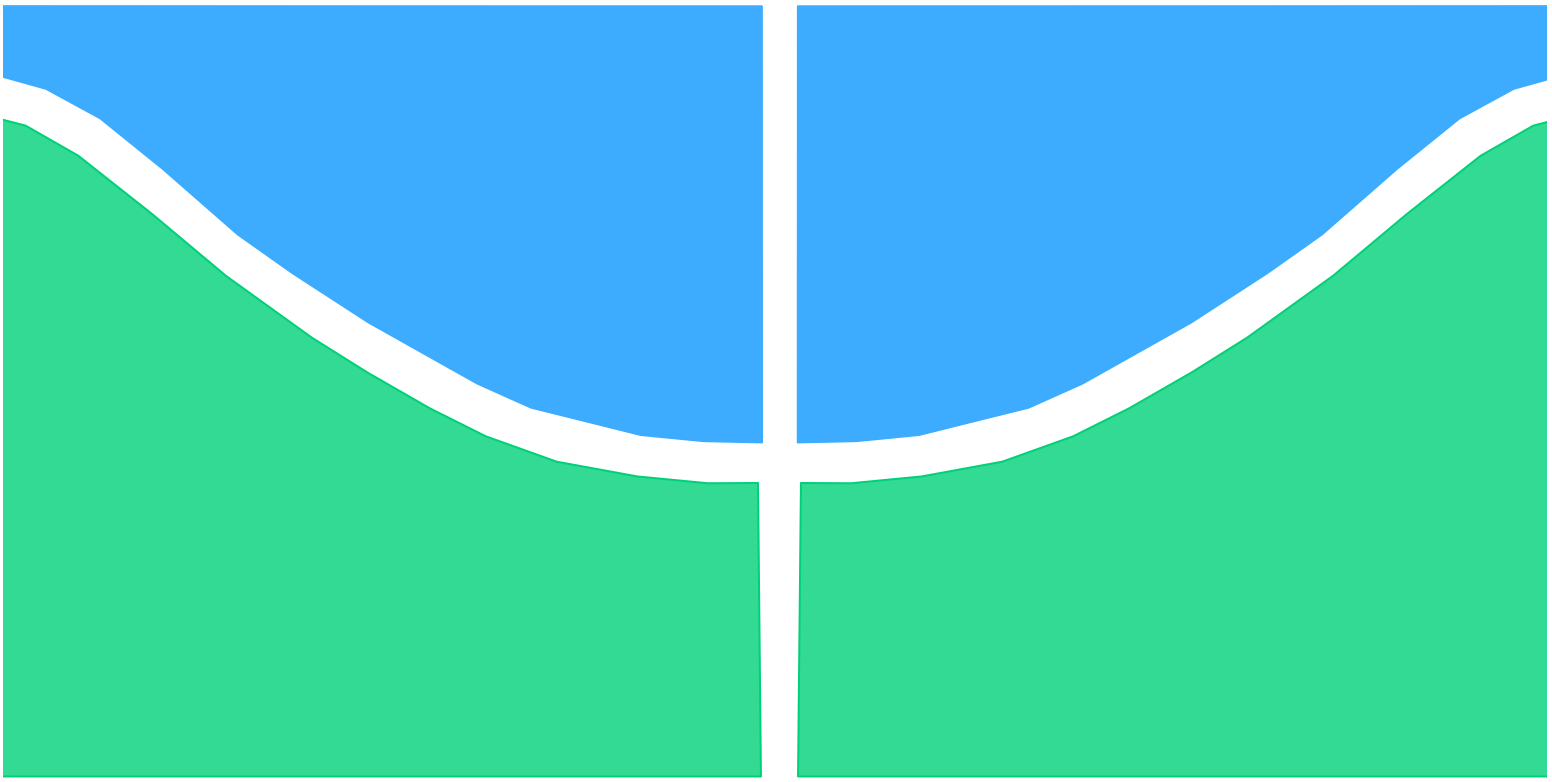


TRABALHO DE GRADUAÇÃO

EFICIÊNCIA ENERGÉTICA DE REDES SEM FIO DE AUTOMAÇÃO PREDIAL ORIENTADA A EVENTOS

Danilo Ferreira Santos

Gabriel Lara de Souza

Brasília, Dezembro de 2015

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

EFICIÊNCIA ENERGÉTICA DE REDES SEM FIO DE AUTOMAÇÃO PREDIAL ORIENTADA A EVENTOS

Danilo Ferreira Santos

Gabriel Lara de Souza

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

Banca Examinadora

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

Prof. Lélio Ribeiro Soares Junior, UnB/ENE _____

Prof. Marcelo Menezes de Carvalho, UnB/ENE _____

Brasília, Dezembro de 2015

FICHA CATALOGRÁFICA

SANTOS, DANILO FERREIRA &
SOUZA, GABRIEL LARA de

Eficiência Energética de Redes Sem Fio de Automação Predial Orientada a Eventos,

[Distrito Federal, 2015]

.... (FT/UnB, Engenheiro, Eletricista, 2015). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Automação 2. Controle
3. Orientada a eventos 4.Sensores sem fio
I. Elétrica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

SANTOS, D. F. & SOUZA, G. L., (2015). Eficiência Energética de Redes Sem Fio de Automação Predial Orientada a Eventos. Trabalho de Graduação em Engenharia Elétrica, Publicação ... , Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, ...

CESSÃO DE DIREITOS

AUTOR: Danilo Ferreira Santos e Gabriel Lara de Souza
TÍTULO DO TRABALHO DE GRADUAÇÃO: Eficiência Energética de Redes Sem Fio de Automação Predial Orientada a Eventos.
GRAU: Engenheiro Eletricista ANO: 2015

Agradecimentos

Agradeço especialmente aos meus pais, por tanto amor e dedicação empenhados em minha educação. Não teria chegado até aqui sem o apoio e sem os ensinamentos que recebi dentro de casa.
A minha madrinha, por acompanhar com muito carinho e paciência minha criação. Sua bondade e amor ao próximo inspiram a minha busca por evolução.
A minha namorada, que esteve presente em todos os momentos da minha graduação. Sua amizade e companheirismo foram fundamentais para que as dificuldades fossem superadas.
Aos meus avós, pela preocupação e afeto.
Ao nosso orientador, pelo tempo e conhecimento compartilhados para a realização deste trabalho.
Ao doutorando Vinícius Galvão Guimarães e ao mestrando Lucas Guilhem de Matos, pela atenção e disponibilidade em ajudar, pois foram essenciais para o desenvolvimento deste projeto.
A todos os familiares que torcem pelo meu sucesso.
Aos amigos que o curso me proporcionou.

Danilo Ferreira Santos

Agradeço primeiramente a Deus, por toda a força de vontade concedida para chegar até aqui e pelo dom da vida.
Aos meus pais, pela torcida e suporte necessários para o meu desenvolvimento em todas as etapas do curso.
A minha namorada, pelo apoio nas horas de desânimo e mesmo nas ausências necessárias para a conclusão do trabalho.
Aos colegas de curso, tão essenciais em várias etapas nestes anos para chegar neste momento.
Ao nosso orientador, pela paciência e tranquilidade em nos guiar.
Ao doutorando Vinícius Galvão Guimarães e ao mestrando Lucas Guilhem de Matos, pela atenção e conhecimento compartilhados que foram de suma importância para que o trabalho fosse realizado.

Gabriel Lara de Souza

RESUMO

A busca por conforto e eficiência faz com que dispositivos conectados através de cabos e fios se tornem ultrapassados. Seguindo essa mesma linha, a automação está utilizando cada vez mais os sensores e atuadores sem fio. Este trabalho estudou a ideia de automação orientada a eventos com a motivação de permitir que esses sensores economizem energia para que suas baterias durem por períodos mais longos, aumentando a vantagem dos sistemas sem fio. Os eventos aqui ditos são oscilações além da faixa de erro aceitável. Essa técnica reduz a quantidade de transmissões, que é o momento de maior consumo energético do módulo. Um código para simulação no programa MatLab foi usado para a compreensão do projeto e da posterior implementação de uma rede real composta por módulos ZigBit que se comunicam por meio do protocolo ZigBee. Esta rede é usada para o controle de temperatura da sala de uma maquete de madeira que simula ambientes de um laboratório. O atuador usado para controlar a temperatura foi um secador de cabelo, acionado por PWM. O trabalho consistiu em comparar métodos com intervalos constantes entre os envios ao sistema orientado a eventos, mostrando que este último consegue o melhor equilíbrio entre eficiência de controle e baixo consumo de corrente.

Palavras-chave: ZigBee, ZigBit, automação predial, automação orientada a eventos, smart sensor, redes sem fio, modo sleep, eficiência energética, true-time, tempo real

ABSTRACT

The search for comfort and efficiency makes devices connected by cables and wires become outdated. Following this same line, automation is increasingly using sensors and wireless actuators. This work's motivation is to allow these sensors to save energy so that your batteries last for longer periods, increasing the advantage of wireless systems. This paper studied the idea of event-driven automation. This technique allows the sensor to remain in sleep mode until some important information should be transmitted. A code for simulation in Matlab program was used for understanding the

design and subsequent implementation of a real network of ZigBit modules that communicate via the ZigBee protocol. This network would be able to control many systems such as lighting and air conditioning. The experiment was conducted in a wooden model simulating environments of a lab. The actuator used to control the temperature was a hair dryer. The work consisted in conducting the event-driven control and measure current consumption at the end device, powered by batteries, showing that even maintaining a good operating system control, there was considerable energy savings. At the end of the tests, battery consumption was analyzed to calculate how long it would be able to power the sensor and then define the generated savings.

Keywords: ZigBee, ZigBit, building automation, event-driven automation, smart sensor, wireless, sleep mode, energy efficiency, TrueTime, Real time Programming.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SÍMBOLOS

Siglas

ACK	Acknowledge
CID	Cluster Identifier
CLH	Cluster Head
DuCy	Duty Cycle
FDMA	Frequency Division Multiple Access
FFD	Full-Function Device
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engeneers
MAC	Media Access Control
NWK	ZigBee Network Layer
PAN	Personal Area Network
PDUs	Protocol Data Units
PHY	Physical
PWM	Pulse-Width Modulation
RFD	Reduced-Function Device
SSP	Synchronous Serial Port
TDMA	Time Division Multiple Access
USB	Universal Serial Bus

1 INTRODUÇÃO

Neste capítulo contextualiza-se o tema escolhido através da apresentação das motivações para a escolha deste e definem-se os objetivos dos estudos realizados.

1.1 CONTEXTUALIZAÇÃO

A automação tem papel essencial na vida cotidiana, e vai desde pequenos sistemas de iluminação ou climatização a grandes indústrias com processos totalmente automatizados. A busca por conforto, comodidade e eficiência tem se tornado muito comum. Além da praticidade, sistemas automatizados podem se tornar muito mais econômicos. É por isso que temos um número crescente de estudos voltados para essa área.

Estudos mostram que a maior parcela do consumo de energia dos Estado Unidos e da Europa se dá nos edifícios, com cerca de 40% da energia final consumida, ficando à frente da indústria e do transporte [1]. Uma pesquisa feita pelo *World Business Council for Sustainable Development* (WBCSD) em 2009 demostrou que é possível cortar expressivamente os gastos energéticos nas construções, economizando o equivalente à energia usada pelo setor de transporte [12].

Reduzir o consumo energético é vital porque preserva as fontes de energia fóssil, diminui os custos para as construtoras e para os consumidores, e pode ser realizado relativamente rápido. A melhor eficiência energética acontece com a otimização de três pilares: a envoltória, o sistema de ar condicionado e o sistema de iluminação [9].

O primeiro passo, a envoltória, é em relação à construção física. Essa etapa é capaz de reduzir bastante os gastos energéticos futuros. Uma configuração com mais iluminação natural evita o uso de lâmpadas artificiais durante o dia, por exemplo. Melhorar a performance térmica também reduz os gastos com ar condicionado e calefação.

Outra forma de economia de energia é a automação de sistemas de ar condicionado. Redes de monitoramento podem ser usadas para identificar padrões de uso e controlar o uso em todo o prédio. Isso evita que equipamentos continuem ligados mesmo quando não tem ninguém no ambiente.

A terceira parte para se chegar a melhor eficiência energética é o controle do sistema de iluminação. Impedir que lâmpadas permaneçam acesas desnecessariamente reduz significativamente o consumo de energia. Um método para garantir essa economia é o uso da informação da ocupação em tempo real para o controle das luzes.

Visando a melhor obtenção dessas informações e padrões de uso, surge a ideia de sensores sem fio. As redes de sensores e atuadores sem fio vem sendo cada vez mais utilizadas na automação predial. A independência de fios permite uma instalação muito mais simples e rápida, principalmente em prédios já prontos e mobiliados. Os sensores podem ser posicionados em pontos mais interessantes sem que se tenham fios passando por todo o ambiente. Isso permite um controle mais preciso de variáveis, como a temperatura por exemplo. Com um sensor posicionado sobre uma mesa na qual se trabalha, o controle do ar condicionado se torna muito mais eficiente, pois o sistema de automação vai garantir que a temperatura esteja sempre na zona de conforto naquela região. Isso não aconteceria com sensores com fio porque estes não garantem necessariamente a temperatura de conforto no local onde o usuário se encontra, pois geralmente os sensores são posicionados próximos a pontos de energia, ou seja, paredes ou teto.

Existe uma forma ainda mais eficaz na utilização de sensores para automação. Na chamada automação predial inteligente, entre outros aspectos, consideram-se os *smart sensors*. Estes módulos sem fio monitoram o ambiente (temperatura, umidade, presença), mas só enviam dados ao nó controlador quando um evento requer a atenção deste. Sendo a transmissão e recepção de dados a maior fonte de consumo do módulo sem fio, reduzir o número de ocorrências dessas tem impacto considerável sobre a vida útil da bateria. Essa tentativa de manter um bom controle do processo enquanto reduz o consumo da bateria é o foco desse trabalho.

1.2 MOTIVAÇÃO

O protocolo de comunicação sem fio utilizado para a execução do projeto foi o ZigBee. Esse é um dos padrões que mais crescem nos últimos anos. Esse protocolo é

baseado no padrão IEEE 802.15.4, que trata de redes sem fio de menor alcance. Esse padrão é comparável às redes Wi-Fi e Bluetooth, porém com menor consumo de energia e menor taxa de transmissão. Compatível com o ZigBee, decidiu-se trabalhar com o módulo ZigBit da Atmel. Sabendo que um dos objetivos centrais deste trabalho é aumentar o tempo de vida da bateria do sensor sem fio, esse módulo é muito interessante, pois sua principal vantagem é o baixo consumo de energia.

Mesmo que o ZigBee e o ZigBit sejam considerados econômicos, ainda é possível melhorar a eficiência da rede de sensores sem fio. Sem nenhuma programação que trate a redução do consumo, a bateria duraria períodos pequenos, muitas vezes não passando de 18h de vida útil [9]. Mesmo com o uso de baterias recarregáveis, evitando o gasto financeiro, ter que trocá-las todos os dias seria incômodo suficiente para impedir o uso destes sensores, pois um dos objetivos da automação é proporcionar maior comodidade ao usuário.

Sendo assim, fazer com que o sistema trabalhe orientado a eventos aumentaria consideravelmente a vida útil das baterias, podendo durar meses e até mesmo anos. Isso tornaria possível o uso de sensores sem fio com mais frequência em projetos de automação.

1.3 OBJETIVO

1.3.1 OBJETIVO GERAL

O objetivo geral desse trabalho é investigar sistemas de automação orientados a eventos, no contexto da automação predial em redes sem fio.

1.3.2 OBJETIVO ESPECÍFICO

Pode-se definir como foco, a criação de uma programação orientada a eventos que permita realizar o controle de temperatura de um ambiente de maneira a reduzir consideravelmente o consumo de energia de dispositivos alimentados por bateria que serão usados como sensores de temperatura. Os resultados obtidos serão comparados ao sistema sem a programação para que se perceba a melhoria proporcionada pelo método.

2 ZigBee

Esse módulo apresenta o Protocolo ZigBee, citando características do seu funcionamento, bem como vantagens e desvantagens.

2.1 REDES DE COMUNICAÇÃO SEM FIO

As redes sem fio, ou redes *wireless,* estão por toda parte e são, basicamente, infraestruturas que permitem transmissão de dados e informações sem a necessidade de cabos. [17] Alguns exemplos de redes sem fio que fazem parte do nosso cotidiano são: WiFi, Bluetooth e a tecnologia dos celulares. Elas permitem que se tenha muito mais mobilidade e conforto por parte do usuário e é fácil perceber porque o uso de tecnologia sem fio vem crescendo de forma expressiva.

Apesar das inúmeras vantagens, as redes sem fio apresentam alguns problemas. São redes suscetíveis a interferências e ruídos. Essas interferências não aconteceriam se fossem usados cabos. Porém, o uso de cabos apresenta diversos outros problemas. Cabos usam conectores que quebram ou apresentam mal contato com facilidade. Os cabos podem se prender em partes da estrutura e se cortarem. Os cabos se desgastam com sal, água e sol. Os cabos exigem uma manutenção que é muitas vezes cara e complicada. Portanto, analisando os prós e contras de cada tipo de rede, fica claro que as redes sem fio são bem mais vantajosas para a maioria dos casos [7].

Quando se fala em controle e automação predial, o uso das redes sem fio se torna uma opção muito interessante. São redes mais baratas e mais fáceis de se instalar quando comparadas as com fios. Além da vantagem econômica, não depender de fios permite uma distribuição muito mais precisa de sensores, que podem ser posicionados em qualquer ponto dentro do alcance da rede. E assim como no mercado de internet o uso do sistema WiFi já é dominante, o uso de redes sem fio para automação também será.

Portanto, se a intenção é usar redes de comunicação sem fio e já conhecemos tantas opções delas, porque trabalhar com o ZigBee?

2.2 ZigBee E O PADRÃO IEEE 802.15.4

O ZigBee é um protocolo de comunicação através de ondas eletromagnéticas voltado para dispositivos de baixo consumo de energia e que se baseia no padrão IEEE 802.15.4. Esse padrão opera numa região que une baixo custo, baixo consumo de energia e baixa complexidade em troca de uma transmissão de dados mais lenta. A Figura 2.1 abaixo mostra o posicionamento do IEEE 802.15.4 em um gráfico de Custo/Complexidade/Consumo de energia X Velocidade de Transmissão e o compara com outros padrões, como o 3G, Bluetooth e WiFi.

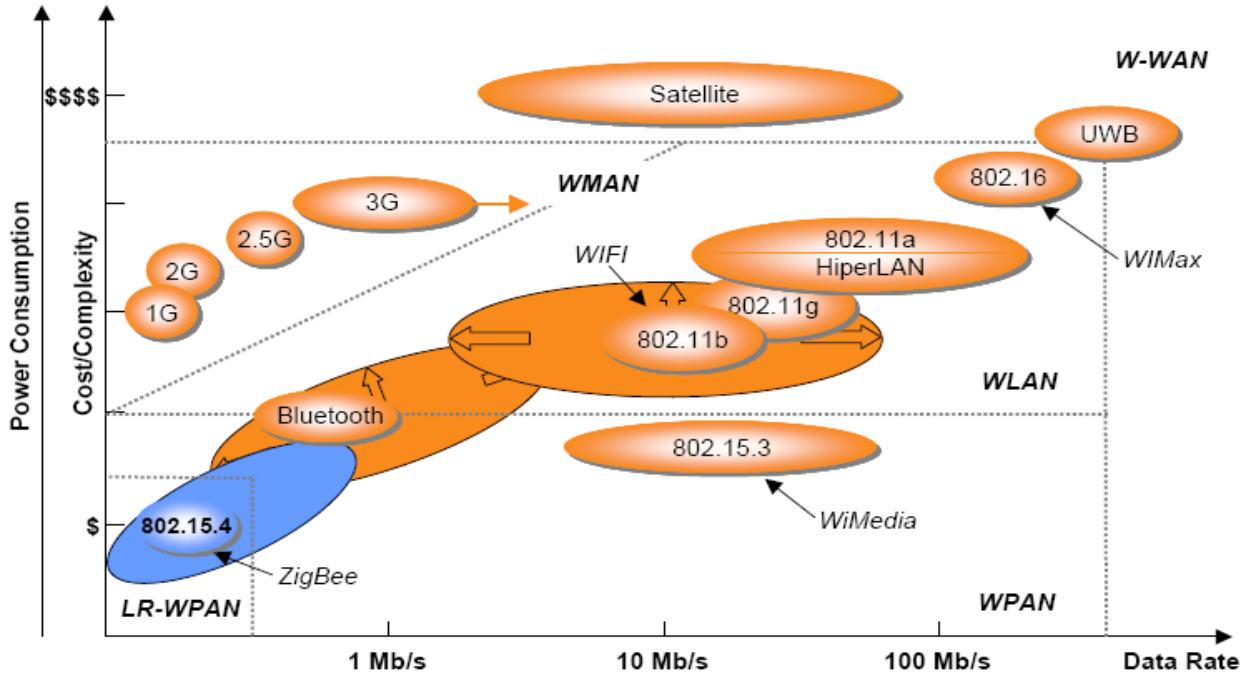


Figura 2.1: Consumo/custo/complexidade x Velocidade de transmissão [15].

Ao observar a Figura 2.1, percebe-se com clareza que para se trabalhar com uma alta taxa de transmissão de dados é preciso de um protocolo mais complexo, que por sua vez é mais caro e consome mais energia. A Tabela 2.1 traduz em números a diferença entre a região onde se encontra o ZigBee, a região do Bluetooth e a região do WLAN.

Tabela 2.1: Propriedades de redes sem fio [15].

	LR-WPAN	Bluetooth	WLAN
Alcance	10-30 m	~10-100 m	~100 m
Taxa de transmissão	<0,25 MBps	1 MBps	~2-11 MBps
Consumo de energia	<BT/10	BT*	>BT
Tamanho	O menor	Pequeno	O maior
Nós/Rede	<<BT	BT	>BT
Custo	~US$ 1	~US$ 10 - 15	~US$ 40

* BT = Bluetooth

Os dados da Tabela 2.1 mostram que o ZigBee trabalha em uma região com algumas desvantagens. Tanto o alcance como a taxa de transmissão do padrão IEEE 802.15.4 são bem inferiores. Em contrapartida, o número de nós por rede é bem maior e o custo da implementação, bem como o consumo de energia são significativamente menores.

Portanto, a escolha do protocolo que será usado deve levar todos esses fatores em consideração. No caso do projeto aqui realizado, que busca maior duração das baterias de um módulo sem fio de um sistema de automação, as ditas desvantagens do ZigBee, como a baixa velocidade de transmissão e o baixo alcance não se aplicam. Por outro lado, as vantagens, como baixo consumo de energia, baixo custo, inúmeros nós por rede e baixa complexidade são muito interessantes.

2.3 MERCADO DE ATUAÇÃO

A rede ZigBee se encaixa num mercado que não é preenchido por outras tecnologias *wireless*. Enquanto a maioria das tecnologias sem fio estão lutando para serem mais rápidas, o ZigBee busca baixas taxas de dados. Enquanto os outros protocolos adicionam cada vez mais funções, o ZigBee espera ter tudo funcionando em um simples controlador de 8 bits. Enquanto as outras tecnologias visam entregar mídias de alta definição, o ZigBee visa controlar a iluminação ou mandar informações de temperatura para um termostato. Enquanto outras tecnologias são programadas para rodar por horas ou dias usando baterias, o ZigBee é programado para rodar por anos [7].

Existem outras diferenças entre o ZigBee e as tecnologias sem fio conhecidas, mas com o que foi citado já é possível perceber que os mercados são distintos. A categoria do mercado que o ZigBee trabalha é “Controle Sem Fio”. Esse mercado

exige algumas características específicas, que ele se encaixa muito bem devido as seguintes qualidades que este apresenta:

- Altamente confiável
- Baixo custo
- Baixo consumo de energia
- Altamente seguro

Para conseguir um baixo custo e baixo consumo de energia, o ZigBee trabalha com baixas taxas de dados. Isso seria uma desvantagem, mas analisando o foco do mercado do ZigBee, percebe-se que não são necessárias altas taxas. Pense em um interruptor de luz. Ele precisa fazer poucas comunicações por dia, podendo até passar dias sem que nenhuma comunicação seja feita. Sendo assim, trabalhar com baixas taxas de dados passa a ser uma solução que faz sentido e que se encaixa perfeitamente nesse mercado.

