UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
BACHARELADO EM ENGENHARIA ELETRÔNICA

GUILHERME PALLARO ROSSI

MÓDULOS DE HARDWARE PARA APLICAÇÃO EM ARQUITETURAS DESCENTRALIZADAS DE AUTOMAÇÃO RESIDENCIAL

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2018

GUILHERME PALLARO ROSSI

MÓDULOS DE HARDWARE PARA APLICAÇÃO EM ARQUITETURAS DESCENTRALIZADAS DE AUTOMAÇÃO RESIDENCIAL

Trabalho de Conclusão de Curso de Graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2 - TCC2, do curso superior de Engenharia Eletrônica do Departamento Acadêmico de Eletrônica (DAELN) da Universidade Tecnológica Federal do Paraná (UTFPR), como requisito parcial para obtencão do título de Engenheiro Eletrônico.

Orientador: Prof. Dr. Márcio Rodrigues da Cunha

CAMPO MOURÃO

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Campo Mourão
Coordenação do Curso de Engenharia Eletrônica

TERMO DE APROVAÇÃO

DO TRABALHO DE CONCLUSÃO DE CURSO INTITULADO

Módulos de hardware para aplicação em arquiteturas descentralizadas de automação residencial.

Guilherme Pallaro Rossi

Trabalho de Conclusão de Curso apresentado no dia 10 de agosto de 2018 ao Curso Superior de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, Campus Campo Mourão. O Candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho _____ (aprovado, aprovado com restrições ou reprovado).

Prof. André Luiz Regis Monteiro
(UTFPR)

Prof. Osmar Tormena Junior
(UTFPR)

Prof. Márcio Rodrigues da Cunha
(UTFPR)
Orientador

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

AGRADECIMENTOS

A Deus, por ter me dado saúde e força para superar as dificuldades.

A esta universidade e seu corpo docente, direção e administração que se empenharam durante os últimos anos em minha formação, fornecendo excelente estrutura, tanto física como humana. Em especial ao meu orientador, Dr. Márcio R. da Cunha, que compartilhou seu conhecimento e experiência, mesmo dispondo de pouco tempo.

Aos meus pais, Reginaldo e Sandra, pelo amor, incentivo e apoio incondicional. Ao meu primo Jorge Luiz, que contribuiu com seus conhecimentos de programação, que foram fundamentais no desenvolvimento do trabalho. Aos meus tios, Paulo e Andréa, que me acolheram durante mais de um ano em sua casa quando comecei os estudos, e até hoje continuam me dando suporte. À Beatriz, minha namorada, que soube entender minhas ausências e me ajudou com seus conselhos.

À empresa AE Tech Soluções, que disponibilizou sua estrutura para a confecção das placas de circuito impresso e contribuiu para aumentar meus conhecimentos durante o tempo do estágio.

Aos meus colegas de turma, em especial Hugo e Alisson, que estiveram comigo desde o início do curso, sempre dispostos a ajudar.

Aos meus avós, tios, primos e toda a família, que sempre estiveram ao meu lado, sem nunca ter negado ajuda. À Natália, por dispor de seu tempo para me ajudar na revisão deste trabalho.

"Buscai primeiro o reino de Deus e a sua justiça,
e tudo o mais vos será acrescentado."
(Bíblia Sagrada, S. Mateus 6:33)

RESUMO

ROSSI, Guilherme Pallaro. **Módulos de hardware para aplicação em arquiteturas descentralizadas de automação residencial.** Trabalho de Conclusão de Curso - Bacharelado em Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Campo Mourão, 2018.

Este trabalho tem como finalidade desenvolver módulos para automação residencial, usando FPGA como dispositivo de controle, para demonstrar sua utilização em aplicações nas quais normalmente são utilizados outros tipos de controladores. O projeto se baseia em uma arquitetura descentralizada, com o uso de dois controladores e duas plantas. Foi desenvolvido um *hardware* para cada FPGA e um aparato de testes, representando uma residência com sensores e atuadores. Para a interligação das partes, também foram desenvolvidas duas placas de interface que fazem as conexões entre a maquete e as FPGAs, além da implementação de um protocolo de rede (utilizando Ethernet e TCP/IP) para futura utilização na interligação das duas centrais. E ainda, como interface homem-máquina, um *software* (contendo uma interface gráfica) foi desenvolvido para o Linux, além da montagem de um circuito com botões funcionando de forma paralela à interface. O sistema se mostrou eficiente, representando como seria o funcionamento desse mesmo projeto instalado em uma residência real. Além disso, concluiu-se que FPGAs podem ser usadas como controladores para sistemas muitos mais complexos do que uma residência com poucos elementos.

Palavras-chave: FPGA; Automação Residencial; Ethernet; Interface Gráfica.

ABSTRACT

ROSSI, Guilherme Pallaro. **Hardware modules for application in descentralized architectures of home automation.** End of Course Work - Bachelor's Degree in Electronic Engineering, Universidade Tecnológica Federal do Paraná. Campo Mourão, 2018.

This work aims to develop modules for home automation using FPGA as a controller device to demonstrate its use in applications in which usually other types of controllers are used. The project is based on a decentralized architecture, using two controllers and two plants. A hardware for each FPGA was developed and also a test apparatus, representing a house with sensors and actuators. For the interconnections between the parts, two interface boards were also manufactured, which make the connections between the model and the FPGAs, besides the implementation of a network protocol (using Ethernet and TCP/IP) for future application in the interconnection of the two control centers. Also, as a human-machine interface, a software (containing a graphical user interface) was developed for Linux, furthermore the assembly of a circuit with buttons working in parallel to the interface. The system proved to be efficient, representing how would be the operation of this same project installed in a real residence. Furthermore, it has been deduced that FPGAs can be used as controllers for more complex systems than a house with a few elements.

Keywords: FPGA; Home Automation; Ethernet; Graphical Interface.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ADC	*Analog-Digital Converter* - Conversor Analógico-Digital
ASIC	*Aplication-Specific Integrated Circuit* - Circuito Integrado de Aplicação Específica
CA	Corrente Alternada
CC	Corrente Contínua
CI	Circuito Integrado
CLP	Controlador Lógico Programável
CPLD	*Complex Programmable Logic Devices* - Dispositivo de Lógica Programável Complexa
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection* - Acesso Múltiplo e Sensoreamento de Portadora com Detecção de Colisão
DAC	*Digital Analog Converter* - Conversor Digital-Analógico
DNS	*Domain Name Service* - Serviço de Nomes de Domínios
DSAP	*Destination Service Access Point* - Ponto de Acesso ao Serviço de Destino
FCS	*Frame Check Sequence* - Sequência de Verificação de Quadros
FPGA	*Field Programmable Gate Array* - Matriz de Portas Programáveis em Campo
FTP	*File Transfer Protocol* - Protocolo de Transferência de Arquivo
GIMP	*GNU Image Manipulation Program*- Programa de Manipulação de Imagem por GNU
GNU	*GNU is Not Unix* GNU não é Unix
GPIO	*General Purpose Input/Output* - Entrada e Saída de Propósito Geral
GTK	*GIMP Toolkit* - Kit de Ferramentas GIMP
HPS	*Hard Processor System*
HTTP	*Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto
I/O	*Input/Output* - Entrada/Saída
ICMP	*Internet Control Message Protocol* - Protocolo de Controle de Mensagem Internet

IEEE	*Institute of Electrical and Electronics Engineers* - Instituto de Engenheiros Eletricistas e Eletrônicos
IP	*Internet Protocol* - Protocolo Internet
ISO	*International Organization for Standardization* - Organização Internacional de Padronização
LAN	*Local Area Network* - Área de Rede Local
LDR	*Light Dependent Resistor* - Resistor Dependente de Luz
LED	*Light Emitting Diode* - Diodo Emissor de Luz
LLC	*Logic Link Control* - Controle de Link Lógico
MAC	*Media Access Control* - Controle de Acesso de Mídia
NTC	*Negative Temperature Coeficient* - Coeficiente Negativo de Temperatura
OSI	*Open Systems Interconnection* - Interconexão de Sistemas Abertos
PCI	Placas de Circuito Impresso
PIR	*Passive Infrared* - Infravermelho Passivo
PLD	*Programmable Logic Device* - Dispositivo de Lógica Programável
PTC	*Positive Temperature Coeficient* - Coeficiente Positivo de Temperatura
PWM	*Pulse Width Modulation* - Modulação por Largura de Pulso
RISC	*Reduced Instruction Set Computer* - Computador com Conjunto Reduzido de Instruções
ROM	*Read Only Memory*- Memória Somente Leitura
RTP	*Real-Time Transport Protocol* - Protocolo de Transporte em Tempo Real
SAR	*Successive Approximation Register* - Registrador de Aproximações Sucessivas
SFD	*Start of Frame Delimiter* - Início do Delimitador de Quadros
SMTP	*Simple Mail Transfer Protocol* - Protocolo de Transferência de Correio Simples
SNAP	*Sub-Network Access Protocol* - Protocolo de Acesso à Sub-rede
SPLD	*Simple Programmable Logic Devices* - Dispositivos de Lógica Programável Simples
SRAM	*Static Random Access Memory* - Memória Estática de Acesso Aleatório
SSAP	*Source Service Access Point* - Ponto de Acesso do Serviço de Origem

STP	*Shielded Twisted Pair* - Par Trançado Blindado
TCP	*Transmission Control Protocol* - Protocolo de Controle de Transmissão
TELNET	Protocolo de Terminal Virtual
UDP	*User Datagram Protocol* - Protocolo de Datagrama do Usuário
UI	*Unnumbered Information* - Informação não Enumerada
UTP	*Unshielded Twisted Pair* - Par Trançado não Blindado
VHDL	*VHSIC Hardware Description Language* - Lingugem de Descrição de Hardware VHSIC
VHSIC	*Very High Speed Integrated Circuit* - Circuito Integrado de Velocidade Muito Alta
XID	*eXchange IDentification* - Identificação de Troca

SUMÁRIO

1 INTRODUÇÃO

A automação residencial, também conhecida como domótica, é um conjunto de tecnologias que permite automatizar processos ou realizar remotamente o controle de equipamentos em uma residência. Pode ser aplicada em diversas áreas, por exemplo, iluminação, climatização, entretenimento, sistemas de segurança, entre outras. Tem por objetivo facilitar as tarefas de seus moradores, aumentar o conforto e a segurança e diminuir gastos com desperdícios (BOLZANI, 2004).

A domótica está inserida dentro do universo da automação, cujo significado é tornar processos automáticos. A automação é um recurso antigo e compreende várias áreas, tais como, industrial, predial, comercial, entre outras. Inicialmente, foi aplicada nas indústrias, sempre buscando facilitar os trabalhos manuais. Os processos de automação evoluíram, principalmente, no século XVIII, com a Revolução Industrial na Inglaterra. Com a evolução da eletrônica e da informática no século XX, tornou-se viável trazer a automação para outras áreas, como a residencial (TEZA, 2002).

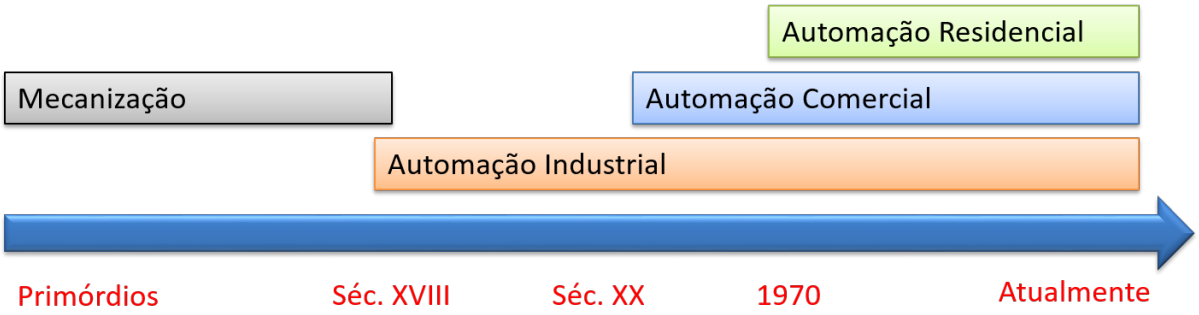
As primeiras residências automatizadas surgiram nos Estados Unidos, na década de 1970. Os primeiros módulos enviavam os sinais através da rede elétrica da residência, para tratar de problemas pontuais, através de soluções simples. Com o avanço e popularização de outras tecnologias, oferta abundante e barata de serviços de comunicação e o ambiente propício para o crescimento dos sistemas domóticos, a busca por tecnologias residenciais aumentou, contribuindo para a evolução deste cenário (MURATORI; BÓ, 2011).

A Figura 1 apresenta uma linha do tempo que resume o surgimento da automação residêncial, que vem desde os primórdios, com a mecanização, com o surgimento da roda d'água, por exemplo, até os dias de hoje. A evolução das tecnologias passou pela Revolução Industrial no século XVIII, com o surgimento das máquinas à vapor, dando início à automação industrial, que ganhou forças com a descoberta da eletricidade até o século XX. Em seguida, o avanço da informática e da eletrônica viabilizou o surgimento da automação comercial, até que essas tecnologias chegaram às residências.

Em uma casa automatizada, são encontrados alguns elementos básicos: controladores, interfaces, sensores e atuadores, protocolos e barramentos. Através dos barramentos, os sensores, atuadores e interfaces se comunicam com os controladores, através de protocolos. A Figura 2 mostra um exemplo de como os elementos se comunicam (ACCARDI; DODONOV, 2012).

