocado em estado alto
```

Figura 23 - Parte do código do arquivo WriteReg_RF

A função **wiringPiSetup** é utilizada para habilitar o uso das funções da biblioteca wiringPi no código, em seguida é configurada a comunicação SPI no canal 0 com 5MHz de velocidade por meio da função **wiringPiSPISetup**, para realizar a escrita em um registrador no módulo nRF24L01 é necessário que o pino CSN esteja em nível baixo e então é enviado o comando para escrita, o comando é 0x20(hexadecimal) e deve ser realizada a operação lógica OR com o registrador que vai ser escrito, após o envio do comando e do registrador são enviados os 8 bits de dados a serem escritos e para completar a escrita o pino CSN é colocado em nível alto novamente.

A Figura 24 mostra o código de configuração do modo TX no módulo RF.

```
system("gpio write 2 0"); //Pino CE colocado em estado baixo

system("sudo ./writeReg_RF reg_config 1 A"); //Configurando o registrador Config para modo de TX

delay(5);
//Configuração dos registradores do módulo
system("sudo ./writeReg_RF reg_en_aa 3 F");
system("sudo ./writeReg_RF reg_en_rxaddr 0 3");
system("sudo ./writeReg_RF reg_setup_aw 0 3");
system("sudo ./writeReg_RF reg_setup_retr 0 3");
system("sudo ./writeReg_RF reg_rf_ch 0 2");
system("sudo ./writeReg_RF reg_rf_setup 0 F");
system("sudo ./writeReg_RF reg_observe_tx 0 0");
system("sudo ./writeReg_RF reg_rpd 0 0");
system("sudo ./writeReg_RF reg_rx_pw_p0 0 0");
system("sudo ./writeReg_RF reg_rx_pw_p1 0 0");
system("sudo ./writeReg_RF reg_rx_pw_p2 0 0");
system("sudo ./writeReg_RF reg_rx_pw_p3 0 0");
system("sudo ./writeReg_RF reg_rx_pw_p4 0 0");
system("sudo ./writeReg_RF reg_rx_pw_p5 0 0");
system("sudo ./writeReg_RF reg_fifo_status 1 1");
system("sudo ./writeReg_RF reg_dynpd 0 1");
system("sudo ./writeReg_RF reg_feature 0 7");
system("sudo ./writeReg_RF cmd_activate 7 3");

system("gpio write 2 1");// Pino CE colocado em estado alto
```

Figura 24 - Código para configuração do módulo nRF24L01 no modo TX

Para a configuração do módulo para o modo TX é necessário que o pino CE esteja em nível baixo e os bits PRIM_RX configurado em 0 e PWR_UP configurado em 1, ambos são bits do registrador Config, feito isso é necessário aguardar a entrada no modo *Standby*, em seguida é realizada a configuração dos outros registradores do módulo e então o pino de CE é colocado em estado alto novamente. Feito isso o módulo estará no modo TX e poderá enviar mensagens para outro módulo RF que esteja configurado em modo RX (receptor). A Figura 25 mostra a ligação física entre o raspberry Pi e o módulo nRF24L01.

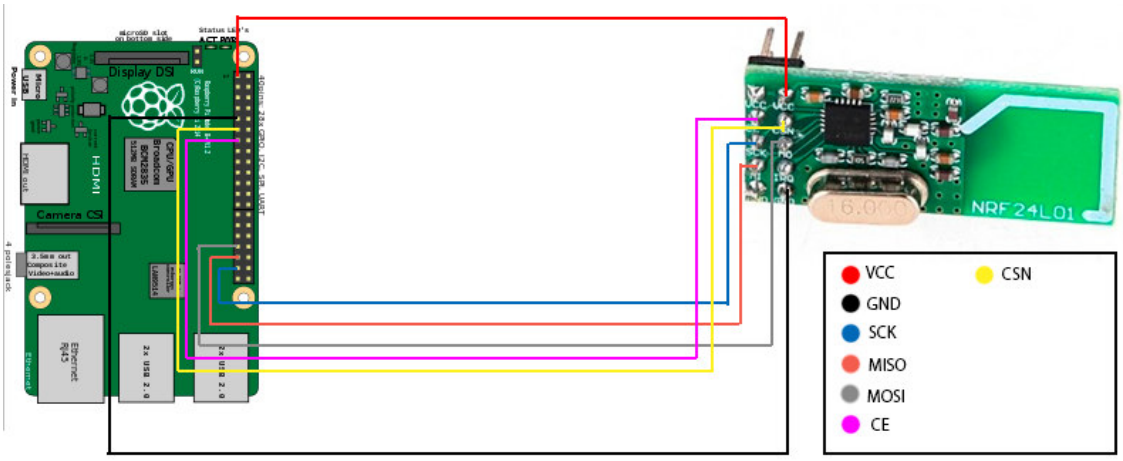

Figura 25 - Ligação entre Raspberry Pi e módulo nRF24L01

4.4 Envio de Mensagem (Raspberry Pi)

Para enviar uma mensagem utilizando o módulo nRF24L01, é necessário primeiramente que esteja configurado em modo TX e então devem ser carregados utilizando o comando **W_TX_PAYLOAD**, são necessários 5 bytes de dados no **FIFO TX** do módulo, após os dados estarem no **FIFO** eles são enviados para o ar com o endereço que está no registrador **TX_ADDR**.

Para realizar a escrita no **FIFO TX** foi é chamado o programa **writeRegFive_RF** que é utilizado enviar o comando **W_TX_PAYLOAD** e enviar 5 bytes de dados que serão posteriormente enviados pelo módulo. Para a utilização desse arquivo é necessária a passagem de alguns parâmetros, são eles o comando seguido dos 5 bytes de dados. A Figura 24 mostra o código do arquivo **writeRegFive_RF.c**, na Figura 26 é mostrado como é utilizado o arquivo **writeRegFive_RF** para enviar uma mensagem com o comando de acender uma lâmpada.

```
wiringPiSetup();
int ret = wiringPiSPISetup(0,5000000);

system("gpio write 0 0"); //pino CSN em nível baixo

unsigned char retSend = wiringPiSPIDataRW(0,data,6); //envio do comando seguindo de 5bytes de dados

system("gpio write 0 1"); //pino CSN em nível alto
```

Figura 26 - Código do arquivo writeRegFive_RF

Para escrita no **FIFO TX** do módulo é necessário que o pino CSN seja colocado em nível baixo, em seguida é utilizada a função wiringPiSPIDataRW para enviar o comando e os 5 bytes de dados. Após o envio dos dados ao módulo o pino de CSN é colocado em nível alto.

```
system("sudo ./writeRegFive_RF cmd_w_tx_payload 1 0 0 0 0 0 0 1 0");
```

Figura 27 - Uso do arquivo writeRegFive_RF

4.5 Interface Gráfica

A interface foi desenvolvida utilizando as linguagens HTML, PHP, Javascript e também a tecnologia AJAX. Inicialmente foi desenvolvido o controle de usuários. A validação usuários é realizada por meio de verificação em banco de dados e a senha possui encriptação MD5. A Figura 28 apresenta a interface de acesso ao sistema.

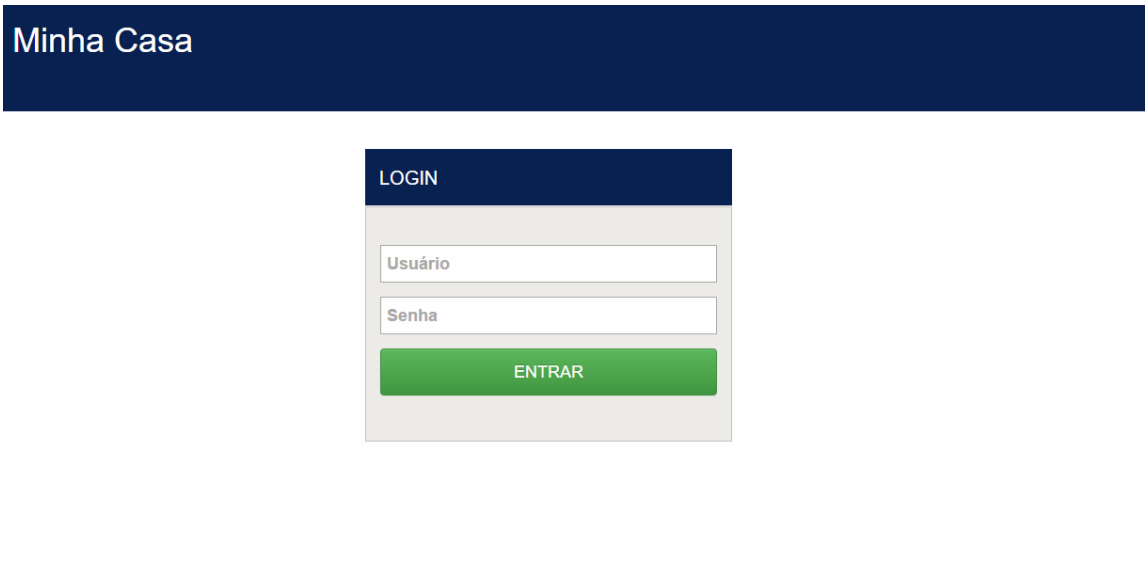

Figura 28 - Tela de Login

O *layout* busca ser simples e intuitivo, cada um dos botões presentes na tela apresentada ao usuário tem uma função específica. Ao ser executada a ação de clique, o botão chama sua respectiva função Javascript que por sua vez chama um arquivo PHP por meio de uma chamada assíncrona com AJAX. O arquivo PHP é responsável por chamar o arquivo executável com o objetivo de enviar a mensagem da ação desejada ao micro controlador Tiva. A Figura 29 exibe a tela principal do sistema.

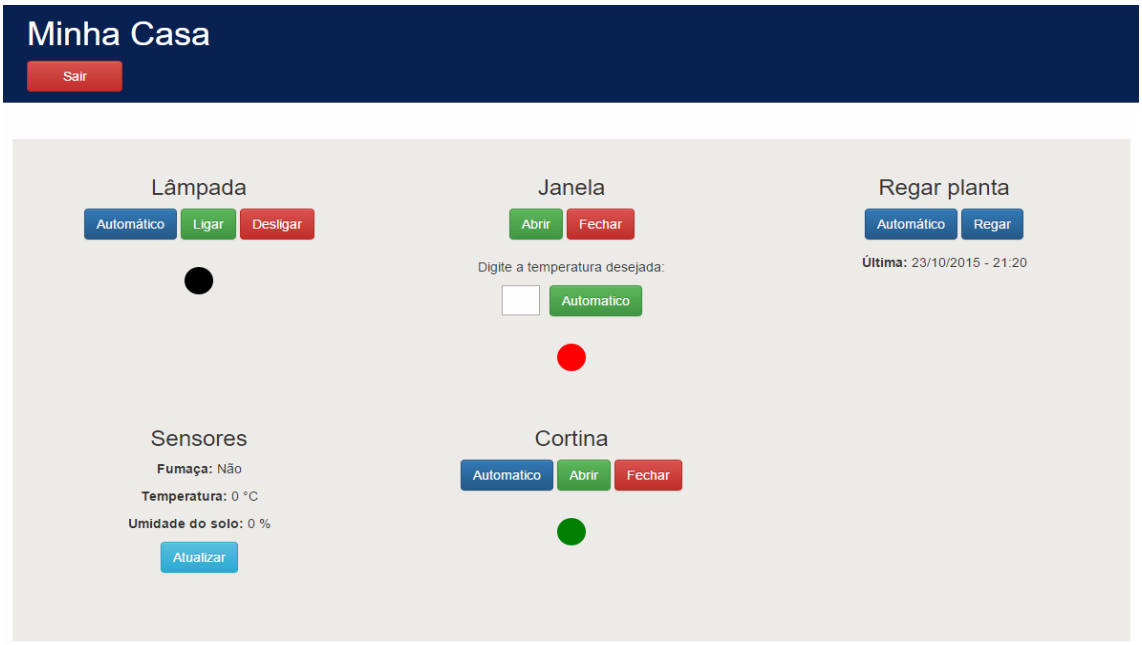

Figura 29 - Tela principal do sistema

O *layout* da página do sistema é responsivo, ou seja, se ajusta à resolução tela de qualquer dispositivo que tenha acesso à Internet permitindo o acesso de diversos dispositivos como *tablets*, *smartphones*, computadores pessoais, *notebooks* e *laptops*.

Quando o usuário clica em algum botão de ação, é executado um código em JAVASCRIPT ligado ao evento de clique do botão, esse código tem por objetivo passar parâmetros através do método POST da arquitetura REST, utilizando AJAX para não haver recarregamento da página, esses parâmetros são passados a um arquivo PHP que tem por objetivo verificar qual ação deve ser executada através dos parâmetros e executar o arquivo correspondente no servidor para envio da mensagem da ação desejada ao microcontrolador Tiva através do módulo nRF24L01.

4.6 Configuração do módulo nRF24L01 no microcontrolador Tiva

Para realizar a configuração do módulo nRF24L01 é necessário utilizar o protocolo SPI, para configuração e envio de dados do microcontrolador ao módulo foi desenvolvida uma biblioteca chamada **tSPI**, a biblioteca possui basicamente duas funções, **init_SPI** utilizada para inicializar a interface SPI do microcontrolador e **send_SPI** utilizada para envio de mensagens desse protocolo. A Figura 30 mostra a função **init_SPI**.