Por serem programáveis, os módulos podem ser aplicados em diferentes sistemas. A programação baseia-se no acontecimento de eventos, que modelam o comportamento da rede e do processamento de dados. Essa maneira de programação, também chamada de *event-driven*, executa tarefas de acordo com uma sequência estabelecida. Ou seja, o fim da execução de uma tarefa leva ao início de uma nova tarefa, trabalhando de maneira linear [9].

Figura 2.2: Aplicações do ZigBee [16].

São inúmeras as possíveis áreas de atuação do protocolo ZigBee. A Figura 2.2 apresenta seis grandes ramos em que a tecnologia ZigBee é interessante. Percebe-se que dentro de cada ramo existem várias funções específicas que podem ser realizadas

pelo ZigBee. Aprofundando mais nos ramos que são tratados nesse trabalho, ou seja, os ramos de automação e controle predial, vemos que o ZigBee é bastante versátil e consegue realizar grande parte de um sistema de automação e controle. No ramo de controle residencial e comercial, por exemplo, são citadas opções de uso na parte de segurança, ventilação, controle de iluminação, controle de acesso, irrigação de jardim e aquecimento. Essas áreas somadas formariam um sistema bem completo e funcional de controle, como mostrado na Figura 2.3.

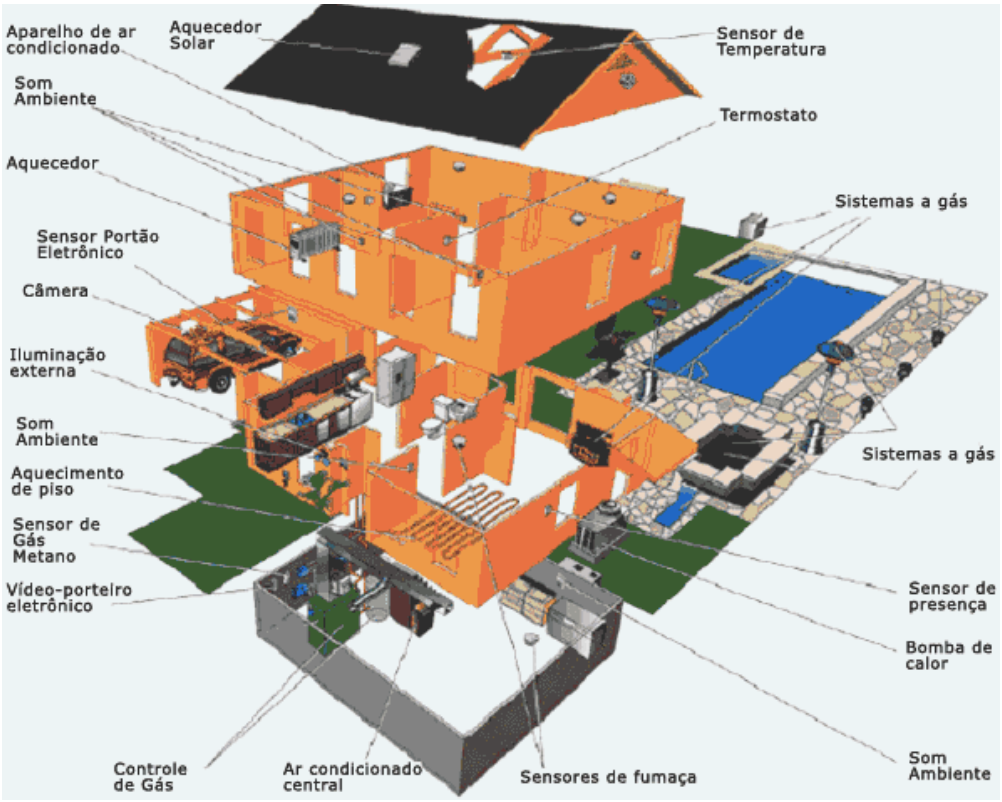


Figura 2.3: Exemplos de aplicações do ZigBee [5].

De maneira mais resumida, algumas das principais características desse protocolo são:

- Um mesmo nó pode executar diferentes papéis em uma mesma rede
- São possíveis configurações em várias topologias de rede
- Habilidade de se auto-organizar e autorreestruturar
- Permite um número elevado de dispositivos conectados à rede
- Bateria com longa duração
- Capacidade de se comunicar de forma transparente com outros sistemas.

2.4 MODOS DE OPERAÇÃO NA REDE

Um sistema ZigBee consiste de vários componentes. O mais básico é o dispositivo final, que pode ser de função completa (FFD – *Full Funciton Device*) ou de função reduzida (RFD – *Reduced Function Device*). Uma rede deve ter pelo menos um FFD operando como coordenador.

Os FFDs podem participar de qualquer topologia e podem operar em três modos: coordenador, roteador ou um dispositivo final. Os RFDs são voltados para aplicações bem simples e não precisam mandar grandes quantidades de dados. Estes não podem ser coordenador e só podem ser de topologia de estrela ou dispositivo final numa topologia de malha. O objetivo de se usar RFDs é a redução de custo e de consumo de energia. Um FFD pode se comunicar com RFDs ou FFDs enquanto o RFD só com um FFD [6].

Dependendo da disponibilidade de funções do dispositivo (RFD ou FFD) e da sua posição na rede, os nós podem ser classificados como: coordenadores, roteadores ou dispositivos finais.

O coordenador é o nó inicial da rede. Quando ligado pela primeira vez como coordenador, o dispositivo analisa todos os canais da frequência de operação até encontrar um PAN ID (*Personal Area Network*) único para trabalhar. Uma vez selecionado um identificador PAN único no seu raio de influência, o coordenador passa a operar em estado ativo para efetuar o controle da rede. Recomenda-se que o coordenador seja alimentado diretamente para que o risco de falha no nó centralizador da rede seja reduzido.

Os roteadores possuem tabelas de roteamento e conseguem encontrar o menor caminho para se chegar ao destino. Caso o roteador não possua o endereço de destino requisitado, ele irá requisitar uma rota e receberá do destino a rota mais eficaz, atualizando a tabela de roteamento. Assim, a rede garante a característica de autorregeneração caso ocorra a queda das funcionalidades de outros nós roteadores na rede. As redes com topologia de malha e ponto a ponto usam os roteadores para terem mais robustez.

Os dispositivos finais não exercem função de roteamento nem de coordenação da rede. Eles se comunicam direto com o roteador ou coordenador. Por possuírem funções mais simplificadas, os dispositivos finais podem ser implementados em microcontroladores com menor memória e potência. É possível também que eles permaneçam boa parte do tempo em estado inativo, ou “dormindo. São comumente

usados para sensores, atuadores e sistemas de controle. A Tabela 2.2 mostra algumas funções de cada modo.

Tabela 2.2: Funções dos modos de operação [16].

Coordenador ZigBee (ZC)	Roteador ZigBee (ZR)	Dispositivo Final ZigBee (ZED)	Função na Camada de Rede
x			Estabelecer uma nova rede ZigBee
x	x		Conceder endereço lógico de rede
x	x		Permitir que dispositivos entrem ou saiam da rede
x	x		Manter lista de vizinhos e rotas
x	x		Rotear pacotes da camada de rede
x	x	x	Transferir pacotes da camada de rede

Pela tabela acima vemos que a única função que cabe ao dispositivo final é a transferência de pacotes da camada de rede. Já os coordenadores e roteadores possuem em comum outras funções mais complexas. Apenas a função de estabelecer uma nova rede ZigBee não pode ser executada pelo roteador, somente pelo coordenador. Esses dados evidenciam melhor o motivo do dispositivo final ser mais simples e, portanto, mais barato e com menor consumo de energia.

2.5 TOPOLOGIA DA REDE

O ZigBee suporta três tipos de rede: em estrela, em malha e ponto a ponto. Cada uma tem vantagens e desvantagens e são indicadas para diferentes usos.

Na estrela, a comunicação é estabelecida entre dispositivos finais e o único centro de controle, chamado coordenador. Algumas aplicações vantajosas para essa topologia incluem automação residencial, brinquedos e jogos. Depois que um FFD é ativado pela primeira vez, ele estabelece a própria rede e se torna o coordenador PAN. Cada rede iniciada escolhe um identificador PAN, que ainda não é usado em nenhuma rede na esfera de influência. Isso permite que cada rede estrela opere independentemente [6].

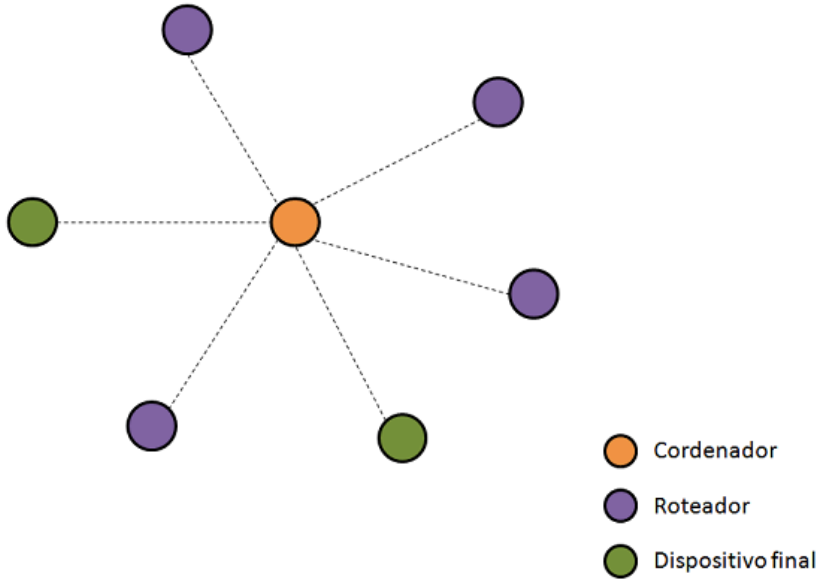


Figura 2.4: Representação de Topologia Estrela [16].

A topologia ponto a ponto é um caso especial em que a maioria dos dispositivos são FFDs. Qualquer um dos FFDs pode agir como um roteador e providenciar sincronização de serviços para outros dispositivos e roteadores. Apesar de permitir quantos roteadores se queira, só um desses pode ser o coordenador.

O coordenador forma a primeira rede se estabelecendo como o principal com o CID de zero, escolhendo um identificador PAN não usado ainda e transmitindo *beacon frames* para os dispositivos vizinhos. Os *beacon frames* têm a função de avisar que a rede está presente. Os candidatos a dispositivos pedem ao coordenador para entrarem na rede e assim a rede vai se formando [6].

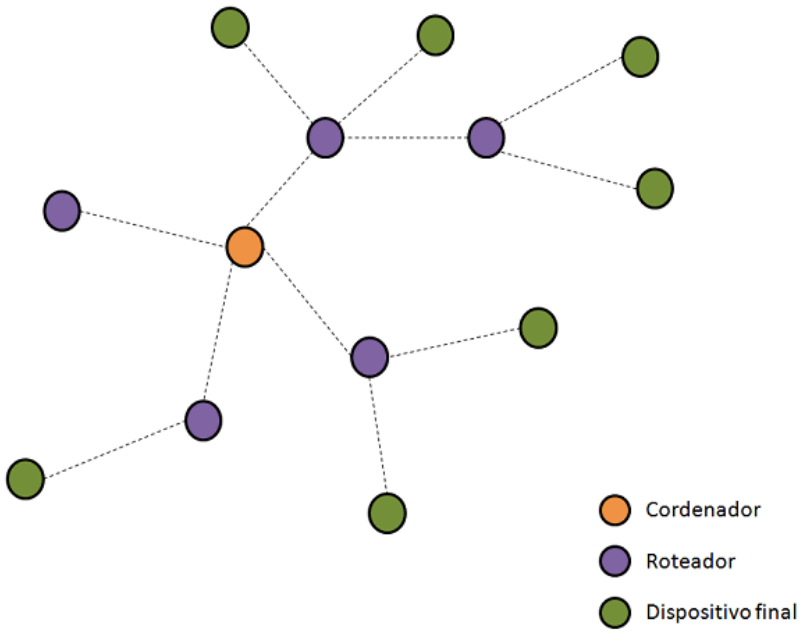


Figura 2.5: Representação de Topologia Ponto a Ponto [16].

A última topologia é a mais complexa, a em malha. Nesta, qualquer dispositivo pode se comunicar com outro desde que estejam em alcance. É o modo mais eficiente de se montar a rede, pois permite vários caminhos ligando um dispositivo aos outros da rede, assim mesmo com alguns dispositivos intermediários com defeito a rede pode funcionar bem. Essa capacidade de auto-organização e autoestruturação faz com que a topologia em malha seja mais robusta. Aplicações para essa topologia seriam o controle e monitoramento industrial e sensores de rede wireless [6].

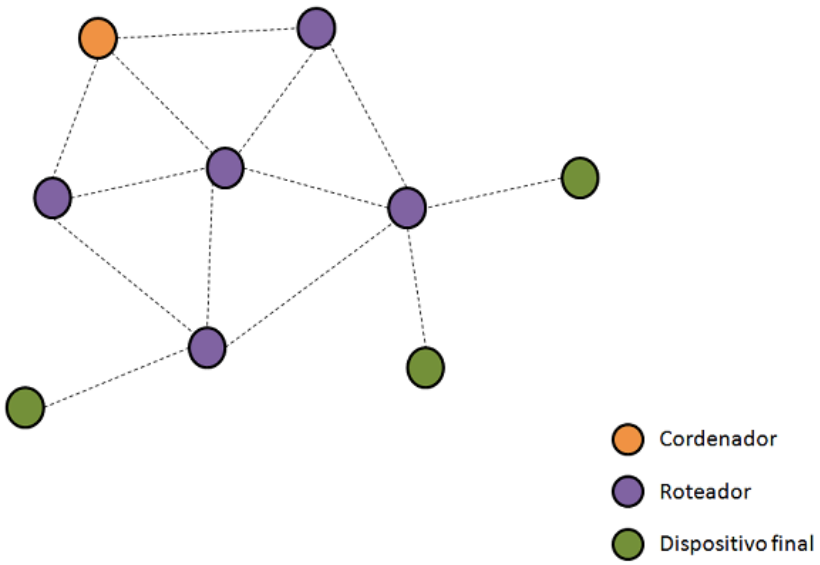


Figura 2.6: Representação de Topologia em Malha [16].

2.6 PADRONIZAÇÃO DO PROTOCOLO 802.15.4 E ZigBee ALLIANCE

Outra qualidade do ZigBee é o protocolo padronizado em relação às camadas de rede e de aplicação. Ou seja, placas de diferentes modelos e até mesmo de diferentes fabricantes são capazes de se encontrar na rede automaticamente, estabelecendo comunicação sem nenhum problema. É o que a ZigBee Alliance chama de ZigBee *Standard*, em português, padronização do ZigBee.

Toda a normalização e criação de modelos a serem seguidos pelos fabricantes é feita pelo grupo ZigBee Alliance. É também o grupo responsável pela maioria das pesquisas no segmento de automação wireless no mundo. Essa padronização dos dispositivos e do protocolo de comunicação é vista pelo grupo como responsável pelo crescimento das redes wireless e pela grande variedade e na inovação dos dispositivos ZigBee.

Enquanto a ZigBee Alliance trabalhou nas camadas superiores, o padrão 802.15.4 ficou responsável pela criação das duas camadas mais baixas da tecnologia, a camada MAC e a camada Física (PHY).

Tabela 2.3: Camadas do protocolo IEEE 802.15.4 [16].

Usuário	Aplicação
ZigBee Alliance	Suporte a Aplicação
	Rede (NWK) / Segurança (SSP)
IEEE 802.15.4	MAC
	PHY

A camada física (PHY) do ZigBee é responsável por permitir a transmissão das PDUs, unidades de dados, através de ondas de rádio.

Sobre a Camada MAC, ela é responsável por preparar para serem transmitidos os dados vindos das camadas superiores.

3 O MÓDULO ZigBit e o BitCloud

Esse capítulo fala sobre as vantagens e parâmetros do módulo ZigBit, bem como do modo Sleep deste. Explica-se, também, a função do BitCloud durante o projeto.

3.1 MÓDULO ZigBit

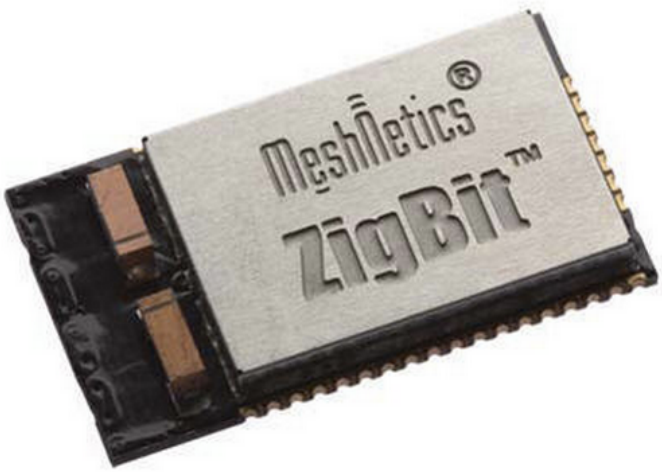


Figura 3.1: Módulo ZigBit [8].

A Figura 3.1 mostra um módulo ZigBit igual aos usados nesse trabalho. Cada ZigBit conta com um ATmega1281, que é um microcontrolador programável com baixo consumo de energia, principalmente no modo sleep. Os módulos são compactos, baseados no protocolo 802.15.4 e a programação tem como base a linguagem C. Os ZigBits são dispositivos baratos em vista da sua funcionalidade e capacidade. Algumas características do ZigBit são: antena com bom alcance, boa performance e uma integração fácil. A Tabela 3.1 apresenta alguns parâmetros gerais do módulo.

Tabela 3.1: Especificações do ZigBit [9].

Parâmetros	Faixa
Tensão de Alimentação	3.0 a 3.6 V
Consumo de Corrente em Modo Recepção	23 mA
Consumo de Corrente em Modo Transmissão	50 mA
Consumo de Corrente em Modo Dormir	<10 µA
Tamanho	38.0 x 13.5 x 2.0 (mm)
Número de Canais	16
Temperatura de Operação	-40 até 85 ºC

Como é possível ver, em modo dormir o consumo de energia do ZigBit é muito baixo. Como um dos focos do projeto é a economia de bateria do sensor sem fio, o uso do ZigBit se mostrou interessante.

Mais informações interessantes sobre o ZigBit serão explicadas mais a frente durante os testes e implementações.

3.2 SLEEP

Como já foi falado anteriormente, para que se mantenha a maleabilidade da rede através do uso de baterias para alimentar o nó sensor, é preciso fazer com que o tempo de vida útil dessas baterias seja elevado. Apesar do ZigBit, aliado ao ZigBee, ter a característica de baixo consumo de energia, nos testes realizados, o consumo de corrente em modo ativo ficou em torno de 20 mA, valor ainda alto se comparado ao que se alcançou no modo dormir, inferior a 40 µA.

Percebe-se, então, que quanto mais o módulo permanecer em modo *sleep*, maior será a economia de energia. Entretanto, se o modo dormir durar muito tempo,

informações importantes para o controle do sistema serão perdidas.

Visando manter o controle funcionando corretamente e consumindo pouca energia da bateria do módulo sensor, pensou-se na opção de usar a automação orientada a eventos. Esse método consiste, resumidamente, em manter o nó em modo *sleep* o maior tempo possível e só enviar informações para o coordenador da rede quando algum evento exigir. Ou seja, na maior parte do tempo o consumo de corrente será próximo a 40 μA.

Os métodos usados para a programação orientada a eventos, bem como para a programação do *sleep*, serão apresentados em mais detalhes durante o capítulo que trata a realização do experimento.

3.3 BitCloud

O BitCloud é uma pilha de biblioteca que auxilia a programação de cada módulo, que neste trabalho será da Atmel, ZigBit. A pilha promove desenvolvimento de firmwares para aplicações com segurança de dados, confiabilidade de rede e escalonamento de tarefas.

O escalonamento é um dos principais recursos do BitCloud. Esse nome se refere a prioridade de execução. Ou seja, se há uma tarefa com maior prioridade, acontece um ajuste para a chamada de um nova tarefa, tudo gerenciado pela pilha. Isso garante que todas as tarefas atinjam seu ponto final. Isso é chamado de Kernel Multi-Task simulado. Com isso, mesmo não tendo um hardware multiprocessado para atender a outros tipos de tarefas ao mesmo tempo, ocorre a simulação em tempo real a partir do escalonamento de tarefas, tornando o processamento de rotina mais rápido, confiável e seguro.

Talvez a principal função do BitCloud nesse trabalho é o fato de permitir ajustar o hardware para a aplicação que se deseja, ou seja, é a partir desta pilha de bibliotecas que podemos programar os dispositivos de rede para consumirem o mínimo possível de energia enquanto exercem o controle do sistema projetado.

4 CONTROLE ORIENTADO A EVENTOS

Esse capítulo fala da solução teórica proposta e analisa uma simulação computacional de um sistema orientado a eventos que servirá de base para o experimento.

4.1 SOLUÇÃO PROPOSTA

Considerando a ideia do trabalho como sendo realizar o controle da temperatura de maneira satisfatória e economizando a energia da bateria do nó sensor, a solução pensada foi unir o modo *sleep* ao sistema orientado a eventos. As vantagens em se manter o nó sem fio "dormindo" sempre que possível são grandes, pois permitem uma drástica redução no consumo de corrente. Mas, como definir em quais momentos o módulo pode entrar nesse modo de operação sem prejudicar a operação da rede?

É aqui que entra a parte da orientação baseada em eventos. É essa programação que irá dizer ao nó quando ele deve dormir ou acordar. O evento aqui citado se refere a temperaturas medidas fora da faixa de erro aceitável. Isso significa que enquanto o controle for capaz de manter a temperatura dentro dessa faixa, não há necessidade de envio de informações para o coordenador. Ou seja, o nó sensor mede a temperatura, identifica se esta representa um evento ou não e a partir disso define se envia o dado coletado. Em seguida volta ao modo *sleep*. A repetição desse procedimento permite a redução das transmissões de dados, economizando bateria.

Quando um evento acontece e o nó sensor envia a temperatura, o coordenador analisa a informação e define a atuação. O atuador aqui usado é um secador de cabelo, que será acionado por PWM. O *Duty Cycle* do PWM dependerá da diferença entre o ponto de operação e o valor medido. Quanto maior a diferença, mais forte o atuador deverá trabalhar e serão necessárias informações com maior frequência para que se controle a subida da temperatura. Quando o sistema estiver próximo do ponto de operação, ou seja, quando a diferença entre o valor medido e o desejado for baixa, o secador de cabelo estará funcionando em potência reduzida apenas para manter a temperatura.

Em resumo, ser orientado a eventos significa que o nó sensor varia o período entre os envios de temperatura ao coordenador de acordo com a necessidade do processo de controle, evitando que se gaste energia com o envio de dados não essenciais.

Com o intuito de se entender melhor esse funcionamento, foi estudado um modelo de simulação computacional de um processo térmico orientado a eventos.

4.2 SIMULAÇÃO COMPUTACIONAL DE UM PROCESSO TÉRMICO ORIENTADO A EVENTOS

Antes de dar início ao experimento proposto por esse trabalho, se fez interessante analisar os resultados gerados por uma simulação de um processo similar ao que será implementado. Esse modelo foi projetado por um Professor da Universidade de Brasília, Adolfo Bauchspiess, que gentilmente forneceu o seu trabalho para ajudar o entendimento da orientação a eventos. Essa simulação é executada no Matlab/Simulink com auxílio do TrueTime. O Truetime é um simulador baseado no Matlab/Simulink para o controle de processos em tempo real. Esta ferramenta facilita o controle de tarefas, assim como kernels, transmissões em rede com ou sem fio e outros tipos de plantas dinâmicas. É escrito em C++ MEX, usa um ambiente de simulação orientado a eventos e que pode funcionar para interrupções externas à rede [4].