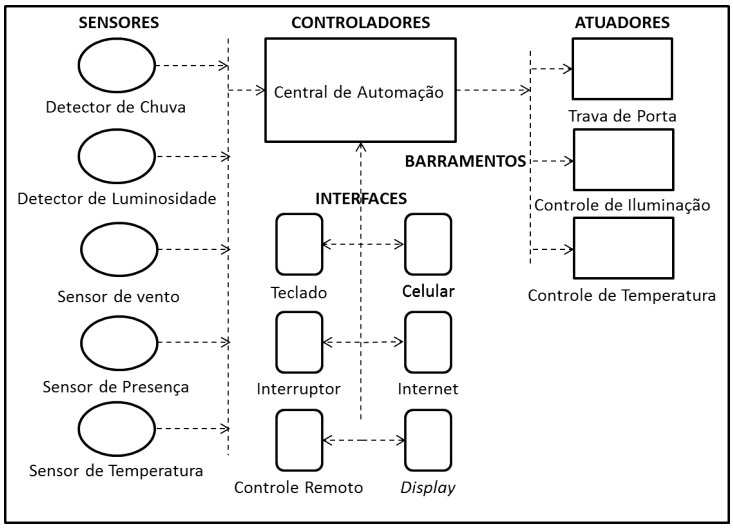
Os controladores são os elementos que recebem informações vindas dos sensores e das interfaces de entrada, e enviam comandos aos atuadores, além de dados para serem apresentados nas interfaces de saída. Existem várias centrais de automação no mercado,

Figura 1 – Linha do tempo do surgimento da automação residencial.



Fonte: Autoria Própria.

Figura 2 – Exemplo de comunicação de elementos básicos da automação residencial.



Fonte: Adaptado de Accardi e Dodonov (2012).

dentre elas, algumas que permitem muitas ações e outras que são mais limitadas. No entanto, quanto mais recursos um sistema oferece, mais complexo se torna o seu circuito interno, o que acaba elevando o seu custo (ROVERI, 2012; ACCARDI; DODONOV, 2012).

As centrais de automação podem ser feitas de várias formas. Segundo Abreu e Valim (2011, p. 4), "muitos sistemas são executados com CLPs (Controlador Lógico Programável)". Já Alvarez e Antunes (2015) implementam um sistema utilizando uma plataforma Arduíno®, a qual possui um microcontrolador da Atmel®. Assim, é possível projetar um controlador de domótica de inúmeras maneiras, inclusive com FPGA (*Field Programmable Gate Array* – Matriz de Portas Programáveis em Campo), que é um dispositivo lógico programável, no qual é possível descrever um *hardware* através de alguma linguagem própria e alterá-lo depois, caso necessário (TOCCI; WIDMER; MOSS, 2011).

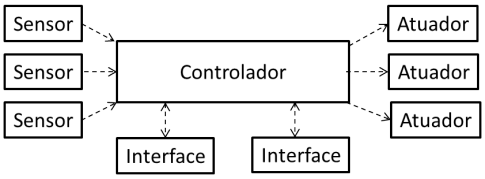
A forma como os sistemas são conectados descrevem a arquitetura da automação,

que pode ser centralizada ou descentralizada. Na centralizada (Figura 3(a)), há somente um controlador e todos os outros elementos são conectados a ele. Isso reduz o custo, mas aumenta a complexidade do sistema e o número de cabos utilizados (ABREU; VALIM, 2011).

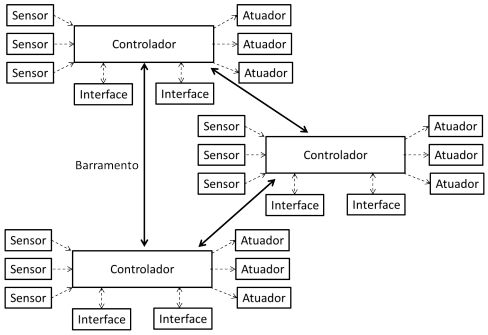
Já na arquitetura descentralizada (Figura 3(b)), há vários controladores e os elementos não são necessariamente conectados a mais de um deles, de forma a dividir o sistema para suprir necessidades complexas. Este modelo deixa o sistema mais imune a falhas, facilita a instalação e a interação com o usuário. No entanto, torna-se mais caro, devido ao número de equipamentos utilizados (ABREU; VALIM, 2011).

Figura 3 – Arquiteturas da Automação.

(a) Arquitetura Centralizada.



(b) Arquitetura Descentralizada.



Fonte: Adaptado de Accardi e Dodonov (2012).

As interfaces são os meios em que o usuário pode interagir com o sistema de automação. Elas podem ser um teclado, um interruptor, um controle remoto, um celular, um computador, uma página da Internet, e assim por diante (ACCARDI; DODONOV, 2012).

Os sensores são os elementos que vão interagir com o meio externo para coletar dados e, normalmente, são necessários nos processos automáticos. Eles podem ser sensores de chuva, vento, temperatura, presença, luminosidade, umidade, detectores de fumaça, entre outros (ACCARDI; DODONOV, 2012).

Os atuadores são os elementos que irão realizar a ação no meio físico. Exemplos comuns são motores para persianas, aquecedores, ventiladores, aparelhos de ar condicionado, lâmpadas, fechaduras magnéticas, sirene, entre outros (ACCARDI; DODONOV, 2012).

De acordo com Tanenbaum e Wetherall (2011, p. 18), “um protocolo é um acordo entre as partes que se comunicam, estabelecendo como se dará a comunicação”. Existem aqueles que foram desenvolvidos para automação residencial e outros que não são específicos, mas que podem ser utilizados. Os principais são: X-10, LONWorks, ZigBee, KNX, IEEE 802.3 (Ethernet) e IEEE 802.11 (Wi-Fi) (ACCARDI; DODONOV, 2012; MURATORI; BÓ, 2014).

Para realizar a comunicação entre os elementos, podem ser utilizados barramentos com ou sem fio. Utilizar tecnologia sem fio é vantajoso por facilitar a instalação e a expansão. No entanto, esta forma é mais vulnerável à interferências, possui acessibilidade menos protegida e têm menores taxas de transferência de dados (RIBEIRO, 2014).

As conexões por cabo oferecem mais segurança em seu tráfego de informações, quase não apresentam problemas de transmissões de dados e interferências em seus sinais e têm um excelente rendimento. Por outro lado, possuem a desvantagem de limitar os pontos de conexão, apresentando uma rede com a instalação mais trabalhosa e oferecendo uma flexibilidade menor, pois os dispositivos precisarão se conectar em locais fixos, além do custo de instalação ser mais elevado (RIBEIRO, 2014).

Em uma automação, a arquitetura do *hardware* do controlador deve se adequar às necessidades do sistema para que haja eficiência. Esse problema deve ser tratado ao escolher o dispositivo de controle, o que causa dificuldades ao projetista, já que cada microcontrolador possui uma arquitetura diferente, elevando o custo do projeto.

A vantagem de um dispositivo FPGA é que, devido à sua possibilidade de alterar o *hardware*, é possível criar várias arquiteturas dentro de um mesmo *chip*, resultando em um dispositivo que se adapte melhor ao sistema. Além disso, a FPGA possui um grande número de entradas e saídas, o que permite gerenciar uma boa quantidade de elementos.

2 OBJETIVOS

2.1 OBJETIVOS GERAIS

Criar módulos de *hardware* para um sistema de automação residencial baseado em arquitetura descentralizada, utilizando FPGAs como controladores.

2.2 OBJETIVOS ESPECÍFICOS

Para que se cumpram os objetivos gerais, são necessárias várias etapas, as quais são apresentadas a seguir:

- Descrever *hardwares* para módulos de controle, que permitam, ao menos, duas entradas de controle distintas.
- Carregar os *hardwares* nas FPGAs, fazendo as devidas associações de sinais entre o sistema de controle e o aparato de testes.
- Desenvolver uma interface gráfica em linguagem de programação para entradas e saídas de dados.
- Implementar a comunicação entre as centrais em protocolo Ethernet e TCP/IP, através do modelo servidor-cliente.
- Desenvolver um *software* para ser executado junto às FPGAs, que servirá para receber e enviar informações para o servidor, atuando como cliente, e que também faça a leitura dos sensores.
- Projetar e fabricar placas para interfacear as FPGAs com o aparato de testes de forma prática e segura, contendo também conversores analógico-digitais para receber os sinais dos sensores.
- Construir um aparato de testes representando uma casa, contendo lâmpadas, tomadas, sensores de presença, temperatura e luminosidade e sirene de alarme, além de chaves para acionar os atuadores de forma paralela à interface gráfica do computador. Utilizar relés e opto-acopladores para ligar e desligar os equipamentos.

2.3 JUSTIFICATIVA

A demonstração de um sistema controlado por FPGA, através de plantas com elementos que representam os usados em uma residência real, permite avaliar a possibilidade

de se utilizar FPGA na criação de centrais de automação, já que ela possibilita trabalhar com arquiteturas específicas de *hardware*.

Além disso, apesar da grande quantidade de trabalhos sobre domótica encontrados na literatura, poucos aplicam FPGA. Portanto, o tema proposto se caracteriza como algo novo e pode preceder uma linha de pesquisa para a área acadêmica, como o estudo da FPGA e suas aplicações, e para a indústria, no desenvolvimento de sistemas de automação para serem comercializados.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados cada um dos elemento básicos da automação residencial. Inicialmente, é abordada a FPGA, que servirá de controlador. Em seguida, são descritas as interfaces homem-máquina (IHM), os sensores e os atuadores. Por se tratar de um sistema contendo parte de controle e potência, é necessária uma interface entre elas. Como será utilizada arquitetura descentralizada, deverá haver a comunicação entre os controladores. Para isso, será utilizado o protocolo Ethernet e o TCP/IP, os quais serão explicados na Seção 3.5, e ainda a forma de implementá-los em linguagem de programação C. Para finalizar, são apresentados os barramentos que serão utilizados no projeto.

3.1 FPGA

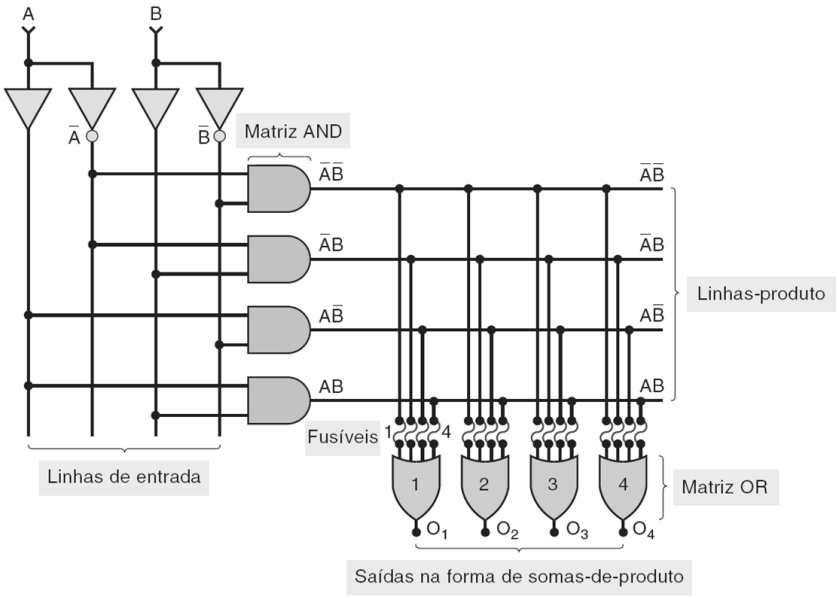
Os sistemas digitais são divididos em várias categorias. As três principais são: lógica padrão, microprocessadores e ASICs (*Aplication-Specific Integrated Circuits*). Os ASICs são circuitos integrados de aplicação específica, ou seja, são projetados para implementar uma aplicação qualquer desejada (TOCCI; WIDMER; MOSS, 2011).

Os ASICs são divididos em quatro sub-categorias: dispositivos lógico programáveis (PLD - *Programmable Logic Device - Dispositivo de Lógica Programável*), matrizes de portas, célula padrão e totalmente personalizados. Os PLDs podem ser adaptados para criar muitos circuitos digitais, dos mais simples aos mais complexos. Comparado aos outros ASICs, adquirir os *softwares* e *hardwares* para programar PLDs necessita investimento de capital mais baixo, pois os dispositivos das outras categorias requerem um contrato com alguma fábrica de circuitos integrados (CIs) para fabricar o chip desejado, o que é caro (TOCCI; WIDMER; MOSS, 2011).

Os PLDs surgiram na década de 1970, inicialmente para poder construir circuitos combinacionais lógicos que fossem programáveis. Esses dispositivos evoluíram e passaram a permitir implementação de circuitos sequenciais. Ao contrário dos microprocessadores, que possuem *hardware* fixo e executam um programa, os PLDs são programáveis em nível de *hardware*, ou seja, seu *hardware* pode ser configurado para atender determinadas especificações (PEDRONI, 2010).

A Figura 4 apresenta um PLD. Neste exemplo, a saída é função de duas entradas. Um PLD funciona basicamente através da "queima" de fusíveis, e considera-se que um fusível "queimado" não conduz. No circuito do exemplo, caso todos os fusíveis estejam intactos, todas as saídas terão nível lógico alto. Através da queima dos fusíveis, obtém-se resultados diferentes, sem alterar a configuração das portas lógicas (TOCCI; WIDMER; MOSS, 2011).