```
void init_SPI(void){

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0); //ativa a interface SPI

    //Configura os pinos utilizados para comunicacao SPI
    ROM_GPIOPinConfigure(GPIO_PA2_SSI0CLK);
    ROM_GPIOPinConfigure(GPIO_PA3_SSI0FSS);
    ROM_GPIOPinConfigure(GPIO_PA4_SSI0RX);
    ROM_GPIOPinConfigure(GPIO_PA5_SSI0TX);

    ROM_GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 |
                       GPIO_PIN_2);
    // Clock da SPI configurado em 5Mhz.
    ROM_SSIClockSourceSet(GPIO_PORTA_BASE, SSI_CLOCK_SYSTEM);
    ROM_SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0,
                           SSI_MODE_MASTER, 5000000, 8);
    // Liga SPI
    ROM_SSIEnable(SSI0_BASE);


    //Limpa os dados iniciais da SPI
    while(ROM_SSIDataGetNonBlocking(SSI0_BASE, &DataRx))
        {
        }
}
```

Figura 30 - Função init_SPI

A Figura 31 mostra a função **send_SPI**.

```
unsigned char send_SPI(unsigned char byte)
{
    pui32DataTx = byte; // byte a ser enviado

    ROM_SSIDataPut(SSI0_BASE, pui32DataTx); //coloca o byte para envio

    while(SSIBusy(SSI0_BASE)){} //aguarda o dado ser enviado

    ROM_SSIDataGet(SSI0_BASE, &pui32DataRx); //pega o retorno

    while(SSIBusy(SSI0_BASE)){} //aguarda o fim do retorno

    pui32DataRx &= 0xFF;

    return(pui32DataRx); //retorna dado recebido
}
```

Figura 31 - Função send_SPI

Para o uso do módulo nRF24L01 foi desenvolvida uma biblioteca chamada **RF24L01** composta por funções para escrita e leitura do modulo. A Figura 32 mostra as funções que compõem a biblioteca no arquivo **RF24L01.h**.

```
void init_Pin(void);
void initRF(void);
void init_Pin_leds(void);
void init_Pin_servo(void);
void pinCE_High(void);
void pinCE_Low(void);
void pinCSN_High(void);
void pinCSN_Low(void);
unsigned char regRead(unsigned char reg);
unsigned char regWrite(unsigned char reg, unsigned char value);
unsigned char regWriteFive(unsigned char reg, unsigned char val1, unsigned char val2, unsigned char val3, unsigned char val4, unsigned char val5);
unsigned char configPRX(void);
void PortDIntHandler(void);
```

Figura 32 - Funções que compõe a biblioteca RF24L01

As funções essenciais para a configuração e uso do módulo são a **regWrite** utilizada para escrever nos registradores do módulo, e a **regWriteFive** utilizada para escrever 5 bytes de dados nos registradores. A Figura 33 exibe a função **regWrite**.

```
unsigned char regWrite(unsigned char reg, unsigned char value){
    unsigned char data;

    pinCSN_Low(); // pino CSN colocado em nivel baixo

    data = send_SPI(CMD_W_REGISTER | reg); // envio do comando para escrita juntamente com o registrador

    send_SPI(value); //envio dos dados

    pinCSN_High(); //pino CSN colocado em nivel alto

    return data;
}
```

Figura 33 - Função regWrite

E a função **regWriteFive** é apresentada na Figura 34.

```
unsigned char regWriteFive(unsigned char reg, unsigned char val1, unsigned char val2, unsigned char val3, unsigned char val4, unsigned char val5)
{
    unsigned char data;

    pinCSN_Low(); // pino CSN colocado em nivel baixo

    data = send_SPI(CMD_W_REGISTER | reg); // envio do comando para escrita juntamente com o registrador
    //envio dos dados
    data = send_SPI(val1);
    data = send_SPI(val2);
    data = send_SPI(val3);
    data = send_SPI(val4);
    data = send_SPI(val5);

    pinCSN_High(); //pino CSN colocado em nivel alto

    return data;
}
```

Figura 34 - Função regWriteFive

Após o pino CSN ser colocado em nível baixo é necessário o comando 0x20 com a operação lógica OR com o registrador que deseja ser escrito, em seguida são enviados os dados, para completar a operação de escrita o pino de CSN é colocado em nível alto.

A configuração do módulo no modo RX é iniciada pela função **initRF** essa função é responsável por iniciar a fila que será utilizada para o armazenamento das mensagens recebidas pelo módulo e dar início a configuração do módulo colocando no modo **standby-I** configurando o pino **PWR_UP** do registrador **config** para o valor 1 em seguida é chamada a função **configPRX** que é utilizada para configurar o módulo no modo **RX**. A Figura 35 apresenta o código da função **initRF** e a Figura 36 mostra a função **configPRX**.

```
void initRF(void){

    criarFila(msgs);

    //Esperar 10.3ms para entrar em PowerDown
    int i;
    for(i=0;i<824000;i++){}

    //PWR_UP = 1 para entrar em modo standby-I
    regWrite(REG_CONFIG,0x0A);

    //Entra em modo PRX
    configPRX();
}
```

Figura 35 - Função initRF

```
pinCE_Low();

//Modo RX com PRIM_UP = 1 && PWR_UP = 1 e PINO IRQ refletindo RX LOW
regWrite(REG_CONFIG,0xBB);

//espera 1.5ms para entrar em StandBy-I
for(i=0;i<120000;i++){}

//Habilita o Endereço do PIPE RX 0
regWrite(REG_EN_RXADDR,0x01);

//Habilita auto Ack
regWrite(REG_EN_AA,0x01);

//SETA O REG_SETUP_AW para endereços de 5bytes
regWrite(REG_SETUP_AW,0x03);

//Seta o REG_SETUP_RETR para retransmitir a cada 250us e 3 tentativas
regWrite(REG_SETUP_RETR,0x03);

//Seta a frequencia do canal de transmissao
regWrite(REG_RF_CH,0x02);

//Setando AIR DATA RATE em 2MB Output POWER TX em 0dbm e ganho LNA
regWrite(REG_RF_SETUP,0x0F);

//Setando reg observe TX (monitora perda de pacotes) para o padrao 0x00
regWrite(REG_OBSERVE_TX,0x00);

//Setando reg CD carrier detect 0x00
regWrite(REG_RPD,0x00);

//Setando regs de quantidade de bytes dos pipes para o padrao
regWrite(REG_RX_PW_P0,0x00);
regWrite(REG_RX_PW_P1,0x00);
regWrite(REG_RX_PW_P2,0x00);
regWrite(REG_RX_PW_P3,0x00);
regWrite(REG_RX_PW_P4,0x00);
regWrite(REG_RX_PW_P5,0x00);

//Setando para o padrao o REG FIFO STATUS
regWrite(REG_FIFO_STATUS,0x11);

//habilita o payload de tamanho dinamico para o data PIPE 0
regWrite(REG_DYNPD,0x01);

//habilita funções de payload dinamico
data = regWrite(REG_FEATURE,0x07);

//Ativa comandos do modo dpl
    pinCSN_Low();

    send_SPI(CMD_ACTIVATE);

    send_SPI(0x73);

    pinCSN_High();


pinCE_High();
```

Figura 36 - Código da função configPRX

A configuração inicia colocando o pino CE em nível baixo, em seguida é configurado o registrador config com os bits **PRIM_RX** em nível 1, **PWR_UP** em nível 1 e o bit **MASK_RX_DR** em nível 1 para entrar em modo RX e para que quando uma mensagem seja recebida ocorra uma transição do nível alto para o nível baixo no pino **IRQ** do modulo, podendo assim entrar na interrupção que irá tratar o recebimento da mensagem que será abordada na seção 5.6 desse trabalho.

Para que a transação entre o TX (módulo ligado ao Raspberry Pi) e o RX (módulo ligado ao Tiva) ocorra é necessário que algum dos 5 registradores de endereço **RX_ADDR_Px** (onde x vai de 0 a 5) esteja configurado com o endereço do módulo TX, no nosso caso o endereço não foi alterado pois utilizaremos o endereçamento padrão que já vem configurado nesses registradores que é o endereço 0xE7E7E7E7E7 contido no registrador **RX_ADDR_P0**.

Também é importante ressaltar que o registrador REG_DYNPD é configurado para um tamanho de pacote dinâmico, ou seja, o módulo pode receber mensagens de 0 a 32 bytes de conteúdo.

Após o termino da execução da função **initRF** o módulo está pronto para receber e validar os pacotes capturados no meio. A Figura 37 mostra a ligação física entre o micro controlador Tiva e o módulo nRF24L01.

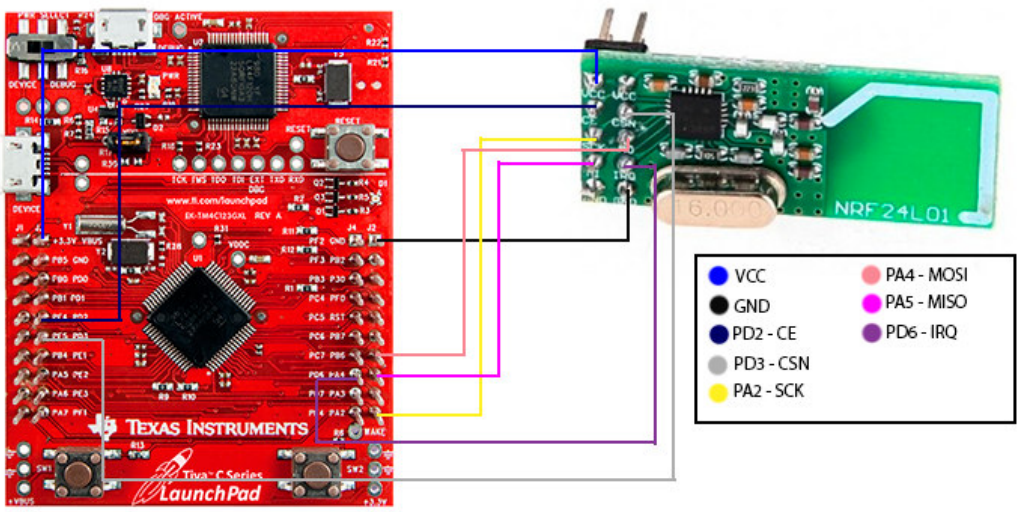

Figura 37 - Ligação física entre Tiva e Módulo nRF24L01

4.7 Recebimento de Mensagens (Microcontrolador Tiva)

Para o armazenamento das mensagens recebidas foi utilizada uma fila simplesmente encadeada, essa fila é montada por funções implementadas em uma biblioteca chamada **Fila**, a biblioteca possui as funções **criarFila** utilizada para criar e inicializar uma fila, **push** utilizada para enfileirar uma mensagem, **pop** utilizada para retirar uma mensagem da fila e **filaVazia** usada para verificar se a fila está vazia. A biblioteca também implementa a estrutura

da mensagem da fila com a **struct fila**. A Figura 38 mostra a estrutura da fila e a listagem das funções da biblioteca.