Os blocos do TrueTime incluem tipos de rede amplamente difundidas, como Ethernet, CAN, TDMA, FDMA, Round Robin (ou Switched Ethernet). Também suporta redes Wireless (802.11b/g WLAN e 802.15.4 – ZigBee) e dispositivos movidos à bateria. Basicamente ele é uma biblioteca que expande as funcionalidades do Matlab/Simulink para simular redes baseadas em processos discretos de controle em tempo real.

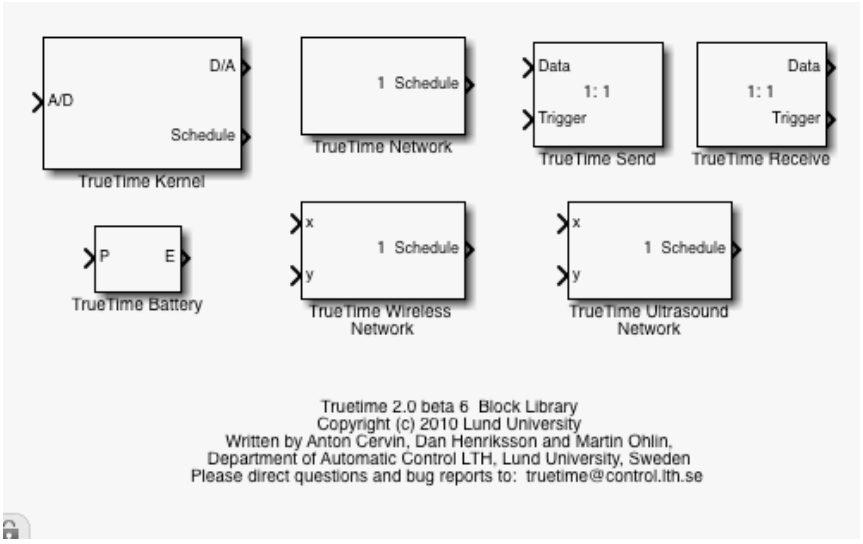


Figura 4.1: Biblioteca de blocos do TrueTime.

Esse modelo simula o comportamento de um sistema orientado a eventos e ajuda na compreensão das consequências de uma perturbação na rede de controle de um processo térmico. Além disso, também se pretende avaliar o consumo de energia e o desempenho de uma rede ZigBee.

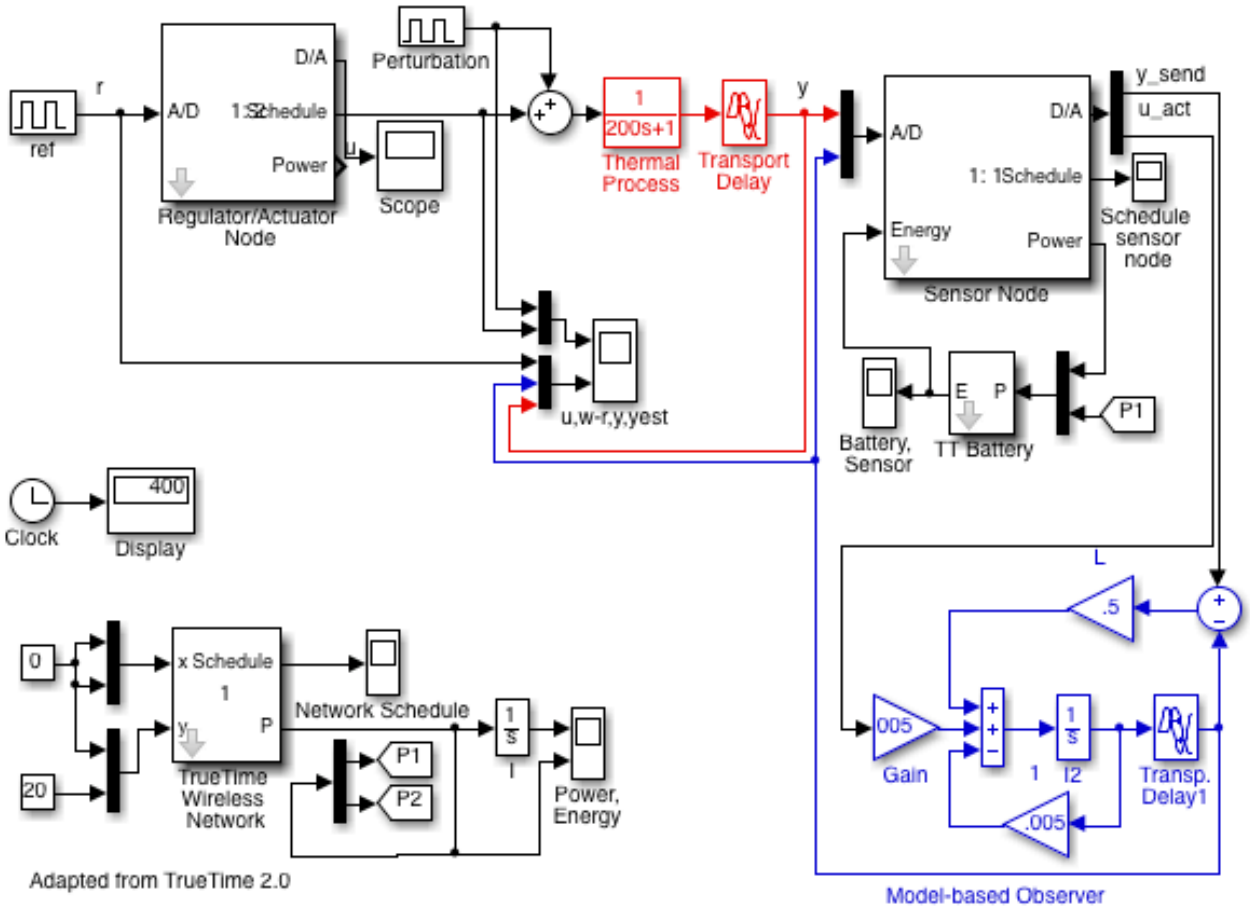


Figura 4.2: Modelo do Simulink.

O primeiro passo para a compreensão da simulação é analisar os blocos individualmente.

Regulator/Actuator Node

Nó que representa o coordenador da rede. Nele também está o atuador. Recebe o sinal de referência e atua no processo térmico.

Sensor Node

Faz as medidas de temperatura de acordo com os eventos, definindo seu escalonamento pelo observador de estados. Está conectado a uma bateria.

TrueTime Wireless Network

Quando um nó tenta transmitir uma mensagem, um sinal de ativação é enviado à entrada desse bloco. Quando a transmissão é terminada, o bloco envia um sinal de ativação na saída correspondente ao nó ao qual a mensagem é destinada. A mensagem é colocada no buffer de recepção do nó.

Thermal Process

Esse bloco contém a equação que descreve o processo térmico de primeira ordem, sendo este objeto de controle.

Perturbation

Gera perturbações aleatórias na rede.

Observer

Simula um observador de estados baseado no processo térmico, prevendo o comportamento dos eventos.

Ao executar essa simulação, com uma taxa de transferência de 250 kbps, temos a saída mostrada na Figura 4.3, considerando-se que cada segundo na simulação equivale a um minuto no processo.

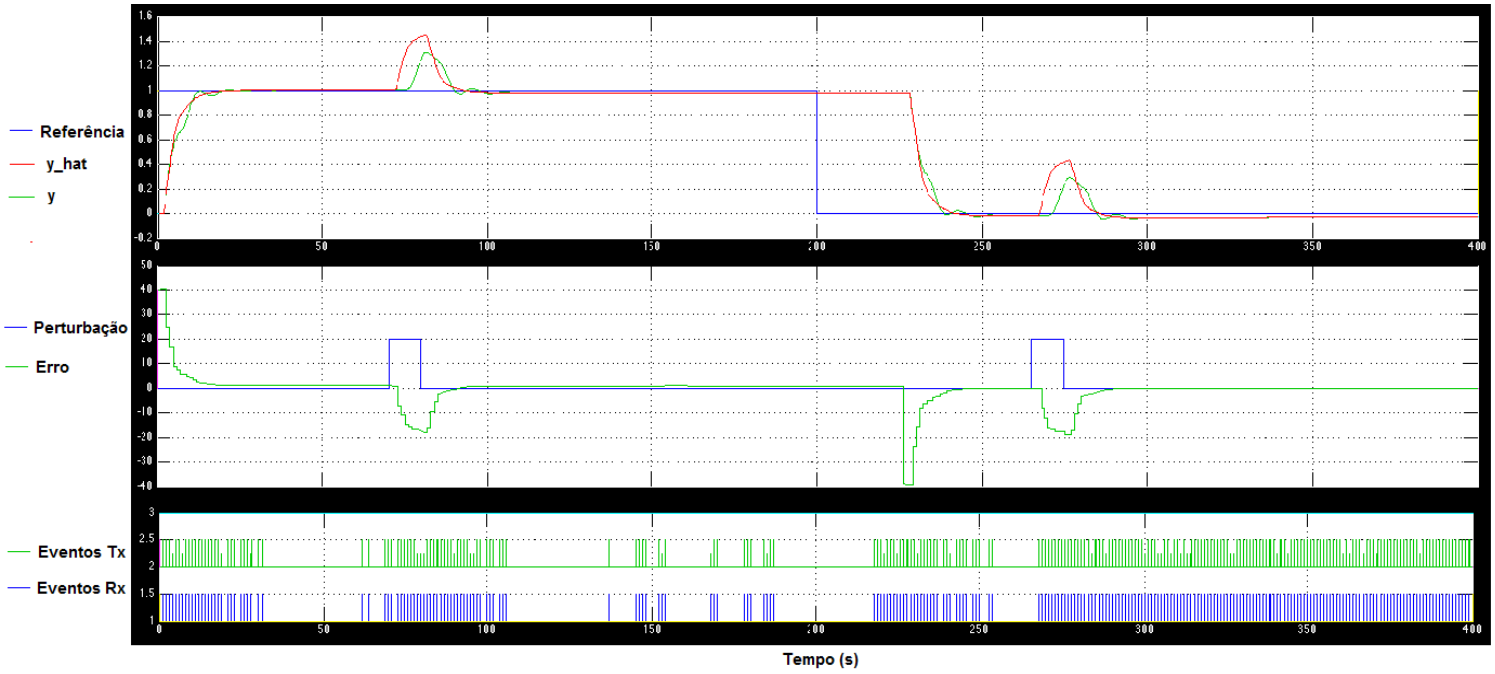


Figura 4.3: Saída rede ZigBee a 250 kbps.

Analisando a Figura 4.3, vê-se que a estratégia de controle proposta pela simulação tem uma onda quadrada variando entre 0 e 1 colocada na entrada no sistema como sinal de referência, com o intuito de simular uma mudança de ponto de operação da temperatura, assim mostrando como o sistema reage a estas variações. É possível visualizar que o processo térmico leva um tempo para alcançar a temperatura desejada, caracterizando um regime transitório um pouco lento, mas que chega ao regime permanente com erro pequeno, assim como é esperado de um sistema de primeira ordem. Partindo do pressuposto que é utilizado um controle PWM, o *duty cycle* (DuCy) no regime transitório de subida é alto e vai diminuindo conforme se aproxima do ponto de operação desejado, tentando não ultrapassá-lo. No regime de descida, acontece o contrário.

A linha azul do gráfico do meio simula alguns eventos gerados aleatoriamente pelo *perturbation*, e pode-se ver a linha verde variando junto, pois ela representa o erro em relação ao ponto ideal em análise. No gráfico superior, o evento gera uma variação na linha verde, que seria o estado real do sistema. A linha vermelha do mesmo gráfico mostra como o observador de estados prevê as condições do sistema, alterando a atuação do sistema antes de um evento realmente acontecer. Pode-se concluir que o observador predita muito bem essas condições visto que tal linha sempre acompanha de perto a linha verde.

É interessante ressaltar que quando o sinal de referência inverte de estado, a linha verde tem um atraso grande para conseguir acompanhar a referência, algo que é bem provável de acontecer em processos reais baseados em redes sem fio.

Analisando o gráfico inferior, que mostra o schedule, ou escalonamento, do nó sensor. Define-se por *schedule* o vetor que descreve se o nó está transmitindo ou não, e em quais momentos isso deve ocorrer. Esses comandos são programados baseados em níveis de tensão, sendo que o nível alto (*high*) significa o ato de transmitir (enviar ou receber). O nível médio (*medium*) quer dizer que o nó está em espera – por uma resposta, ACK ou outro pacote que esteja próximo de vir. E o nível baixo (*low*), que o nó está dormindo (*sleep*).

Também é interessante analisar o schedule do sensor de temperatura, que seria a corrente atingida para os eventos transmitidos. Quando, no esquema PWM, o sistema está num estado em que ele precisa de velocidade para atingir o ponto de operação, ou seja, o erro em relação ao ponto desejado está alto, o sensor envia informações de temperatura com maior frequência, possibilitando que o coordenador possa ajustar o DuCy da melhor forma possível, pois ele e o atuador estão no mesmo nó. Se o sistema está com erro baixo em regime permanente, o sensor pode prolongar seu tempo de *sleep,* enviando informações mais esparsas. É interessante lembrar que o sensor "acordar" para fazer medições não gasta tanta energia, mas a transmissão sim é um momento de grande gasto.

Grande parte do sistema de orientação a eventos está nessa capacidade do sensor de enviar mensagens em maior frequência quando necessário. Pois quando o sistema está em regime permanente, com DuCy baixo, e um evento ocorre, o sensor deve atualizar o sistema rapidamente para que se possa corrigir o erro através do PWM.

Além da análise feita em relação aos sinais e ao erro, fez-se uma estimativa teórica e linear do decaimento da bateria, mostrado na Figura 4.5. O comportamento linear visto aqui não ocorre em baterias reais, pois elas apresentam diferentes fatores que influenciam a queda da capacidade efetiva [14]. Ainda assim, espera-se que no processo real a vida útil da bateria também se prolongue, como visto no caso teórico.

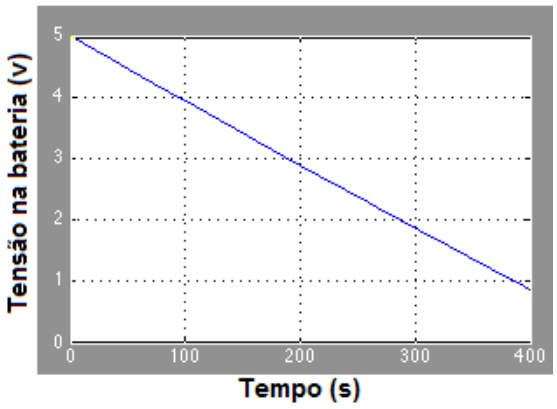


Figura 4.4: Forma de decaimento da bateria do sensor.

5 DESENVOLVIMENTO DO PROJETO

Esse módulo explica a montagem do experimento desde a parte física até a parte de programação. Apresentam-se aqui os equipamentos, bem como os softwares utilizados.

5.1 MONTAGEM FÍSICA E HARDWARE USADO

Para realizar os testes, buscou-se reproduzir um processo similar ao visto na simulação computacional, porém de maneira mais simplificada, sem o uso do observador para prever os eventos.

Essa seção apresenta os equipamentos utilizados para o desenvolvimento do trabalho e justifica, por meio da análise das características específicas, a escolha de cada um deles. Sempre que necessário, serão mostrados esquemáticos elétricos e de visualização das ligações feitas, ajudando a compreensão do sistema.

5.1.1 MAQUETE DE TESTES

Para a realização dos testes será necessário variar o *setpoint* da temperatura e esperar que o atuador agisse até alcançar o valor desejado, repetindo o processo algumas vezes. Usar um sistema real de ar condicionado seria incomodo para os ocupantes do ambiente, além de dificultar a obtenção de dados em situações específicas. Sendo assim, o melhor cenário encontrado para o experimento foi uma maquete em madeira e vidro que simula um ambiente com algumas salas. Essa maquete foi desenvolvida por integrantes do Laboratório de Robótica e Automação da Universidade de Brasília. Uma foto da maquete pode ser vista na Figura 5.1. Nesta figura é possível ver que o atuador usado foi um secador de cabelo que aquece a sala em estudo, onde se encontra o sensor, até a temperatura desejada. O acionamento do secador de cabelo é feito pela placa coordenador/atuador e se dá através do uso de um relé de estado sólido.

Figura 5.1: Maquete usada nos testes [2].

5.1.2 RELÉ DE ESTADO SÓLIDO

Um relé de estado sólido é um dispositivo semicondutor muito usado para ligar ou desligar dispositivos. Ele funciona como um interruptor eletrônico, sem partes mecânicas. Sua operação se dá por meio de tiristores que comutam quando uma determinada corrente passa por eles, eliminando a necessidade de contatos metálicos no interior do relé [18].

O relé de estado sólido utilizado foi o P240D4-17 da OPTO 22. Ele pode ser alimentado com tensões de 3 a 32 VDC e a tensão de operação AC na saída é 24-280 V [13].

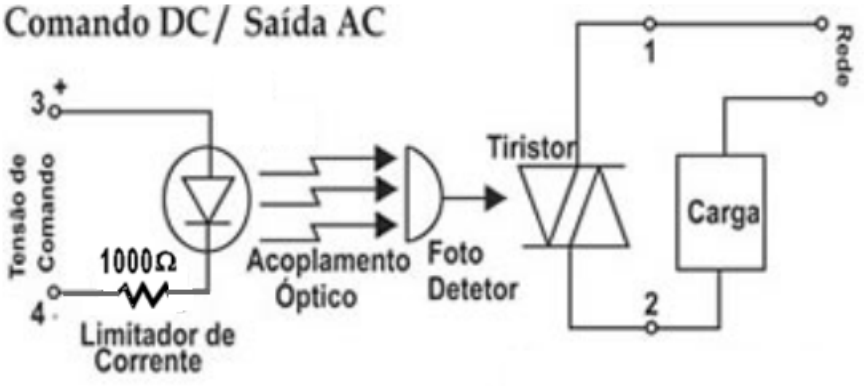


Figura 5.2: Esquemático interno do relé de estado sólido modelo P240D4 [13].

As conexões do sistema de atuação estão representadas nos esquemáticos das Figuras 5.3 e 5.4.

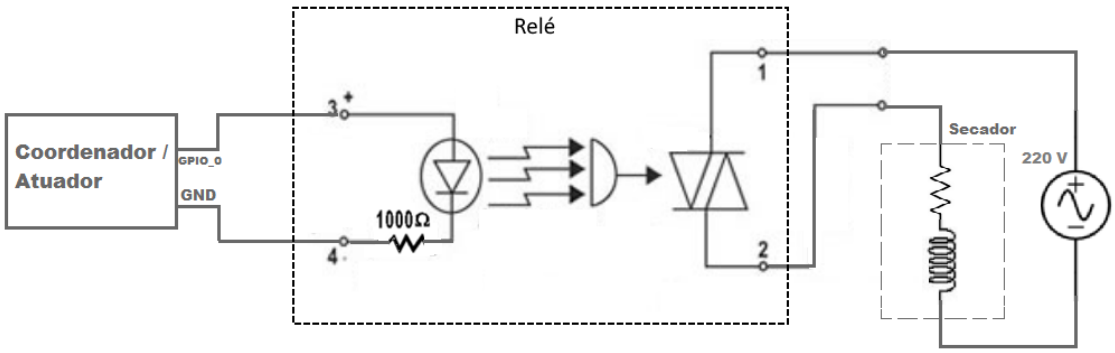


Figura 5.3: Esquemático elétrico da ligação do atuador.

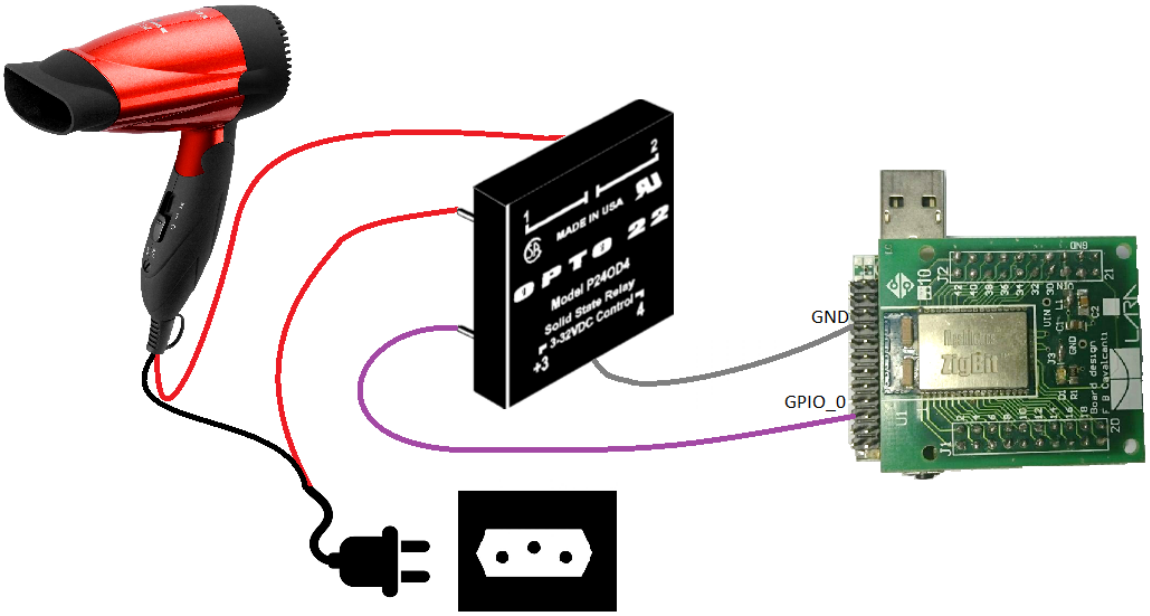


Figura 5.4: Visualização das ligações do circuito de atuação.

Com o circuito montado, temos que quando o atuador precisa ligar o secador, o sinal é enviado ao relé pelo pino GPIO_0, excitando a entrada. Assim que se tem tensão suficiente na entrada, o relé passa a operar e permite a corrente para o secador pelo pino 2, ligando-o. Quando for a hora de desligar o secador, o atuador interrompe o envio da tensão para o relé e este abre o contato.

5.1.2.1 ACIONAMENTO DO RELÉ – PWM

A sigla PWM vem do inglês, *Pulse-Width Modulation,* que em português significa modulação por largura de pulso, ou seja, é feito o controle de potência através da largura do pulso de um sinal periódico, de alta frequência em relação à banda passante do processo. Essa técnica é comum para o controle de velocidade de motores elétricos [10].

O funcionamento do PWM pode ser simplificado usando a ideia de uma chave liga e desliga simples. Quando a chave está ligada, 100% da potência é aplicada a carga e quando a chave está desligada a potência é 0. Se for possível controlar o tempo em que a chave permanece ligada e desligada, pode-se então controlar a potência média entregue a carga. Um exemplo seria uma chave ligada 40% do tempo e desligada 60%. Nesse caso, a potência média é 40%. Quanto mais tempo a chave permanecer ligada, maior será a potência média. A razão entre o tempo em que o pulso está em nível lógico alto e o tempo total do pulso é o *DutyCycle* (DuCy), definido em porcentagem.

A aplicação do PWM nesse projeto se dá no acionamento do relé que controla o atuador. O DuCy foi ajustado através de testes e tem seu valor variando de acordo com a situação em que se encontra o controle da rede. Detalhes sobre o método aplicado para definir *DutyCycle* serão informados na seção 5.2.2, que explica a utilização do PWM no experimento, e no Capítulo 6, que trata dos testes realizados.

5.1.3 MÓDULOS SEM FIO

Antes de se estabelecer uma rede, é necessário definir qual a proposta de utilização para esta. Nesse trabalho, espera-se que a rede seja capaz de controlar a temperatura de um ambiente usando um nó sensor sem fio e um atuador. Sabendo disso, conclui-se que pelo menos dois dispositivos são necessários, um como dispositivo final medindo a temperatura e o outro para trabalhar como coordenador/atuador recebendo os dados enviados pelo sensor e definindo se aciona o atuador ou não.