Figura 4 – Exemplo de um PLD.



Fonte: TOCCI; WIDMER; MOSS, 2011.

No entanto, os fusíveis são apenas uma representação de como funciona essa "programação" do *hardware*, pois, na realidade, isso é feito a partir de transistores, baseado nos vários tipos de memória de semicondutor. Dependendo da forma como for feito, o PLD pode ser volátil ou não volátil, ou seja, respectivamente, perdem a configuração do *hardware* ao ser desligado da energia ou não (PEDRONI, 2010; TOCCI; WIDMER; MOSS, 2011).

Os PLDs se dividem em três tipos diferentes: dispositivos lógicos programáveis simples (SPLD - *Simple Programmable Logic Devices*), dispositivos lógicos programáveis complexos (CPLD - *Complex Programmable Logic Devices*) e matrizes de portas programáveis em campo (FPGA - *Field Programable Gate Arrays*) (TOCCI; WIDMER; MOSS, 2011).

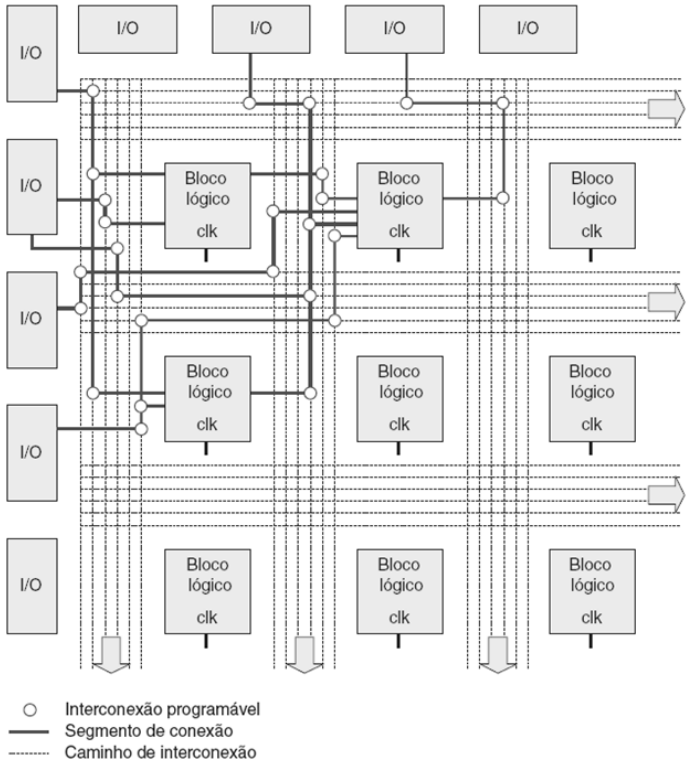
Tocci, Widmer e Moss (2011) atribuem os CPLDs e as FPGAs como dispositivos lógicos programáveis de alta capacidade, devido à sua grande quantidade de recursos lógicos. A FPGA, apesar de mais cara, apresenta uma eficiência melhor que a do CPLD, devido à sua arquitetura, podendo ser usada para projetos complexos e de alto desempenho (PEDRONI, 2010).

A maioria das FPGAs é volátil, pois sua arquitetura é baseada em SRAM (*Static Random Access Memory* - Memória Estática de Acesso Randômico). Dessa forma, é necessário transferir a programação do *hardware* para a FPGA, a partir de uma memória externa, quando o dispositivo é energizado (TOCCI; WIDMER; MOSS, 2011).

A Figura 5 apresenta uma visão da arquitetura simplificada de uma FPGA. De

acordo com Tocci, Widmer e Moss (2011), a programação do *hardware* contém informações que definem o funcionamento de cada bloco lógico, quais blocos de I/O (*Input/Output* - Entrada/Saída) são entradas e saídas e também como são interconectados.

Figura 5 – Arquitetura simplificada de uma FPGA.



Fonte: TOCCI; WIDMER; MOSS, 2011.

Tocci, Widmer e Moss (2011, p. 754) afirmam que as FPGAs têm a disponibilidade de propriedade intelectual, ou seja, "projetos predefinidos de blocos digitais complexos usados com seus próprios blocos de projeto para satisfazer às necessidades das aplicações". Esses blocos podem ser disponibilizados pelos próprios fabricantes da FPGA ou por terceiros. Como exemplo, existe a família de processadores embutidos versáteis chamados Nios II, da Altera®. Com o Nios II, é possível instalar um *software* para ser executado na FPGA.

De acordo com Pedroni (2010), a síntese dos circuitos de FPGA pode ser feita, dentre outras, a partir das linguagens VHDL e Verilog. Existe também a ferramenta Qsys, que auxilia na utilização e personalização de blocos pré-desenvolvidos e disponibilizados para os usuários de FPGA. O Nios II, por exemplo, é gerado à partir dessa ferramenta.

3.1.1 VHDL

VHDL significa *VHSIC Hardware Description Language - Linguagem de Descrição de Hardware VHSIC*, onde a sigla VHSIC vem de *Very High Speed Integrated Circuit* -

Circuito Integrado de Altíssima Velocidade. É uma linguagem de descrição de *hardware* que não depende de tecnologia e de fabricante, pois é padronizada pelo IEEE (*Institute of Electrical and Electronics Engineers* - Instituto de Engenheiros Eletricistas e Eletrônicos). O código descreve a estrutura ou o comportamento desejado, a partir do qual o compilador define um circuito físico (PEDRONI, 2010).

De acordo com Pedroni (2010), a estrutura do código VHDL se divide em três partes: declarações de bibliotecas e pacotes, entidade e arquitetura.

- **Declarações de bibliotecas e pacotes:** Esta é a primeira parte do código, na qual deve conter uma lista com as bibliotecas e pacotes necessários para que o compilador consiga processar o projeto.

- **Entidade:** A segunda parte do código é utilizada para especificar as portas de entrada e saída do circuito. Além disso, podem ser declaradas constantes genéricas globais.

- **Arquitetura:** Esta é a última parte de um código VHDL. É nela que se escreve o código que descreve o circuito. A descrição do *hardware* é feita a partir de instruções e operadores, que são aplicados aos valores de entrada para gerar a saída.

Um *hardware* possui dois tipos de lógicas: a combinacional e a sequencial. Na combinacional, os sinais são associados diretamente com os elementos lógicos, resultando em uma execução em paralelo. Na lógica sequencial, os sinais estão conectados aos elementos através de *flip-flops*, fazendo com que essas partes dependam de outro evento ocorrer para serem executadas (PEDRONI, 2010).

Em VHDL, um código combinacional deve ser escrito apenas dentro do ambiente da arquitetura. Para ser sequencial, as linhas devem estar escritas dentro do ambiente *process*, no qual é necessário declarar um sinal que ele irá monitorar um evento. O código sequencial na FPGA é semelhante ao uso de interrupções em microcontroladores (PEDRONI, 2010).

VHDL, além das entradas e saídas declaradas na entidade, possui também sinais intermediários, declarados na arquitetura. Podem ser *signal*, *variable*, *shared variable*, dentre outros. Também é possível criar blocos de memória. Os mais usados são os *signal*, que podem ser descritos tanto na parte combinacional, quanto na sequencial (PEDRONI, 2010).

É muito comum na descrição de um *hardware* o uso das portas lógicas e *flip-flops*. As principais portas lógicas são NOT, AND, OR, XOR e suas complementares NAND, NOR e XNOR. O *flip-flop* mais comum é o tipo D. Mais detalhes sobre esses elementos são tratados no Apêndice A.

Um código descrito em VHDL permite a criação de componentes, que são utilizados para reaproveitamento de código. Uma ou mais instâncias de um mesmo bloco podem ser criadas dentro de outro bloco através da criação de componentes, economizando linhas de código (PEDRONI, 2010).

3.1.2 Verilog

Verilog é uma outra linguagem de descrição de *hardware*. A estrutura de seu código se baseia em módulos e sua sintaxe é semelhante à linguagem de programação C (PALNITKAR, 1996).

A descrição de um módulo se inicia com a declaração dos pinos de entrada e saída. Em seguida, são declarados os sinais intermediários, que podem ser *wires* ou *regs*. Na sequência, inicia-se o código que contém a lógica em desenvolvimento(PALNITKAR, 1996).

Assim como em VHDL, Verilog também pode trabalhar em lógica combinacional e sequencial. Para usar lógica combinacional, deve se acrescentar o comando *assign* na frente da linha que a descreve. Se for usar sinais intermediários, utiliza-se *wires*(PALNITKAR, 1996).

A lógica sequencial é descrita dentro do ambiente criado com o comando *always*, semelhante ao *process* de VHDL. Tudo o que estiver dentro desse ambiente dependerá de uma borda de algum outro sinal, já que, na síntese, é criado um *flip-flop*. Nessa parte, os sinais intermediários utilizados são os *regs*(PALNITKAR, 1996).

Esta linguagem também permite a criação e uso de portas lógicas, comparadores e operadores aritméticos. Por ser uma linguagem com sintaxe mais enxuta, seu uso é muito comum para associar os sinais entre os módulos, quando o projeto necessita mais que um. Em VHDL, isso também é possível, mas o código fica mais carregado. VHDL já é mais usada na parte da criação da lógica, pois possui um pouco mais de recursos do que Verilog (PALNITKAR, 1996).

3.1.3 Nios II

O Nios II é um processador *softcore* desenvolvido pela Altera®. Ele é um processador RISC (*Reduced Instruction Set Computer* - Computador com Conjunto de Instruções Reduzido), baseado na arquitetura Harvard. Possui conjunto de instruções, dados e espaço de endereçamento de 32 bits, além de 32 registradores de propósito geral. Dessa forma, é possível realizar diversas tarefas, mantendo o foco no projeto do *software* (PIBER, 2017; INTEL FPGA, 2016).

Um processador ser *softcore* significa que ele é descrito em linguagem de descrição de *hardware* e que pode ser sintetizado e customizado em uma FPGA. Por esse motivo,

o Nios II possui a vantagem de ser flexível, pois ele pode se comunicar facilmente com outros periféricos e permite alterar o conjunto de instruções e as estruturas internas do processador, como tamanho da memória *cache*, priorização de interrupções, dentre outros (SILVA, 2014).

O termo RISC significa, em português, computador com conjunto de instruções reduzido, o que resulta numa unidade de controle simples, de baixo custo e rápida. Para isso, as arquiteturas RISC optam por instruções menos complexas possíveis, tendo pouca variedade e poucos endereços, o que traz de benefício a previsibilidade, resultando no ganho de desempenho (PRODUÇÃO VIRTUAL, 2017).

Na arquitetura Harvard, as memórias que contêm as instruções de programa são separadas das memórias de dados, o que permite que um processador acesse as duas simultaneamente, obtendo um bom desempenho, já que ele pode buscar uma nova instrução enquanto executa outra (MACÊDO, 2012).

Além de possuir instruções simples de multiplicação e divisão em 32 bits, o Nios II também tem instruções dedicadas para calcular multiplicações em 64 e 128 bits, o que permite realizar as operações de forma mais rápida e eficaz (PIBER, 2017; INTEL FPGA, 2016).

Em sua composição, o processador *softcore* da Altera® também contém acesso a vários periféricos *on-chip* e interface para memória e outros periféricos *off-chip*, formando um conjunto de elementos que visam melhorar o desempenho. Periféricos *on-chip* definem-se como aqueles implementados em linguagem de descrição de *hardware* e os *off-chip* são os implementados em CI's presentes nas placas dos kits (PIBER, 2017; INTEL FPGA, 2016; SILVA, 2014).

Além de seus periféricos, o Nios II também disponibiliza um ambiente de desenvolvimento baseado no *GNU is Not Unix* (GNU) C/C++ e o *software Build Tools* para o ambiente Eclipse. Para isso, utiliza-se a ferramenta Qsys, uma interface gráfica para habilitar as configurações do Nios II (MAGALHÃES, 2015).

A Altera® disponibiliza três versões diferentes para o Nios II, que se diferenciam pelo desempenho e economia. Todas são compatíveis em código entre si, mas suas implementações são diferentes. A primeira delas, a mais barata, denominada *Economy*, é a menor e mais simples. Não possui *cache* nem *pipeline* (PIBER, 2017; INTEL FPGA, 2016).