```
typedef struct fila
{
    volatile unsigned char data1;
    unsigned char data2;
    unsigned char data3;
    unsigned char data4;
    unsigned char data5;
    struct fila *prox;
}Fila;

void criarFila(Fila *f);
int filaVazia(Fila *f);
Fila *push(Fila *f, unsigned char d1, unsigned char d2, unsigned char d3, unsigned char d4, unsigned char d5);
Fila* pop(Fila* ini);
```

Figura 38 - Estrutura da fila e funções da biblioteca fila

A estrutura da fila consiste em 5 variáveis para armazenamento dos 5 bytes recebidos nas mensagens e um ponteiro que irá apontar para a próxima mensagem da fila.

Quando o módulo encontra um pacote valido o pino IRQ sofre uma transição do nível alto para o nível baixo e ativa uma interrupção na porta D do micro controlador, essa interrupção é tratada pela função **PortDintHandler**, nessa função as mensagens são retiradas do **FIFO RX** do módulo nRF24L01 através da comunicação SPI e colocadas na fila de mensagens. A Figura 39 mostra a função de tratamento de interrupção **PortDintHandler**.

```
status = GPIOIntStatus(GPIO_PORTD_BASE,true);

if( (status & GPIO_INT_PIN_6) == GPIO_INT_PIN_6){ //verifica se a interrupção foi no pino 6

    GPIOIntClear(GPIO_PORTD_BASE, GPIO_INT_PIN_6);// limpa flag de interrupção

    pinCSN_Low(); //coloca pino CSN em nivel baixo

    send_SPI(CMD_R_RX_PL_WID); //envia comando para leitura do tamanho dos dados recebidos

    pinCSN_High();//coloca pino CSN em nivel alto

    pinCSN_Low(); //coloca pino CSN em nivel baixo

    send_SPI(CMD_R_RX_PAYLOAD); ////envia comando para leitura dos dados do FIFO RX
    //leitura dos dados
    data1 = send_SPI(0xFF);
    data2 = send_SPI(0xFF);
    data3 = send_SPI(0xFF);
    data4 = send_SPI(0xFF);
    data5 = send_SPI(0xFF);
    data6 = send_SPI(0xFF);

    pinCSN_High(); //coloca pino CSN em nivel alto

    statusTMP = regRead(REG_STATUS);

    regWrite(REG_STATUS, 0x4E); //escreve no registrador de status

    // coloca as mensagens recebidas na fila
    if(data1 != 0x40){
        msgs = push(msgs,data1,data2,data3,data4,data5);
    }
    else{
        msgs = push(msgs,data2,data3,data4,data5,data6);
    }

}
```

Figura 39 - Função PortDintHandler

O comando para descarregar as mensagens recebidas pelo módulo é o **CMD_R_RX_PAYLOAD**, após o comando enviado são enviados dados com valor 0xFF apenas para recebermos os dados da mensagem, após a mensagem ser recebida ela é colocada na fila de mensagens.

4.8 Leitura de Sensores

Para a leitura dos sensores será usado o conversor ADC do micro controlador Tiva, possuindo 12 bits de resolução e tensão de referência de 3V, isso é transforma grandezas elétricas em um valor de 0 a 4096. O ADC possui 8 canais, nesse trabalho usaremos 4 para leitura dos sensores de luminosidade (fotoresistor), temperatura (LM35) e fumaça (MQ-2).

Existem 4 sequencias para uso do ADC cada uma utiliza uma quantidade de canais do ADC, a sequência 0 utiliza os 8 canais, a sequência 1 utiliza 4 canais, a sequência 2 utiliza 4 canais e a sequência 3 utiliza 1 canal. Nesse trabalho iremos utilizar a sequência 2. A inicialização e configuração é mostrada na Figura 40.

```
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);

//Pinos de ADC = PE0, PE1, PE2, PE3
ROM_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0 | GPIO_PIN_3 | GPIO_PIN_2 | GPIO_PIN_1);

ROM_ADCReferenceSet(ADC0_BASE,ADC_REF_INT); //referencia de 3V.

ROM_ADCSequenceDisable(ADC0_BASE, 1);
ROM_ADCSequenceDisable(ADC0_BASE, 2);
ROM_ADCSequenceDisable(ADC0_BASE, 0);
ROM_ADCSequenceDisable(ADC0_BASE, 3);

ROM_ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);

ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_CH0);
ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_CH2);
ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_CH1);
ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_CH3 | ADC_CTL_IE |
                             ADC_CTL_END);
ROM_ADCSequenceEnable(ADC0_BASE, 2);
ROM_ADCIntClear(ADC0_BASE, 2);
```

Figura 40 - Inicialização e configuração ADC

O canal 0 do ADC corresponde ao sensor de luminosidade, o canal 3 corresponde ao higrômetro, o canal 2 corresponde ao sensor de fumaça, e o canal 1 corresponde ao sensor de temperatura. Após a inicialização e configuração é possível realizar a leitura das entradas analógicas como mostra a Figura 41.

```
ROM_ADCProcessorTrigger(ADC0_BASE, 2); //inicia a conversao
while(!ROM_ADCIntStatus(ADC0_BASE, 2, false)){} //aguarda a conversao terminar
ROM_ADCIntClear(ADC0_BASE, 2); //limpa interrupcao da sequencia 2
ROM_ADCSequenceDataGet(ADC0_BASE, 2, pui32ADC0Value); //recebe os dados da conversao e coloca no vetor pui32ADC0Value
```

Figura 41 - Aquisição de dados do ADC

O início da conversão é via software por meio da função **ROM_ADCProcessorTrigger**, em seguida é aguardada o termino da conversão e então a interrupção é limpa, após esses passos os dados são recebidos do ADC e colocados em um vetor. Nesse processo são lidos todos os canais do ADC e colocados nesse vetor.

4.9 Uso dos Atuadores

O uso dos atuadores é feito no código principal do micro controlador Tiva, a estrutura do código foi feita com uma abordagem em laço infinito que passa por 8 etapas dentro de seu código, em sua etapa inicial lê uma mensagem recebida que está na fila de mensagens e então avança para as outras etapas, cada uma das outras etapas corresponde a um atuador, quando chega na última etapa retorna a etapa inicial, avançando novamente pelas outras etapas, formando um laço infinito. A Figura 42 mostra o fluxograma do código em laço infinito sistema.

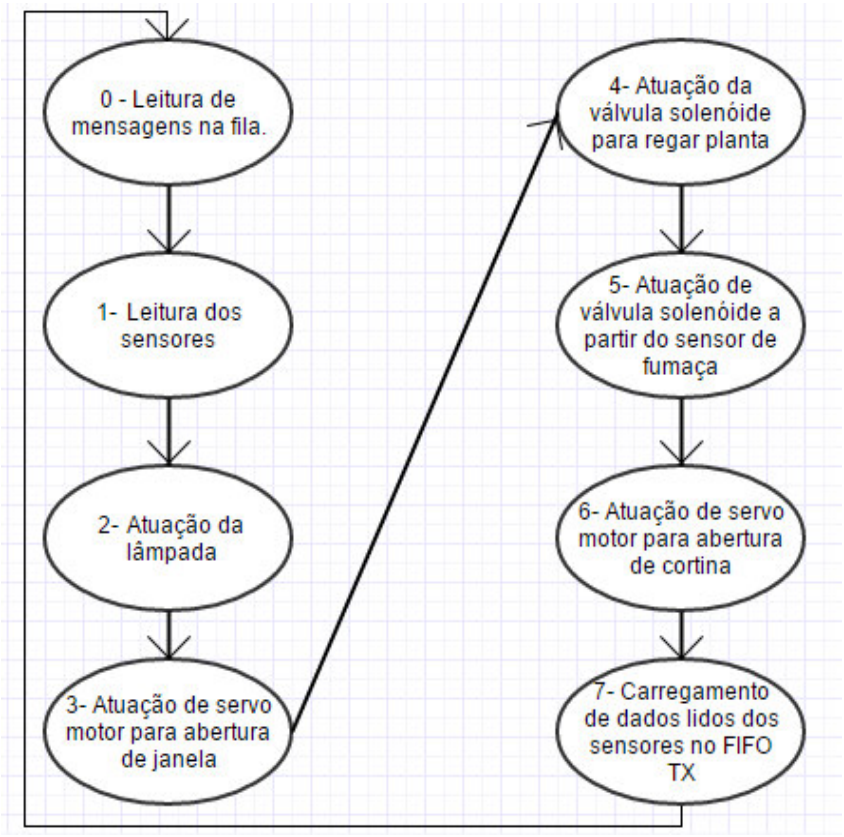

Figura 42 - Fluxograma do código em loop infinito

A etapa inicial (etapa 0), é a etapa responsável pela retirada das mensagens da fila, após lida e armazenada em variáveis a mensagem é retirada da fila e a então avança para a próxima etapa. A Figura 43 mostra o código da etapa 0.

```
case 0:
    if((filaVazia(msgs))==0){
        mdata1 = msgs->data1;
        mdata2 = msgs->data2;
        mdata3 = msgs->data3;
        mdata4 = msgs->data4;
        mdata5 = msgs->data5;
        msgs = pop(msgs);

    }
    estado = 1;
    break;
```

Figura 43 - Etapa 0 do laço infinito

A etapa 1, é responsável pela leitura dos sensores de temperatura e fumaça. Nessa etapa são lidos os valores convertidos do ADC e salvos em duas variáveis chamadas **Temp** para temperatura e **Fum** para o nível de fumaça no ambiente. Posteriormente esses valores são utilizados para acionar atuadores. A Figura 44 mostra o código da etapa 1.

```
case 2: // sensor de temperatura e fumaça
    ROM_ADCProcessorTrigger(ADC0_BASE, 2);
    while(!ROM_ADCIntStatus(ADC0_BASE, 2, false)){}
    ROM_ADCIntClear(ADC0_BASE, 2);
    ROM_ADCSequenceDataGet(ADC0_BASE, 2, pui32ADC0Value);
    //conversao da temperatura
    TempAux = ((pui32ADC0Value[2])*1000)/4096;
    Temp = ((300 * ((float)TempAux/100)));
    //conversao do nivel de fumaca
    Fum = ((pui32ADC0Value[1]-200)*10000)/4096;

    estado = 3;
    break;
```

Figura 44 - Código da etapa 1

Em seguida o sistema avança para a etapa 2, que é responsável pelo atuador da lâmpada, nessa etapa a lâmpada é acionada, desligada ou colocada em modo automático, a mensagem para ligar a lâmpada é 0100000001, para desligar é 0100000002 e para o modo automático é 0100000099.

Para a lâmpada ser acionada é utilizado um pino de saída do micro controlador, o pino é colocado em nível alto para acionar o rele que aciona a lâmpada, para desligar a lâmpada o pino é colocado em nível baixo. Quando colocado no modo automático o micro controlador configura uma variável chamada **ledAuto** que indica se a lâmpada está no modo automático ou não. A Figura 45 mostra o circuito utilizado para acionar o rele, esse circuito também é utilizado para acionar reles de outros atuadores do sistema.

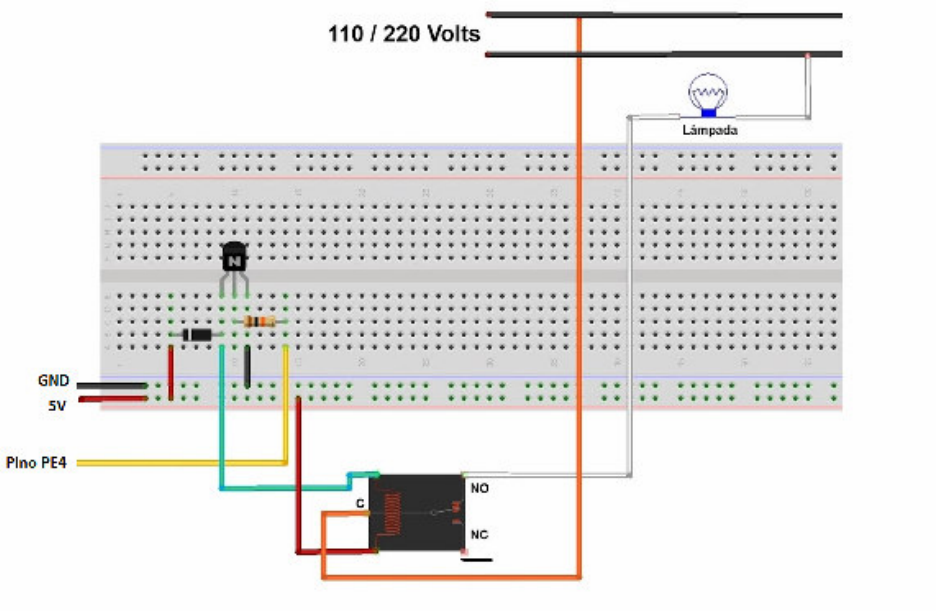

Figura 45 - Circuito usado para acionar o relé

No modo automático é realizada a leitura do fotoresistor, os valores lidos vão de 0 a 4095, quanto maior esse valor maior a quantidade de luz, por meio de testes foi estipulado que o valor de 2000 seria usado para indicar o acionamento ou não da lâmpada. Quando a leitura está abaixo de 2000 a lâmpada é acesa o que indica pouca luz no local, quando está acima de 2000 a lâmpada permanece apagada pois indica que o ambiente está iluminado. Para sair do modo automático o usuário deve acender ou apagar a lâmpada por meio dos comandos enviados via *web*. A Figura 46 mostra o código do modo automático da lâmpada.