Sabendo o que cada dispositivo precisa fazer, o próximo passo é definir quais módulos serão usados em cada um. Nos dois casos, um módulo será de uso comum. É a placa ZigBit Channel 1.0 que terá a função de gravadora e coordenadora. Essa placa foi desenvolvida em estudos anteriores por alunos da Universidade de Brasília. Ela é alimentada por uma entrada USB, que além da alimentação, é responsável pela

transferência da programação do computador para as placas e pelo envio de dados das placas para o computador. A Figura 5.5 abaixo mostra esse módulo.

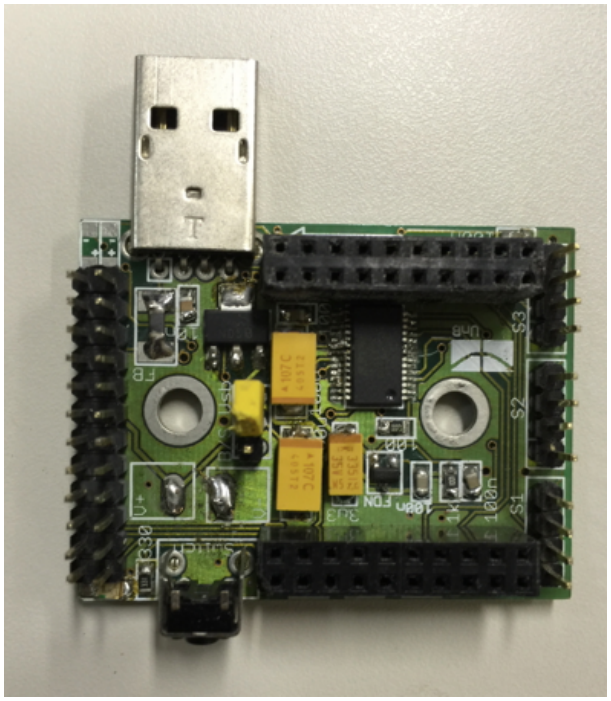


Figura 5.5: Placa ZigBit Channel 1.0, LARA/UnB, usada como Gravadora/Coordenadora.

A Figura 5.5 mostra que além dos dispositivos já mencionados, a placa também possui pinos machos, que serão usados para acionar o atuador e pinos fêmeas que serão usados para acoplar uma segunda placa, que contém o ZigBit. Essa segunda placa é chamada de ZigBit Breakout e também foi projetada por alunos da Universidade de Brasília, Felipe Brandão Cavalcanti e Vinícius Galvão Guimarães. Todos os dois nós usarão uma placa ZigBit Breakout semelhante à mostrada na Figura 5.6.

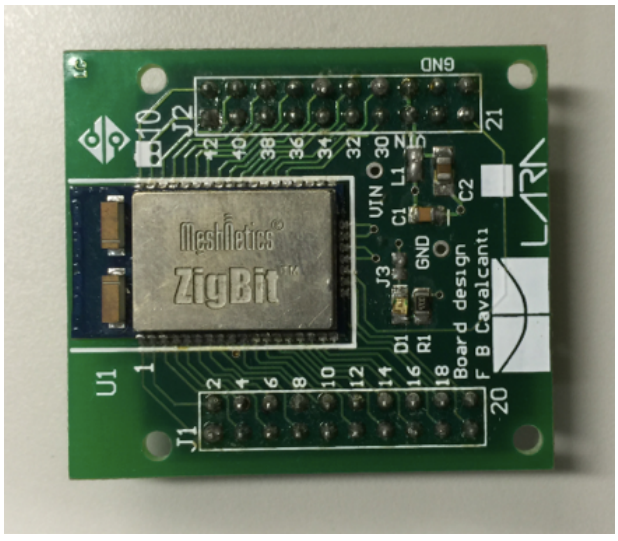


Figura 5.6: Placa ZigBit Breakout, LARA/UnB.

No caso do dispositivo final que será usado para medir a temperatura do ambiente controlado, a placa ZigBit Channel 1.0 não possui a conexão USB. Sua alimentação é feita por duas baterias AA de aproximadamente 1.5 V cada, como mostra a Figura 5.7.

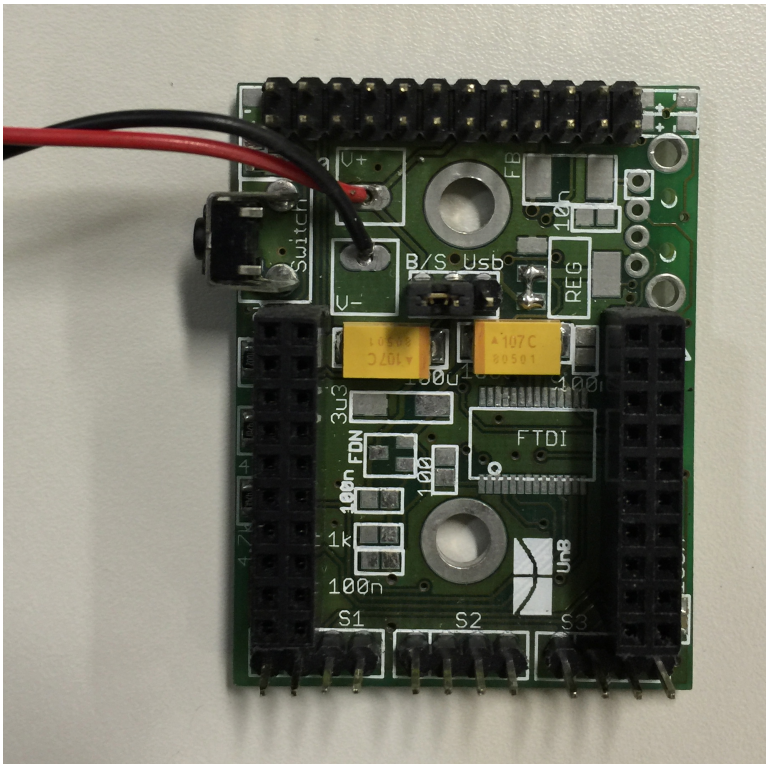
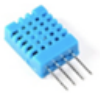


Figura 5.7: Placa ZigBit Channel 1.0 usada como dispositivo final.

5.1.4 SENSOR DHT22

As opções possíveis de sensores de temperatura para o projeto eram: o LM35, o DHT11, o DHT22 e o SHT71. O LM35 é analógico e por isso era o menos interessante para o projeto. O SHT71 é o mais preciso e com melhor qualidade entre todos os sensores analisados, o que implica em um custo bem mais alto. Como o trabalho aqui realizado não exige uma precisão tão alta, concluiu-se que o custo-benefício do SHT71 era desvantajoso. A decisão ficou entre o DHT11 e o DHT22. Os dois são semelhantes. A Tabela 5.1 ilustra algumas das diferenças entre eles e ajuda a entender qual o melhor sensor para ser usado nesse projeto.

Tabela 5.1: Comparação entre os sensores DHT11 e DHT22 [3].

	DHT11	DHT22
		
Alimentação	3 - 5.5 V	3.3 - 6 V
Faixa de leitura - Umidade	20 - 80%	0 - 100%
Precisão - Umidade	5%	5%
Faixa de leitura - Temperatura	0 - 50 °C	-40 - 125 °C
Precisão - Temperatura	+/- 2 °C	+/- 0,5 °C
Intervalo entre medições	1 s	2 s
Resolução	1 °C	0,1 °C

Vê-se que, apesar de o DHT22 ser mais lento que o DHT11, ele é capaz de medir uma faixa mais ampla de temperatura, com mais precisão e maior resolução. Como o projeto poderia trabalhar com temperaturas acima da faixa de leitura do DHT11, a opção do DHT22 se mostrou a mais sensata. Vale acrescentar também que durante a realização de testes com o sensor DHT11, percebeu-se que mesmo possuindo bytes destinados às partes decimais da humidade e da temperatura, ele só enviava as partes inteiras, o que prejudicava ainda mais a precisão.

O sensor escolhido, o DHT22, é de baixo consumo de corrente (2,5 mA durante medições, e 100-150 µA em *standby*), e possui internamente um sensor de umidade capacitivo e um termistor, além de um conversor analógico/digital para comunicação com o microcontrolador. Ele possui 4 pinos, dos quais apenas 3 são usados para conexão ao microcontrolador, como mostra a Figura 5.8 [3]. A comunicação é feita através de um único pino, diferentemente da técnica I2C, usada no SHT71. Quando o sensor recebe um pedido de medição, ele envia 5 *bytes* por medida, sendo os dois primeiros os dados da umidade, os dois seguintes os dados da temperatura e o quinto byte confere erro de *checksum*.

Figura 5.8: Pinagem do DHT22 [3].

Para conectar o sensor ao módulo, usou-se um resistor de 5,1 kΩ entre o pino 2, que envia dados do sensor, e o pino 1, do VCC. As imagens abaixo mostram o esquema da ligação das conexões entre a placa alimentada por bateria, o sensor e o multímetro.

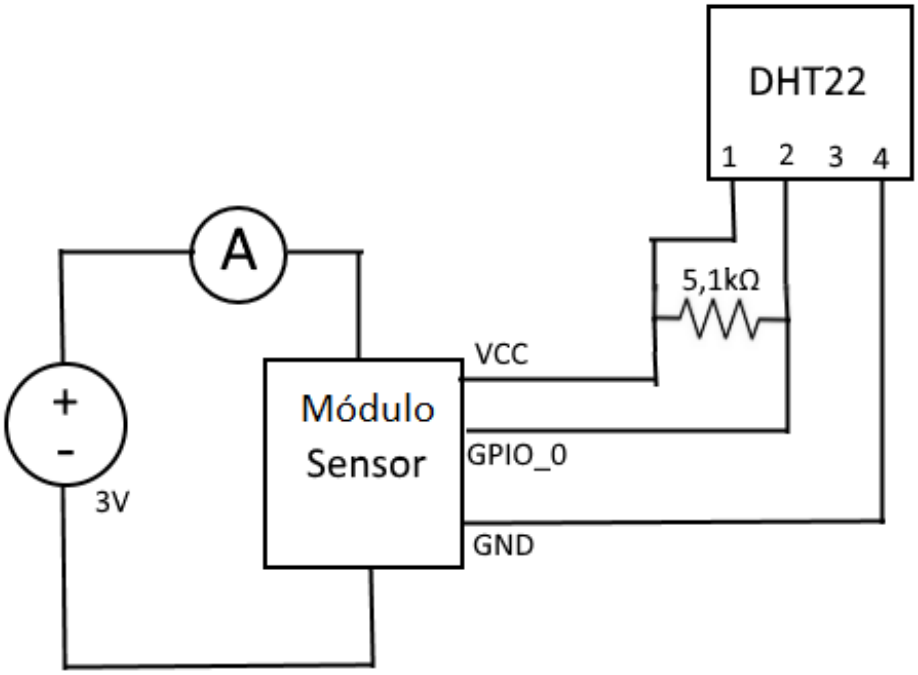


Figura 5.9: Esquemático elétrico da ligação do nó sensor.

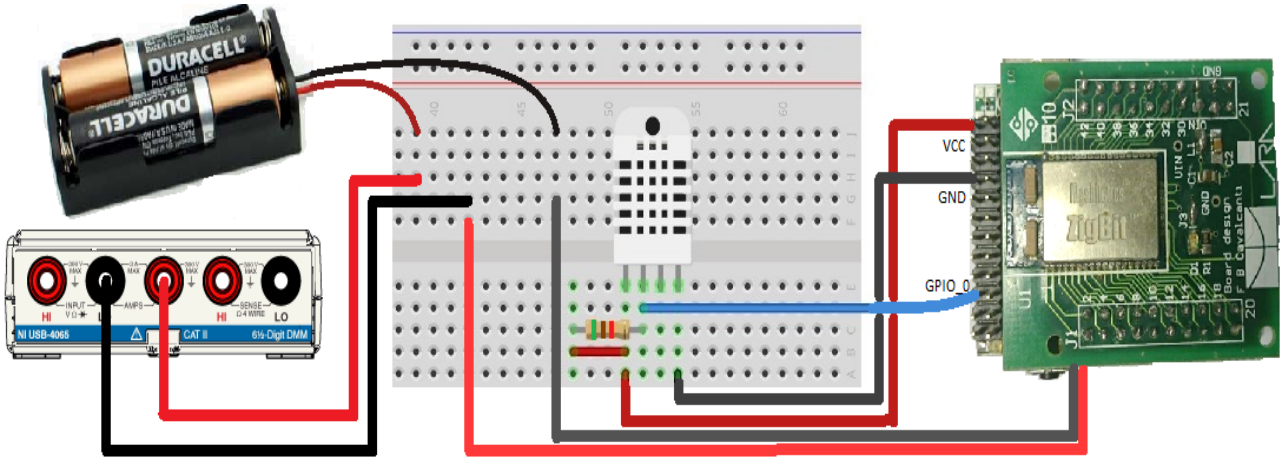


Figura 5.10: Visualização da ligação do sensor e multímetro.

5.1.5 MULTÍMETRO

Como um dos objetivos do trabalho é medir o consumo de corrente para cada método de controle testado, o multímetro precisa ser capaz de salvar as medições para que se consiga gerar gráficos de comparação. Como multímetros digitais com conexão direta ao computador são de alto custo, a opção foi utilizar o equipamento disponível nos laboratórios de Circuitos Elétricos da Universidade de Brasília.

Figura 5.11: Multímetro digital NI USB-4065 DMM [19].

O modelo é o NI USB-4065 DMM da empresa National Instruments. Algumas de suas características são relacionadas nas tabelas abaixo [11].

Tabela 5.2: Especificações do Multímetro.

Faixa máxima de corrente	
Alcançe	-3 A - 3 A
Sensibilidade	3 µA
Saída analógica	
Quantidade de canais	0
Temporização/Trigger/Sincronização	
Trigger	Digital

Entrada analógica	
Quantidade de canais	1
Resolução da entrada analógica	22 bits
Dígitos de resolução	6.5 digit
Faixa máxima de tensão	
Alcançe	-300 V - 300 V
Sensibilidade	1 mV

5.1.6 PROCESSO TÉRMICO INSTRUMENTADO

Para facilitar o entendimento da montagem física da rede, montou-se o esquemático da Figura 5.12, unindo todos os itens que compõem o sistema. Esse esquema foi feito usando como base uma planta baixa simples da maquete. Levando em conta que os detalhes das conexões já foram explicados individualmente, espera-se dessa figura apenas uma visão geral do posicionamento dos equipamentos.

Os termos J1, J2, J3 e J4 são referentes as janelas da maquete. As portas estão representadas por P1, P2, P3, P4 E P5. Essas representações serão usadas para detalhar eventos criados durante os testes. Alguns testes serão que com orientação a eventos.

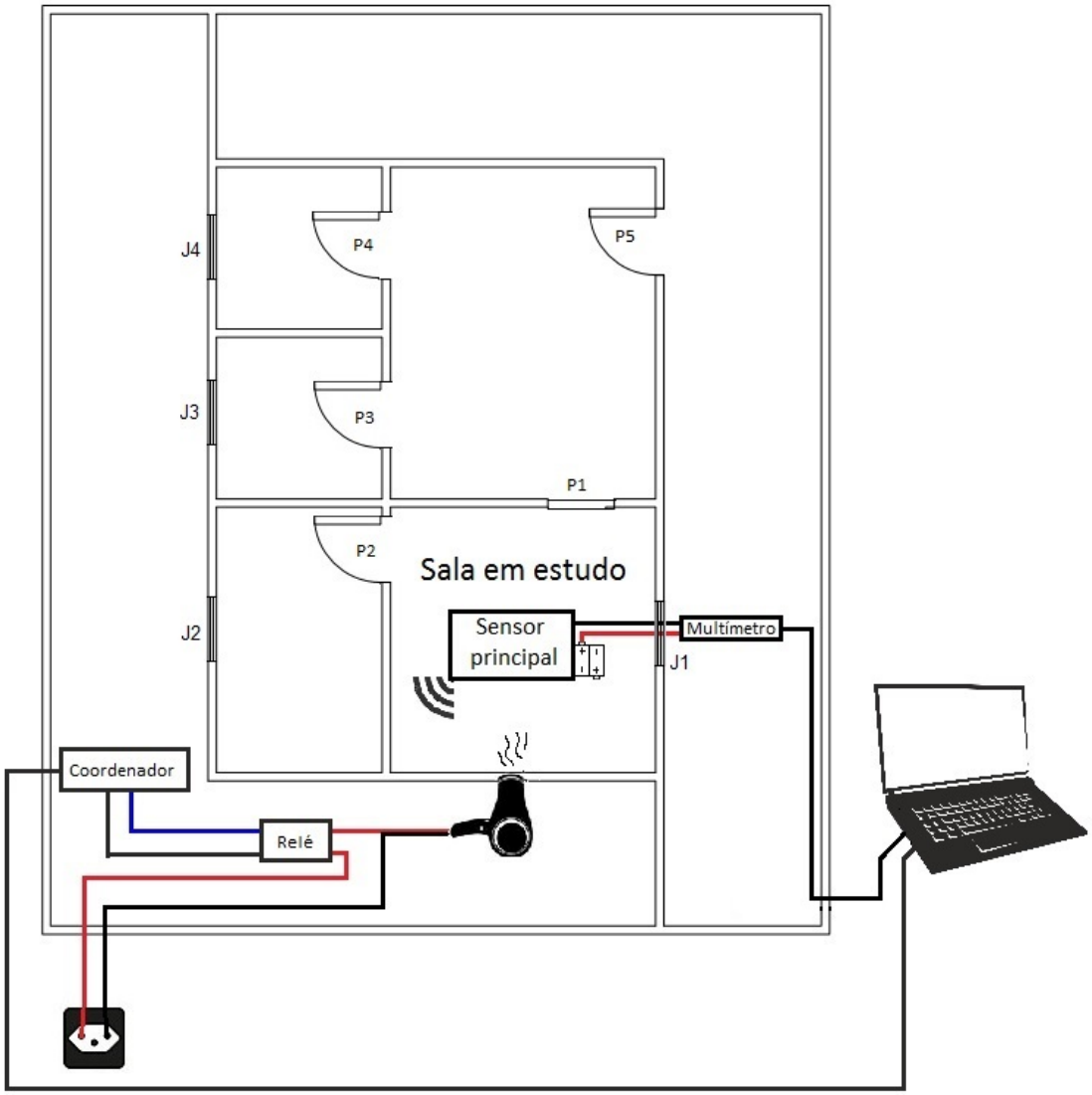


Figura 5.12: Esquema completo da rede física.

5.2 CONSTRUÇÃO DO AMBIENTE DE TESTES

Aqui serão apresentados os softwares utilizados, bem como a estruturação do ambiente de testes. Será montado um passo a passo da programação feita desde a criação de uma rede até se chegar no controle orientado a eventos.

5.2.1 SOFTWARE UTILIZADOS

Alguns softwares se fizeram necessários durante o projeto. São softwares encontrados facilmente para download grátis através da internet e não é o foco do

trabalho explicar em detalhes o funcionamento de cada um. Portanto, será indicada apenas a utilidade destes no desenvolvimento das atividades aqui relatadas.

Simulink

Usado para a simulação do experimento.

AVR Studio

Software IDE (Ambiente de Desenvolvimento Integrado) usado para a programação dos módulos ZigBit.

MatLab

Útil para ler os dados enviados pelo sensor ao coordenador e gerar os gráficos.

CoolTerm

Escreve na tela do computador as informações passadas pelos módulos e facilita o acompanhamento do processo. Além de permitir que dados recebidos sejam enviados para um arquivo de texto para possíveis tratamentos matemáticos.

LabVIEW SignalExpress

Software usado para adquirir os dados medidos pelo multímetro.

Bootloader PC Tool

Carrega os códigos compilados nos módulos ZigBit.

5.2.2 PROGRAMAÇÃO DA REDE

Estabelecer comunicação e programar o sensor

A programação da rede se deu por etapas, seguindo uma lógica de prioridades. Obviamente, o primeiro passo foi estabelecer uma comunicação entre coordenador e o dispositivo final. Usou-se como base para isso um exemplo de projeto incluso no Bitcloud, o Peer2peer. Feito isso, buscou-se realizar a programação do sensor para que esse enviasse ao coordenador os valores de temperatura medidos. Para isso, as informações de uma biblioteca voltada para o uso do sensor DHT22 junto ao Arduino serviram de ajuda para o desenvolvimento de uma biblioteca própria para o Bitcloud.

Após essas duas primeiras etapas, uma rede básica para medir a temperatura e imprimi-la no monitor já estava funcionando e decidiu-se que a redução do consumo de energia, por meio do modo *sleep*, seria o próximo passo.

Aplicar o modo *sleep*

Para entrar em modo *sleep*, a placa deve estar configurada como dispositivo final, o que já acontecia na rede básica pré-estabelecida. Entrar em modo *sleep,* de maneira simplificada, significa que os canais de transmissão e recepção sem fio do ZigBit são desativados. Isso mantém o gasto de energia em função quase que exclusivamente do consumo interno do controlador. As medições realizadas mostraram um consumo de corrente entre 30 μA e 40 μA enquanto dormindo e próximo de 30 mA durante a transmissão, ou seja, com o canal de transmissão ativo, o consumo é aproximadamente 1000 vezes maior.

A programação aqui usada é para que o nó sensor realize a medição, a envie para o coordenador e logo em seguida "durma". Após certo tempo ele irá acordar para repetir o processo. Isso implica que, quando estiver dormindo, nenhum valor de temperatura será coletado. A implementação desse modo *sleep* será de grande valia para aumentar a vida útil da bateria.

Conectando o atuador

A partir daqui os passos não estão mais diretamente relacionados com a economia de bateria, mas sim com o controle do processo. Para que se controle a temperatura, é preciso conectar um atuador à rede. Como explicado anteriormente, o atuador aqui usado é um secador de cabelo, cuja função é aquecer um ambiente da maquete. Por praticidade, o nó atuador foi programado junto ao coordenador. A primeira ideia para esse sistema foi utilizar um controle Liga-Desliga simples, apenas para testar a interação entre coordenador/atuador e dispositivo final. Foi estabelecido uma temperatura fixa, ou *setpoint*, e quando o valor de temperatura medido estivesse abaixo desse *setpoint* o atuador agia, e quando o valor medido atingisse o ponto desejado o atuador desligava. Esse tipo de controle não é tão interessante, pois, por trabalhar sempre nos extremos de potência do secador, ou 100% ou 0%, provoca muita oscilação da temperatura.

Aplicando o PWM

Com a comunicação da rede funcionando perfeitamente até esse ponto, a etapa seguinte era melhorar o controle da temperatura, reduzindo o erro e aumentando a velocidade. O que pode ser traduzido em um funcionamento mais suave. Para isso, é preciso utilizar uma atuação melhor que o método on-off, pois este causava muita oscilação da temperatura. Deseja-se que o secador de cabelo trabalhe com potência variável de acordo com a necessidade da automação. Sendo assim, quando a temperatura estiver próxima do ponto de operação, o secador deve apresentar uma potência suficiente para que a curva se mantenha dentro da faixa limite sem muita oscilação. Isso é importante por diminuir a quantidade de informações necessárias para manter o controle, ou seja, menos envios serão feitos, economizando bateria.

O método usado para obter esse resultado foi programar direto pelo BitCloud o controle PWM, pois ele permite variar a potência de operação do secador de cabelo de acordo com a relação entre a temperatura medida e o *setpoint.*

O Zigbit tem como microcontrolador o ATMega1281 [22], que permite 4 modos de PWM possíveis, o *fast mode*, o *phase correct mode* e outras duas derivações desses.

Fast mode : O temporizador conta de zero até o valor máximo, também chamado de *top*, que é predefinido como 255 geralmente, e então é zerado no próximo ciclo do *clock.* Quando o temporizador atinge o valor comparativo, ele alterna o estado da saída. O estado é reiniciado quando o temporizador é zerado. O estado inicia em cada ciclo como 1 ou 0 dependendo se está selecionada a saída não-invertida ou invertida, respectivamente.