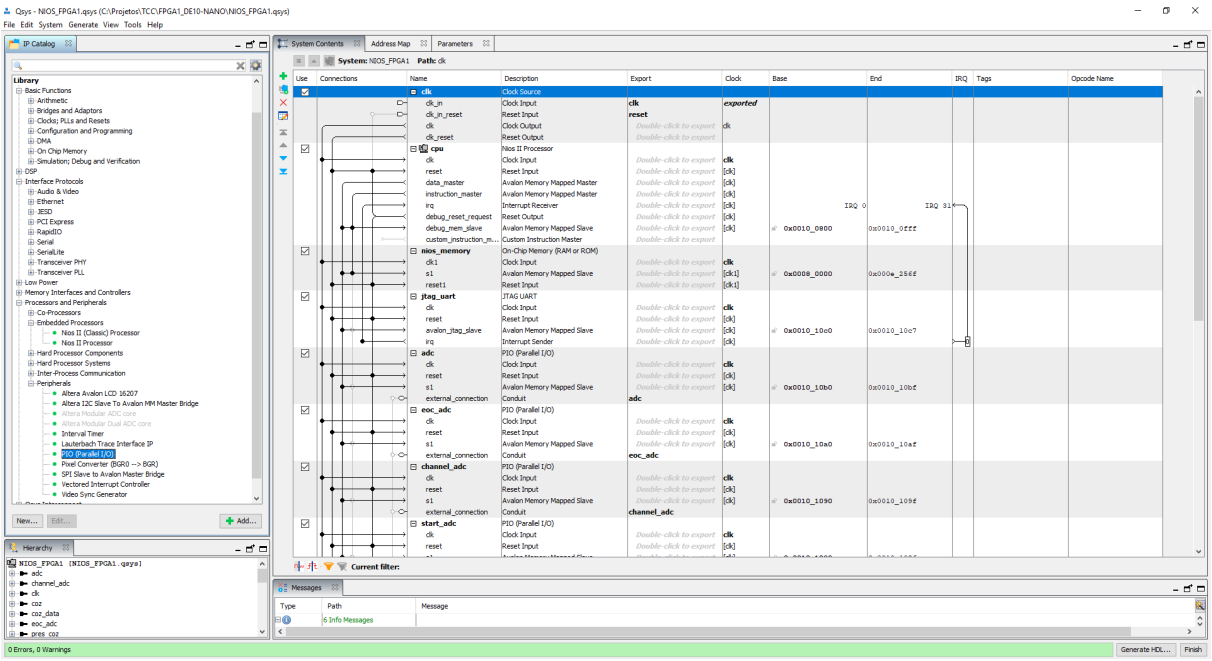
A segunda, intermediária entre a mais simples e a mais complexa, chama-se *Standard*. Esta inclui *cache* de dados, predição estática de saltos e cinco níveis de *pipeline* (PIBER, 2017).

A última, chamada *Fast*, é a mais completa, incluindo *cache* de dados de instrução, predição dinâmica de saltos e seis níveis de *pipeline* (PIBER, 2017).

3.1.4 Qsys

A Altera® disponibiliza uma plataforma para auxiliar no desenvolvimento de projetos que utilizam blocos *on-chip*, chamada Qsys. Esta ferramenta traz uma biblioteca com diversos blocos de *hardware* pré-desenvolvidos, tais como o processador *softcore* Nios II, funções básicas, memórias, blocos para processamento digital de sinais, protocolos de interface, alguns periféricos, como PIO (*Parallel Input-Output* - Entrada e Saída Paralela), dentre outros blocos. A Figura 6 apresenta a plataforma Qsys, com o projeto deste trabalho aberto para exemplificar.

Figura 6 – Plataforma Qsys.



Fonte: Autoria Própria

Um projeto no Qsys é feito adicionando-se os blocos no ambiente *System Contents*, configurando-os e fazendo as ligações com outros blocos ou exportando sinais externos. Por exemplo, para se adicionar o Nios II, também é necessário um *clock*, um bloco de memória *on-chip* para instruções e dados, uma interface de comunicação serial (JTAG-UART) e periféricos de saída, como PIO (TERASIC TECNOLOGIES INC., 2013).

PIOs são os blocos necessários para integrar o *hardware* descrito ao *software* programado no Nios 2, ou conectar o processador aos periféricos de saída *off-chip*. Eles são equivalentes aos *ports* dos microcontroladores, mas aqui são personalizáveis. Pode-se adicionar quantos o desenvolvedor desejar, dentro dos limites da FPGA. Cada PIO pode conter até 32 bits e serem configurados como entrada ou saída do processador (TERASIC TECNOLOGIES INC., 2013).

Ao final, uma síntese do projeto feito no Qsys é realizada para criar um bloco contendo os elementos adicionados. Esse bloco deve ser incluso no projeto feito no Quartus

para ser utilizado. Um projeto pode conter apenas blocos feitos pelo Qsys ou também integrá-los com outros blocos desenvolvido nas linguagens de descrição de *hardware* (TERASIC TECNOLOGIES INC., 2013).

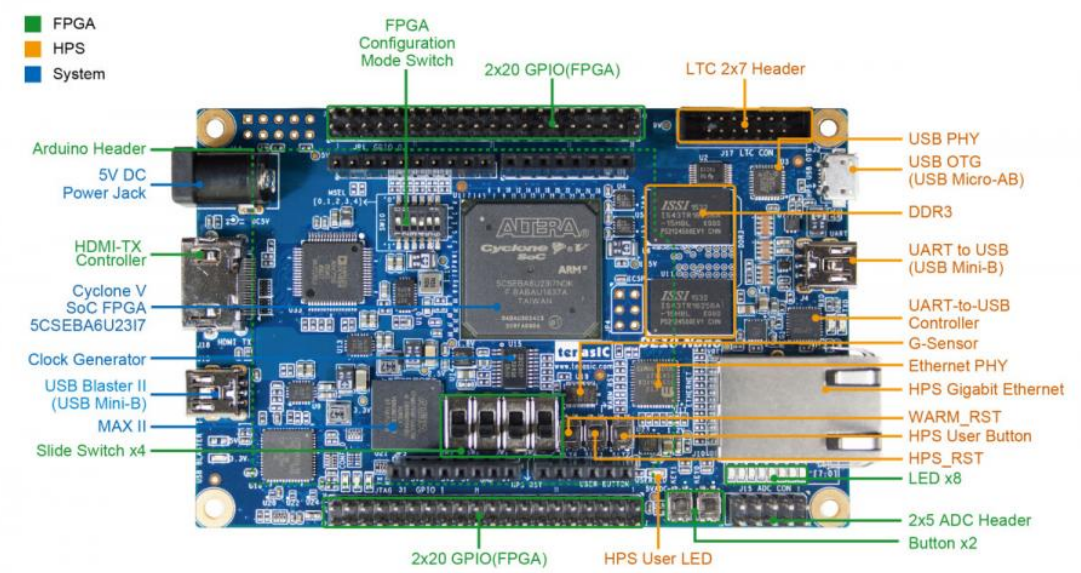
3.1.5 Kit de Desenvolvimento DE10-Nano

No mercado, existem vários kits de desenvolvimento para FPGAs, que são desenvolvidos para facilitar o trabalho do projetista. Os kits disponibilizam os periféricos *off-chip*, como memórias, conectores, transceptores, chaves, interruptores, diodos emissores de luz (LED - *Light Emitting Diode*) e assim por diante (TERASIC TECNOLOGIES INC., 2017).

A Terasic desenvolve vários kits para FPGAs da Altera. Eles diferem quanto ao chip FPGA e os periféricos. O kit utilizado neste trabalho é o DE10-Nano, o qual contém um chip Cyclone V (TERASIC TECNOLOGIES INC., 2017).

A placa DE10-Nano disponibiliza saídas HDMI, USB, Ethernet com padrão de conector RJ45 e duas portas de entrada e saída de propósito geral (GPIO - *General Purpose Input/Output* com 40 pinos cada uma, sendo 36 saídas ou entradas de propósito geral, 2 terras, 1 saída de 3,3 V e um de 5 V em cada conector. A Figura 7 apresenta uma imagem deste kit (TERASIC TECNOLOGIES INC., 2017).

Figura 7 – Kit de desenvolvimento DE10-Nano.



Fonte: TERASIC TECNOLOGIES INC., 2017.

Além disso, o kit também disponibiliza 8 canais de conversores analógico-digital com resolução de 12 bits e ainda 4 chaves do tipo interruptor e 2 do tipo *push-button* (TERASIC TECNOLOGIES INC., 2017).

Outro recurso importante deste kit é o próprio chip Cyclone V, que integra, em um mesmo encapsulamento, uma FPGA e um microprocessador ARM. Assim, em um mesmo chip, há recursos de *hardware* e *software*. Ao contrário do Nios II que é um *softcore*, o ARM é um *hardcore*, ou seja, sua arquitetura não é personalizável, mas possui diversos recursos (TERASIC TECNOLOGIES INC., 2017).

Com um cartão SD, é possível instalar um Linux para ser processado nesse ARM, o que aumenta mais ainda os recursos desse kit. O conector e transceptor Ethernet estão conectados diretamente ao ARM, permitindo que o projetista implemente um protocolo da mesma forma que faria em um computador, poupando trabalho (TERASIC TECNOLOGIES INC., 2017).

Através de uma das portas USB, o ARM também pode se comunicar com outro dispositivo via serial UART, permitindo que o usuário acesse o Linux através de outro computador. Outro modo de acessá-lo é conectando um monitor na saída HDMI e teclado e mouse na USB OTG. Assim, o usuário interage com o chip diretamente por ele (TERASIC TECNOLOGIES INC., 2017).

O Cyclone V ainda disponibiliza três pontes de conexão entre o ARM e a FPGA, denominadas *FPGA to HPS Bridge*, *HPS to FPGA Bridge* e *HPS to FPGA Lightweight Bridge*, onde HPS significa *Hard Processor System*. As duas primeiras possuem 32, 64 ou 128 bits, conforme configurado e a última possui apenas 32. Essas pontes fazem a comunicação entre as partes através de mapeamento de memória, por isso é necessário incluir periféricos para associar os PIOs a alguma memória (TERASIC TECNOLOGIES INC., 2017).

A ponte *HPS to FPGA Lightweight* permite a passagem de dados nos dois sentidos e pode ser mapeada de forma mais simples. Devido a isso, é a mais utilizada. Para aplicação que necessitam de mais bits, o uso se torna mais complexo, além de haver pouco material para dar instruções de como utilizá-las (TERASIC TECNOLOGIES INC., 2017).

3.2 INTERFACES HOMEM-MÁQUINA

O homem precisa interagir com a máquina para poder controlá-la ou verificar suas informações. No caso de um sistema de automação residencial, essas interfaces podem ser feitas, dentre outras formas, através de chaves mecânicas e interface gráfica em um *software*. A seguir, são explicadas essas duas formas.

3.2.1 Chaves Mecânicas

Em qualquer área da eletrônica é muito comum o uso de chaves para acionar alguma coisa. As chaves basicamente interrompem um circuito, podendo conectar um

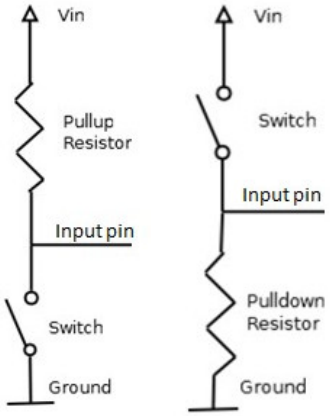
nó a outro ou deixá-lo sem conexão. Dois tipos comuns são os interruptores e as chaves *push-button* (ALMEIDA, 2014).

As primeiras são chaves que ficam travadas em uma determinada posição, até que haja uma força sobre ela para alterar seu estado. Possuem o mesmo princípio dos interruptores de lâmpadas das residências (ALMEIDA, 2014).

As chaves *push-button* possuem uma mola que as fazem retornar a seu estado normal quando não há mais força exercida sobre ela. São como os interruptores de campainha utilizados também em residências (ALMEIDA, 2014).

Para impedir flutuação em circuitos lógicos digitais, podem ser utilizados os resistores de *pull-up* e *pull-down*, conforme apresentados na Figura 8 (CUNHA, 2018).

Figura 8 – Resistores de *pull-up* e *pull-down*.



Fonte: CUNHA, 2018.

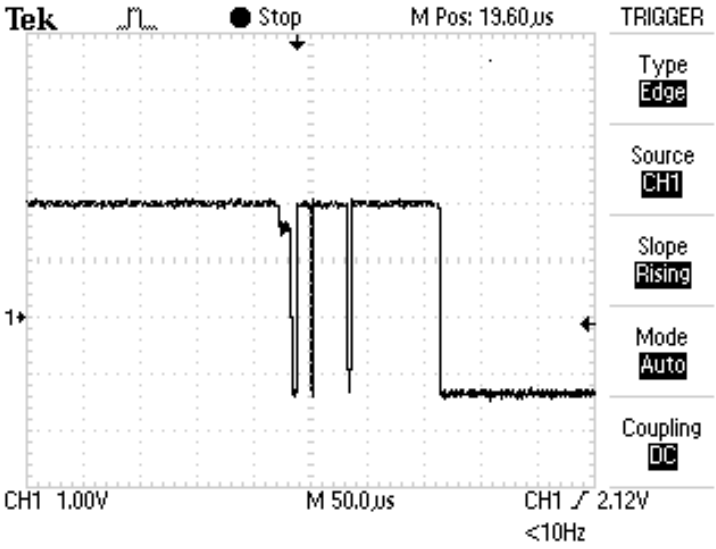
Um efeito comum e indesejado quando se utilizam chaves mecânicas é o *bouncing*. Esse efeito são oscilações indevidas que podem gerar acionamentos acidentais. A Figura 9 mostra como é essa oscilação, que ocorre quando a chave comuta (ALMEIDA, 2014).

Para tratar esse efeito, existem diversas soluções, como colocar capacitores em paralelo com a chave ou fazer um tratamento via *software* em microcontroladores ou via *hardware* em FPGA (ALMEIDA, 2014).

3.2.2 Interface Gráfica

Existem inúmeras maneiras de se criar alguma interface no computador. Nessas interfaces, são necessários *hardware* e *software*. O usuário irá ter o contato direto com a interface através do *hardware*, que pode ser um teclado, um mouse, uma tela, entre outros. O *software* é a parte que implementa os processos computacionais para controlar os dispositivos de *hardware*, construir os objetos de interfaces em que os usuários também

Figura 9 – Efeito *bouncing*.