```
//marca flag de modo automatico
ledAuto = 1;
//leitura do adc
ROM_ADCProcessorTrigger(ADC0_BASE, 2);
while(!ROM_ADCIntStatus(ADC0_BASE, 2, false)){}
ROM_ADCIntClear(ADC0_BASE, 2);
ROM_ADCSequenceDataGet(ADC0_BASE, 2, pui32ADC0Value);
//verificacao do valor do fotoresistor
if(pui32ADC0Value[0] <= 2000){
    ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4, GPIO_PIN_4);
}else if(pui32ADC0Value[0] > 2000)
{
    ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4, 0);
}
```

Figura 46 - Código do modo automático da lâmpada

Em seguida a o sistema entra na etapa 3, essa etapa é responsável por abrir e fechar uma janela, utilizando um servo motor com capacidade de rotação de 180 graus, a abertura e fechando é feita com um sinal PWM enviado ao servo motor, a largura de banda desse sinal

indica a posição em que o motor irá ficar. A abertura da janela pode ser feita de modo "manual" ou automático. No modo "manual" o usuário decide utilizando-se da interface do sistema se abre ou fecha a janela. No modo automático o usuário informa a temperatura na qual a janela abre se a temperatura ambiente for superior a escolhida pelo usuário e fecha se for menor. A Figura 47 mostra o código do modo automático.

```
autoTemp = (int)mdata4*10;

if(Temp >=  autoTemp){
    ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 8); //largura de pulso para abrir
    ROM_PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
    ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0); //habilita saída da pwm

}else{
    ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 2.5); //largura de pulso para fechar
    ROM_PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
    ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0); //habilita saída da pwm
}
```

Figura 47 - Modo automático da janela

O sistema então avança para a etapa 4, essa etapa é responsável pela abertura de uma válvula solenóide ligada a rede de água para regar uma planta, ao entrar nessa etapa a válvula solenóide é aberta por aproximadamente 4 segundos e fecha após o termino desse tempo, a data e a hora em que a planta foi regada ficam salvas no banco de dados e são exibidas na interface do sistema para o usuário. A abertura da válvula solenóide é feita utilizando o circuito de acionamento com relé (o mesmo utilizado para acionar a lâmpada). A Figura 48 mostra o código da etapa 4.

```
ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_0, GPIO_PIN_0); //abre solenoide
for(auxSol=0; auxSol < 10000000; auxSol++){} //aguarda aproximadaente 10 segundos
ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_0, 0); // fecha solenoide
```

Figura 48 - Código de abertura de solenóide para regar uma planta

Ainda na etapa 4 temos o modo automático para abertura da válvula solenóide se a umidade do solo medida pelo higrômetro está baixa, no nosso caso adotamos uma umidade mínima de 30% para que a válvula solenóide seja aberta, ela permanece aberta por aproximadamente 2 segundos e fechada por 2 segundos até que a umidade lida pelo sensor passe do valor estabelecido. Quando o modo automático é acionado pelo usuário e o microcontrolador recebe a mensagem para esse modo a variável **regarFlag** recebe o valor de 1, que indica que o modo automático está ativo. A Figura 49 contém o código do modo automático da válvula solenóide.

```
    if(regarFlag == 1){
        if(Umidade < 30){
            ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7, GPIO_PIN_7); //abre solenoide
            for(auxSol=0; auxSol < 2000000; auxSol++){} //aguarda aproximadamente 2 segundos
            ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7, 0); // fecha solenoide
            for(auxSol=0; auxSol < 2000000; auxSol++){} //aguarda aproximadamente 2 segundos
        }
    }
```

Figura 49 - Código do modo automático para regar planta

Após esse procedimento o sistema entra na etapa 5, essa etapa é responsável por verificar o nível de fumaça do ambiente pela medição do sensor e acionar uma válvula solenóide ligada a rede de água para um sistema contra incêndio, a válvula solenóide permanece aberta até que os níveis de fumaça se estejam baixos. O sensor mede de 300 a 10000 ppm (partes por milhão) de partículas de fumaça no ar, a partir de alguns testes foi estipulado o valor de 2500 ppm para acionamento da válvula solenóide. A leitura do sensor é feita na etapa 2 e armazenada na variável fumaça após conversão da entrada do ADC para ppm. O circuito para acionamento da válvula solenóide é o circuito de acionamento por relé (o mesmo utilizado para acionar a lâmpada). A Figura 50 mostra o código da etapa 5.

```
if(Fum > 2500){ //verifica nivel de fumaça
    ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5, GPIO_PIN_5); //aciona a solenoide
}else{
    ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5, 0); // fecha solenoide
}
estado = 6;
```

Figura 50 - Código para acionamento de solenóide de acordo com nível de fumaça

O sistema então avança para a etapa 6, é a etapa responsável pela abertura das cortinas, a abertura é feita por meio de um servo motor com capacidade de rotação 180 graus (o mesmo utilizado para a abertura da janela). O código de abertura e fechamento funciona da mesma maneira do que o código da etapa 3. O modo automático da abertura das cortinas leva em consideração o nível de luminosidade do ambiente, quando o ambiente externo está claro a cortina será aberta, se o ambiente estiver escuro a cortina será fechada. A Figura 51 mostra o código do modo automático da etapa 6.

```
if(cortinaAuto !=0){
    if(cortinaAuto >  2000){ //quando esta claro fora de casa abre a cortina
        ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 8);
        ROM_PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, true);
        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);

    }else{ //quando esta escuro fora de casa fecha a cortina
        ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 2.5);
        ROM_PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, true);
        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
    }
}
```

Figura 51 - Código do modo automático da cortina

A variável cortinaAuto possui o valor de luminosidade obtido na entrada do ADC e é usado para verificar se o ambiente está escuro ou claro para a tomada de decisão. A partir de testes foi obtido o valor de 2000 como limiar entre claro e escuro.

O sistema então avança para a etapa 7, etapa responsável por carregar as leituras dos sensores no FIFO TX para posterior envio ao PTX para exibição ao usuário na página, após receber uma mensagem os dados dos sensores são carregados no FIFO em seguida o PTX envia uma mensagem que não toma nenhuma ação dentro do laço infinito apenas para obter os dados dos sensores por meio do pacote ACK. A Figura 52 mostra o código de carregamento dos dados lidos dos sensores no FIFO TX.

```
pinCSN_Low();
send_SPI(CMD_FLUSH_TX); //Limpa o FIFO de envio
pinCSN_High();

pinCSN_Low();

send_SPI(CMD_W_ACK_PAYLOAD); //Comando para carregar dados no FIFO do ACK
send_SPI((unsigned char)sendFum); //Dados do sensor de fumaça
send_SPI((unsigned char)sendTemp); //Dados do sensor de Temperatura
send_SPI((unsigned char)Umidade); //Dados do sensor de Umidade do solo
send_SPI(0x00);
send_SPI(0x00);

pinCSN_High();
```

Figura 52 - Código para carregamento de dados dos sensores no FIFO TX

Após passar pelas 8 etapas o sistema retorna a etapa 0 para pegar outra mensagem da fila de mensagens. Os modos “manuais” somente são executados se as mensagens são endereçadas a eles, quando algum atuador está no modo automático o código desse modo sempre será executado no laço infinito do sistema. O quadro 3 mostra as mensagens que são enviadas do servidor web, seus respectivos atuadores e suas funções.

Mensagem	Atuador	Função
0100000001	Lâmpada	Ligar
0100000002	Lâmpada	Desligar
0100000099	Lâmpada	Modo automático
0300000001	Janela	Abrir
0300000002	Janela	Fechar
030000XX99	Janela	Modo automático (XX é a temperatura informada pelo usuário)
0400000001	Solenóide	Regar planta
0400000099	Solenóide	Modo automático para regar planta
0600000001	Cortina	Abrir
0600000002	Cortina	Fechar
0600000099	Cortina	Modo automático
0700000000	Nenhum	Carregar dados lidos dos sensores no FIFO do ACK
0800000000	Nenhum	Não possui função especifica, usada apenas para obter os dados dos sensores após carregamento no FIFO do ACK no lado do RX.

Quadro 5 - Tabela de mensagens

4.10 Recuperação do sistema

Uma outra funcionalidade implementada no sistema é que ele é capaz de recuperar seu último estado antes de um desligamento e reconfigurar os atuadores de acordo com esse estado. Toda ação efetuada pelo usuário é salva em um banco de dados, tornando possível a recuperação do último estado caso ocorra um desligamento do sistema.

É importante mostrar a estrutura do banco de dados para que se possa ter uma melhor compreensão de como o arquivo que realiza a recuperação funciona. A Figura 53 mostra a estrutura do banco de dados do sistema.

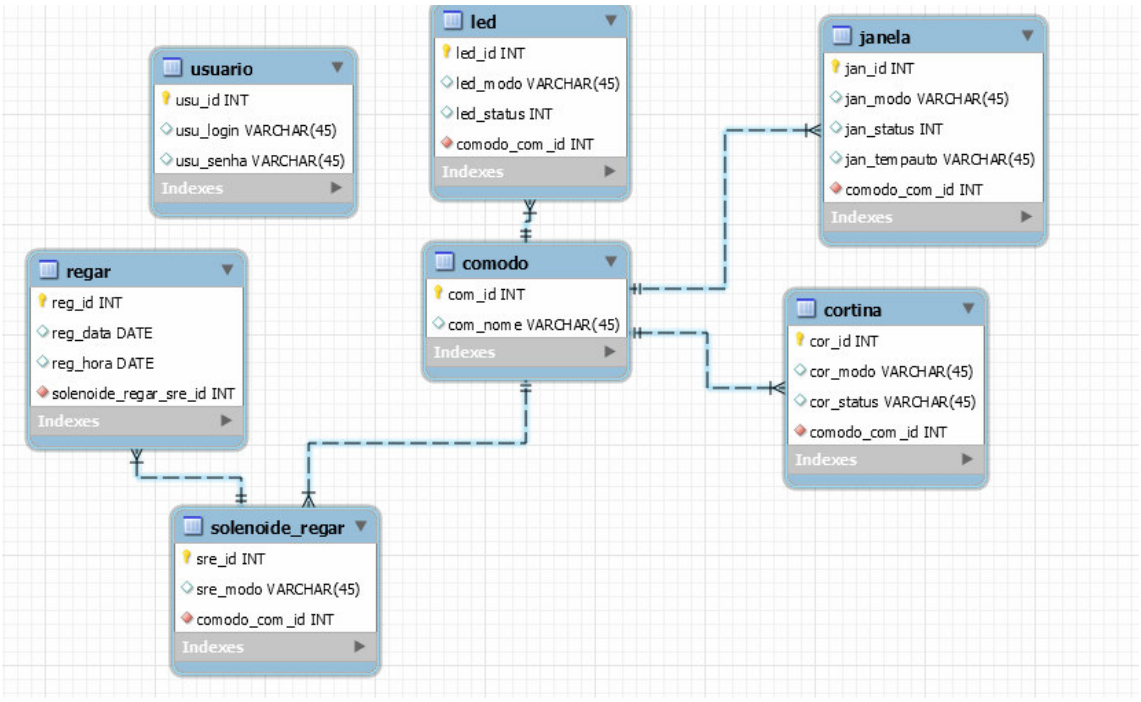

Figura 53 - Estrutura do banco de dados do sistema

A tabela usuário é responsável por guardar os dados de acesso ao sistema, a tabela comodo refere-se aos cômodos onde estarão os atuadores, as tabelas led (lâmpadas), janela (servo motores), solenóide_regar (solenóides) e cortina (servo motores) são usadas para seus correspondentes atuadores, essas tabelas possuem um campo comodo_com_id que indica em qual como estão os atuadores cadastrados. No nosso caso usaremos apenas um cômodo para desenvolvimento do sistema.

A tabela regar serve para gravar quando foi a última data e hora que alguma solenóide para regar plantas foi acionada manualmente.

Com a estrutura do banco de dados já apresentada podemos entrar no código de recuperação do sistema, o arquivo responsável pela recuperação do sistema é um arquivo php chamado **recover.php** ele é executado no boot do sistema após a configuração do módulo nRF24L01, para isso foi adicionado a seguinte linha ao final do script **rc.local** do sistema operacional raspbian.

/usr/bin/php5 /var/www/recover.php

Esse arquivo **recover.php** é responsável por recuperar as informações salvas no banco de dados e enviar mensagens para o microcontrolador para que os atuadores estejam configurados da maneira que estavam salvas no banco antes de um eventual desligamento do

sistema. A Figura 54 mostra a recuperação dos dados do atuador lâmpada, a Figura 55 mostra o tratamento dos dados recuperados e envio das mensagens para o micro controlador.