Phase Correct mode: Tem funcionamento parecido, mas neste modo o temporizador contar até o valor máximo e, ao invés de zerar, começa a contar no sentido contrário, do valor máximo até zero. Desta maneira, o funcionamento é similar ao *Fast mode*, sendo que a frequência do PWM é a metade da primeira para mesmas condições. O valor máximo também vem predefinido com o controlador.

Fast mode e Phase Correct mode com valor máximo variável: De ambos modos apresentados anteriormente, derivam-se mais dois, nos quais o *top* do registrador pode ser definido pelo usuário. A documentação do BitCloud é pobre em relatar como ele funciona, fazendo com que a biblioteca “pwm.h” seja a única fonte de

informação sobre os modos de operação. Analisando essa biblioteca, acredita-se que o *Phase Correct mode* com o valor máximo variável é o utilizado pelo ZigBit.

O parâmetro top já foi explicado como sendo o valor máximo do registrador e foi definido como 75. E o *prescaler*, que significa o valor por quanto o *clock* será dividido, foi definido como 256. O Bitcloud tem programado como *prescaler* algumas opções, como: 1, 8, 64, 256 e 1024. Para calcular a frequência [20], com o *prescaler* sendo 256, o top em 75 e o *clock* do processador sendo 16 MHz, temos:

$$f_{PWM} = 16\ x\ 10^6 / 256 / 75 / 2 / 2 = 208{,}33 \text{ Hz}$$

Variação do *Setpoint*

Para aproximar o processo dos testes de uma situação real, trabalhou-se com a variação do *setpoint.* A intenção era simular um ambiente real refrigerado por um sistema de ar condicionado acionado por PWM e com o ponto de operação sendo a temperatura escolhida pelo usuário. Um exemplo dessa situação é um escritório onde trabalham duas pessoas, uma de manhã e outra a tarde, sendo que a da manhã deseja que a temperatura fique próxima de 25˚C e a da tarde prefere 20˚C. Para simular essa variação, foi gerada uma onda quadrada com valores de 60˚C em alta e 50˚C em baixa, sendo que a cada 20 minutos esses valores se alternam. Esses *setpoints* foram escolhidos empiricamente, através da busca de valores que permitissem que o tempo de queda da temperatura medida não fosse tão longo, pois o aquecimento é via atuador e o resfriamento é de forma natural por troca de calor com o ambiente externo mais frio.

Com o que foi dito até agora, têm-se que quando a rede é estabelecida, o sensor mede a temperatura, a compara com o *setpoint*, envia para o coordenador uma mensagem e entra em modo *sleep*. O coordenador analisa a informação recebida e define se o atuador deve agir e se for agir, define também com qual *Duty Cycle*. Esse processo se repete em intervalos de tempo constantes entre as medições, pois o sensor ainda não tem programação que o indique em qual momento ele deve ou não transmitir os valores medidos ao coordenador. E é essa a função da próxima parte.

Automação orientada a eventos

A última etapa da programação da rede é voltada para o controle orientado a eventos. É aqui que será informado ao nó sensor a quantidade de informação necessária, ou seja, qual o período entre os envios.

Variar a quantidade de informações enviadas ao coordenador interfere diretamente no controle do sistema. Quanto menor for o tempo entre os envios, mais fácil e preciso será o controle, porém com maior consumo de bateria. Por consequência, reduzir a quantidade de transferências torna o sistema mais econômico. Entretanto, se essa redução for feita sem planejamento prévio, dados importantes podem não ser coletados, afetando negativamente o controle da temperatura.

O funcionamento do sistema orientado a eventos já foi explicado na seção 4.1 e resta apenas definir o que será considerado um evento. Deseja-se que a temperatura opere sempre dentro de um faixa entre ±1 ˚C em relação ao ponto de operação. Sabendo disso, definiu-se três tipos de evento diferentes para a comparação dos resultados.

Tabela 5.3: Definição dos eventos

	CASO 1	**CASO 2**	**CASO 3**
EVENTO	Erro > |0,1 ˚C|	Erro > |0,5 ˚C|	Erro > |1 ˚C|

5.3 ABORDAGENS DE CONTROLE

A proposta desse trabalho é usar uma técnica de controle que permita comparar sistemas orientados a eventos aos sistemas não orientados. Isso implica que não se faz necessário encontrar a melhor entre as técnicas existentes, basta que se escolha uma capaz de evidenciar a comparação. Sendo assim, optou-se por trabalhar com opções mais simples e que não exigem tanto aprofundamento em relação a identificação da planta, permitindo que se investisse mais tempo e esforço no objetivo central.

Foram definidas duas opções a serem testadas, controle liga-desliga e controle proporcional com PWM. O diagrama de controle usado é simplificado pela Figura 5.14, sendo que os dois tipos foram executados individualmente, trabalhando em função da lógica de cada um.

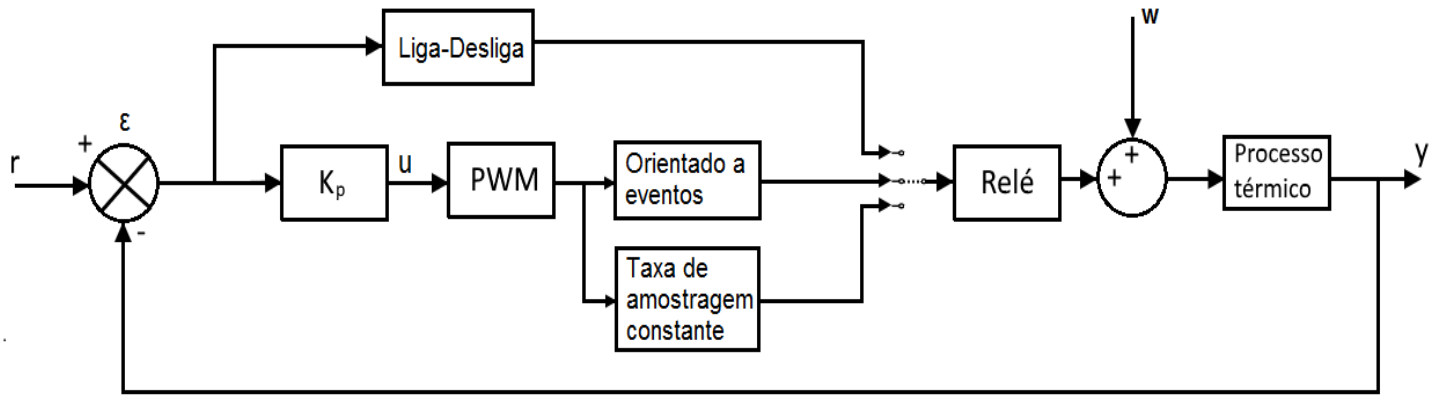


Figura 5.13: Sistemas de controle aplicados ao processo térmico.

"r" é o *setpoint*;

"ε" é o erro entre o *setpoint* e a temperatura medida

"u" é o sinal contínuo entre 0 e 1 que controla a ação de contole;

"w" são as perturbações por aberturas de portas e janelas;

"y" é a temperatura medida;

5.2.1 CONTROLE LIGA-DESLIGA

O controle liga-desliga possui uma lógica bem simples. Ele funciona baseado no erro entre a entrada e a saída. No caso desse trabalho, a entrada é o valor da temperatura desejada, o *setpoint,* e a saída é a temperatura medida pelo sensor. A lei de controle é:

$$\begin{cases} 1, se\ (r-y) > 0 \\ 0, se\ (r-y) < 0 \end{cases}$$

Sendo assim, quando a temperatura medida for menor que a do ponto de operação o secador liga, e só desliga quando a temperatura medida passar desse ponto.

5.2.2 Controle proporcional

Nessa técnica, a ação de controle é proporcional ao erro. Quanto maior o erro, maior a ação corretiva. Compatível com o controle proporcional, opera-se o PWM, permitindo que o secador de cabelo não opera só em 1 ou 0 como no liga-desliga. O funcionamento geral do PWM já foi explicado anteriormente e resta agora entender como o processo definirá o *Duty Cycle* para controlar a potência do secador.

Pelo diagrama da Figura 5.14, vê-se que o sinal que controla o PWM é chamado de "u". Na lei geral do controle proporcional, esse sinal é definido pela equação

$$u = Kp(r - y),$$

que basicamente multiplica o erro entre a entrada e a saída por um ganho.

No entanto, essa lei não será aplicada a todas as faixas de temperatura nos testes, pois definiu-se uma banda proporcional. Essa banda também pode ser chamada de faixa de controle, ou seja, é a faixa de temperatura que se deseja ou que se pode controlar.

Essa faixa foi defina com largura de 2,5 ˚C, entre *setpoint* + 0,7 ˚C e *setpoint* – 1,8 ˚C. Ou seja,

$$setpoint - 1{,}8\,^{\circ}C < banda\ proporcional\ < setpoint + 0{,}7\,^{\circ}C.$$

Baseando-se nessa banda e na lei de controle citada, será definido um algoritmo capaz de automatizar o sistema. Os valores fora da banda não obedecem a esse algoritmo e terão atuações predefinidas.

Quando a temperatura estiver abaixo da banda, o atuador estará funcionando com 100% da potência para que se chegue o mais rápido possível na região controlada. Quando o valor é superior à banda, isto é, o ambiente esquenta além dos 0,7 ˚C acima do ponto de operação, tem-se uma situação que o sistema não consegue mais controlar, pois a única ação de controle é o aquecimento e para que a temperatura caia novamente, o atuador desliga até que se atinja a faixa de controle novamente. A Figura 5.15 mostra as diferentes faixas usadas.

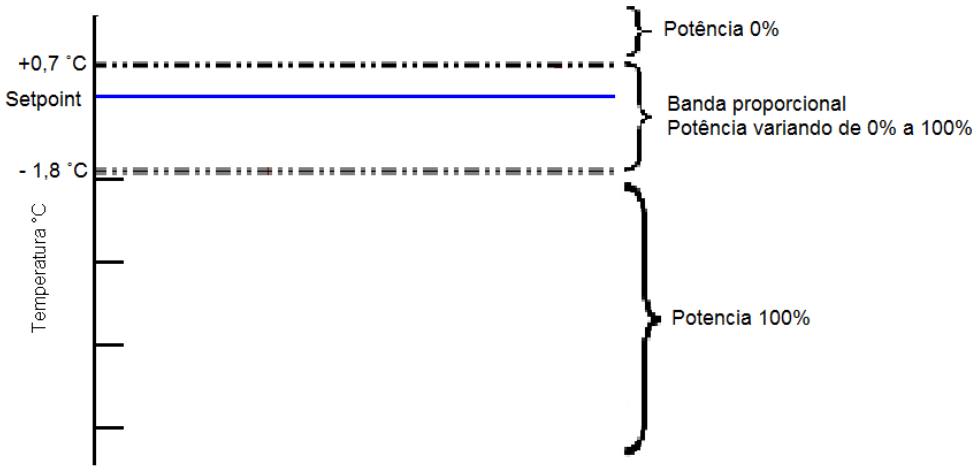


Figura 5.14: Banda proporcional

O algoritmo de controle desse sistema é:

$$\begin{cases} Se\ erro > 1{,}8\ ^{\circ}C, atua\ com\ 100\%\ da\ potência \\ Se\ erro\ < -0{,}7\ ^{\circ}C, desliga\ o\ atuador \\ Se - 0{,}7\ ^{\circ}C < e < 1{,}8\ ^{\circ}C, aplica\ u = Kp(r - y) \end{cases}$$

Pelo algoritmo, tem-se que o PWM irá variar o DuCy de 100% na temperatura igual a *setpoint* – 1,8 ˚C até 0% na temperatura igual a *setpoint* + 0, 7. Isso implica que o valor a ser considerado para "*r*" será deslocado 0,7 ˚C acima do *setpoint*. Esse deslocamento foi pensado para permitir que *u* assuma valores maiores que zero mesmo acima da temperatura que se deseja controlar, mantendo uma potência baixa nessa região. Sem esse deslocamento, a potência do secador seria muito baixa quando a temperatura se aproximasse do ponto de operação e zero sempre que ultrapasse esse ponto, dificultando a manutenção do controle nesse faixa.

A equação de *u,* levando em conta essa alteração, é a seguinte:

$$u = \mathrm{Kp}[(setpoint + 0{,}7) - temp\ medida] =$$

$$= \frac{1}{0{,}7 - (-1{,}8)}[(setpoint + 0{,}7) - temp\ medida]$$

$$= \frac{1}{2{,}5}[(setpoint + 0{,}7) - temp\ medida]$$

$$= 0{,}4[(setpoint + 0{,}7) - temp\ medida].$$

5.3 SEQUÊNCIA DE TAREFAS DA REDE

Para facilitar o entendimento do escalonamento de tarefas da rede, construiu-se o esquemático da Figura 5.15, mostrando o ciclo de funcionamento da rede orientada a eventos com controle proporcional e acionamento do atuador por PWM.

Figura 5.15: Sequência de tarefas da rede.

6 RESULTADOS

Este capítulo apresenta os resultados obtidos, analisa os gráficos e compara com o esperado.

Antes de apresentar os resultados, é importante que se entenda a lógica usada para a obtenção destes. Afim de comprovar a eficiência da rede orientada a eventos, decidiu-se comparar o comportamento desta ao de redes em que a taxa de envios é constante, ou seja, o nó sensor mede a temperatura e envia para o coordenador com períodos fixos, sem interromper as transmissões quando operando próximo do valor desejado.

Serão gerados dois gráficos para cada teste, o da variação da temperatura no tempo e o do sinal *u* no tempo. Além desses gráficos, será calculado consumo médio de corrente em cada caso. Por serem experimentos bem semelhantes, a análise dos resultados será simples, dependendo apenas dessa comparação entre os diferentes cenários.

Gráfico da variação da temperatura no tempo

Esse gráfico mostra a curva da temperatura medida junto com a onda quadrada do *setpoint,* que altera o valor a cada 20 minutos. Esse gráfico é usado para saber se o controle do processo foi satisfatório ou não. Quanto mais próximo do ponto de operação, significa que o controle foi mais preciso. Devido as subidas e descidas de temperatura serem lentas, deu-se foco para a análise da região de regulação de cada curva, desprezando os erros maiores nas regiões de aquecimento e resfriamento. Como já explicado no capítulo anterior, planeja-se que a qualidade do controle diminua cada vez mais com o aumento do período entre as medições, até chegar num ponto que se torna insatisfatório. Foram adicionadas linhas tracejadas em ± 1 ˚C em relação ao ponto de operação, tornando mais evidente a faixa de erro aceitável. Nesses gráficos, a curva azul é o *setpoint* e a vermelha é o comportamento da temperatura.

Gráfico do sinal u no tempo

Esse gráfico mostra a resposta do algoritmo de controle em relação as temperaturas medidas. Quanto menor o período entre os envios das temperaturas ao coordenador, mais rápido o sinal *u* será capaz de variar, mostrando uma curva mais suave. Quanto menor o valor de *u,* menor o erro da temperatura naquele ponto.

Corrente média consumida

A terceira informação a ser estudada é do consumo médio de corrente. O multímetro usado acompanha todos os valores da corrente que é gasta pelo nó. Os valores medidos e armazenados pelo LabVIEW SignalExpress foram exportados em forma de planilha e tratados no MatLab, informando o valor da corrente média consumida. Observou-se que os valores comumente consumidos durante o *sleep*, durante a medição e durante a transmissão foram, respectivamente, 40 μA, 2 mA e 24,5 mA. Quanto maior o número de envios no período, maior será o gasto de energia.

O resfriamento da sala de estudo da maquete é feito por perda de calor, principalmente para o ambiente externo, que variou a temperatura entre 23,5 até 26,5 ˚C. Feitas as considerações gerais a respeito dos dados a serem observados e do que se espera em cada situação, basta confirmar na prática os resultados desejados.

6.1 CONTROLE SEM ORIENTAÇÃO A EVENTOS

Os resultados dos sistemas atuando com PWM e sem orientação a eventos devem mostrar que, para período fixos pequenos entre os envios, a temperatura fica bem próxima do ponto de operação. No entanto, quando esse período aumenta, o controle fica prejudicado e a atuação passa a ficar parecida com o método Liga-Desliga. Os casos analisados operam com intervalo de 2, 5, 15 e 30 segundos entre as transmissões.

CASO COM INTERVALOS DE 2 SEGUNDOS

Ressalta-se aqui que, como mostrado na Tabela 5.1, o intervalo mínimo entre medidas que deve se respeitar para o bom funcionamento do sensor DHT22 é de 2 segundos. Portanto, o intervalo entre as medições partiu desse valor. Abaixo estão os gráficos gerados para essa configuração.

Analisando o gráfico da Figura 6.1, pode-se ver que com a atuação do PWM a curva da temperatura acompanha muito bem o ponto de operação, mantendo um erro extremamente pequeno na região de regulação. Alguns erros de medição por se trabalhar com medições no limite da velocidade do sensor geraram picos aleatórios, mas que não interferem a visualização geral do resultado. Devido a apresentar erro pequeno, o sinal u não atingiu valores altos e teve um comportamento com poucas variações na região de regulação.

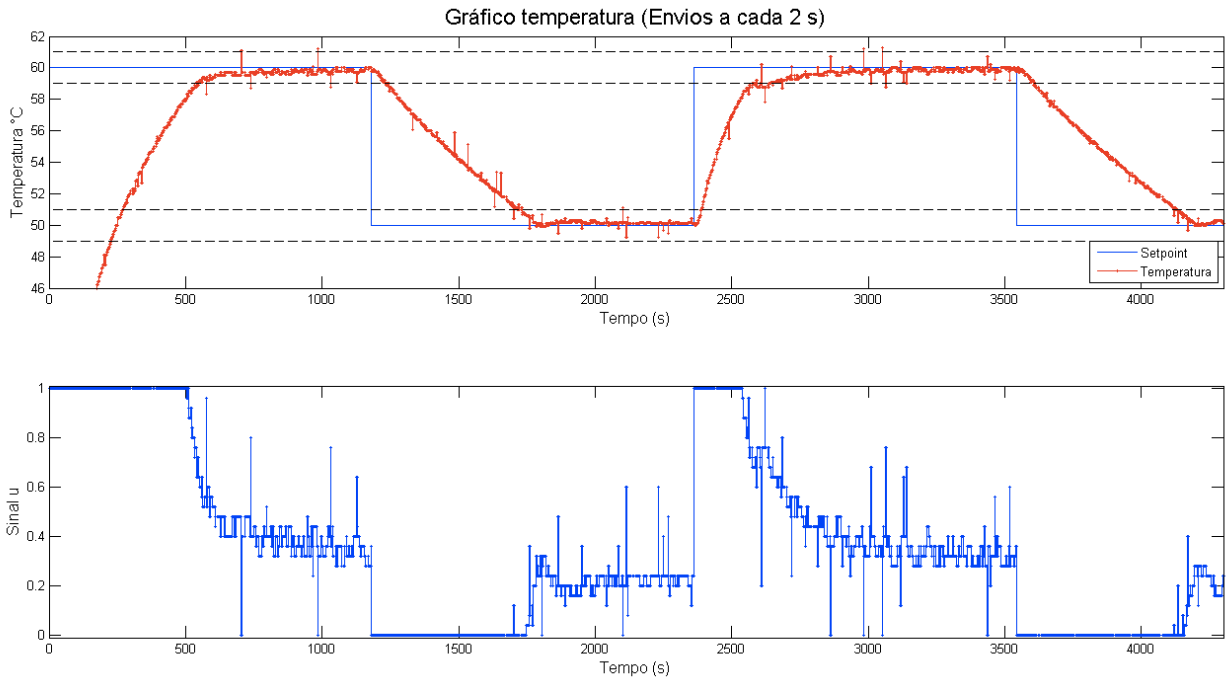


Figura 6.1: Variação da temperatura com controle P + PWM e com intervalo entre envios de 2 s.

O cálculo da corrente média consumida foi estimado em 31 mA. Além da grande quantidade de medições, um agravante para o aumento do gasto de energia é o fato de que para um intervalo de 2 segundos, o sensor tem pouco tempo para entrar no modo *sleep* e rapidamente tem que sair dele para transmitir. Assim sendo, a corrente não se estabiliza ao redor de um ponto fixo e varia entre 65 µA e 95 µA, um valor muito pequeno para correntes, mas que não é o melhor que o ZigBit consegue alcançar.

Conclui-se desse intervalo de 2 segundos atuando com PWM que o controle da temperatura foi excelente e o consumo de corrente foi alto.

Em relação ao controle com Liga-Desliga mostrado na Figura 6.2, nota-se que a temperatura não consegue se estabilizar e apresenta variação bem maior. Quanto ao consumo médio de corrente, o valor foi de aproximadamente 37 µA, bem próximo do caso com atuação por PWM.

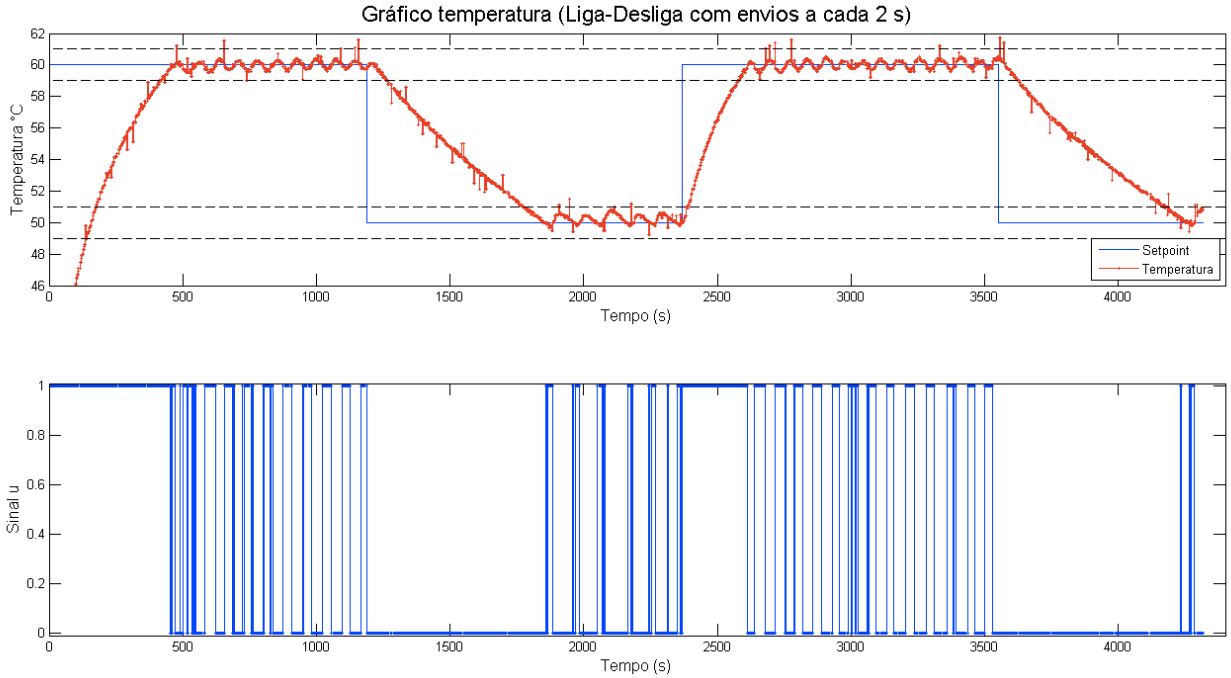


Figura 6.2: Variação da temperatura com controle Liga-Desliga e com intervalo entre envios de 2 s.

Analisando os dois tipos de controle, fica claro que o PWM permite um controle muito mais estável e preciso da tempearatura.

CASO COM INTERVALOS DE 5 SEGUNDOS

Com o aumento do intervalo entre as medições e envios, o sensor demora mais a perceber algumas alterações na temperatura, fazendo com que a atuação atrase um pouco, causando maiores variações do sinal *u*. Apesar da piora em relação ao caso de 2 segundos, a curva da temperatura medida ainda se mantém próxima do ponto de operação.

Por outro lado, um menor número de medidas deve melhorar a economia de energia, pois como foi dito, transmitir é o que gasta mais energia no Zigbit. Como intervalo entre as transmissões foi um pouco maior, pôde-se observar que a corrente no modo *sleep* se estabiliza melhor em torno de um ponto, aproximadamente 40 µA, ao contrário do que acontecia no caso anterior. O consumo reduziu pela metade, chegando em 1,6 mA.