Fonte: ALMEIDA, 2014.

podem interagir, criar mensagens e símbolos que representam as informações do sistema e interpretar os comandos dos usuários (LEITE, 2000).

Existem diversos tipos de interação, que vão desde os mais simples que contêm apenas texto, até os mais complexos, com elementos gráficos. Alguns deles podem ser úteis em um *software* de interface para domótica.

As linguagens de comandos representam uma opção que possibilita ao usuário enviar instruções diretamente ao sistema através de comandos específicos. Os comandos são passados pelo teclado e podem ser teclas com funções especiais, um caractere ou mesmo palavras. São interfaces simples, mas requerem que os usuários conheçam os comandos (LEITE, 2000).

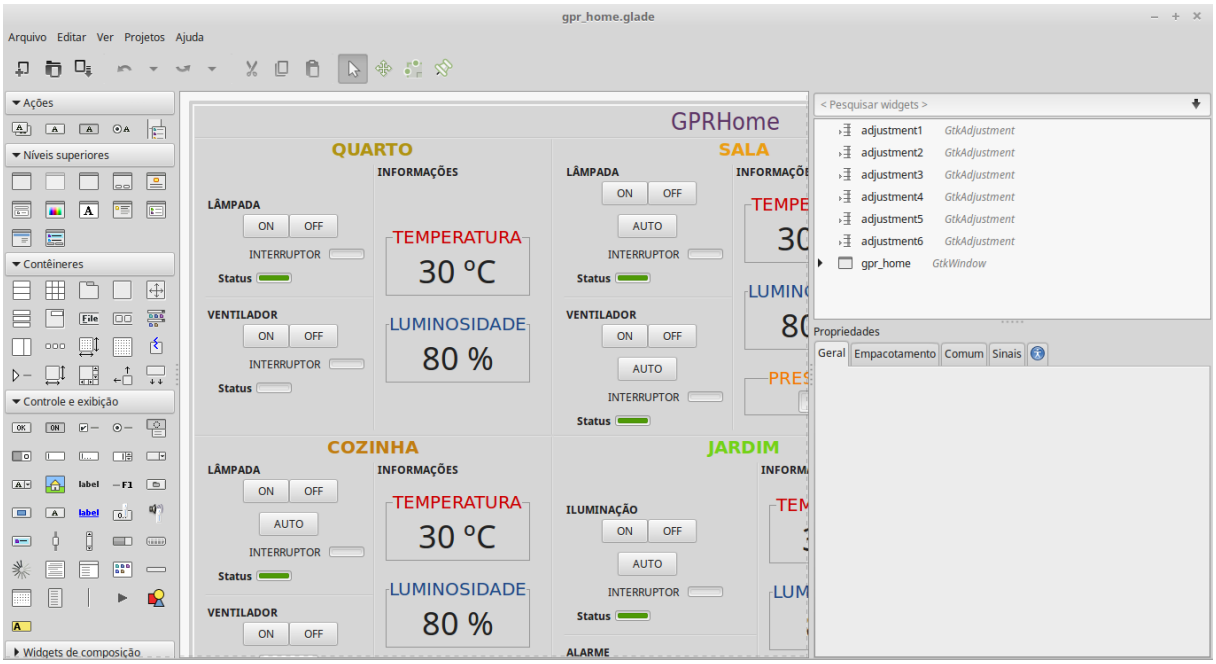
Leite (2000) também afirma que outra possibilidade são as interfaces por menus. Nesta, o conjunto de funções oferecidas pelo *software* é mostrado ao usuário através da tela. O usuário deve selecionar uma delas através de teclas alfanuméricas, do mouse, ou de teclas especiais. Esta interface também é simples, porém não é necessário que o usuário memorize os comandos, já que esses já são apresentados na tela.

Além dessas, existem as interfaces mais complexas, as quais possuem janelas, ícones gráficos, menus e apontadores. Nestes tipos, o usuário pode interagir com mouse e teclado, e é necessário um *software* mais complexo para gerenciar o ambiente gráfico. Grande parte das aplicações utilizam esse tipo de interface (LEITE, 2000).

3.2.2.1 Biblioteca GTK e Glade

O desenvolvimento de interfaces gráficas mais complexas não é tão trivial, por isso precisam de bibliotecas que auxiliam no processo. Para Linux, há uma biblioteca chamada GTK (*GIMP Toolkit* - Kit de Ferramentas GIMP (*GNU Image Manipulation Program* - Programa de Manipulação de Imagens para GNU)). A GTK possui funções que auxiliam na criação de janelas, campos de texto, botões e outros *widgets* utilizados nas interfaces gráficas. Existem duas formas de se trabalhar com ela: usando apenas seus comandos, sem auxílio de componentes gráficos, ou com auxílio de outro *software* para criar os *widgets*. O Glade é o *software* que facilita as tarefas(PROGRAMMER'S NOTES, 2015). A Figura 10 apresenta sua aparência.

Figura 10 – Interface do *software* Glade.



Fonte: Autoria Própria

A criação da interface gráfica se inicia com seu desenho no Glade. Após concluído, associam-se sinais para os *widgets* que terão interação com o programa, como botões, campos de texto de entrada e rótulos de saída. O Glade gera um arquivo `.xml` com as instruções de como o sistema operacional deve desenhar esta interface (PROGRAMMER'S NOTES, 2015).

Em um código à parte, escrito normalmente em C, esse arquivo `.xml` é incluído e seus sinais são associados à variáveis presentes no código. Graças à biblioteca GTK, manipulando essas variáveis, o programador estará mexendo com a interface (PROGRAMMER'S NOTES, 2015).

Por último, vêm as funções, que podem estar associadas aos botões ou ser "independentes". A execução das funções é feita após todas as declarações e associações.

Existe a função `g_idle_add`, que é executada sempre que o programa está em tempo ocioso (sem botões sendo apertados ou outras informações sendo passadas), a função `g_timeout_add_seconds`, que é chamada sempre que ocorre o término de uma contagem de tempo, e a função `gtk_main`, que faz o programa entrar em um laço de repetição aguardando os botões serem apertados (PROGRAMMER'S NOTES, 2015; GNOME DEVELOPER, 2014).

As funções `g_idle_add` e `g_timeout_add_seconds` são responsáveis por chamar outras funções. Elas apenas identificam quando tem algo ocioso ou o tempo limite passado por parâmetro foi atingido. Quando requisitadas, a execução é direcionada para as funções que são passadas em seus parâmetros. Elas devem ser chamadas antes da `gtk_main` (PROGRAMMER'S NOTES, 2015). Quando essas duas funções são chamadas, o processador cria uma *thread* para cada uma delas, para ficar monitorando quando esses eventos acontecem. Normalmente, elas chamam funções para mostrar informações na tela, como textos, sinalizadores, barras de *status*, relógio, dentre outras informações que não dependem dos botões (GNOME DEVELOPER, 2014).

De acordo com Tanenbaum e Wetherall (2011), *threads* são responsáveis por dividir a execução de um programa, para que seja feita em "paralelo". Isso depende dos processadores e núcleos disponíveis, mas o sistema operacional sabe como proceder. Mesmo que não tenha processadores suficientes para trabalhar em paralelo, é feita uma administração do tempo, para não travar em tarefas, dando o efeito de que o código está em paralelo.

No código da interface gráfica, a última função a ser chamada é a `gtk_main`. Ela é executada pela *thread* principal, que fica em *loop* esperando sinais dos botões, por isso a execução do programa fica "travada" depois de sua chamada até que o comando `gtk_main_quit` seja executado, que acontece quando se fecha a janela da interface. Devido a isso, as outras funções do programa que não dependem da interface devem possuir suas próprias *threads*, senão elas só serão executadas quando a interface se fechar (PROGRAMMER'S NOTES, 2015).

Cada botão possui uma função. Dentro das funções, ficam os códigos que devem ser executados quando o botão for pressionado (PROGRAMMER'S NOTES, 2015).

Com o auxílio dessas funções, o restante fica a critério da criatividade do programador.

3.3 SENSORES E ATUADORES

Em qualquer automação é necessário determinar as condições dos sistemas. Para isso, utilizam-se os sensores, dispositivos capazes de obter os valores das variáveis físicas do ambiente monitorado, e atuadores, os quais alteram determinada variável ambiente.

De acordo com Thomazini e Albuquerque (2005, p. 17), o termo “sensor” se refere a “dispositivos sensíveis a alguma forma de energia do ambiente que pode ser luminosa, térmica, cinética, relacionando informações sobre uma grandeza que precisa ser medida, como temperatura, pressão, velocidade, corrente, aceleração, posição etc.”

Ja os atuadores são elementos que modificam determinada variável controlada. Eles recebem um sinal vindo do controlador e agem no sistema controlado. Podem ser válvulas, relés, cilindros, motores, solenoides, entre outros (THOMAZINI; ALBUQUERQUE, 2005).

Os sensores, geralmente, precisam ter seu sinal de saída condicionado para que possa ser utilizado em um sistema de controle, pois eles nem sempre fornecem as características elétricas necessárias. Esse tratamento é realizado por um circuito de interface, que pode ser, por exemplo, um amplificador. Quando o sensor já vem conectado ao circuito de interface, o dispositivo é chamado de transdutor (THOMAZINI; ALBUQUERQUE, 2005).

Aqueles sensores que têm suas características alteradas ao interagirem com uma grandeza física são chamados de analógicos. Também é possível discretizar o sinal de um sensor analógico, passando-o por um circuito conversor, transformando-o em um transdutor digital. Além desses, existem os sensores que são construídos para terem sua saída digital sem passar por conversão, como é o caso de alguns sensores de vazão. Os sensores analógicos podem assumir qualquer valor ao longo do tempo, dentro de uma faixa de operação, enquanto os sensores digitais apresentam menor quantidade de valores (THOMAZINI; ALBUQUERQUE, 2005).

Existem sensores analógicos para medir pressão, temperatura, umidade, luminosidade, entre outros, pois essas são grandezas físicas que podem assumir qualquer valor. Já os sensores digitais podem ser usados para detectar passagem de objetos, medir a vazão de fluidos, verificar presença humana e assim por diante. Para isso, utilizam-se sensores ultrassônicos, sensores com saída em modulação por largura de pulso (PWM - *Pulse Width Modulation*), sensores infravermelhos, entre outros. (THOMAZINI; ALBUQUERQUE, 2005).

Quanto aos atuadores, alguns são mais comuns de ser usados na automação residencial. As lâmpadas e ventiladores, por exemplo, são elementos que atuam, respectivamente, na luminosidade e calor e são normalmente acionados por relés. Já os alarmes atuam produzindo som, além de poderem acionar outros dispositivos de segurança. O Apêndice C trata desses elementos com mais detalhes.

Muitas vezes, é necessária a conversão dos sinais, seja de um sensor para um controlador ou, então, de um controlador para um atuador. Para isso, existem os conversores analógico-digitais (ADC - *Analog-Digital Converter*) e os digital analógicos (DAC - *Digital-Analog Converter*). No entanto, parte da informação é perdida quando a conversão é realizada (THOMAZINI; ALBUQUERQUE, 2005). A subseção 3.4.2 tratará com mais

detalhes sobre os conversores analógico-digitais.

A seguir, serão descritos os sensores utilizados neste trabalho, os quais são comuns de serem encontrados em sistemas de domótica.

3.3.1 Sensor de Presença

Existem vários tipos de sensores de presença, dentre eles, ópticos, ultrassom e infravermelho. Eles podem ser usados tanto para detectar a presença de objetos, como de pessoas (THOMAZINI; ALBUQUERQUE, 2005). Na automação residencial é comum utilizar sensores infravermelhos passivos (PIR - *Passive Infrared*), que podem ser encontrados de várias formas diferentes, mas com princípios de funcionamento semelhantes.

Os sensores PIRs são basicamente constituídos de um sensor piroelétrico que, de acordo com Thomazini e Albuquerque (2005), pode ser um fotodiodo ou fototransistor, os quais são capazes de detectar níveis de radiação infravermelha (ADAFRUIT INDUSTRIES, 2014). Seu funcionamento é possível graças ao efeito descoberto por Willian Hershel, em 1800.

Hershel era um astrônomo que, em um de seus experimentos de observação do Sol, notou que a temperatura variava com a cor, fato que o motivou a estudar as temperaturas do espectro de cores. Com seu aparato experimental, Hershel descobriu que a temperatura era maior nas frequências mais baixas que o vermelho, as quais eram invisíveis (OLIVEIRA; SILVA, 2014).