```php
//Status da lampada

$queryLampada = mysql_query("SELECT * FROM led WHERE led_id = 1");

$resultLampada = mysql_fetch_array($queryLampada);

$modoLampada = $resultLampada['led_modo'];

$statusLampada = $resultLampada['led_status'];
```

Figura 54 - Recuperação dos dados do atuador lâmpada

```php
//Recover Lampada

if($modoLampada == 'automatico')
{

    passthru("sudo /var/www/writeRegFive_RF2 cmd_w_tx_payload 1 0 0 0 0 0 0 0 9 9",$return);
    print_r($return);

}else if(($modoLampada == 'manual')&&($statusLampada == 1))
{

    passthru("sudo /var/www/writeRegFive_RF2 cmd_w_tx_payload 1 0 0 0 0 0 0 0 1 0",$return);
    print_r($return);

}else if(($modoLampada == 'manual')&&($statusLampada == 0))
{

    passthru("sudo /var/www/writeRegFive_RF2 cmd_w_tx_payload 1 0 0 0 0 0 0 0 2 0",$return);
    print_r($return);

}
```

Figura 55 - Código para envio de ação ao micro controlador

4.11 Adição de microcontroladores

É possível ainda realizar a comunicação com mais de um microcontrolador utilizando o mesmo Paspberry Pi apenas mudando o endereço de configuração de envio do módulo nRF24L01 ligado ao Raspberry Pi (PTX).

Para realizar a troca do endereço de envio do módulo PTX é necessário primeiramente colocá-lo em modo Power Down para reconfiguração dos registradores do módulo, para isso o bit PWR_UP é colocado em nível lógico baixo (0), após a entrada nesse modo é feita a mesma configuração descrita no subcapítulo 4.3 com adição de uma linha de código para alteração do endereço de transmissão presente no registrador TX_ADDR, no lado do mircrocontrolador adicionado é necessário também configurar o módulo nRF24L01 ligado a

ele (PRX) seguindo o mesmo procedimento descrito no subcapitulo 4.6, com a adição de uma linha de código para alteração do endereço no registrador RX_ADDR_P0 e deve conter o mesmo endereço que foi colocado no TX_ADDR do PTX.

Assim é possível o envio de mensagens do Raspberry Pi para dois (ou mais) microcontroladores com seus respectivos módulos de comunicação configurados com endereços distintos, devido a capacidade dos módulos descartarem pacotes que não tem o seu endereçamento igual aos configurados nos módulos PRX.

4.12 Testes

Os testes foram realizados em uma rede interna, com o servidor *web* configurado com um endereço IP fixo e ligado diretamente ao computador por meio de um cabo de rede, assim foi possível acessar a interface *web* através do navegador com o endereço IP configurado no servidor e realizar os testes em todo o sistema.

4.13 Protótipo

Foi desenvolvido um protótipo capaz de mostrar o funcionamento do sistema e seus atuadores. As Figuras 56, 57, 58 e 59 mostram a aparência física do protótipo.

Figura 56 - Visão frontal do protótipo

Figura 57 - Visão interna do protótipo com a engrenagem da cortina

Figura 58 - Visão da cortina e lâmpada

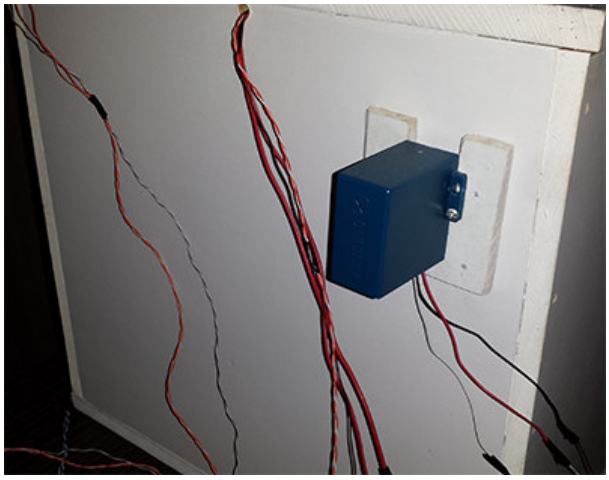

Figura 59 - Visão exterior do protótipo

5 CONCLUSÃO

As tecnologias utilizadas nesse trabalho permitem que o usuário tenha acesso a dados sobre a sua residência (temperatura, umidade do solo de um jardim e os níveis de fumaça do ambiente), que ele possa tomar decisões e atuar em alguns aspectos da sua residência (acender/apagar uma lâmpada, regar uma planta, abrir/fechar uma cortina e abrir/fechar uma janela), bem como colocar esses aspectos em um modo automático no qual os dados obtidos sobre a sua residência são utilizados para a atuação autônoma de alguns objetos.

Todas essas possibilidades são apresentadas ao usuário por meio de uma página *Web*, que possui um *layout* responsivo sendo capaz adaptar-se a qualquer dispositivo que possua Internet. O acesso a está página é garantido por meio de um usuário e senha (criptografada com o algoritmo MD5) cadastrados no banco de dados.

O Raspberry Pi possui poder computacional suficiente para aguentar a carga de processamento de um servidor Web, banco de dados MySQL, compilação e processamento da linguagem PHP e linguagem C, realizar a comunicação através do protocolo SPI. O dispositivo mostrou-se importante para suportar a carga do servidor e realizar o processamento por traz da página de controle do sistema que permite o envio e recebimento de dados através do módulo nRF24L01.

O uso do módulo de comunicação por radiofrequência nRF24L01 permitiu que o servidor web (Raspberry Pi) possa realizar a troca de mensagens, tanto envio quanto recebimento, com o microcontrolador (Tiva) responsável pela leitura dos sensores e acionamento dos atuadores, o que abre a possibilidade de serem adicionados mais microcontroladores responsáveis por outros aspectos da residência.

O microcontrolador Tiva é o responsável por receber as mensagens, salva-las em uma fila, e posteriormente executar as ações relacionadas a essas mensagens, o dispositivo é importante pois possui interface de comunicação SPI, utilizada para troca de mensagens com o módulo nRFL01, possui modulo gerador de PWM, conversor analógico digital de 12 bits com 8 canais, pinos de propósito geral, capacidade de memória para suportar uma fila, pinos de alimentação para os sensores, dentre muitas outras características. Isso permitiu realizar a leitura de sensores, acionar atuadores e armazenar mensagens.

O desenvolvimento do protocolo de comunicação *enhanced shockburst* para a troca de mensagens no sistema trouxe a capacidade de comunicação por radiofrequência entre um Raspberry Pi (servidor *web*) e um (ou mais) microcontrolador Tiva essa comunicação é considerada o núcleo desse trabalho, tendo em vista que uma mensagem é um comando

enviado pelo usuário para o microcontrolador. Olhando apenas para o início e o fim do processo a comunicação permitiu que um comando enviado pelo usuário através da *web* seja capaz de ser transmitido do servidor (Raspberry Pi) para o controle de qualquer cômodo (microcontrolador Tiva), independente da finalidade do comando enviado.

Como trabalhos futuros novos microcontroladores podem ser adicionados a residência utilizando o mesmo servidor e novas funcionalidades como a utilização de sensores de corrente para medir o consumo de tomadas, utilização de sensores de fluxo de água para saber o consumo de torneiras e a utilização de sensores de chuva para fechar uma janela que esteja aberta no momento de alguma precipitação.

O protótipo desenvolvido buscou mostrar como o sistema pode trazer praticidade, conforto e facilidade em alguns aspectos simples do dia-a-dia do usuário e também possibilitar que seja possível realizar ações apenas tendo um dispositivo que possa conectar à Internet e uma conexão.

REFERÊNCIAS

AARTS, Emili; RUYTER, Boris de. **New research perspectives on ambient intelligence**. J. Ambient Intell. Smart Environ., v. 1, n. 1, p. 5–14, 2009

ACAMPORA, Giovanni; COOK, Diane J.; RASHIDI, Parisa; VASILAKOS, Athanasios V. A survey on ambient intelligence in healthcare. **IEEE**, v. 101, n. 12, December 2013, p. 2420-2494.

AL-KUWARI, Ali Mohammed A. H.; ORTEGA-SANCHEZ, Cezar; SHARIF, Atif; POTDAR, Vidyasagar. **User Friendly Smart Home Infrastructure: BeeHouse**. In: 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011), 31 May -3 June 2011, p. 257-262.

ATHREYA, Arjun P; TAGUE, Patrick. **Network Self-Organization in the Internet of Things**. In: 2013 IEEE International Workshop of Internet-of-Things Networking and Control (IoT-NC), 2013, p. 25-33.

ATZORI Luigi; IERA Antonio; MORABITO Giacomo; **The Internet of Things: A survey;** Computer Networks; Computer Networks, v. 54, n. 15, p. 2787–2805, 2010.

BARI; Nima; MANI, Ganapathy; BERKOVICH; Simon**. Internet of Things as a Methodological Concept;** 2013 Fourth International Conference on Computing for Geospatial Research and Application; p.48-55.

BARROS, Edna; CAVALCANTE, Sérgio; **Introdução aos sistemas embarcados;** Grupo de Engenharia da Computação GRECO – Centro de Informática CIn – Universidade Federal de Pernambuco – UFPE; p.1-2.

BORGES, L. P.; Dores, R. C., **Automação predial sem fio utilizando bacnet/zigbee com foco em economia de energia**. 2010, 76f. Trabalho de conclusão de curso – Curso de Gradua;cão em Engenharia de Controle e Automação – UNB, Brasília, 2010.

BRUGNARI, A.; MAESTRELLI, L. H. M., **AUTOMAÇÃO RESIDENCIAL via WEB**. 2010, 36f. Trabalho de conclusão de curso – Curso de Graduação em Engenharia de Computação – PUC-PR, Curitiba, 2010.

CHLOUBA, Tomáš. **Design patterns in ambient intelligence**. In: 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009, p. 277-282.

COETZEE Louis; EKSTEEN, Johan; **The Internet of Things – Promise for the Future? An Introduction;** IST-Africa 2011 Conference Proceedings; 2011, p.1-9.

DHINGRA, Vandana; ARORA, Anita; **Pervasive Computing: Paradigm for New Era Computing;** First International Conference on Emerging Trends in Engineering and Technology; p.349-354, 2008.

ESTELLER-CURTO, Roger; CERVERA, Enric; POBIL, Angel P. del; MARIN Raul; **Proposal of a REST-based architecture server to control a robot**. In: Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, p.708-710, 2012.

FAN, Peng-fei; ZHOU, Guang-zhao. Analysis of the Business Model Innovation of the Technology of Internet of Things in Postal Logistics, **IEEE**, 2011, p. 532-536.

HANWEI ELETRONICS CO. Technical data MQ-2 gas sensor. Disponível em: <https://www.seeedstudio.com/depot/datasheet/MQ-2.pdf>. Acessado em 20 de setembro de 2015.

HENDERSON, Gordon. Wiring Pi Library. Disponível em: < http://wiringpi.com >. Acessado em 10 de setembro de 2014.

INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems and EPoSS, **The Internet of Things in 2020: A Road map for the future**, September 5, 2008, p. 1-32. Disponível em: <http://www.smart-systems-integration.org/public/documents/publications/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v3.pdf>. Acesso em: 05 out. 2014.

IST Advisory Group. **Scenarios for Ambient Intelligence**. In: 2010 European Commission, p. 1-58. Disponível em: <ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf>. Acesso em: 25 set. 2014.

ITU, Internet Reports. **2005: The Internet of Things**, 2005, ITU Internet Report. Disponível em: <http://www.itu.int/osg/spu/publications/internetofthings/>. Acesso em: 05 out. 2014.

JIA, Xiaolin; FENG, Quanyuan; MA, Chengzhen. **An efficient anti-collision protocol forRFID tag identification**, IEEE Communications Letters, v.14, n.11,p.1014-1016, 2010.

JIA, Xiaolin; FENG, Quanyuan; FAN, Taihua; LEI, Quanshui. **RFID technology and its applications in Internet of Things (IOT)**. IEEE, 2012, p. 1282-1285.

KO, Hoon; RAMOS, Carlos; **A Survey of context classfication for intelligent systems research for Ambient Intelligence;** 2010 International Conference on Complex, Intelligent and Software Intensive Systems. p.746-747.

LEGNER, Christine; THIESSE, Frédéric. **RFID-based maintenance at Frankfurt airport**. In IEEE Pervasive Computing, v. 5, n 1, p. 34-39, 2006.

MORGANTI, Avelino. **Desenvolvimento de um sistema de automação residencial com interface web,para controle e segurança de residências**, PIBITI – CNPq - Unoesc, 2014

NORDIC SEMICONDUCTOR. nRF24L01 Single Chip 2.4GHz Transceiver Product Specification. Disponível em: < https://www.nordicsemi.com/chi/content/download/2730/34105/file/nRF24L01_Product_Specification_v2_0.pdf >. Acessado em 3 de junho de 2015.

OLIVEIRA, Diego Vanderlei Guerreiro de; PETREK, Felipe Joly. Sistema de automação residencial controlado via web. 2014. 165 f. Trabalho de Conclusão de Curso (Graduação) – Universidade Tecnológica Federal do Paraná, Curitiba, 2014.

RAMOS, Carlos; AUGUSTO, Juan Carlos; SHAPIRO, Daniel. Ambient intelligence the next step for artificial intelligence. **IEEE Intelligent Systems**, v. 23, n. 2, Nov. 2008, p. 15-18.

SADRI, Fabri. Ambient intelligence: a survey. **ACM Comput. Surv**., v. 43, n. 4, p. 36:1–36:66, Oct. 2011.

SCHNEIDER DE OLIVEIRA, André; SOUZA DE ANDRADE, Fernando**. Sistemas Embarcados: Hardware e Firmware na prática.** 2ª Edição. Editora Érica 2010 - 320p.

SEED WIKI.Grove – Moisture Sensor.Disponível em <http://www.seeedstudio.com/wiki/Grove_-_Moisture_Sensor> Acessado em 25 de outubro de 2015.

SHENZHEN SENBA OPTICAL & ELECTRONIC CO., LTD. GL55 Series Photoresistor. Disponível em: < http://akizukidenshi.com/download/ds/senba/GL55%20Series%20Photoresistor.pdf >. Acessado em 30 de setembro de 2015.

GALASSI, Thomas Tadeu. UM ESTUDO EXPLORATÓRIO SOBRE SISTEMAS OPERACIONAIS EMBARCADOS. Disponível em: < http://www.fatec.edu.br/revista/wp-content/uploads/2013/06/Um-estudo-explorat%C3%B3rio-sobre-sistemas-operacionais-embarcados.pdf >. Acessado em 25 de outubro de 2015.

TEXAS INSTRUMENTS. Datasheet LM35. Disponível em: < http://www.ti.com/lit/ds/symlink/lm35.pdf >. Acessado em 20 de setembro de 2015.

TEXAS INSTRUMENTS. Tiva™ TM4C123GH6PM Microcontroller DATA SHEET. Disponível em: < http://www.ti.com/lit/ds/spms376e/spms376e.pdf >. Acessado em 05 de junho de 2015.

VINICIUS DE MELO EUZÉBIO, Michel; RIBEIRO DE MELLO, Emerson; **DroidLar: Automação residencial através de um celular Android**; 2011; p.1-3

WEBER, Rolf H. Internet of Things - Need for a New Legal Environment?. **Computer Law & Security Review**, 2009, p. 522-527.

WEISER, Mark. **Hot topics: Ubiquitous computing**, IEEE computer, October 1993, p. 71-72.

WU, Miao; LU, Ting-lie; LING Fei-Yang; SUN Ling; DU Hui-Ying; **Research on the architecture of Internet of things**. In: 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), 2010, p. V5-484-V5-487.

APÊNDICE A

LISTAGEM DO CÓDIGO PRINCIPAL DO MICROCONTROLADOR TIVA