Portanto, a temperatura ainda permaneceu controlada e com menor gasto de bateria em relação ao intervalo de 2 segundos.

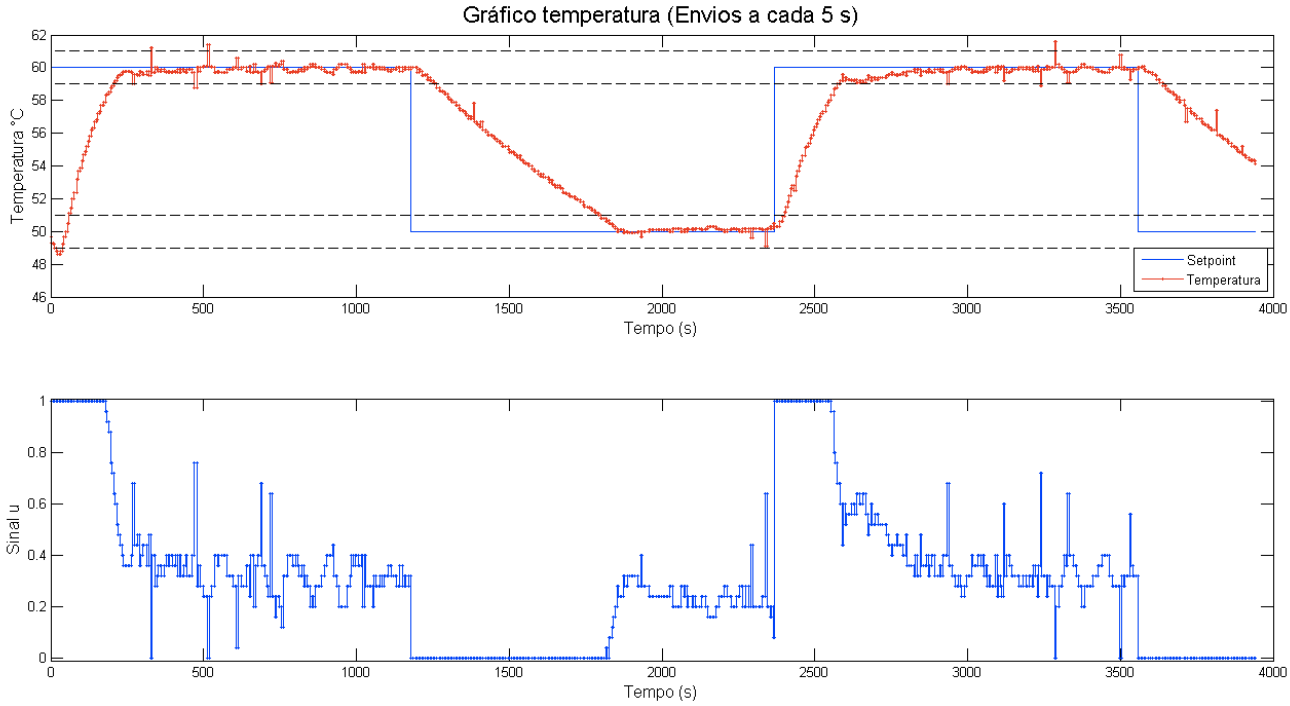


Figura 6.3: Variação da temperatura com controle P + PWM e com intervalo entre envios de 5 s.

CASO COM INTERVALOS DE 15 SEGUNDOS

Triplicou-se o intervalo entre os envios para 15 segundos. A Figura 6.4 mostra que o controle da temperatura se tornou mais complicado. Os erros foram bem mais visíveis, com oscilações maiores do sinal *u*. O aumento do intervalo já começa, então, a fazer diferença na eficiência.

A corrente média consumida foi, como esperado, menor que os casos anteriores e atingiu o valor de 508,57 µA.

Comparando a controle P + PWM da Figura 6.4 ao controle Liga-Desliga da Figura 6.5, percebe-se a vantagem do primeiro método, que conseguiu manter a

temperatura com erro bem menor. No caso Liga-Desliga, para o *setpoint* igual a 50 ˚C, observou-se erros bem altos, por volta de 2,5 ˚C. Ou seja, o controle não foi eficiente.

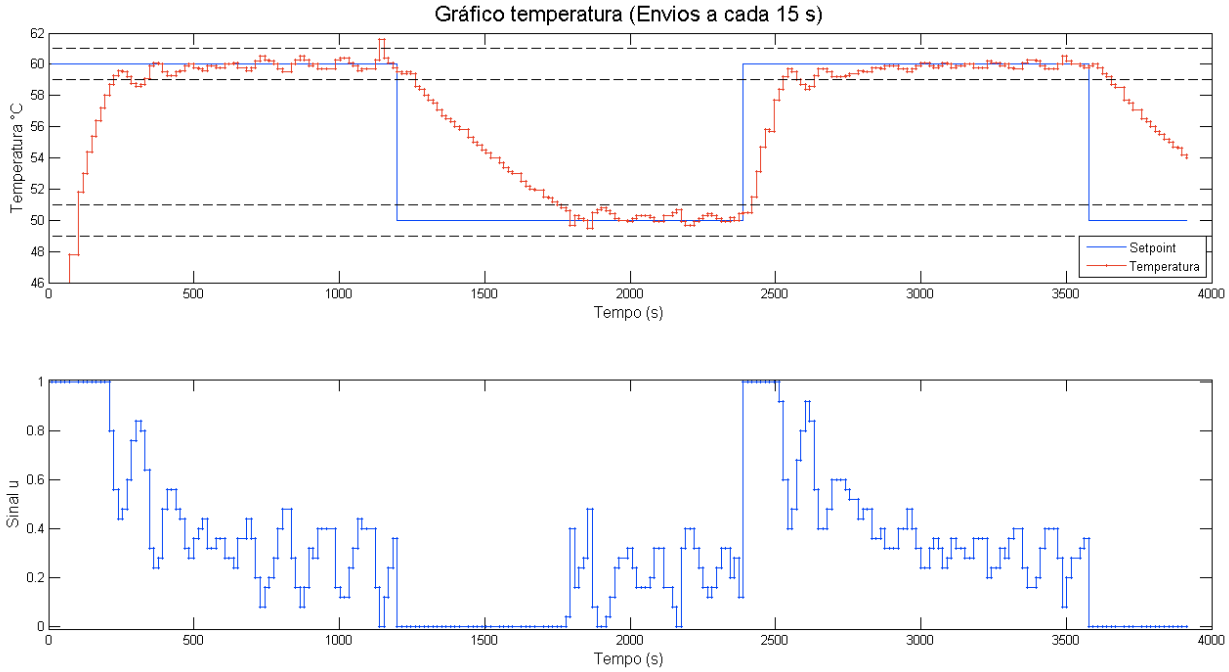


Figura 6.4 Variação da temperatura com controle P + PWM e com intervalo entre envios de 15 s.

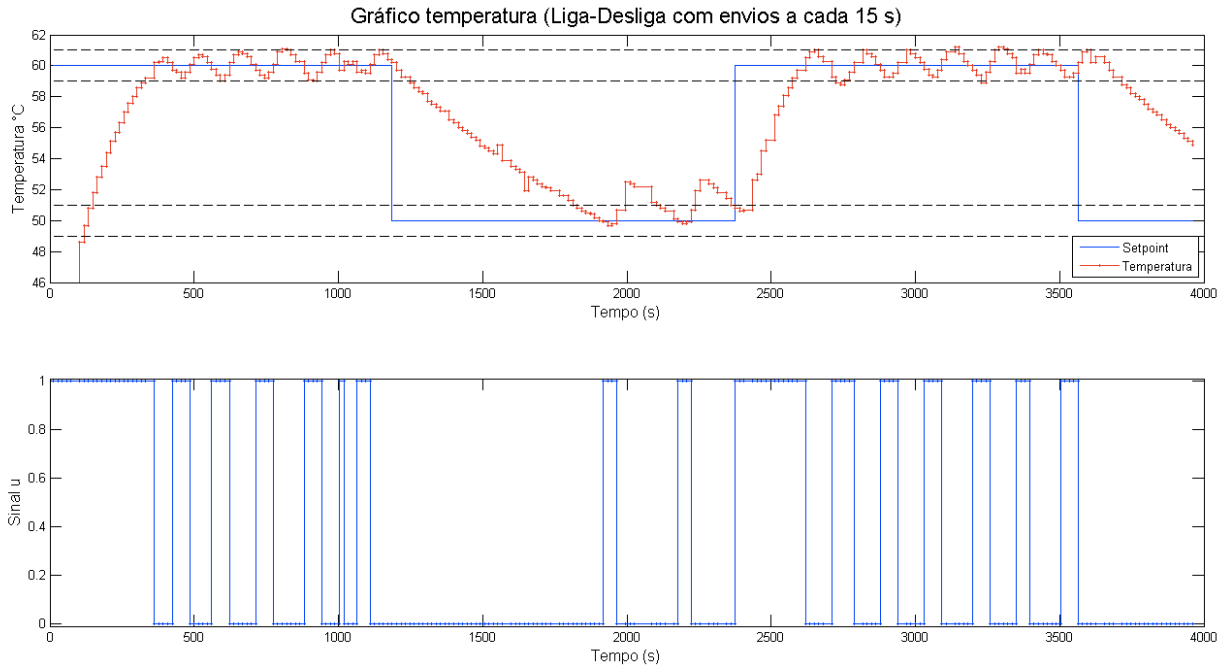


Figura 6.5: Variação da temperatura com controle Liga-Desliga e com intervalo entre envios de 15 s.

CASO COM INTERVALOS DE 30 SEGUNDOS

Quando se aumentou o intervalo entre os envios para 30 segundos, o atraso na percepção das mudanças de temperatura foi longo o suficiente para fazer com que a variação ficasse bem evidente. O sinal *u* não conseguiu se manter estável numa faixa intermediária, e mostrou altos e baixos devido aos erros maiores.

Como a intenção era manter a temperatura sempre dentro da faixa de erro aceitável e em diversos pontos o erro ultrapassou esse limite, pode-se se concluir que o controle não foi satisfatório. Sendo assim, a redução do consumo, que foi estimado em 308,14 µA, não justifica o uso desse intervalo maior entre as transmissões.

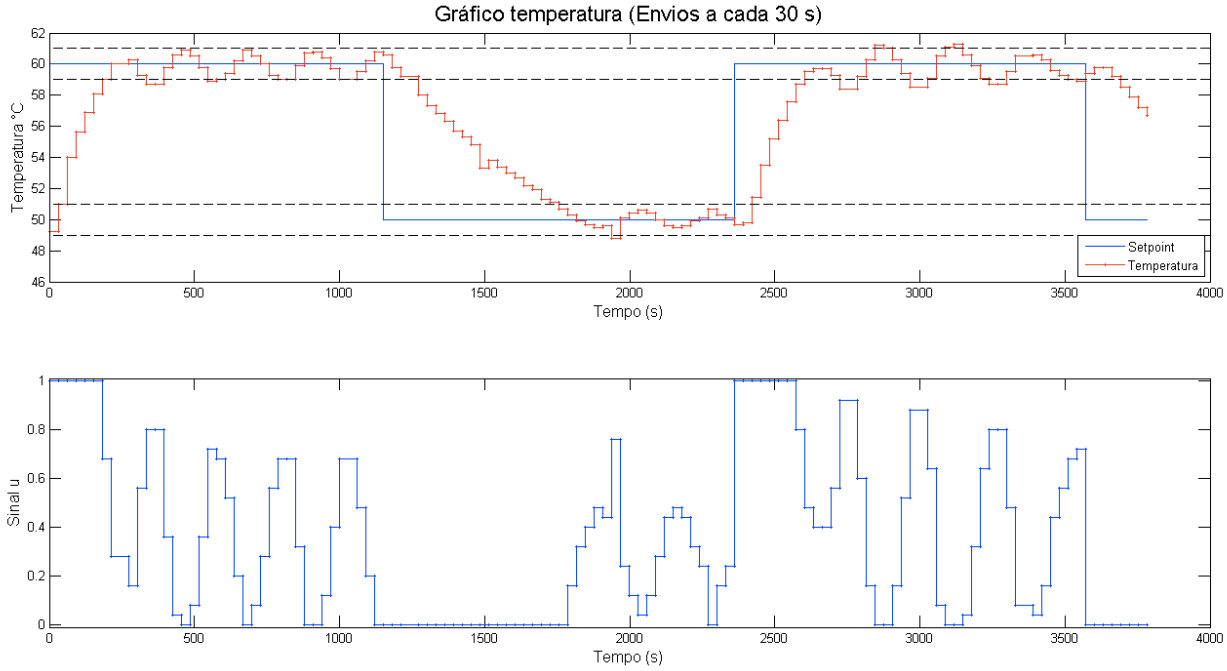


Figura 6.6: Variação da temperatura com controle P + PWM e com intervalo entre envios de 30 s.

Comparando todos estes testes feitos, observou-se que quanto maior o intervalo, pior era o controle e menor era o consumo de corrente, resultado exatamente igual ao esperado. Percebeu-se que o controle atuando com PWM e intervalo de 30 segundos apresentou comportamento com características de um controle liga-desliga.

Dentre os testes feitos, o com intervalo de 15 segundos foi o que obteve o melhor equilíbrio entre eficiência de controle e economia de bateria. Por esse motivo, esse caso será usado para comparação com o sistema orientado a eventos.

6.2 CONTROLE ORIENTADO A EVENTOS

O conceito de evento já foi explicado e os diferentes eventos a serem considerados foram mostrados na Tabela 5.3. Cada caso será analisado separadamente, através dos mesmos gráficos e informações geradas nos casos sem orientação a eventos.

Acrescenta-se aqui que, nos testes orientados a eventos, o período entre as medições e entre os envios será diferente para cada faixa de temperatura. Definiu-se as nove regiões seguintes.

Região 1 → temperatura $\geq$ *setpoint* + 2,5 ˚C
Intervalo entre envios = 20 s

Região 2 → *setpoint* + 1,5 ˚C $\leq$ temperatura < *setpoint* + 2,5 ˚C
Intervalo entre envios = 8 s

Região 3 → *setpoint* + 1 ˚C $\leq$ temperatura < *setpoint* + 1,5 ˚C
Intervalo entre envios = 5 s

Região 4 → *setpoint* + 0,5 ˚C < temperatura < *setpoint* + 1 ˚C
Intervalo entre envios = 3 s

Região 5 → *setpoint* - 0,5 ˚C $\leq$ temperatura $\leq$ *setpoint* + 0,5 ˚C
Intervalo entre envios = 10 s

Região 6 → *setpoint* – 1 ˚C $\leq$ temperatura < *setpoint* – 0,5 ˚C
Intervalo entre envios = 3 s

Região 7 → *setpoint* – 1,5 ˚C $\leq$ temperatura < *setpoint* – 1 ˚C
Intervalo entre envios = 4 s

Região 8 → *setpoint* – 2,5 ˚C $\leq$ temperatura < *setpoint* – 1,5 ˚C
Intervalo entre envios = 8 s

Região 9 → temperatura < *setpoint* - 2,5 ˚C
Intervalo entre envios = 20 s

CASO 1

Evento = Erro > |0,1 ˚C|

Com um evento representado por uma faixa de erro menor, a quantidade de transmissões será maior e o controle tende a ficar mais estável, porém o consumo de energia será mais alto. A Figura 6. mostra o resultado encontrado nesse teste.

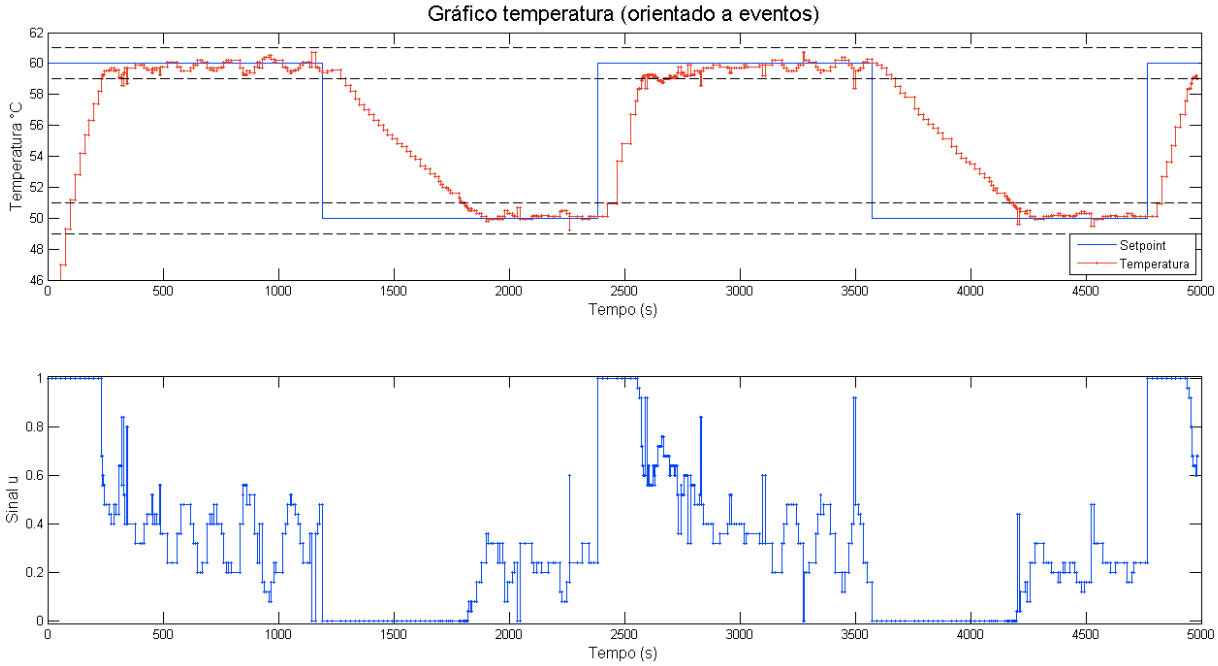


Figura 6.7: Orientado a eventos com evento igual a erro > |0,1 ˚C|.

Percebe-se que mesmo ficando sem transmitir por uma faixa de temperatura, ainda foi possível um bom controle. A variação do sinal *u* no tempo mostra que os erros não foram tão altos e a atuação se manteve numa região intermediária de potência.

Porém, como esperado, o consumo médio de corrente não foi tão baixo, ficando em 747,98 µA.

CASO 2

Evento = Erro > |0,5 ˚C|

Comparando o resultado desse teste ao Caso 1, observou-se uma histerese maior da temperatura, em função do maior período sem que o coordenador receba informações. Por ter um erro maior, consequentemente o sinal *u* atinge valores mais

altos, indicando a necessidade de uma atuação mais forte. Por ter menos transmissões que o caso 1, o consumo médio de corrente foi menor, igual a 623,78 µA.

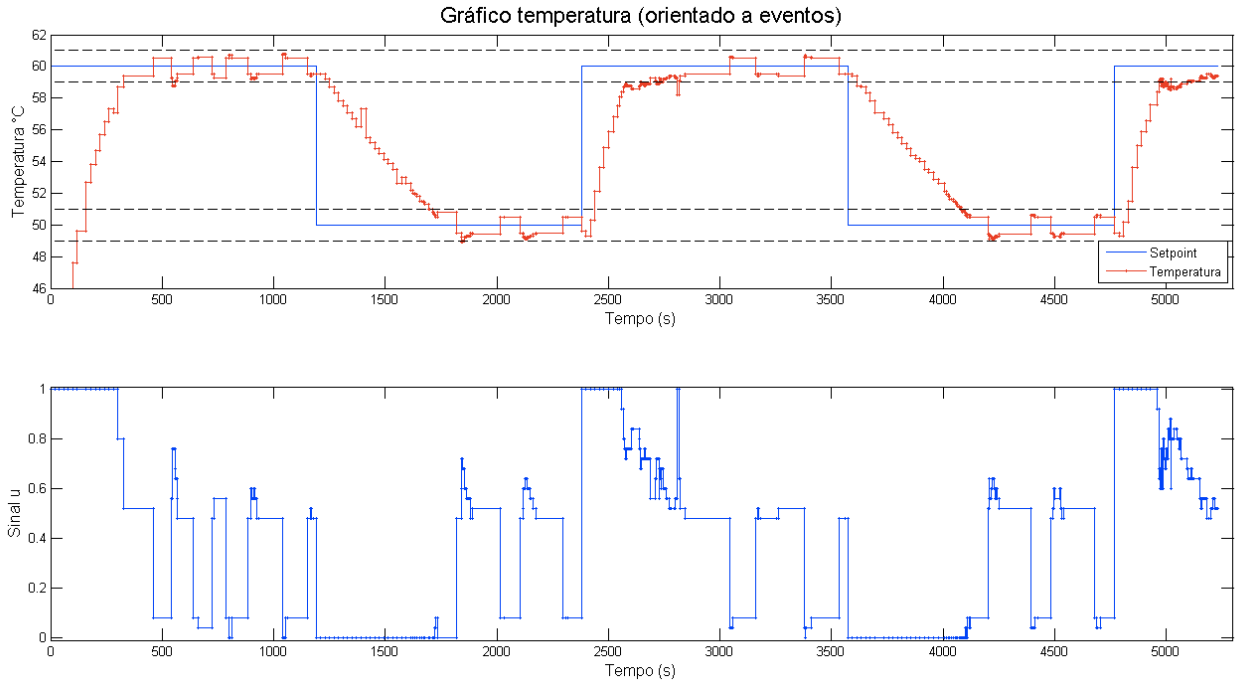


Figura 6.8: Orientado a eventos com evento igual a erro > |0,5 ˚C|.

CASO 3

Evento = Erro > |1 ˚C|

Como o nó sensor não envia a temperatura até que essa esteja maior que 1 ˚C ou menor que -1 ˚C, imagina-se que até que seja percebido o evento e enviada a ordem de atuação, a temperatura varie um pouco além do faixa de erro aceitável, que é exatamente a mesma faixa sem eventos.

Analisando a Figura 6.9, em especial a variação de *u,* vê-se que nesse caso o erro foi bem maior, obrigando a atuação com potência mais alta. Outro fato interessante que pode ser visto no gráfico do sinal *u* é que sempre que é gerado um evento com erro maior que 1 ˚C a potência vai para 0, pois o algoritmo de controle define que para valores medidos acima de *setpoint* + 0,7 se desligue o atuador.

A vantagem desse caso 3 está no consumo de corrente, que foi o menor, 460 µA em média.

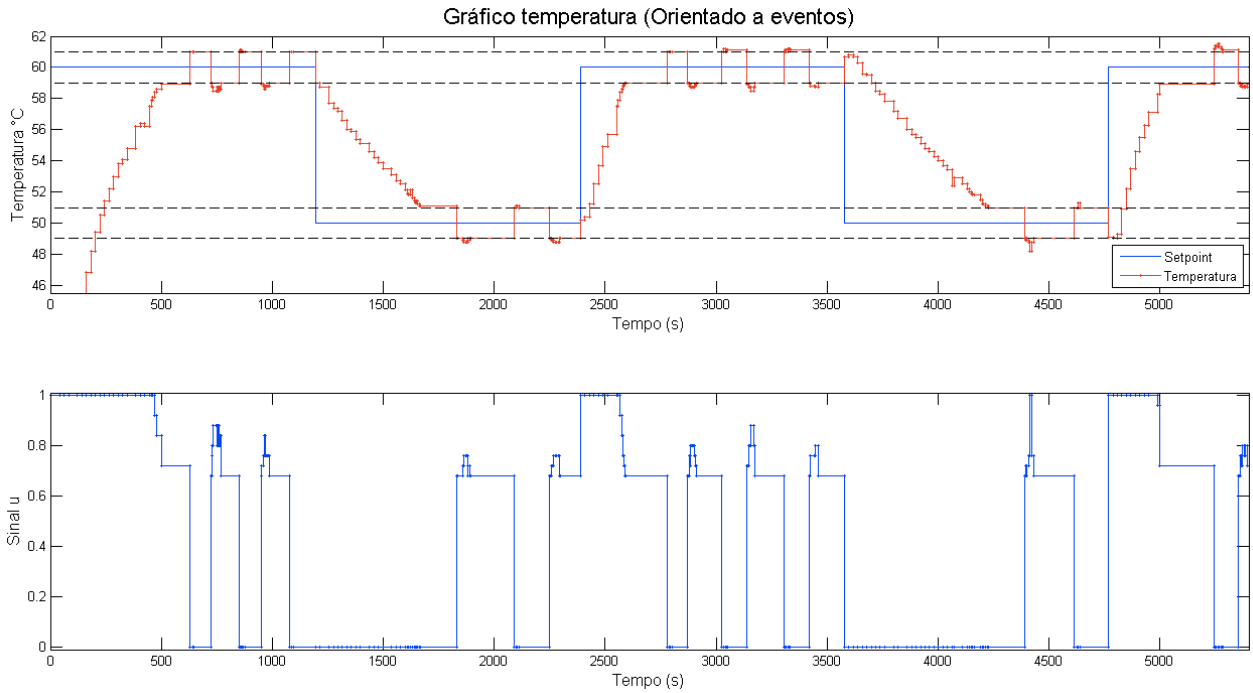


Figura 6.9: Orientado a eventos com evento igual a erro > |1 ˚C|.