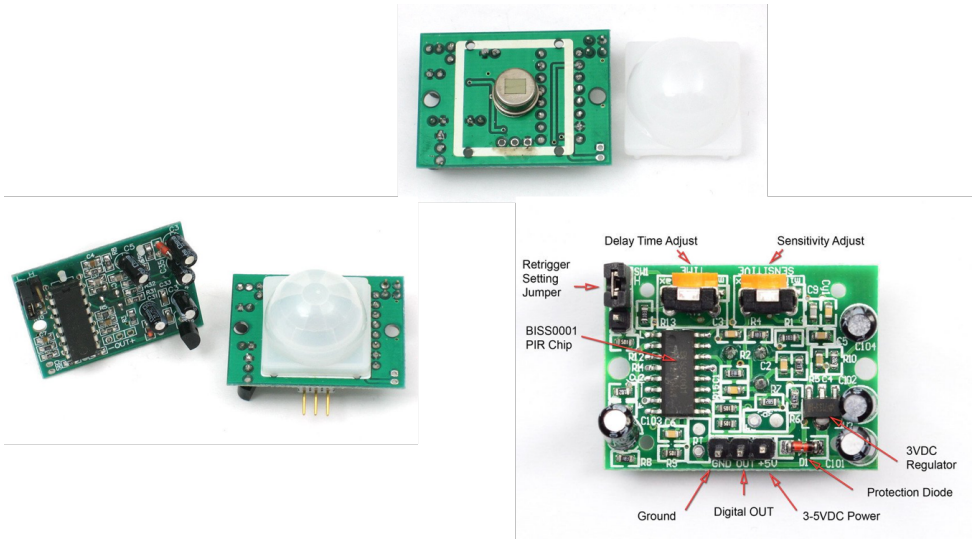
A radiação infravermelha causa vibrações nos átomos de um material físico, o que causa aumento da temperatura e variações nas características físicas, químicas e propriedades elétricas. Devido a isso, é possível utilizar o infravermelho como um sinal para detectar presença, já que todos os corpos emitem calor (VANZETTI, 1972 apud LOMBARDI, 2006).

Para se tornar aplicável, esses sensores trabalham em conjunto com outros componentes de suporte, como resistores, capacitores e circuitos integrados, formando um transdutor. O sensor de presença da Adafruit® é composto por um sensor piroelétrico dividido em duas metades, as quais estão ligadas de forma que se anulam mutuamente. Se uma metade perceber mais ou menos radiação infravermelha do que a outra, a saída será alta ou baixa (ADAFRUIT INDUSTRIES, 2014).

Esse modelo trabalha junto com o CI BISS0001, que lê a saída do sensor e faz um pequeno processamento para emitir um pulso de saída digital do sensor analógico. Além disso, o transdutor possui dois potenciômetros: um para ajustar o ângulo de detecção e outro para determinar o tempo que a saída fica em alta após a detecção. Sua sensibilidade permite um alcance de até seis metros e ângulo entre 110° e 70° de detecção. O componente é alimentado com tensão de 5V e sua saída digital é de 3.3V. A Figura 11 apresenta o

transdutor (ADAFRUIT INDUSTRIES, 2014).

Figura 11 – Sensor de Presença Adafruit.



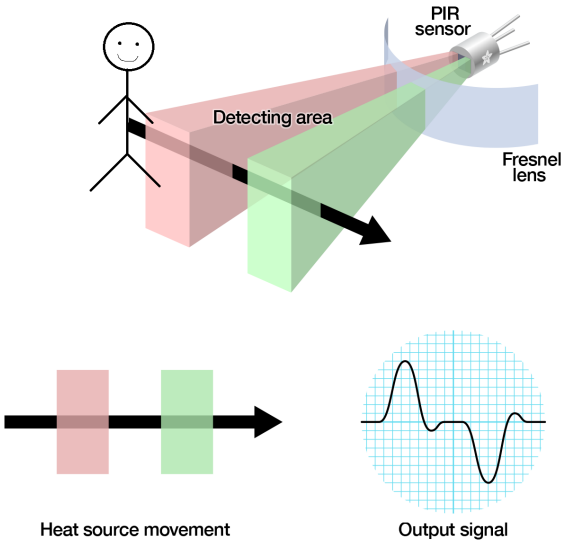
Fonte: Adaptado de Adafruit Industries (2014).

O transdutor pode ser configurado para trabalhar no modo de não-reativação ou no de reativação. O primeiro deles gera pulsos com durações fixas cada vez que detecta movimento, mas o pulso não é prolongado se houver movimento enquanto está em nível lógico alto. Esse modo pode ser utilizado para gerar interrupções em microcontroladores (ADAFRUIT INDUSTRIES, 2014).

No modo de reativação, a saída gera um sinal de nível lógico alto que terá a duração mínima conforme a regulagem do potenciômetro. No entanto, se outro movimento for detectado durante esse estado, a contagem do tempo se reinicia. Assim, a saída ficará em nível alto o tempo todo em que houver movimento, somando o tempo de atraso configurado no potenciômetro. O tempo de atraso pode variar de $2,5$ s a 250 s aproximadamente. Esse modo é utilizado na maioria das aplicações (ADAFRUIT INDUSTRIES, 2014).

Como já dito, o sensor da Adafruit® possui duas partes, cada uma feita de um material sensível a infravermelho. Há também uma lente, que basicamente determina a distância que o sensor detecta. Quando o sensor está ocioso, ambas as partes detectam a mesma quantidade de infravermelho, que pode ser a quantidade irradiada dos móveis ou paredes. Quando algum corpo quente, como pessoa ou animal, atravessa seu alcance, ele primeiro intercepta em uma metade do sensor PIR, gerando uma mudança de diferencial positivo entre as duas metades. Quando o corpo deixa a área sensível, o oposto acontece, de forma que o sensor gera uma mudança diferencial negativa. Essas mudanças de pulso são o que é detectado. A Figura 12 ilustra o funcionamento (ADAFRUIT INDUSTRIES, 2014).

Figura 12 – Funcionamento do sensor de presença.



Fonte: Adaptado de Adafruit Industries (2014).

3.3.2 Sensores de Luminosidade

Na automação residencial, podem ser usados sensores de luminosidade. Dentre eles, os mais comuns são os fotorresistores.

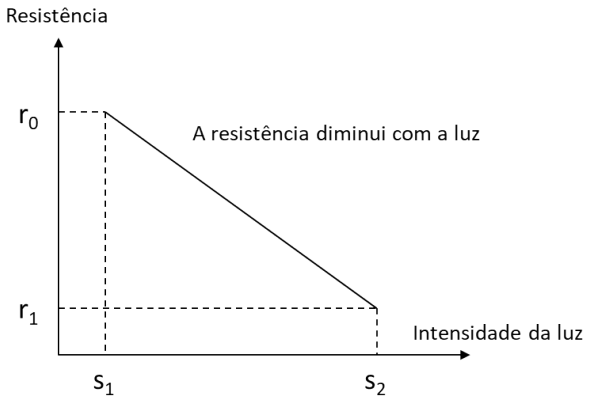
O LDR é um fotorresistor, cujo significado é Resistor Dependente de Luz. É um sensor que tem a sua resistência alterada quando recebe uma incidência de luz. Isso ocorre, porque, em certos materiais, acontece a liberação de portadores de carga, melhorando a condutividade elétrica, sempre que a luz incide sobre eles. No caso do LDR, o material utilizado é o sulfeto de cádmio (THOMAZINI; ALBUQUERQUE, 2005).

O sulfeto de cádmio apresenta uma resistência muito elevada no escuro (milhões de ohms). Quando recebe iluminação direta, ou seja, luz forte, sua resistência cai para centenas de milhares de ohms. Essa variação permite obter a incidência de luz no ambiente onde o sensor está colocado (THOMAZINI; ALBUQUERQUE, 2005).

Um gráfico mostrando a variação da resistência conforme a luminosidade é mostrado na Figura 13.

Os LDRs não são componentes polarizados, ou seja, a corrente pode circular nos dois sentidos, sem haver diferença nas variações da resistência com a luz. Sua sensibilidade se altera de acordo com o tamanho do componente. Uma superfície maior deixa o dispositivo mais sensível à luz, assim como uma capacidade maior de dissipar calor, permitindo que ele trabalhe com correntes maiores. Como exemplo, um LDR de um centímetro dissipa tipicamente 100mW, podendo ser aplicada uma tensão máxima entre seus terminais de 150 V (THOMAZINI; ALBUQUERQUE, 2005). A Figura 14 apresenta uma foto de um dos modelos deste sensor.

Figura 13 – Variação de resistência do LDR com a intensidade luminosa.



Fonte: Adaptado de Thomazini e Albuquerque (2005).

Figura 14 – Sensor LDR.

Fonte: Autoria Própria.

O tempo de resposta do LDR é lento, o que o impede de ser utilizado em algumas aplicações. Mas, em casos mais simples, como sensores de luz ambiente e detectores de níveis de iluminação, ele é muito útil (THOMAZINI; ALBUQUERQUE, 2005).

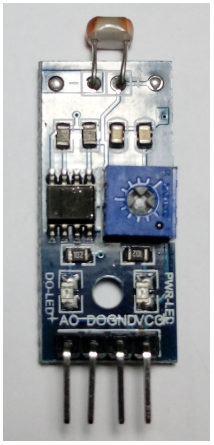
O LDR normalmente é usado em conjunto com outro resistor, formando um divisor de tensão. Com a variação de luz, a resistência do sensor varia e, consequentemente, a tensão no circuito. Existe um módulo de LDR desenvolvido para a plataforma Arduino, que já contem as trilhas, o LDR e o resistor. A Figura 15 apresenta esse módulo.

Pode-se ver na imagem que o módulo possui 4 pinos: VCC (5 V), GND, A0 (saída analógica) e D0 (saída digital). A saída digital depende de um comparador feito com o amplificador operacional LM383, cuja regulagem de sensibilidade pode ser feita através de um potenciômetro.

A saída analógica possui um circuito mais simples, que serve apenas para converter luminosidade em tensão. O esquemático é apresentado na Figura 16.

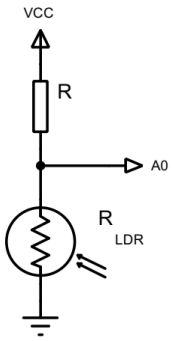
Utilizando as Leis de Kirchhoff, tem-se que a tensão em A0 é a descrita pela

Figura 15 – Módulo de sensor LDR.



Fonte: Autoria Própria.

Figura 16 – Esquemático da parte analógica do módulo de sensor LDR.



Fonte: Autoria Própria.

Equação 1.

$$A0 = \frac{VCC \times R_{LDR}}{R + R_{LDR}} \tag{1}$$

Observando a fórmula, é possível notar que a tensão no pino analógico irá diminuir se R_{LDR} decair, o que acontece quando se tem incidência de luz mais forte sobre o sensor. Como o módulo é alimentado com 5V, a tensão de saída será zero no limite máximo de medição de luz e 5V no limite mínimo.

Pode-se fazer um cálculo de porcentagem para indicar o nível de luminosidade. Se considerar que esse sinal vai ser passado por um conversor analógico-digital antes de ser tratado por um controlador, a tensão será convertida em um código que dependerá da resolução do conversor. Para um conversor de 8 bits, com tensão de referência igual a 5V (mesma tensão do módulo), a sua saída será $2^8 - 1 = 255$ quando o sensor estiver enviando 5 V, ou seja, 0 % de luminosidade. Para 100 %, a saída será igual a zero. Aplicando-se propriedades matemáticas, chega-se à Equação 2, que calcula a porcentagem de luz no

ambiente, na qual *code* é o código informado pelo ADC.

$$lum_{\%} = \frac{255 - code}{255} \times 100\ \% \tag{2}$$

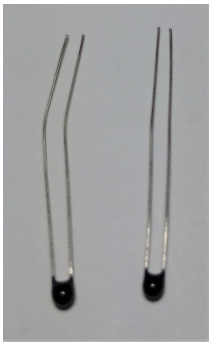
Essa equação pode ser usada pelo controlador para indicar a luminosidade do ambiente.

3.3.3 Sensores de Temperatura

Para medir a temperatura ambiente, os termistores são sensores que podem ser utilizados. Eles são resistores que variam sua resistência elétrica conforme a temperatura. Existem dois tipos básicos: os de coeficiente positivo de temperatura (PTC - *Positive Temperature Coeficient*) e os de coeficiente negativo de temperatura (NTC - *Negative Temperature Coeficient*) (THOMAZINI; ALBUQUERQUE, 2005).

A Figura 17 apresenta uma foto do sensor NTC.

Figura 17 – Termistor NTC.



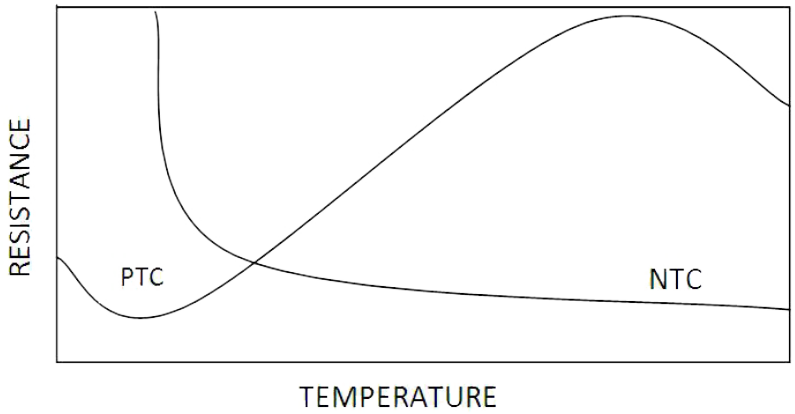
Fonte: Autoria Própria

Os PTCs aumentam sua resistência conforme a temperatura cresce, exceto em uma faixa de temperatura mais baixa, conforme é apresentado na Figura 18. Já os NTCs funcionam de forma contrária, ou seja, sua resistência diminui quando a temperatura se eleva, como é possível observar na imagem (THOMAZINI; ALBUQUERQUE, 2005).