```
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include "inc/hw_memmap.h"
#include "driverlib/rom.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/pwm.h"
#include "utils/uartstdio.h"
#include "RF24L01.h"
#include "tSPI.h"
#include "fila.h"

void init_UART(void){
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
        ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
        ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
        ROM_UARTConfigSetExpClk(UART0_BASE, ROM_SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                         UART_CONFIG_PAR_NONE));
}

int
main(void)
{
        int estado = 0;
        int ledAuto = 0;
        int pinLED = 0;
        uint32_t pui32ADC0Value[4];
        float TempAux;
        float Temp;
        int autoTemp=0;
        int cortinaAuto=0;
        int cortinaFlag=0;
        int regarFlag=0;
        int Fum;
        int Umidade;
        int auxSol=0;
        int auxCnt=0;
        int sendFum = 0;
        float sendTemp = 0;

        unsigned char mdata1 = 0x00,mdata2 = 0x00,mdata3 = 0x00,mdata4 = 0x00,mdata5 = 0x00;

        ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
            SYSCTL_XTAL_16MHZ);
    init_Pin();
    init_Pin_leds();
    init_Pin_servo();
    init_UART();
```

```
    init_SPI();
    initRF();

    while(1){
        //Máquina de Estados
        switch(estado){
        case 0:
            if((filaVazia(msgs))==0){
                mdata1 = msgs->data1;
                mdata2 = msgs->data2;
                mdata3 = msgs->data3;
                mdata4 = msgs->data4;
                mdata5 = msgs->data5;
                msgs = pop(msgs);
            }
            estado = 1;
            break;
        case 1: //Lampada
            if(mdata1==0x01){ // se a msg eh para esse atuador
                if(mdata5 != 0x99){
                    //modo Manual
                    if((mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 == 0x00)&&(mdata5
== 0x01)){
                        pinLED =
ROM_GPIOPinRead(GPIO_PORTE_BASE,GPIO_PIN_4);
                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4,
GPIO_PIN_4);
                            pinLED =
ROM_GPIOPinRead(GPIO_PORTE_BASE,GPIO_PIN_4);
                        mdata1 = 0x00;
                        mdata2 = 0x00;
                        mdata3 = 0x00;
                        mdata4 = 0x00;
                        mdata5 = 0x00;

                    }else if((mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 ==
0x00)&&(mdata5 == 0x02)){
                        pinLED =
ROM_GPIOPinRead(GPIO_PORTE_BASE,GPIO_PIN_4);
                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4, 0);
                        pinLED =
ROM_GPIOPinRead(GPIO_PORTE_BASE,GPIO_PIN_4);
                        mdata1 = 0x00;
                        mdata2 = 0x00;
                        mdata3 = 0x00;
                        mdata4 = 0x00;
                        mdata5 = 0x00;
                    }
                    //limpa flag de modo automatico
                    ledAuto = 0;
                }else if(mdata5 == 0x99){
                    //marca flag de modo automatico
                    ledAuto = 1;
                    //leitura do adc
                    ROM_ADCProcessorTrigger(ADC0_BASE, 2);
                    while(!ROM_ADCIntStatus(ADC0_BASE, 2, false)){}
                    ROM_ADCIntClear(ADC0_BASE, 2);
                    ROM_ADCSequenceDataGet(ADC0_BASE, 2, pui32ADC0Value);
                    //verificacao do valor do fotoresistor
                    if(pui32ADC0Value[0] <= 2000){
```

```
                                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4,
GPIO_PIN_4);
                                    }else if(pui32ADC0Value[0] > 2000){
                                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4, 0);
                                    }
                                    mdata1 = 0x00;
                                    mdata2 = 0x00;
                                    mdata3 = 0x00;
                                    mdata4 = 0x00;
                                    mdata5 = 0x00;
                                }
                            }else{ // se nao for
                                if(ledAuto == 1){
                                    //execucao modo automatico
                                    ROM_ADCProcessorTrigger(ADC0_BASE, 2);
                                    while(!ROM_ADCIntStatus(ADC0_BASE, 2, false)){}
                                    ROM_ADCIntClear(ADC0_BASE, 2);
                                    ROM_ADCSequenceDataGet(ADC0_BASE, 2, pui32ADC0Value);
                                    if(pui32ADC0Value[0] <= 2000){
                                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4,
GPIO_PIN_4);
                                    }else if(pui32ADC0Value[0] > 1000){
                                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4, 0);
                                    }
                                }
                            }
                            estado = 2;
                            break;

                    case 2: // sensor de temperatura e fumaça
                            ROM_ADCProcessorTrigger(ADC0_BASE, 2);
                            while(!ROM_ADCIntStatus(ADC0_BASE, 2, false)){}
                            ROM_ADCIntClear(ADC0_BASE, 2);
                            ROM_ADCSequenceDataGet(ADC0_BASE, 2, pui32ADC0Value);
                            //conversao da temperatura
                            cortinaAuto = pui32ADC0Value[0];
                            TempAux = ((pui32ADC0Value[2])*1000)/4096;
                            Temp = ((300 * ((float)TempAux/100)));
                            //conversao do nivel de fumaca
                            Fum = ((pui32ADC0Value[1]-200)*10000)/4096;
                            Umidade = 100-(pui32ADC0Value[3]*100/4096);
                            estado = 3;
                            break;

                    case 3: // Servo-motor
                            if(mdata1==0x03){
                                if((mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 ==
0x00)&&(mdata5==0x01)){
                                    ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 4);
                                    ROM_PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
                                    ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
                                    autoTemp = 0;
                                    mdata1 = 0x00;
                                    mdata2 = 0x00;
                                    mdata3 = 0x00;
                                    mdata4 = 0x00;
                                    mdata5 = 0x00;
                                }else if((mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 ==
0x00)&&(mdata5==0x02)){
```

```
                                        ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 2.5);
                                        ROM_PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
                                        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
                                        autoTemp = 0;
                                        mdata1 = 0x00;
                                                mdata2 = 0x00;
                                                mdata3 = 0x00;
                                                mdata4 = 0x00;
                                                mdata5 = 0x00;
                                }else if((mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata5==0x99)){ //Automatico
                                        autoTemp = (int)mdata4*10;
                                        if(Temp >=  autoTemp){
                                                ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 8); //largura de pulso para abrir
                                                        ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_0_BIT, true);
                                                        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
//habilita saída da pwm

                                        }else{
                                                ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 2.5); //largura de pulso para fechar
                                                        ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_0_BIT, true);
                                                        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
//habilita saída da pwm
                                        }
                                        mdata1 = 0x00;
                                        mdata2 = 0x00;
                                        mdata3 = 0x00;
                                        mdata4 = 0x00;
                                        mdata5 = 0x00;
                                }
                        }else{
                                if(autoTemp != 0){
                                        if(Temp >=  autoTemp){
                                                ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 8);
                                                        ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_0_BIT, true);
                                                        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);

                                        }else{
                                                ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_OUT_0) / 2.5);
                                                        ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_0_BIT, true);
                                                        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
                                        }
                                }
                        }

                        estado = 4;
                        break;

                case 4: //Solenoide
                        if ((mdata1 == 0x04)&&(mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 ==
0x00)&&(mdata5==0x01)){
```