A comparação entre os três casos conclui que reduzir a faixa de evento melhora o controle, porém aumenta o consumo de bateria. Olhando o resultado encontrado, escolheu-se o caso 2 como o mais vantajoso, pois foi o que melhor equilibrou eficiência x consumo.

6.2.1 INTERVALO DE 15 SEGUNDOS COM EVENTO = ERRO > |0,5 ˚C|

Baseando-se no fato de que, nos controles não orientados a eventos, o intervalo de 15 segundos apresentou melhor equilíbrio e que a faixa de evento mais vantajosa foi a do caso 2, decidiu-se unir os dois parâmetros. Programou-se um controle que realiza as medições com períodos fixos de 15 segundos, mas só envia se for um evento, ou seja, se o erro for maior que |0,5 ˚C|.

A Figura 6.10 mostra que o resultado foi interessante. A qualidade do controle foi um pouco inferior ao teste com intervalo de 15 segundos sem orientação a eventos, mostrado na Figura 6.4, mas ainda foi satisfatória. No entanto, o que mais chama atenção é a queda no consumo de corrente, que passou de 508 µA para 366 µA

aproximadamente. Essa redução é significativa e mostra que o uso do método com orientação a eventos permite sim economizar energia.

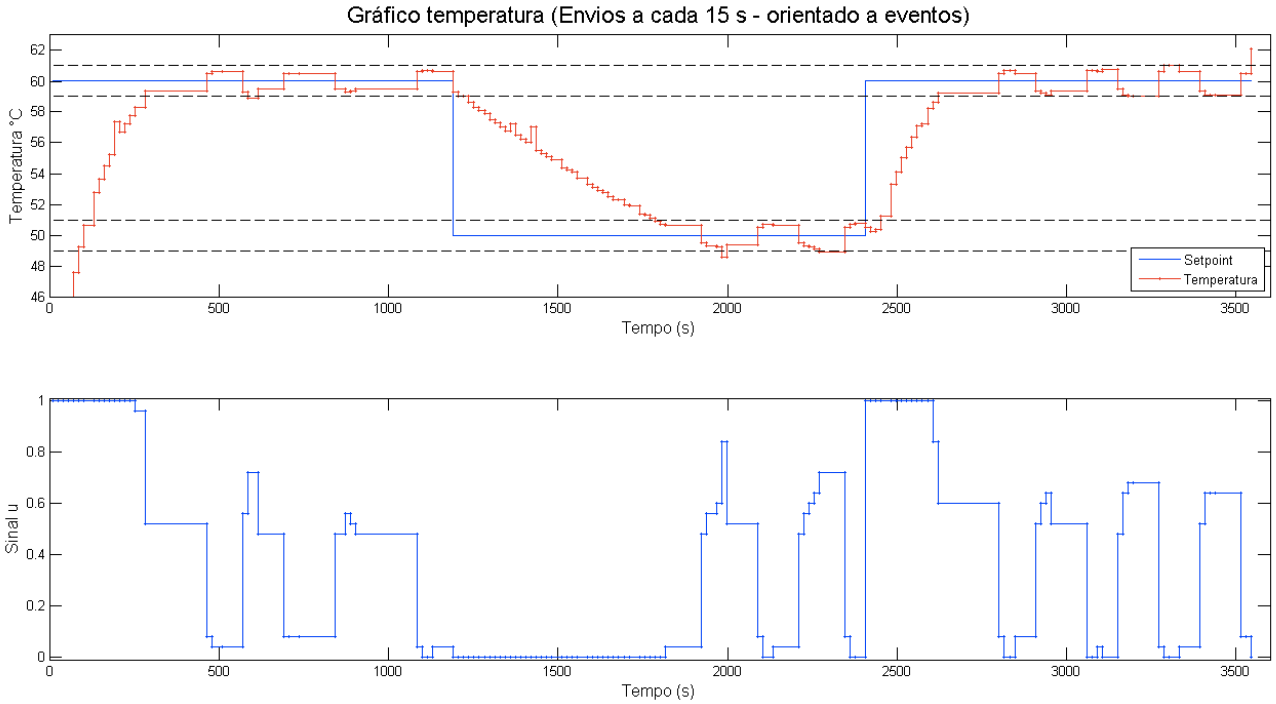


Figura 6.10: Intervalo de medição de 15 s e evento igual a erro > |0,5 ˚C|.

6.2.2 GERANDO PERTURBAÇÕES

Nessa etapa, serão criados eventos que influenciam a temperatura do ambiente em estudo, por exemplo, abertura e fechamento de portas e janelas. A intenção é avaliar a capacidade do sistema em manter o controle de maneira satisfatória, corrigindo erros que esses eventos podem causar em um ambiente real. Utilizou-se aqui um sistema orientado a eventos para identificar a velocidade de percepção do erro e posterior atuação de correção.

Perturbação 1

A primeira perturbação aconteceu no segundo 1020 e foi a abertura do vidro que cobre a maquete durante 15 segundos.

Perturbação 2

Aconteceu no segundo 2220 e foi causada pela abertura da janela J2 e das portas P1, P2 e P5 durante 3 minutos e meio.

Perturbação 3

Se passou no segundo 2970 e foi a mudança da posição do sensor, que antes ficava sobre uma caixa de papelão próximo ao teto da sala de estudo e foi colocado no chão da sala por 8 minutos.

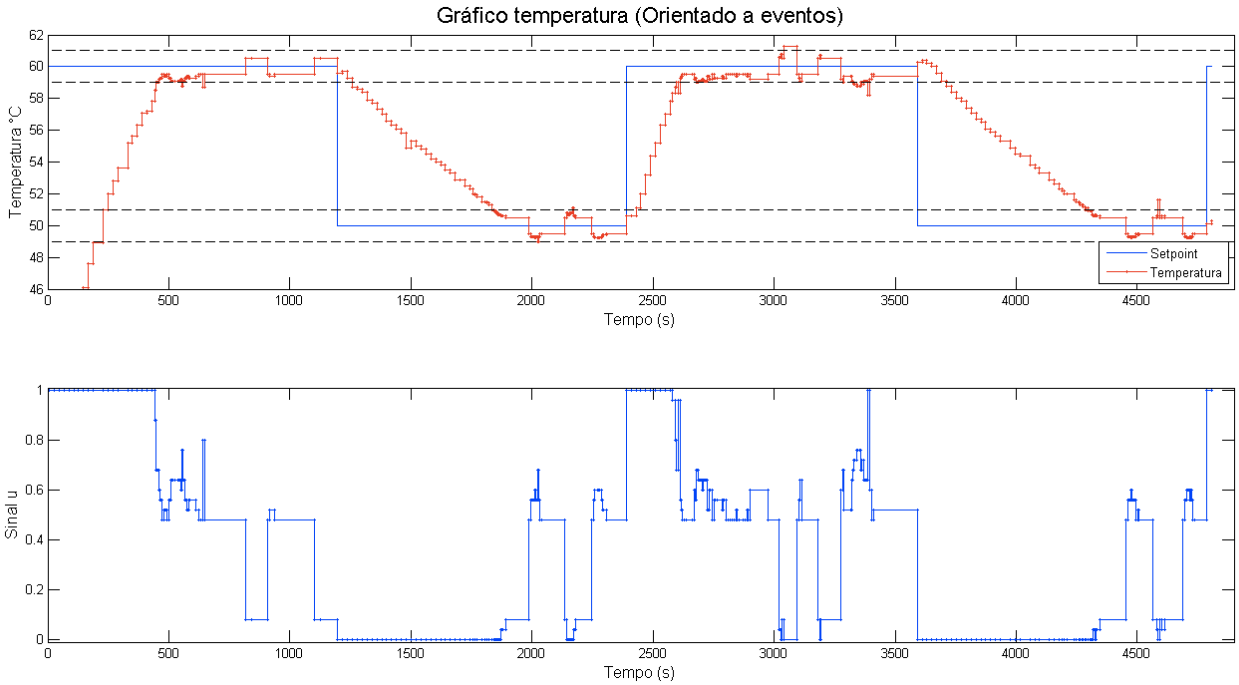


Figura 6. 11: Sistema orientado a eventos com perturbações

A Figura 6.11 mostra a visão geral do comportamento. Percebe-se que o controle, por mais que tenha sofrido algumas variações maiores por conta das perturbações, se manteve próximo do ponto de operação e as frequências de transmissões, como esperado, foram variadas devido a orientação baseada em eventos.

6.3 COMPARAÇÃO ENTRE OS MÉTODOS

Usando os dados de consumo de corrente, quantidade de envios, tempo de duração e tensão da bateria, obteve-se os parâmetros mostrados na Tabela 6.1. Analisando, de maneira subjetiva, a capacidade dos sistemas testados em manter o

controle dentro da faixa de erro aceitável, fez-se uma escala da qualidade de controle para ajudar na comparação das vantagens de cada um.

Tabela 6.1: Parâmetros de comparação dos sistemas testados.

Sistema	Corrente média (mA)	Potência média (mW)	Envios/h	Qualidade do Controle
Não orientado/30 s	0,3081	0,9676	114,22	Ruim
Orientado/15 s	0,3663	1,1502	132,05	Bom
Orientado/erro > |1 ˚C|	0,462	1,4507	161,53	Regular
Não orientado/15 s	0,5086	1,5969	220,79	Bom
Liga-Desliga/15 s	0,527	1,6549	218,15	Ruim
Orientado/erro > |0,5 ˚C|	0,6238	1,9587	240,89	Bom
Orientado/Perturbação	0,6259	1,9654	234,43	Bom
Orientado/erro > |0,1 ˚C|	0,748	2,3487	303,26	Bom
Não orientado/5 s	1,6	5,024	641,88	Muito Bom
Não orientado/2 s	3,1	9,734	1505,38	Excelente
Liga-Desliga/2 s	3,7	11,618	1501,63	Regular

A Tabela 6.1 apresenta os sistemas em ordem crescente de consumo médio de corrente. O sistema com intervalos fixos de 30 segundos foi o mais interessante em relação à economia de bateria, porém obteve o pior controle. Em contra partida, o sistema com controle P + PWM e com intervalos de 2 segundos conseguiu um controle excelente, mas em troca de um altíssimo consumo de energia.

Pensando, então, no equilíbrio entre eficiência no controle e economia de energia, o caso que mais se destaca é o orientado a eventos com intervalo de transmissão fixo em 15 segundos. Porém, espera-se que esse sistema não responda tão bem às perturbações quanto os orientados a eventos com intervalo variável entre as transmissões.

Os dados da tabela comprovam que os sistemas com controle orientado a eventos se posicionaram com comportamento satisfatório e gasto intermediário, assim como esperado. Afirmando-se como a uma boa opção para aliar eficiência no controle e economia de energia.

7 CONCLUSÃO

O uso de redes sem fio em sistemas de automação predial orientados a eventos traz inúmeras vantagens, mas esbarra no rápido consumo das baterias. Tendo isso em vista, o objetivo geral desse trabalho foi estudar a eficiência energética de redes sem fio em sistemas de automação predial orientados a eventos. Realizou-se o controle de um processo térmico de tal maneira que se atingisse um equilíbrio entre eficiência x consumo de corrente, a fim de prolongar drasticamente a vida útil das baterias.

Para isso, o nó sensor alimentado por bateria foi programado para identificar o erro entre a temperatura desejada e a medida, definindo, baseado em sua estratégia de controle, se esse valor representa ou não um evento a ser transmitido. Esse método orientado a eventos foi aplicado ao controle proporcional + PWM, e comparado com o funcionamento de sistemas não orientados a eventos.

Com os resultados encontrados, pode-se concluir que os sistemas não-orientados a eventos com controle P + PWM e com intervalos curtos entre as transmissões apresentaram erros menores. Em contrapartida, o sistema orientado a eventos com controle P + PWM proporcionou uma redução no número de transmissões, e portanto, o consumo médio de corrente foi menor.

Outro ponto importante a se analisar é o fato de que as respostas às perturbações nos sistemas orientados a eventos tendem a ser melhores que os não-orientados, pois quando opera fora da faixa de eventos o número de transmissões aumenta, percebendo melhoras as alterações da temperatura.

Levando-se em consideração o erro, o consumo e a resposta a perturbações, o uso do sistema orientado a eventos se mostra vantajoso, apesar de resultados não tão expressivos quanto o esperado.

7.1 TRABALHOS FUTUROS

Apesar do sistema orientado a eventos e com controle proporcional + PWM ter se mostrado mais econômico sem perder completamente a qualidade, acredita-se que a implementação de um observador de estados ao processo pode reduzir ainda mais o número de eventos e, por consequência, a quantidade de transmissões.

Outra opção para que a automação fique mais precisa é a utilização de uma técnica de controle mais complexa e eficiente, permitindo que o comportamento em regime permanente seja mais estável, mantendo por mais tempo a temperatura dentro da faixa sem envios.

Por fim, para trabalhos futuros sugere-se utilizar a orientação a eventos em outros processos, além do controle de temperatura aqui realizado.

8 REFERÊNCIAS

[1] ADAMA, V. R., Wireless Sensor Network Architecture for Smart Buildings, Jyothishmathi Institute of Technology & Science, Karimnagar, Andhra Pradesh, India, 2006.

[2] ARAÚJO, H. A. S., MELO, M. C. C., Estudo do Controle Adaptativo na Eficiência Energética de Climatização Utilizando Ambiente Predial, 2013, Trabalho de Graduação em Engenharia de Controle e Automação, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF.

[3] Arduinoecia, Sensor de temperatura e umidade DHT22 (AM2302), 2015,. Acessado em 10/2015. Disponível em: <http://www.arduinoecia.com.br/2015/02/sensor-de-temperatura-e-umidade-dht22.html>.

[4] CERVIN, A., HENRIKSSON, D., OHLIN, M., TrueTime 2.0 beta – Reference Manual, 2010, Department of Automatic Control, Lund University.

[5] DINIZ, M., Automação residencial: "a casa do futuro", 2014. Acessado em 11/2015. Disponível em: <https://marisadiniz.files.wordpress.com/2014/02/automac3a7c3a3o_residencial.gif>

[6] ERGEN, S. C, ZigBee/IEEE 802.15.4 Summary, 2004.

[7] GISLASON, Drew, ZigBee Wireless Networking, 2008, Newnes.

[8] LEONOV, A., Parking lot gets smart with ZigBee. Acessado em 11/2015. Disponível em: <http://www.automatedbuildings.com/news/sep07/articles/meshnetics070821030505leonov.htm>.

[9] MATOS, L. G., FILHO, N. C. M., Economia de energia e gravação remota de dispositivos ZigBee visando a automação predial, Trabalho de Graduação em Engenharia Mecatrônica, Universidade de Brasília, Brasília, DF, Brasil, 2013.

[10] MECAWEB, PWM – Modulação por largura de pulso. Acessado em 11/2015. Disponível em: <http://mecaweb.com.br/eletronica/content/e_pwm>.

[11] NATIONAL INSTRUMENTS, NI 4065 Specifications, <http://sine.ni.com/nips/cds/view/p/lang/pt/nid/204061>.

[12] NGUYEN, T. A., AIELLO, M., Energy intelligent buildings based on user activity: A survey, 2012, Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, Nijenborgh, AG Groningen, The Netherlands, p. 245-257.

[13] OPTO 22, Solid-State Relays: Data Sheet

[14] SCHNEIDER, K. K., SAUSEN, P. S., SAUSEN, A., Análise Comparativa do Tempo de Vida de Baterias em Dispositivos Móveis a partir da Ulitlização de Modelos Analíticos, 2011, Mestrado em Modelagem Matemática, Universidade Regional do Noroeste do Estado do Rio Grande do Sul, Ijuí, RS, Brasil.

[15] SHAMMAA, S., 802.15.4, Telecommunications Eng. Dept., Dr Pramode Verma

[16] VASQUES, B. L. R. P., COUTINHO, I. B. A., LIMA, M. F., CARNEVAL, V. P. O., ZigBee, Universidade Federal do Rio de Janeiro, 2010.
[17] WIKIPEDIA, Redes sem fio. Acessado em 10/2015. Disponível em: <https://pt.wikipedia.org/wiki/Rede_sem_fio>.
[18] WIKIPEDIA, Relés de estado sólido. Acessado em 12/2015. Disponível em: <https://pt.wikipedia.org/wiki/Relé_de_estado_sólido>.
[19] https://c.gongkong.com/NI. Acessado em 12/2015.
[20] SHIRRIFF, K., BADGER, P., Secrets of Arduino PWM. Acesso em 11/2015. Disponível em: <https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM>
[21] Doxygen, BitCloud SDK for ATZB-DK-24/ATZB-DK-A24/ATZB-DK-900. Acessado em 09/2015. Disponível em: <http://www2.ee.ic.ac.uk/t.clarke/projects/Resources/BitCloud/BitCloud_ZDK_1_4_1/Documentation/HTML%20help/main.html>.
[22] Atmel, Datasheet ATmega 640/V-1280/V-1281/V-2560/V-2561/V. Acessado em: 11/2015. Disponível em: <http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf>.

ANEXO I

Código do módulo Coordenador

```
/*  Programa que funciona como coordenador numa rede de controle de temperatura
orientada a eventos.
Autor: Gabriel Lara de Souza email: gbr1990@gmail.com
Utilizar no configuration.h:
#define CS_UID 0x0LL
#define CS_NWK_ADDR 0x0000
#define CS_MAX_CHILDREN_AMOUNT 4
#define CS_MAX_CHILDREN_ROUTER_AMOUNT 2
*Para Mais informações a respeito das funções e do funcionamento da rede ZigBee,
recomenda-se a leitura da API da BitCloud.  */
/*****************************************************************************
**********************************************
 Nota, incluir também as bibliotecas exclusivas da aplicação e os .Cs que sejam
customizados
   É importante ao compilar o programa verificar se existe o caminho
referenciado para o include das bibliotecas.
    *****************************************************************************
**********************************************/
 #include <types.h>
 #include <taskManager.h>
 #include <configServer.h>
 #include <zdo.h>
 #include <peer2peer.h>
 #include <serialInterface.h>
 #include <appTimer.h>
 #include <adc.h>
 #include <pwm.h>
/*****************************************************************************
 **********************************************
   Definição de variáveis globais
   *****************************************************************************
   **********************************************/
static AppState_t appState = APP_INITIAL_STATE;         //Estado da aplicação.
Usado como referência no escalonador de tarefas da aplicação.
static ZDO_StartNetworkReq_t networkParams;          //Parâmetros da rede que
será gerada/acessada
static SimpleDescriptor_t simpleDescriptor = { APP_ENDPOINT, APP_PROFILE_ID, 1,
1, 0, 0 , NULL, 0, NULL };   //Simple Descriptor. Parâmetros do Nó
static APS_RegisterEndpointReq_t endpointParams;        //Parâmetros para
registro do nó na rede.
static HAL_UsartDescriptor_t appUsartDescriptor;        //Parâmetros para a
configuração da porta Serial
static uint8_t usartRxBuffer[100];            //Buffer de recepção da porta
serial
static HAL_AppTimer_t delayTimer;           //Parâmetros para configuração do
Timer

//Funções do PWM
static HAL_PwmDescriptor_t *descriptor ;
static HAL_PwmUnit_t pwmUnit;
static HAL_PwmPrescaler_t prescaler;
static uint16_t top;
static uint16_t cmpValue;
static uint8_t tempState;

static uint16_t nwkAddr;              // Receberá o Short_Adress do nó
static AppMessageBuffer_t appMessageBuffer;          // Buffer utilizado para o
envio de Dados Wireless.
```

```
/************************************************************************
 ************************************************
    Function Prototype Section
************************************************************************
************************************************/
static void initNetwork(void);             // Função que inicializa os
parâmetros de rede e define o papel do nó na rede que se fromará/acessará
static void startNetwork(void);             // Inicializa/Acessa a rede.
static void APS_DataIndication(APS_DataInd_t* dataInd);  //Indicação de dao
recebido.
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t* confirmInfo);
//Confirmação de formação de rede. Callback
static void initSerialInterface(void);           //Define os parâmetros de
comunicação serial

static void usartBytesReceived(uint16_t readBytesLen);       //Recebimento de
dados via serial
static void APS_DataIndication(APS_DataInd_t* indData);       //Função que trata
o recebimento de dados.
void ZOD_WakeUpInd(void);              //Função de Stub Para o Coordenador.
/************************************************************************
 ************************************************
    Stub Functions
************************************************************************
************************************************/
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams)
{
  (void)nwkParams;
  }

void ZDO_BindIndication(ZDO_BindInd_t *bindInd)
 {
   (void)bindInd;
    }

void ZDO_UnbindIndication(ZDO_UnbindInd_t *unbindInd)
 {
   (void)unbindInd;
   }


static void usartWriteConf(void)
 {

 }
void ZDO_WakeUpInd(void)
 {

 }
/************************************************************************
 ************************************************
   Rotinas para inicialização de rede.

************************************************************************
 ************************************************/
//Inicializa a rotina de preparação da rede
static void initNetwork(void)
 {
    CS_WriteParameter(CS_NWK_UNIQUE_ADDR_ID, &(bool){true});
      DeviceType_t deviceType;   //Papel do nó na rede.
         // Short Address
          CS_ReadParameter(CS_NWK_ADDR_ID, &nwkAddr);
  // Função do nó. Se for 0, o nó é coordenador, senão, é EndDevice.
   if (0 == nwkAddr)  {
    #ifdef _SECURITY_
         {
               ExtAddr_t extAddr;
```

```
              CS_ReadParameter(CS_APS_TRUST_CENTER_ADDRESS_ID, &extAddr);
              CS_WriteParameter(CS_UID_ID, &extAddr);
               }
       #endif //_SECURITY_
    deviceType = DEVICE_TYPE_COORDINATOR;
       }
       else
        {
            deviceType = DEVICE_TYPE_END_DEVICE;
                }
                  // Define o tipo do nó
        CS_WriteParameter(CS_DEVICE_TYPE_ID, &deviceType);
  // Muda o estado para a referência do escalonador de tarefas,
  appState = APP_NETWORK_JOINING_STATE;
  }


//Quando um pedido de inicialização de rede é feito e a função é executada,
então essa função é chamada para exercer confirmação
void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t* confirmInfo)
{
if (confirmInfo->status == ZDO_SUCCESS_STATUS)
       {
        BSP_OnLed(LED_RED);
        appState = APP_NETWORK_JOINED_STATE;     // Configura o nó a ser
registrado
        endpointParams.simpleDescriptor = &simpleDescriptor;
        endpointParams.APS_DataInd = APS_DataIndication;     // Register
endpoint
        APS_RegisterEndpointReq(&endpointParams);
        SYS_PostTask(APL_TASK_ID);
        }
        else
        {
        //Volta ao escalonador de tarefas para continuar o processo.
        SYS_PostTask(APL_TASK_ID);
        }
        }
//Função que inicia a rede.
static void startNetwork(void)
{
//Determina a função de confirmação de envio.
networkParams.ZDO_StartNetworkConf = ZDO_StartNetworkConf;
//Começa a Rede
ZDO_StartNetworkReq(&networkParams);
}

/************************************************************************
*************************************************
   Configuração de Comunicação via USART/UART
   ************************************************************************
   *************************************************/
//Parâmetos para início de interface serial
static void initSerialInterface(void)
{
appUsartDescriptor.tty             = USART_CHANNEL;      //DEfine a maneira de
transmissao
appUsartDescriptor.mode            = USART_MODE_ASYNC;     //Modo de
transmissão. Sugestão para o modo assíncrono.
appUsartDescriptor.baudrate        = USART_BAUDRATE_38400;    //BaudRate.
appUsartDescriptor.dataLength      = USART_DATA8;      //Tamanho do Elemento de
transmissão.
appUsartDescriptor.parity          = USART_PARITY_NONE;     //Paridade
appUsartDescriptor.stopbits        = USART_STOPBIT_1;      //Bits de Parada
appUsartDescriptor.rxBuffer        = usartRxBuffer;      //Buffer de REcepção
appUsartDescriptor.rxBufferLength  = sizeof(usartRxBuffer);    //Tamanho do
Buffer de Recepção
```

```
appUsartDescriptor.txBuffer        = NULL;         // Callback mode. Quando a
transmissão acaba... chama uma função de confirmação
appUsartDescriptor.txBufferLength  = 0;
appUsartDescriptor.rxCallback      = usartBytesReceived;     //Callback de
recepção de mensagens
appUsartDescriptor.txCallback      = usartWriteConf;      //Confirmação de envio
de dados
appUsartDescriptor.flowControl     = USART_FLOW_CONTROL_HARDWARE;   //Controle
de fluxo.

  HAL_OpenUsart(&appUsartDescriptor);
  }
//Essa função é só um eco para o teste. no funcionamento real da rede, é só uma
função de //Stub

static void usartBytesReceived(uint16_t readBytesLen)
{
       READ_USART(&appUsartDescriptor, appMessageBuffer.message.data,
APP_MAX_PACKET_SIZE);
       WRITE_USART(&appUsartDescriptor,(uint8_t*)&appMessageBuffer.message.data,
sizeof(appMessageBuffer.message.data));
       (void)readBytesLen;

       }


/************************************************************************
*********************************************
   Configurar parâmetros de troca de mensagens por rede
   ************************************************************************
   *********************************************/
// Todo dado recebido é escrito na porta serial
static void APS_DataIndication(APS_DataInd_t* indData)
       {

              AppMessage_t *appMessage = (AppMessage_t *) indData->asdu;
              WRITE_USART(&appUsartDescriptor,appMessage->data,indData-
>asduLength-1);

              uint16_t recValue = (appMessage->data[6]-'0')*100;
              recValue += (appMessage->data[7]-'0')*10;
              recValue += appMessage->data[8]-'0';

              cmpValue = recValue;

              //Funções do PWM

              GPIO_0_make_out();
              pwmUnit = PWM_UNIT_1;
              HAL_OpenPwm(pwmUnit);

              descriptor->channel = PWM_CHANNEL_0;
              descriptor->polarity = PWM_POLARITY_INVERTED;
              descriptor->unit = PWM_UNIT_1;
              HAL_StartPwm(descriptor);

              top = 75;
              prescaler = PWM_PRESCALER_256;
              HAL_SetPwmFrequency(pwmUnit,top,prescaler);

              HAL_SetPwmCompareValue(descriptor, cmpValue);



       }
/************************************************************************
*********************************************
```

```
   Task Handler Function
*****************************************************************************
    ***********************************************/


void APL_TaskHandler(void)
       {
                      switch (appState)
                       {
                                              case APP_INITIAL_STATE:
// nó inicial do nó
                                              initSerialInterface();
                                              initNetwork();
                                              SYS_PostTask(APL_TASK_ID);    //
Executa o próximo passo.
                                              break;

                                              case APP_NETWORK_JOINING_STATE:
                                              startNetwork();
                                              break;

                                              case APP_NETWORK_LEAVING_STATE:
                                              break;

                                              case APP_NETWORK_JOINED_STATE:
                                              //StartTimer();
                                              break;

                                              default:
                                              break;
                               }
       }
/*****************************************************************************
***********************************************
   Main Function
   *****************************************************************************
   ***********************************************/
//Inicializa o Micro-Controlador e chama a tarefa.
int main(void)
       {
              SYS_SysInit();
                      for(;;)
                         {
                         SYS_RunTask();
                          }
       }
```