Thomazini e Albuquerque (2005) também apresentam os sensores eletrônicos, que são desenvolvidos especialmente para montagem em placas de circuito impresso. Podem ser usados diodos ou transistores, os quais possuem parâmetros que variam com a temperatura, ou então circuitos integrados próprios, que geram em sua saída um sinal modulado proporcional à temperatura.

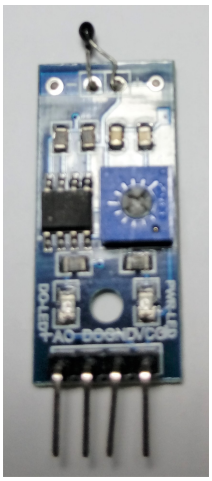
Da mesma forma que o LDR, o NTC também possui um módulo pronto (conforme Figura 19), desenvolvido para Arduino. A fórmula para a tensão de saída também é descrita pela Equação 1, apenas trocando a resistência do LDR pela resistência do NTC.

Figura 18 – Curvas típicas dos termistores.



Fonte: EAGER LEARNING, 2015.

Figura 19 – Módulo do termistor NTC.



Fonte: Autoria Própria

Assim, conhecendo a tensão e sabendo o valor de R, é possível calcular a resistência do sensor, com a Equação 3.

$$R_{NTC} = \frac{A0 \times R}{VCC - A0} \tag{3}$$

De acordo com Thomazini e Albuquerque (2005), a temperatura medida pelo sensor pode ser calculada pela Equação 4, de Steinhart & Hart,

$$T = \frac{1}{A + B + \ln(R) + C[\ln(R)]^3} \tag{4}$$

onde T é a temperatura em Kelvin e A, B e C são os coeficientes. Mais detalhes sobre como calcular os coeficientes são apresentados no Apêndice B.

3.4 INTERFACE ENTRE SISTEMA DE CONTROLE E PLANTA

A automação residencial é composta pela planta, onde estão os sensores e atuadores, e pela parte de controle. Os níveis de tensão entre essas partes são bem diferentes, por isso é necessário um circuito que faça a interface entre eles de forma segura. Além disso, os sensores medem suas grandezas e as convertem em sinais analógicos, sendo necessária sua conversão para a forma digital. Esta seção apresenta os circuitos que fazem esse interfaceamento.

3.4.1 Opto-acopladores e relés

Para acionar uma lâmpada ou qualquer outro elemento por um controlador digital, é necessário um circuito que permita esse acionamento. Em geral, os controladores não possuem tensão e potência necessárias para manter ligado um atuador. Dessa forma, deve-se usar uma interface entre as duas partes (controlador e atuador), que permita operar com dois níveis diferentes de tensão. Dentre as diversas opções de interface, existem os relés (SOUZA, 2014).

O relé é um dispositivo eletromecânico que funciona como um interruptor. Seu funcionamento depende das várias partes que o compõe, as quais são: bobina, que é constituída por um fio de cobre enrolado em um núcleo de material ferromagnético; armadura fixa, que atua como suporte; armadura móvel, que se desloca quando um campo eletromagnético é induzido na bobina; mola de rearme e terminais de entrada e saída. Os terminais de saída são compostos pelo terminal comum, o normalmente aberto e o normalmente fechado (MUNDO DA ELÉTRICA, 2018).

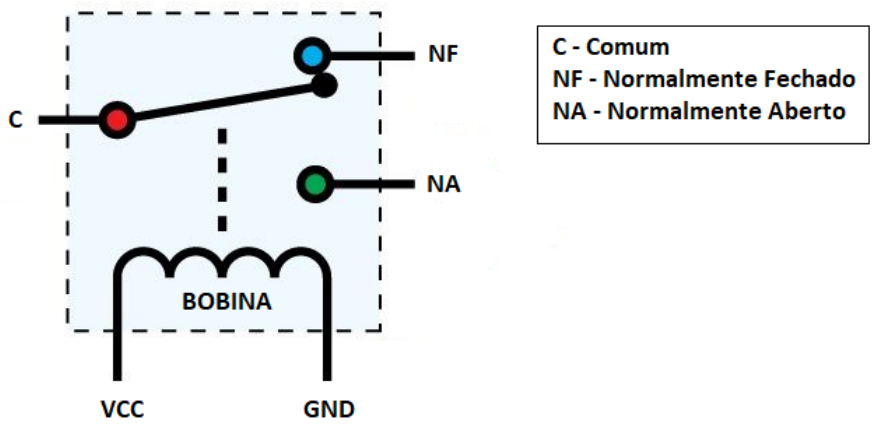
Quando uma tensão é aplicada aos terminais de entrada, uma corrente passa pela bobina e induz um campo eletromagnético. Esse campo atrai a armadura móvel, desconectando o terminal comum do terminal normalmente fechado e conectando-o ao terminal normalmente aberto. A Figura 20 ilustra a composição de um relé de forma simplificada (MUNDO DA ELÉTRICA, 2018).

Além do relé, que permite um controlador acionar circuitos de potência mais elevada, também é necessário um dispositivo de proteção que isole totalmente o circuito de potência e o de controle. Os opto-acopladores são exemplos de componentes para essa função. A Figura 21 apresenta o esquemático do opto-acoplador PC817, da *Sharp*.

O funcionamento do optoacoplador baseia-se no efeito fotoelétrico, no qual um feixe de luz infravermelha, produzido pelo LED quando se aplica uma tensão entre os terminais 1 e 2, polariza a base de um fototransistor, permitindo a condução entre coletor e emissor (terminais 3 e 4) (GONZAGA, 2015).

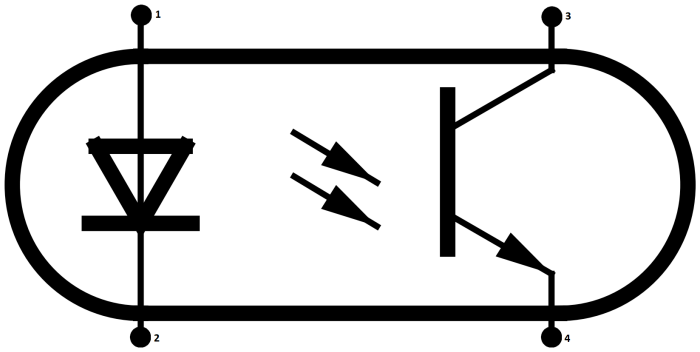
No mercado, existem módulos que já vêm com o circuito completo, incluindo opto-acopladores e relés. A Figura 22 apresenta um módulo de relés desenvolvido para a

Figura 20 – Esquemático de um relé.



Fonte: Adaptado de Tecnología (2018).

Figura 21 – Esquemático do optoacoplador PC817C.

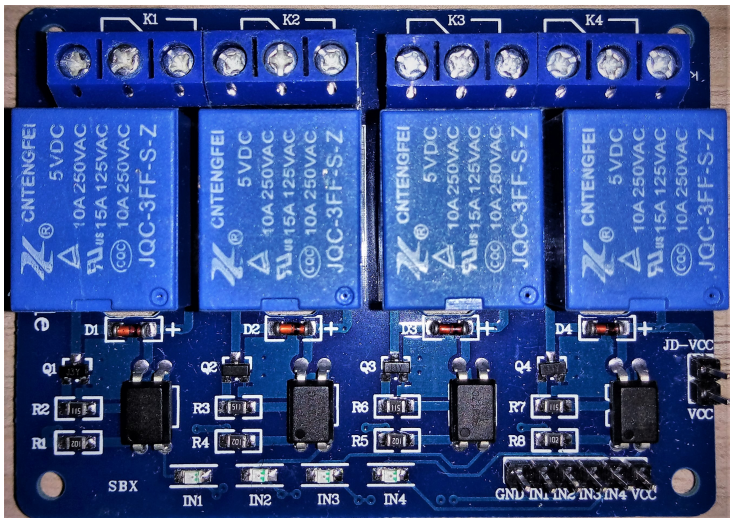


Fonte: Adaptado de Sharp (2018).

plataforma Arduíno, mas que pode ser utilizado com diversos controladores. Este módulo utiliza o relé JQC-3FF-S-Z, que precisa receber 5 V para atracar e o opto-acoplador PC817C, cujo LED infravermelho possui tensão direta de 1,2 V e corrente de 20 mA (HONGFA RELAY, 2018; SHARP, 2018).

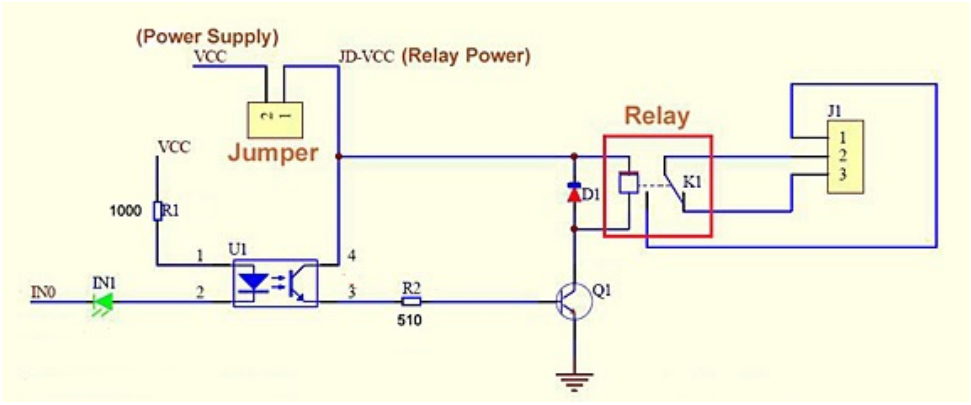
Este módulo possui quatro canais. A Figura 23 mostra o circuito esquemático para um dos canais deste módulo. Para isolar totalmente os lados de potência e controle, deve-se remover o *jumper* que conecta o JD-VCC ao VCC. Assim, é necessário utilizar duas fontes, uma para os relés e outra para o LED dos opto-acopladores. O módulo opera em nível lógico baixo, ou seja, aplicando 0V a qualquer um dois canais, uma corrente irá fluir entre VCC e IN, acendendo o LED infravermelho, que por sua vez polariza o fototransistor. Assim uma corrente irá fluir entre JD-VCC e a base do transistor Q, fazendo-o entrar em condução e permitindo que uma corrente flua pela bobina do relé, atracando-o. O diodo colocado em paralelo é chamado diodo de roda livre e serve para que a bobina possa ser descarregada em segurança quando o relé for desligado (AUTOMALABS, 2012).

Figura 22 – Módulo de relés de quatro canais.



Fonte: Autoria própria.

Figura 23 – Circuito esquemático do módulo de relés.



Fonte: Adaptado de Automalabs (2012).

3.4.2 Conversor Analógico Digital

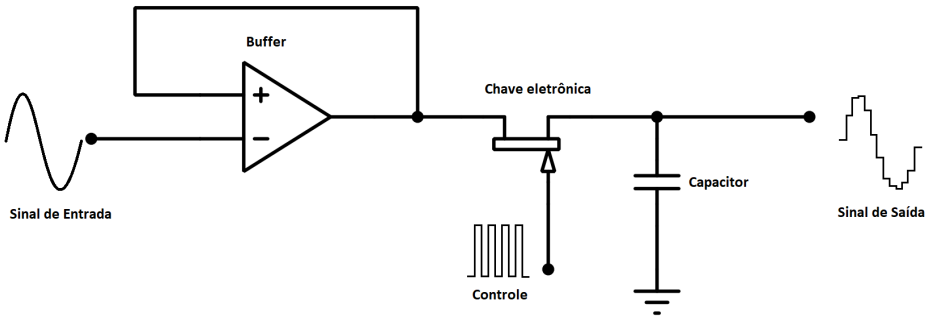
No mundo real, as grandezas físicas são variadas, como temperatura, luminosidade, pressão, umidade, dentre outras. Assim como apresentado na seção 3.3 sobre sensores, essas grandezas, geralmente relacionadas a alguma forma de energia, precisam ser convertidas em sinais elétricos para que possam ser tratadas por dispositivos eletrônicos. Essa transformação para sinal elétrico é tarefa dos sensores, que trabalham em conjunto com outros componentes que condicionam o sinal de saída, como níveis de tensão e corrente, formando os transdutores.

Através do sinal elétrico de saída do transdutor, é possível determinar o valor da grandeza física mensurada, pois o sinal é proporcional a ela. No entanto, este sinal ainda é analógico e contínuo no tempo, o que o impede de ser interpretado por circuitos digitais. Para que isso seja possível, o sinal precisa ser amostrado e quantizado, pois os sistemas

digitais trabalham com números finitos de valores. Essa conversão é realizada pelos ADCs, que transformam um sinal analógico, contínuo no tempo, para um sinal digital, discreto no tempo (PUHLMANN, 2015).

A conversão se inicia com o processo de amostragem, através do circuito de *sample and hold* (amostragem e retenção), que lê o sinal de entrada em um determinado instante e mantém esse valor na saída por um tempo. Esse valor de saída é o que será quantizado. O circuito de *sample and hold* mais simples é constituído por um *buffer*, através de um amplificador operacional, uma chave eletrônica, como um transistor, e um capacitor. A Figura 24 ilustra esse circuito (BRAGA, 2015).

Figura 24 – Circuito de *sample and hold*.