```
                        ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7, GPIO_PIN_7); //abre
solenoide
                        for(auxSol=0; auxSol < 5000000; auxSol++){} //aguarda aproximadaente 10
segundos
                        ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7, 0); // fecha solenoide
                        regarFlag = 0;
                        mdata1 = 0x00;
                        mdata2 = 0x00;
                        mdata3 = 0x00;
                        mdata4 = 0x00;
                        mdata5 = 0x00;
                }else if ((mdata1 == 0x04)&&(mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 ==
0x00)&&(mdata5==0x99)){
                        regarFlag = 1;
                        if(Umidade < 30){
                                ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7, GPIO_PIN_7);
//abre solenoide
                                        for(auxSol=0; auxSol < 2000000; auxSol++){} //aguarda
aproximadaente 10 segundos
                                        ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7, 0); //
fecha solenoide
                        }
                        mdata1 = 0x00;
                        mdata2 = 0x00;
                        mdata3 = 0x00;
                        mdata4 = 0x00;
                        mdata5 = 0x00;
                }

                if(regarFlag == 1){
                        if(Umidade < 30){
                                        ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7,
GPIO_PIN_7); //abre solenoide
                                        for(auxSol=0; auxSol < 2000000; auxSol++){} //aguarda
aproximadamente 2 segundos
                                        ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7, 0); //
fecha solenoide
                                        for(auxSol=0; auxSol < 2000000; auxSol++){} //aguarda
aproximadamente 2 segundos
                                }
                }
                estado = 5;
                break;
        case 5:
                if(Fum > 2500){ //verifica nivel de fumaça
                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5, GPIO_PIN_5); //aciona a
solenoide
                }else{
                        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5, 0); // fecha solenoide
                }
                estado = 6;
                break;
        case 6:
                if(mdata1==0x06){
                                        if((mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 ==
0x00)&&(mdata5==0x01)){
                                                ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 8);
                                                ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_1_BIT, true);
```

```
                                        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
                                        cortinaFlag = 0;
                                        cortinaAuto = 0;
                                        mdata1 = 0x00;
                                        mdata2 = 0x00;
                                        mdata3 = 0x00;
                                        mdata4 = 0x00;
                                        mdata5 = 0x00;
                                    }else if((mdata2 == 0x00)&&(mdata3 == 0x00)&&(mdata4 ==
0x00)&&(mdata5==0x02)){
                                        ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 2.5);
                                        ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_1_BIT, true);
                                        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);
                                        cortinaFlag = 0;
                                        cortinaAuto = 0;
                                        mdata1 = 0x00;
                                        mdata2 = 0x00;
                                        mdata3 = 0x00;
                                        mdata4 = 0x00;
                                        mdata5 = 0x00;
                                    }else if((mdata2 == 0x00)&&(mdata3 ==
0x00)&&(mdata5==0x99)){ //Automatico
                                    if(cortinaAuto >  2000){
                                        ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 8);
                                            ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_1_BIT, true);
                                            ROM_PWMGenEnable(PWM0_BASE,
PWM_GEN_0);
                                    }else{
                                        ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1,
ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 2.5);
                                            ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_1_BIT, true);
                                            ROM_PWMGenEnable(PWM0_BASE,
PWM_GEN_0);
                                    }

                                    cortinaFlag = 1;
                                    mdata1 = 0x00;
                                    mdata2 = 0x00;
                                    mdata3 = 0x00;
                                    mdata4 = 0x00;
                                    mdata5 = 0x00;
                                    }
                                }else{
                                    if(cortinaFlag !=0){
                                        if(cortinaAuto >  2000){ //quando esta claro fora de casa
abre a cortina
                                            ROM_PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_1, ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 8);
                                            ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_1_BIT, true);
                                            ROM_PWMGenEnable(PWM0_BASE,
PWM_GEN_0);

                                        }else{ //quando esta escuro fora de casa fecha a cortina
```

```
                                                ROM_PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_1, ROM_PWMGenPeriodGet(PWM0_BASE, PWM_GEN_0) / 2.5);
                                                ROM_PWMOutputState(PWM0_BASE,
PWM_OUT_1_BIT, true);
                                                ROM_PWMGenEnable(PWM0_BASE,
PWM_GEN_0);
                                        }
                                }
                        }
                estado = 7;
                break;

        case 7:
                if(mdata1 == 0x07){
                        sendFum = (Fum * 100) / 10000;
                        sendTemp = (Temp*100)/1000;
                        pinCSN_Low();
                        send_SPI(CMD_FLUSH_TX); //Limpa o FIFO de envio
                        pinCSN_High();

                        pinCSN_Low();