ANEXO II

Código do módulo dispositivo final

```
/*  Programa que funciona como end device numa rede de controle de temperatura
orientada a eventos, incluindo o sensor DHT22
Autor: Gabriel Lara de Souza email: gbr1990@gmail.com
Utilizar no configuration.h valores DIFERENTES de:
#define CS_UID 0x0LL
#define CS_NWK_ADDR 0x0000
 */
/*****************************************************************************
***********************************************
 Nota, incluir também as bibliotecas exclusivas da aplicação e os .Cs que sejam
customizados
   É importante ao compilar o programa verificar se existe o caminho
referenciado para o include das bibliotecas.
    ******************************************************************************
***********************************************/
#include <types.h>
#include <taskManager.h>
 #include <configServer.h>
 #include <zdo.h>
 #include <peer2peer.h>
 #include <serialInterface.h>
 #include <appTimer.h>
 #include <adc.h>
 #include "dht22.h"
/*****************************************************************************
 ***********************************************
   Definição de variáveis globais
   ******************************************************************************
   ***********************************************/
static AppState_t appState = APP_INITIAL_STATE;         //Estado da aplicação.
Usado como referência no escalonador de tarefas da aplicação.
static ZDO_StartNetworkReq_t networkParams;          //Parâmetros da rede que
será gerada/acessada
static SimpleDescriptor_t simpleDescriptor = { APP_ENDPOINT, APP_PROFILE_ID, 1,
1, 0, 0 , NULL, 0, NULL };   //Simple Descriptor. Parâmetros do Nó
static APS_RegisterEndpointReq_t endpointParams;        //Parâmetros para
registro do nó na rede.
static HAL_AppTimer_t delayTimer;           //Parâmetros para configuração do
Timer
static APS_DataReq_t apsDataReq;           // Parâmetros de Configuraçõ de Envio
de dado na rede
static int i = 0;               // Variável inteira para Utilização geral
static ZDO_SleepReq_t zdoSleepReq;           //Parâmetros para pedido de Sleep.
static HAL_UsartDescriptor_t appUsartDescriptor;        //Parâmetros para a
configuração da porta Serial
static uint8_t usartRxBuffer[100];            //Buffer de recepção da porta
serial
static int contdata;              //Variável para reforço no enviao de mensagem.
static ZDO_ZdpReq_t leaveReq;             //Parâmetros para deixar a rede.
static uint8_t msgBuffer[10];
static uint8_t tempState;
int16_t setpoint = 600;
int32_t sleep_timer = 3000L;                     //variável que armazena o tempo
de sleep

static uint16_t nwkAddr;              // Receberá o Short_Adress do nó
static AppMessageBuffer_t appMessageBuffer;          // Buffer utilizado para o
envio de Dados Wireless.

/*****************************************************************************
 ***********************************************
```

```
    Function Prototype Section
******************************************************************************
***********************************************/
static void initNetwork(void);             // Função que inicializa os
parâmetros de rede e define o papel do nó na rede que se fromará/acessará
static void startNetwork(void);             // Inicializa/Acessa a rede.
static void APS_DataIndication(APS_DataInd_t* dataInd);  //Indicação de dao
recebido.
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t* confirmInfo);
//Confirmação de formação de rede. Callback
static void zdpLeaveResp(ZDO_ZdpResp_t *zdpResp);      //Resposta à tentativa de
deixar a rede.
static void initSerialInterface(void);           //Define os parâmetros de
comunicação serial
static void usartBytesReceived(uint16_t readBytesLen);       //Recebimento de
dados via serial
static void APS_DataIndication(APS_DataInd_t* indData);       //Função que trata
o recebimento de dados.
static void networkSendData(void);          //Função que prepara as
configurações e envia um dado na rede.
static void APS_DataConf(APS_DataConf_t* confInfo);      //Função que verifica
se o dado foi enviado corretamente. //
static void StartTimer(void);
static void SwitchSet(void);
void ZOD_WakeUpInd(void);              //Função de Stub Para o Coordenador.
static void ZDO_SleepConf(ZDO_SleepConf_t *conf);
static void StartSleep(void);
static void EnvioMsg(void);

/******************************************************************************
 ***********************************************
    Stub Functions
******************************************************************************
***********************************************/
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams)
{
  (void)nwkParams;
  }

void ZDO_BindIndication(ZDO_BindInd_t *bindInd)
 {
   (void)bindInd;
    }

void ZDO_UnbindIndication(ZDO_UnbindInd_t *unbindInd)
 {
   (void)unbindInd;
   }


static void usartWriteConf(void)
 {

 }

/******************************************************************************
 ***********************************************
    Rotinas para inicialização de rede.

******************************************************************************
 ***********************************************/
//Inicializa a rotina de preparação da rede
static void initNetwork(void)
 {
    CS_WriteParameter(CS_NWK_UNIQUE_ADDR_ID, &(bool){true});
      DeviceType_t deviceType;   //Papel do nó na rede.
         // Short Address
```

```
         CS_ReadParameter(CS_NWK_ADDR_ID, &nwkAddr);
  // Função do nó. Se for 0, o nó é coordenador, senão, é EndDevice.
   if (0 == nwkAddr)  {
    #ifdef _SECURITY_
         {
               ExtAddr_t extAddr;
              CS_ReadParameter(CS_APS_TRUST_CENTER_ADDRESS_ID, &extAddr);
              CS_WriteParameter(CS_UID_ID, &extAddr);
               }
       #endif //_SECURITY_
    deviceType = DEVICE_TYPE_COORDINATOR;
       }
       else
        {
            deviceType = DEVICE_TYPE_END_DEVICE;
               #define CS_RX_ON_WHEN_IDLE false
                }
                  // Define o tipo do nó
        CS_WriteParameter(CS_DEVICE_TYPE_ID, &deviceType);
  // Muda o estado para a referência do escalonador de tarefas,
  appState = APP_NETWORK_JOINING_STATE;
  }


//Quando um pedido de inicialização de rede é feito e a função é executada,
então essa função é chamada para exercer confirmação
void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t* confirmInfo)
{
if (confirmInfo->status == ZDO_SUCCESS_STATUS)
       {
        BSP_OnLed(LED_RED);
        appState = APP_NETWORK_JOINED_STATE;     // Configura o nó a ser
registrado
        endpointParams.simpleDescriptor = &simpleDescriptor;
        endpointParams.APS_DataInd = APS_DataIndication;     // Register
endpoint
        APS_RegisterEndpointReq(&endpointParams);
        SYS_PostTask(APL_TASK_ID);

        CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID, &sleep_timer);

        }
        else
        {
        //Volta ao escalonador de tarefas para continuar o processo.
        SYS_PostTask(APL_TASK_ID);
        }
        }
//Função que inicia a rede.
static void startNetwork(void)
{
//Determina a função de confirmação de envio.
networkParams.ZDO_StartNetworkConf = ZDO_StartNetworkConf;
//Começa a Rede
ZDO_StartNetworkReq(&networkParams);
}
//Parâmetros para saída de rede.
static void leaveNetwork(void)
{
       ZDO_MgmtLeaveReq_t *zdpLeaveReq = &leaveReq.req.reqPayload.mgmtLeaveReq;
       APS_UnregisterEndpointReq_t unregEndpoint;
       unregEndpoint.endpoint = endpointParams.simpleDescriptor->endpoint;
       APS_UnregisterEndpointReq(&unregEndpoint);
       leaveReq.ZDO_ZdpResp =  zdpLeaveResp;
       leaveReq.reqCluster = 1;
       leaveReq.dstAddrMode = SHORT_ADDR_MODE;
       leaveReq.dstNwkAddr = 0;
```

```
       zdpLeaveReq->deviceAddr = 0;
       zdpLeaveReq->rejoin = 1;
       zdpLeaveReq->removeChildren = 1;
       zdpLeaveReq->reserved = 0;
       ZDO_ZdpReq(&leaveReq);
  //Tenta se reunir à rede.
  appState = APP_NETWORK_JOINING_STATE;
  SYS_PostTask(APL_TASK_ID);
  }
  static void zdpLeaveResp(ZDO_ZdpResp_t *zdpResp)
  {
 //Stub Function
  (void)zdpResp;
   }
static void networkSendData(void)
  {
  //for(i=0; i<sizeof(msgBuffer); i++)
  //appMessageBuffer.data[i] = msgBuffer[i];
  //i=0;
    apsDataReq.dstAddrMode = APS_SHORT_ADDRESS;               // Modo de envio
baseado no Shot Address
       apsDataReq.dstAddress.shortAddress = 0x0000;    //Endereço do lugar para
qual ser enviado.
       apsDataReq.profileId = simpleDescriptor.AppProfileId;     // Profile ID
       apsDataReq.dstEndpoint = simpleDescriptor.endpoint;       // EndPoit de
destino
       apsDataReq.clusterId = APP_CLUSTER_ID;                    // ID do
cluster de destino
       apsDataReq.srcEndpoint = simpleDescriptor.endpoint;       // EndPoint da
fonte
       apsDataReq.asdu = (uint8_t*) &appMessageBuffer.message.data;      //
Ponteiro para o buffer de aplicação
       // actual application message length
       //apsDataReq.asduLength = sizeof(msgBuffer)+1;
       apsDataReq.txOptions.acknowledgedTransmission = 1;        // Transmissão
com reconhecimento ativada.
       #if APP_FRAGMENTATION
              apsDataReq.txOptions.fragmentationPermitted = 1;
              #else
              apsDataReq.txOptions.fragmentationPermitted = 0;
       #endif // APP_FRAGMENTATION
       apsDataReq.radius = 0;                                    // Usa o raio
máximo possível.
       apsDataReq.APS_DataConf = APS_DataConf;                   // Confirmação.
       APS_DataReq(&apsDataReq);         //Chamada para o envio de informação.
 }

//Callback do envio de dados.
static void APS_DataConf(APS_DataConf_t* confInfo)
 {
   if(ZDO_SUCCESS_STATUS != confInfo->status)
     {
           contdata++;
                 if(contdata==5)
                 {
                     BSP_OnLed(LED_YELLOW);
                        HAL_StopAppTimer(&delayTimer);
                        contdata=0;
                        leaveNetwork();
                                    }
              else
                   networkSendData();
                      }
       else
         contdata=0;
         }
```

```
/*************************************************************************
***
                                                                        *
            Função que define o tempo para alternar os setpoints
                                                                        *
*************************************************************************
**/
static void StartTimer(void)
{
             delayTimer.interval = 1190000L;
             delayTimer.mode     = TIMER_REPEAT_MODE;
             delayTimer.callback = SwitchSet;
             HAL_StartAppTimer(&delayTimer);
}

/*************************************************************************
                                *
                    Função que alterna os setpoints
                                                        *
*************************************************************************
*/
static void SwitchSet(void)
{
       int16_t toggleSet = (500 ^ 600);
       setpoint ^= toggleSet;
}
static void StartSleep(void)
 {
    zdoSleepReq.ZDO_SleepConf = ZDO_SleepConf;  // Confrimação de Sleep
       ZDO_SleepReq(&zdoSleepReq);     //Inicialização do modo de Sleep
         }
static void ZDO_SleepConf(ZDO_SleepConf_t *conf)
 {
    if (ZDO_SUCCESS_STATUS != conf->status)     // A pilha não conseguiu dormir
       SYS_PostTask(APL_TASK_ID);      // Chama a tarefa de novo, para tentar
dormir outra vez.
        }
void ZDO_WakeUpInd(void)
 {
        EnvioMsg();
 }

/*************************************************************************
************************************************
   Configuração de Comunicação via USART/UART
   ************************************************************************
   ************************************************/
//Parâmetos para início de interface serial
static void initSerialInterface(void)
{
appUsartDescriptor.tty             = USART_CHANNEL;      //DEfine a maneira de
transmissao
appUsartDescriptor.mode            = USART_MODE_ASYNC;     //Modo de
transmissão. Sugestão para o modo assíncrono.
appUsartDescriptor.baudrate        = USART_BAUDRATE_38400;    //BaudRate.
appUsartDescriptor.dataLength      = USART_DATA8;      //Tamanho do Elemento de
transmissão.
appUsartDescriptor.parity          = USART_PARITY_NONE;     //Paridade
appUsartDescriptor.stopbits        = USART_STOPBIT_1;      //Bits de Parada
appUsartDescriptor.rxBuffer        = usartRxBuffer;      //Buffer de REcepção
appUsartDescriptor.rxBufferLength  = sizeof(usartRxBuffer);    //Tamanho do
Buffer de Recepção
appUsartDescriptor.txBuffer        = NULL;         // Callback mode. Quando a
transmissão acaba... chama uma função de confirmação
appUsartDescriptor.txBufferLength  = 0;
appUsartDescriptor.rxCallback      = usartBytesReceived;     //Callback de
recepção de mensagens
```

```
appUsartDescriptor.txCallback      = usartWriteConf;      //Confirmação de envio
de dados
appUsartDescriptor.flowControl     = USART_FLOW_CONTROL_HARDWARE;   //Controle
de fluxo.

  HAL_OpenUsart(&appUsartDescriptor);
  }
//Essa função é só um eco para o teste. no funcionamento real da rede, é só uma
função de //Stub

static void usartBytesReceived(uint16_t readBytesLen)
{
       READ_USART(&appUsartDescriptor, appMessageBuffer.message.data,
APP_MAX_PACKET_SIZE);
       WRITE_USART(&appUsartDescriptor,(uint8_t*)&appMessageBuffer.message.data,
sizeof(appMessageBuffer.message.data));
       (void)readBytesLen;
}
/************************************************************************
*************************************************
   Configurar parâmetros de troca de mensagens por rede
   ************************************************************************
   *************************************************/
// Todo dado recebido é escrito na porta serial
static void APS_DataIndication(APS_DataInd_t* indData)
       {
              AppMessage_t *appMessage = (AppMessage_t *) indData->asdu;
              //WRITE_USART(&appUsartDescriptor,indData->asdu,indData-
>asduLength);
       }
/************************************************************************
*************************************************
   Task Handler Function  ************************************************
************************************************************************/
static void EnvioMsg(void)
{

       uint16_t temp[2];

       uint8_t ans = dht22_read(temp);

       int16_t result = setpoint - temp[0];

       int16_t compare;

       //Trecho abaixo define a lei de controle

       if (result < -7)
       {
              compare = 0;
       }


       else {
              if (result > 18)
              {
              compare = 75;
              }
              else
              {
              compare = (result + 7) * 3;
              }
              }
       //Funções para variar o tempo entre medições

              if (result > 25)
              {
```

```
                    sleep_timer = 20000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }

             if (result > 15 && result <= 25)
             {
                    sleep_timer = 8000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }
             if (result > 10 && result <= 15)
             {
                    sleep_timer = 4000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }

             if (result > 5 && result <= 10)
             {
                    sleep_timer = 3000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }

             if (result <= 5 && result > 3)
             {
                    sleep_timer = 10000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }

             if (result <= 3 && result >= 0 )
             {
                    sleep_timer = 10000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }
             if (result < 0 && result >= -5)
             {
                    sleep_timer = 10000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }
             if (result < -5 && result > -10)
             {
                    sleep_timer = 3000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }

             if (result <= -10 && result > -15)
             {
                    sleep_timer = 5000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }

             if (result <= -15 && result > -25)
             {
                    sleep_timer = 8000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }

             if (result <= -25)
             {
```

```
                    sleep_timer = 20000L;
                    CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID,
&sleep_timer);
             }


             if(ans != 5) //Condição usada para que o sensor não envie em caso
de erro de checksum na leitura
             {
                    if((result >= 5 ) || (result <= -5))     //Condição que
define um evento
                    {
                    sprintf(msgBuffer,
"%03u%03u%03d\n",temp[0],setpoint,compare);


                    int k = 0;
                    for(;k<strlen(msgBuffer);k++)
                    {
                    appMessageBuffer.message.data[k+1]=msgBuffer[k];
                    }

                    apsDataReq.asduLength =
sizeof(msgBuffer)+sizeof(appMessageBuffer.message.messageId);
                    networkSendData();
                    }
             }

       SYS_PostTask(APL_TASK_ID);

}


void APL_TaskHandler(void)
       {
                    switch (appState)
                     {
                                          case APP_INITIAL_STATE:
// nó inicial do nó
                                          initSerialInterface();
                                          initNetwork();
                                          SYS_PostTask(APL_TASK_ID);    //
Executa o próximo passo.
                                          break;

                                          case APP_NETWORK_JOINING_STATE:
                                          startNetwork();
                                          break;

                                          case APP_NETWORK_LEAVING_STATE:
                                          break;

                                          case APP_NETWORK_JOINED_STATE:
                                          StartSleep();
                                          StartTimer();
                                          break;

                                          default:
                                          break;
                           }
       }
/************************************************************************
*************************************************
   Main Function
   ************************************************************************
   *************************************************/
//Inicializa o Micro-Controlador e chama a tarefa.
```

```
int main(void)
       {
              SYS_SysInit();
                     for(;;)
                        {
                        SYS_RunTask();
                         }
       }
```

ANEXO III

Biblioteca do sensor DHT22

dht22.h:

```
/*
Biblioteca para Bitcloud do sensor DHT22 adaptada de bibliotecas para Arduino
Autor: Gabriel Lara de Souza
email: gbr1990@gmail.com
*/

#ifndef dht22_h
#define dht22_h



uint16_t dht22_read(uint16_t *result);

#endif
//
// END OF FILE
//
```

dht22.c:

```
/*
Programa que faz parte da biblioteca dht22.h
Baseado em bibliotecas para Arduino
Autor: Gabriel Lara de Souza
email: gbr1990@gmail.com
*/

#include <zdo.h>
#include <halW1.h>
#include <appTimer.h>
#include "dht22.h"
#include <gpio.h>
#include <halAppClock.h>;


uint16_t dht22_read(uint16_t *result)
{
  uint16_t humidity;
  uint16_t temperature;

      // BUFFER TO RECEIVE
        uint8_t bits[5];
        uint8_t cnt = 7;
        uint8_t idx = 0;
              uint64_t ti;
              uint64_t tf;

        // EMPTY BUFFER
        for (int i=0; i< 5; i++) bits[i] = 0;


              // REQUEST SAMPLE

              GPIO_0_make_out();
```

```
        GPIO_0_clr();
        int k = 0;
    for(;k < 72; k++) __delay_us(250);

        GPIO_0_set();

        __delay_us(40);

        GPIO_0_make_in();


    // ACKNOWLEDGE or TIMEOUT
    unsigned int loopCnt = 50000;
    while(!GPIO_0_read())
            if (loopCnt-- == 0) return 1;

    loopCnt = 50000;
    while(GPIO_0_read())
            if (loopCnt-- == 0) return 2;

    // READ OUTPUT - 40 BITS => 5 BYTES or TIMEOUT
    for (int i=0; i<40; i++)
    {
            loopCnt = 50000;
            while(!GPIO_0_read())
                    if (loopCnt-- == 0) return 3;


                        halGetSystemTimeUs(&ti);
            loopCnt = 50000;
            while(GPIO_0_read())
                    if (loopCnt-- == 0) return 4;

                        halGetSystemTimeUs(&tf);
            if ((tf-ti) > 40) bits[idx] |= (1 << cnt);
            if (cnt == 0)   // next byte?
            {
                    cnt = 7;    // restart at MSB
                    idx++;      // next byte!
            }
            else cnt--;
    }

    // WRITE TO RIGHT VARS
    // as bits[1] and bits[3] are always zero they are omitted in formulas.
    humidity = bits[0];
        humidity = (humidity << 8) | bits[1];

        result[1] = bits[0];
        result[1] = (result[1] << 8) | bits[1];

    result[0] = bits[2];
        result[0] = (result[0] << 8) | bits[3];

        temperature = bits[2];
        temperature = (temperature << 8) | bits[3];


    uint8_t sum = bits[0] + bits[1] + bits [2] + bits[3] ;

    if (bits[4] != sum) return 5;
    return temperature;
}
//
// END OF FILE
//
```