                        send_SPI(CMD_W_ACK_PAYLOAD); //Comando para carregar dados no FIFO do
ACK
                        send_SPI((unsigned char)sendFum); //Dados do sensor de fumaça
                        send_SPI((unsigned char)sendTemp); //Dados do sensor de Temperatura
                        send_SPI((unsigned char)Umidade); //Dados do sensor de Umidade do solo
                        send_SPI(0x00);
                        send_SPI(0x00);
                        pinCSN_High();
                }
                estado = 0;
                break;
        }
  }
  //return(0);
}
```

APÊNCIDE B

LISTAGEM DO CÓDIGO DA BIBLIOTECA RF24L01

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/rom.h"

#include "driverlib/adc.h"
#include "driverlib/systick.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/ssi.h"
#include "driverlib/pwm.h"
#include "utils/uartstdio.h"
#include "driverlib/uart.h"
#include "driverlib/gpio.h"
#include "tSPI.h"
#include "RF24L01.h"
#include "fila.h"

int statusCSN;
//Fila inicio;
//Fila ultimo;
Fila *msgs;

void PortDIntHandler(void){
	uint32_t status = 0;
	unsigned char statusTMP = 0x00;
	unsigned char data1,data2,data3,data4,data5,data6 = 0x00;
	status = GPIOIntStatus(GPIO_PORTD_BASE,true);
	if( (status & GPIO_INT_PIN_6) == GPIO_INT_PIN_6){
		GPIOIntClear(GPIO_PORTD_BASE, GPIO_INT_PIN_6);
		pinCSN_Low();
		send_SPI(CMD_R_RX_PL_WID);
		pinCSN_High();
		pinCSN_Low();
		send_SPI(CMD_R_RX_PAYLOAD);
		data1 = send_SPI(0xFF);
		data2 = send_SPI(0xFF);
		data3 = send_SPI(0xFF);
		data4 = send_SPI(0xFF);
		data5 = send_SPI(0xFF);
		data6 = send_SPI(0xFF);
		pinCSN_High();
		statusTMP = regRead(REG_STATUS);
		regWrite(REG_STATUS, 0x4E);
		if(data1 != 0x40){
			msgs = push(msgs,data1,data2,data3,data4,data5);
		}
		else{
			msgs = push(msgs,data2,data3,data4,data5,data6);
		}
	}
}
```

```
void init_Pin(void){
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
         //PD2 = CE = 0
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_2);
        ROM_GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_2, 0);
         //PD1 = CSN = 1
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_3);
        ROM_GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_3, GPIO_PIN_3);
        //IRQ PIN com interrupção em borda de descida
        ROM_GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_6);
        ROM_GPIOIntTypeSet(GPIO_PORTD_BASE,GPIO_PIN_6,GPIO_FALLING_EDGE);
        GPIOIntRegister(GPIO_PORTD_BASE,PortDIntHandler);
        GPIOIntEnable(GPIO_PORTD_BASE, GPIO_INT_PIN_6);
}

void init_Pin_servo(void){
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); //PB6
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        ROM_GPIOPinConfigure(GPIO_PB6_M0PWM0);
        ROM_GPIOPinConfigure(GPIO_PB7_M0PWM1);
        ROM_GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_6);
        ROM_GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_7);
        ROM_PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_UP_DOWN |
       PWM_GEN_MODE_NO_SYNC);
        ROM_PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, 228028);
        ROM_PWMGenConfigure(PWM0_BASE, PWM_GEN_1, PWM_GEN_MODE_UP_DOWN |
                PWM_GEN_MODE_NO_SYNC);
        ROM_PWMGenPeriodSet(PWM0_BASE, PWM_GEN_1, 228028);
}

void init_Pin_leds(void)
{
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
        //Pinos de ADC = PE0, PE1, PE2, PE3
        ROM_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0 | GPIO_PIN_3 | GPIO_PIN_2 |
GPIO_PIN_1);
        ROM_ADCReferenceSet(ADC0_BASE,ADC_REF_INT); //referencia de 3V.
        ROM_ADCSequenceDisable(ADC0_BASE, 1);
        ROM_ADCSequenceDisable(ADC0_BASE, 2);
        ROM_ADCSequenceDisable(ADC0_BASE, 0);
        ROM_ADCSequenceDisable(ADC0_BASE, 3);
        ROM_ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
        ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_CH0);
        ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_CH2);
        ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_CH1);
        ROM_ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_CH3 | ADC_CTL_IE |
                        ADC_CTL_END);
        ROM_ADCSequenceEnable(ADC0_BASE, 2);
        ROM_ADCIntClear(ADC0_BASE, 2);
        //Pino para led - PD X
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_4);
        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_4,0);
        //Pino para solenoide - PE 0
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);
        ROM_GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_7,0);
        //Pino para solenoide Sensor de Fumaça - PE 5
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_5);
        ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5,0);
```

```
}
void initRF(void){
	criarFila(msgs);
	//Esperar 10.3ms para entrar em PowerDown
	int i;
	for(i=0;i<824000;i++){}
	//PWR_UP = 1 para entrar em modo standby-I
	regWrite(REG_CONFIG,0x0A);
	//Entra em modo PRX
	configPRX();
}

void pinCE_High(void){
	ROM_GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_2, GPIO_PIN_2);
}
void pinCE_Low(void){
	ROM_GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_2, 0);
}

void pinCSN_High(void){
	ROM_GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_3, GPIO_PIN_3);
	statusCSN = ROM_GPIOPinRead(GPIO_PORTD_BASE,GPIO_PIN_3);
}

void pinCSN_Low(void){
	ROM_GPIOPinWrite(GPIO_PORTD_BASE,GPIO_PIN_3, 0);
	statusCSN = ROM_GPIOPinRead(GPIO_PORTD_BASE,GPIO_PIN_3);
}

unsigned char regRead(unsigned char reg){

	unsigned char data;
	pinCSN_Low();
	data = send_SPI((CMD_R_REGISTER & 0x1F) | (reg));
	send_SPI(CMD_NOP);
	pinCSN_High();
	return(data);
}

unsigned char regWrite(unsigned char reg, unsigned char value){
	unsigned char data;
	pinCSN_Low(); // pino CSN colocado em nivel baixo
	data = send_SPI(CMD_W_REGISTER | reg); // envio do comando para escrita juntamente com o
registrador
	send_SPI(value); //envio dos dados
	pinCSN_High(); //pino CSN colocado em nivel alto
	return data;
}

unsigned char regWriteFive(unsigned char reg, unsigned char val1, unsigned char val2, unsigned char
val3, unsigned char val4, unsigned char val5)
{
	unsigned char data;
	pinCSN_Low(); // pino CSN colocado em nivel baixo
	data = send_SPI(CMD_W_REGISTER | reg); // envio do comando para escrita juntamente com o
registrador
	//envio dos dados
	data = send_SPI(val1);
	data = send_SPI(val2);
	data = send_SPI(val3);
```

```
        data = send_SPI(val4);
        data = send_SPI(val5);
        pinCSN_High(); //pino CSN colocado em nivel alto
        return data;
}

unsigned char configPRX(){
        int i;
        unsigned char data;
        pinCE_Low();
        //Modo RX com PRIM_UP = 1 && PWR_UP = 1 e PINO IRQ refletindo RX LOW
        regWrite(REG_CONFIG,0xBB);

        //espera 1.5ms para entrar em StandBy-I
        for(i=0;i<120000;i++){}
        //Habilita o Endereço do PIPE RX 0
        regWrite(REG_EN_RXADDR,0x01);
        //Habilita auto Ack
        regWrite(REG_EN_AA,0x01);
        //SETA O REG_SETUP_AW para endereços de 5bytes
        regWrite(REG_SETUP_AW,0x03);
        //Seta o REG_SETUP_RETR para retransmitir a cada 250us e 3 tentativas
        regWrite(REG_SETUP_RETR,0x03);
        //Seta a frequencia do canal de transmissao
        regWrite(REG_RF_CH,0x02);
        //Setando AIR DATA RATE em 2MB Output POWER TX em 0dbm e ganho LNA
        regWrite(REG_RF_SETUP,0x0F);
        //Setando reg observe TX (monitora perda de pacotes) para o padrao 0x00
        regWrite(REG_OBSERVE_TX,0x00);
        //Setando reg CD carrier detect 0x00
        regWrite(REG_RPD,0x00);
        //Seta o endereço do PIPE 0
        //regWriteFive(REG_RX_ADDR_P0,0xE7,0xE7,0xE7,0xE7,0xE7);
        //Seta o endereço do PIPE 1
        //regWriteFive(REG_RX_ADDR_P1,0xA3,0xA3,0xA3,0xA3,0xA3);
        //Seta o endereço do PIPE 2
        //regWrite(REG_RX_ADDR_P2,0xC2);
        //Seta o endereço do PIPE 3
        //regWrite(REG_RX_ADDR_P3,0xC3);
        //Seta o endereço do PIPE 4
        //regWrite(REG_RX_ADDR_P4,0xC4);
        //Seta o endereço do PIPE 5
        //regWrite(REG_RX_ADDR_P5,0xC5);
        //Seta o REG_TX_ADDR para o mesmo que o PIPE 0
        //regWriteFive(REG_TX_ADDR,0xE7,0xE7,0xE7,0xE7,0xE7);
        //Setando regs de quantidade de bytes dos pipes para o padrao
        regWrite(REG_RX_PW_P0,0x00);
        regWrite(REG_RX_PW_P1,0x00);
        regWrite(REG_RX_PW_P2,0x00);
        regWrite(REG_RX_PW_P3,0x00);
        regWrite(REG_RX_PW_P4,0x00);
        regWrite(REG_RX_PW_P5,0x00);
        //Setando para o padrao o REG FIFO STATUS
        regWrite(REG_FIFO_STATUS,0x11);
        //habilita o payload de tamanho dinamico para o data PIPE 0
        regWrite(REG_DYNPD,0x01);
        //habilita funções de payload dinamico
        data = regWrite(REG_FEATURE,0x07);
        //Ativa comandos do modo dpl
        pinCSN_Low();
```

```
        send_SPI(CMD_ACTIVATE);
        send_SPI(0x73);
        pinCSN_High();
        pinCE_High();
        return(data);
}

unsigned char configPTX(){
        int i;
        unsigned char data;
        pinCE_Low();
        regWrite(REG_CONFIG,0x08);

        for(i=0;i<120000;i++){}
        //Modo RX com PRIM_UP = 1 && PWR_UP = 1 e PINO IRQ refletindo RX LOW
        regWrite(REG_CONFIG,0x1A);
        //espera 1.5ms para entrar em StandBy-I
        for(i=0;i<120000;i++){}
        //Habilita o Endereço do PIPE RX 0
        regWrite(REG_EN_RXADDR,0x03);
        //Habilita auto Ack
        regWrite(REG_EN_AA,0x3F);
        //SETA O REG_SETUP_AW para endereços de 5bytes
        regWrite(REG_SETUP_AW,0x03);
        //Seta o REG_SETUP_RETR para retransmitir a cada 250us e 3 tentativas
        regWrite(REG_SETUP_RETR,0x03);
        //Seta a frequencia do canal de transmissao
        regWrite(REG_RF_CH,0x02);
        //Setando AIR DATA RATE em 2MB Output POWER TX em 0dbm e ganho LNA
        regWrite(REG_RF_SETUP,0x0F);
        //Setando reg observe TX (monitora perda de pacotes) para o padrao 0x00
        regWrite(REG_OBSERVE_TX,0x00);
        //Setando reg CD carrier detect 0x00
        regWrite(REG_RPD,0x00);
        //Seta o endereço do PIPE 0
        //regWriteFive(REG_RX_ADDR_P0,0xE7,0xE7,0xE7,0xE7,0xE7);
        //Seta o endereço do PIPE 1
        //regWriteFive(REG_RX_ADDR_P1,0xA3,0xA3,0xA3,0xA3,0xA3);
        //Seta o endereço do PIPE 2
        //regWrite(REG_RX_ADDR_P2,0xC2);
        //Seta o endereço do PIPE 3
        //regWrite(REG_RX_ADDR_P3,0xC3);
        //Seta o endereço do PIPE 4
        //regWrite(REG_RX_ADDR_P4,0xC4);
        //Seta o endereço do PIPE 5
        //regWrite(REG_RX_ADDR_P5,0xC5);
        //Seta o REG_TX_ADDR para o mesmo que o PIPE 0
        //regWriteFive(REG_TX_ADDR,0xE7,0xE7,0xE7,0xE7,0xE7);
        //Setando regs de quantidade de bytes dos pipes para o padrao
        regWrite(REG_RX_PW_P0,0x00);
        regWrite(REG_RX_PW_P1,0x00);
        regWrite(REG_RX_PW_P2,0x00);
        regWrite(REG_RX_PW_P3,0x00);
        regWrite(REG_RX_PW_P4,0x00);
        regWrite(REG_RX_PW_P5,0x00);
        //Setando para o padrao o REG FIFO STATUS
        regWrite(REG_FIFO_STATUS,0x11);
        //habilita o payload de tamanho dinamico para o data PIPE 0
        regWrite(REG_DYNPD,0x01);
        //habilita funções de payload dinamico
```

```
        data = regWrite(REG_FEATURE,0x07);
        //Ativa comandos do modo dpl
        pinCSN_Low();
        send_SPI(CMD_ACTIVATE);
        send_SPI(0x73);
        pinCSN_High();
        pinCE_High();
        return(data);
}

unsigned char configPRX2(){
        int i;
        unsigned char data;

        pinCE_Low();
        regWrite(REG_CONFIG,0x08);
        for(i=0;i<120000;i++){}
        //Modo RX com PRIM_UP = 1 && PWR_UP = 1 e PINO IRQ refletindo RX LOW
                regWrite(REG_CONFIG,0xBB);
                //espera 1.5ms para entrar em StandBy-I
                for(i=0;i<120000;i++){}
                //Habilita o Endereço do PIPE RX 0
                regWrite(REG_EN_RXADDR,0x01);
                //Habilita auto Ack
                regWrite(REG_EN_AA,0x01);
                //SETA O REG_SETUP_AW para endereços de 5bytes
                regWrite(REG_SETUP_AW,0x03);
                //Seta o REG_SETUP_RETR para retransmitir a cada 250us e 3 tentativas
                regWrite(REG_SETUP_RETR,0x03);
                //Seta a frequencia do canal de transmissao
                regWrite(REG_RF_CH,0x02);
                //Setando AIR DATA RATE em 2MB Output POWER TX em 0dbm e ganho LNA
                regWrite(REG_RF_SETUP,0x0F);
                //Setando reg observe TX (monitora perda de pacotes) para o padrao 0x00
                regWrite(REG_OBSERVE_TX,0x00);
                //Setando reg CD carrier detect 0x00
                regWrite(REG_RPD,0x00);
                //Seta o endereço do PIPE 0
                //regWriteFive(REG_RX_ADDR_P0,0xE7,0xE7,0xE7,0xE7,0xE7);
                //Seta o endereço do PIPE 1
                //regWriteFive(REG_RX_ADDR_P1,0xA3,0xA3,0xA3,0xA3,0xA3);
                //Seta o endereço do PIPE 2
                //regWrite(REG_RX_ADDR_P2,0xC2);
                //Seta o endereço do PIPE 3
                //regWrite(REG_RX_ADDR_P3,0xC3);
                //Seta o endereço do PIPE 4
                //regWrite(REG_RX_ADDR_P4,0xC4);
                //Seta o endereço do PIPE 5
                //regWrite(REG_RX_ADDR_P5,0xC5);
                //Seta o REG_TX_ADDR para o mesmo que o PIPE 0
                //regWriteFive(REG_TX_ADDR,0xE7,0xE7,0xE7,0xE7,0xE7);
                //Setando regs de quantidade de bytes dos pipes para o padrao
                regWrite(REG_RX_PW_P0,0x00);
                regWrite(REG_RX_PW_P1,0x00);
                regWrite(REG_RX_PW_P2,0x00);
                regWrite(REG_RX_PW_P3,0x00);
                regWrite(REG_RX_PW_P4,0x00);
                regWrite(REG_RX_PW_P5,0x00);
                //Setando para o padrao o REG FIFO STATUS
                regWrite(REG_FIFO_STATUS,0x11);
```

```
                //habilita o payload de tamanho dinamico para o data PIPE 0
                regWrite(REG_DYNPD,0x01);
                //habilita funções de payload dinamico
                data = regWrite(REG_FEATURE,0x07);
                //Ativa comandos do modo dpl
                        pinCSN_Low();
                        send_SPI(CMD_ACTIVATE);
                        send_SPI(0x73);
                        pinCSN_High();
                pinCE_High();
                return(data);
}
```

APÊNDICE C

LISTAGEM DO CÓDIGO DA BIBLIOTECA FILA

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>

typedef struct fila
{
        unsigned char data1;
        unsigned char data2;
        unsigned char data3;
        unsigned char data4;
        unsigned char data5;
        struct fila *prox;
}Fila;

void criarFila(Fila *f)
{
        f->prox = NULL;
}

int filaVazia(Fila *f){
        if(f==NULL)
        return 1;
        else
        return 0;
}

Fila *push(Fila *f, unsigned char d1, unsigned char d2, unsigned char d3, unsigned char d4, unsigned char
d5)
{
  Fila *novo = (Fila*)malloc(sizeof(Fila));
  Fila *temp = f;
  int verif = 0;
  novo->data1 = d1;
  novo->data2 = d2;
  novo->data3 = d3;
  novo->data4 = d4;
  novo->data5 = d5;
  novo->prox = NULL;

  if(filaVazia(f)==1){
          return novo;
  }

  while(temp->prox != NULL){
          temp = temp->prox;
  }

  temp->prox = novo;

  return f;
}

Fila* pop(Fila* ini)
{
        if(filaVazia(ini)==1){
```

```
            return NULL;
        }
        Fila *tmp = ini->prox;
        free(ini);
        return tmp;
}
```

APÊNDICE D

LISTAGEM DO CÓDIGO DA BIBLIOTECA tSPI

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/rom.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/ssi.h"
#include "utils/uartstdio.h"
#include "driverlib/uart.h"
#include "RF24L01.h"

#define num_Bytes_tx_rx 1

//Variaveis Globais

uint32_t pui32DataTx;
uint32_t pui32DataRx;
uint32_t DataRx;
uint32_t ui32Index;


void init_SPI(void){

        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0); //ativa a interface SPI
        //Configura os pinos utilizados para comunicacao SPI
        ROM_GPIOPinConfigure(GPIO_PA2_SSI0CLK);
        ROM_GPIOPinConfigure(GPIO_PA3_SSI0FSS);
        ROM_GPIOPinConfigure(GPIO_PA4_SSI0RX);
        ROM_GPIOPinConfigure(GPIO_PA5_SSI0TX);
        ROM_GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 |
                GPIO_PIN_2);
        // Clock da SPI configurado em 5Mhz.
        ROM_SSIClockSourceSet(GPIO_PORTA_BASE, SSI_CLOCK_SYSTEM);
        ROM_SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0,
                SSI_MODE_MASTER, 5000000, 8);
        // Liga SPI
        ROM_SSIEnable(SSI0_BASE);
        //Limpa os dados iniciais da SPI
        while(ROM_SSIDataGetNonBlocking(SSI0_BASE, &DataRx))
          {
          }
}

unsigned char send_SPI(unsigned char byte)
{
        pui32DataTx = byte; // byte a ser enviado
        ROM_SSIDataPut(SSI0_BASE, pui32DataTx); //coloca o byte para envio
        while(SSIBusy(SSI0_BASE)){} //aguarda o dado ser enviado
        ROM_SSIDataGet(SSI0_BASE, &pui32DataRx); //pega o retorno
        while(SSIBusy(SSI0_BASE)){} //aguarda o fim do retorno
        pui32DataRx &= 0xFF;
        return(pui32DataRx); //retorna dado recebido
}
```