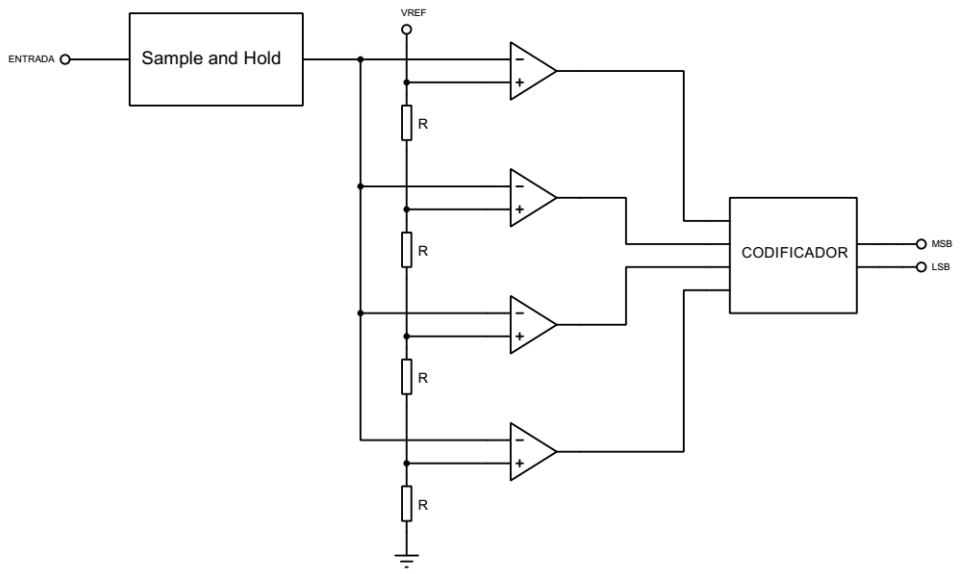
Fonte: Adaptado de Braga (2015).

O sinal analógico é amplificado pelo *buffer* de entrada, que também tem por finalidade isolar o circuito de conversão. Em sua saída, há uma chave eletrônica que determina o instante exato em que a leitura do sinal deve ser feita. A chave conduz por um tempo determinado pela frequência ou taxa de amostragem, permitindo que o sinal carregue o capacitor. Dessa forma, quando a chave deixa de conduzir, esperando a próxima leitura, o capacitor armazena o valor da grandeza analógica que está sendo convertida. O valor de tensão sobre esse capacitor é o sinal de entrada para o quantizador. A taxa de amostragem deve obedecer o critério de Nyquist, ou seja, deve ser maior que o dobro da frequência do sinal analógico. (BRAGA, 2015).

A quantização é o processo de converter os valores instantâneos, que são obtidos na saída do circuito de amostragem e retenção, em valores digitais. Essa parte é a que determina a resolução do ADC, ou seja, a quantidade de valores que podem ser lidos dentro da faixa de conversão. De acordo com Braga (2015), existem diversos circuitos de quantização. O circuito mais simples é o sistema de conversão simultânea, o qual usa comparadores e resistores. A Figura 25 apresenta um exemplo desse circuito.

Nesse circuito, tem-se uma escala de 4 valores possíveis de saída, o que pode ser representado por um sistema de 2 bits. Portanto, trata-se de um ADC de 2 bits. Os comparadores possuem, em suas entradas de referência, tensões escalonadas pelos resistores, as quais determinam os níveis em que eles devem comutar. Dessa forma, há 4 tensões

Figura 25 – Circuito de quantização de 2 bits de resolução.



Fonte: Adaptado de Braga (2015).

escalonadas, de 1/4 a 4/4 de V_{REF}, que é a máxima tensão que o circuito pode medir em sua entrada (BRAGA, 2015).

Junto ao quantizador, é necessário uma matriz codificadora, a partir de portas NOT, portas AND e portas OR, seguindo a lógica *booleana*, para converter em binário os níveis lógicos obtidos nas saídas dos comparadores, já que eles são sequenciais. Essa é a forma mais simples, no entanto, funcional de se fazer um quantizador. Existem outras formas mais complexas, que utilizam essa como base, para se construir um conversor analógico digital (BRAGA, 2015).

Os conversores analógico digitais podem ser encontrados de diversas formas. Existem aqueles que já vêm construídos internamente a um microcontrolador e também os CIs dedicados a essa função. Os ADCs possuem, em geral, uma ou mais entradas analógicas, uma ou mais saídas, entradas para sinal de *clock* e pinos para tensões de referência e alimentação. Eles podem ter as saídas na forma serial ou paralela. A forma serial normalmente utiliza menor quantidade de pinos e necessita de um protocolo para transmissão de dados. Na forma paralela, há mais pinos, cuja quantidade depende de quantos *bits* de conversão possui o ADC. Além disso, os ADCs podem ter entradas para seleção de canal e outras entradas e saídas para configuração e sinalização.

O ADC0809 é um CI específico para fazer conversão de analógico para digital. Ele possui oito *bits* de resolução e oito canais de entrada. Sua saída é na forma paralela, com um pino para cada *bit*, especificados para os níveis de significância do resultado, variando do *bit* menos ao mais significativo. Além disso, o CI possui três *bits* de entrada para seleção de canais e *bits* para ativar o canal escolhido, iniciar a conversão, ativar as saídas

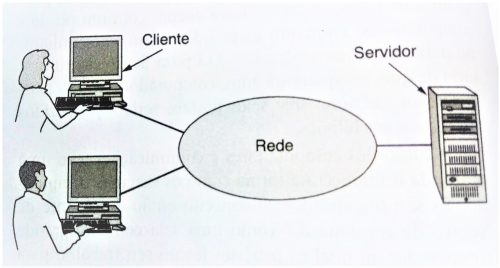
e indicar que o resultado da conversão está pronto para ser lido. O ADC0809 também é composto pelas entradas de tensão de referência e alimentação e do sinal de *clock* (TEXAS INSTRUMENTS, 2013). O Apêndice D trata com mais detalhes sobre este componente.

3.5 REDES DE COMPUTADORES

As redes de computadores estão inseridas na domótica através da computação ubíqua, ou seja, a computação embutida no dia a dia das pessoas. Com ela muitos dispositivos estão interconectados, sendo eles computadores, sensores e atuadores, eletrodomésticos, entre outros. Essa conexão permite realizar processos automáticos, aumenta a interação entre usuários e dispositivos, melhora a eficiência energética, e possui muitas outras vantagens. Dessa forma, domótica e a computação ubíqua estão muito relacionadas nos dias de hoje (TANENBAUM; WETHERALL, 2011).

Um modelo de rede muito utilizado por empresas, mas que pode ser aplicado à automação residencial, é o cliente-servidor. Nesse modelo, os dados são armazenados em computadores com recursos mais avançados, chamados servidores. Outras máquinas mais simples, chamadas clientes, acessam esses dados através da rede. A Figura 26 apresenta uma representação desse modelo (TANENBAUM; WETHERALL, 2011).

Figura 26 – Exemplo de rede com dois clientes e um servidor.



Fonte: TANENBAUM; WETHERALL, 2011.

Na domótica, que está inserida dentro da rede doméstica, são usadas as tecnologias de rede local. Esta é chamada de *Local Area Network* (LAN). Podem ser usadas LANs sem fio, principalmente nos lugares onde a instalação de cabos é muito trabalhosa, mas as LANs com fio oferecem um melhor desempenho em velocidade e segurança. Elas podem ser feitas a partir de fios de cobre ou fibra óptica e são implementadas com base em um padrão, sendo o mais comum o IEEE 802.3, conhecido como Ethernet (TANENBAUM; WETHERALL, 2011).

3.5.1 Arquiteturas de Rede

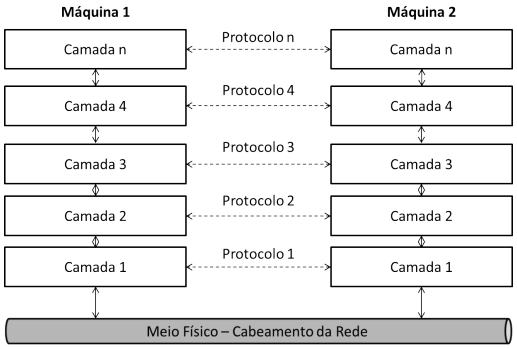
Para diminuir a complexidade do projeto, as redes são organizadas em uma pilha de camadas. Cada uma delas possui um nome, número, conteúdo e função diferentes e

elas diferem de uma rede para outra. Porém, o objetivo de cada camada em todas as redes é oferecer serviços às camadas superiores, isolando essas camadas dos detalhes de implementação (TANENBAUM; WETHERALL, 2011).

A camada n de uma máquina se comunica com a camada n da outra máquina, através de um protocolo que elas têm em comum. No entanto, a informação não é passada diretamente de uma máquina para outra. De acordo com Torres (2009, p. 45), "na transmissão de um dado, cada camada pega as informações passadas pela camada superior, acrescenta informações pelas quais ela seja responsável e passa os dados para a camada imediatamente inferior", processo conhecido como encapsulamento. Na recepção, cada camada n retira as informações que a camada n do transmissor acrescentou.

A Figura 27 ilustra o que foi descrito anteriormente. As setas com linhas contínuas representam o caminho real dos dados, enquanto as tracejadas indicam a comunicação virtual.

Figura 27 – Comunicação entre as camadas de uma arquitetura de rede.



Fonte: Adaptado de Tanenbaum e Wetherall (2011).

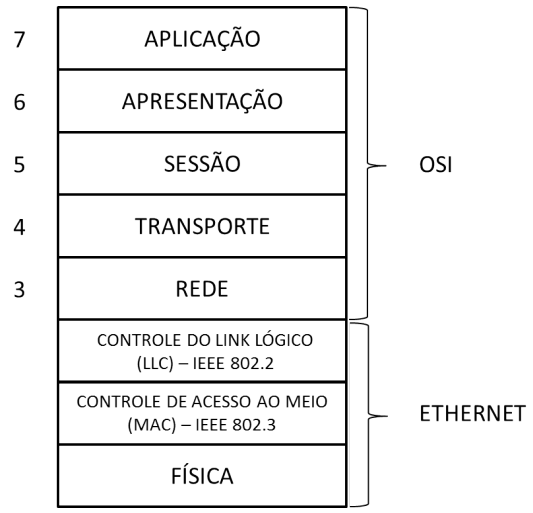
Pode-se dizer que cada camada possui um protocolo. Os protocolos podem ser divididos em dois grupos principais. Os de baixo nível são aqueles que cuidam da comunicação física da rede, por exemplo, protocolos Ethernet e WiFi, mas existem outros. Os que estão acima deles são os de alto nível, que tratam as informações antes de serem enviadas. O exemplo mais comum é o conjunto de protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol* - Protocolo de Controle de Transmissão/Protocolo Internet), mas existem outros (TORRES, 2009).

Para facilitar a interconexão de sistemas de computadores, a ISO (*International Organization for Standardization* - Organização Internacional de Normalização) desenvolveu um modelo de referência chamado OSI (*Open Systems Interconnection* - Interconexão de Sistemas Abertos). Dessa forma, os fabricantes passaram a ter um modelo para seguir na hora de criar seus protocolos (TORRES, 2009). O modelo OSI é detalhado no Apêndice E.

3.5.2 Protocolo Ethernet

O protocolo de baixo nível mais utilizado em redes locais é o Ethernet. Ele opera nas camadas um e dois do modelo OSI, a fim de definir a parte física da rede. A camada dois do modelo OSI (Link de Dados) é dividida em duas subcamadas no padrão Ethernet: uma inferior, chamada Controle de Acesso ao Meio (MAC - *Media Access Control*), e uma superior, denominada Controle do *Link* Lógico (LLC - *Logic Link Control*). A subcamada MAC segue o padrão IEEE 802.3 e a LLC segue o IEEE 802.2. A Figura 28 relaciona o padrão Ethernet ao modelo OSI (TORRES, 2009).

Figura 28 – O padrão Ethernet e o modelo OSI.



Fonte: Adaptado de Torres (2009).

Por estar nas camadas de baixo nível, a função do Ethernet é, de acordo com Torres (2009), inserir os dados entregues pelos protocolos de alto nível dentro de quadros que serão enviados através da rede, chamados de quadros Ethernet, além de definir como os dados vão ser transmitidos fisicamente, por exemplo, o formato do sinal.

A seguir, serão descritas as funções das camadas Ethernet.

3.5.2.1 Controle do Link Lógico (LLC, IEEE 802.2)

A camada de LLC tem a função de receber os dados repassados por algum protocolo de alto nível e acrescentar a informação de qual deles foi o responsável por gerar os dados. Dessa forma, a máquina receptora saberá para qual protocolo de alto nível devem ser entregues os pacotes de dados recebidos (TORRES, 2009).

De acordo com Torres (2009), esta camada acrescenta um cabeçalho de oito bytes aos dados vindos da camada superior, sendo três para o cabeçalho LLC e cinco para o cabeçalho *Sub Network Access Protocol* (SNAP - Protocolo de Acesso à Sub-Rede), divididos da seguinte maneira:

- **DSAP (*Destination Service Access Point* - Ponto de Acesso do Serviço de Destino) [1 byte]:** indica o protocolo de destino. Caso o cabeçalho SNAP esteja sendo usado, é atribuído o valor binário 10101010 ao DSAP.

- **SSAP (*Source Service Access Point* - Ponto de Acesso do Serviço de Origem) [1 byte]:** indica o protocolo de origem. Caso o cabeçalho SNAP esteja sendo usado, também é atribuído o valor binário 10101010 ao SSAP.

- **Controle [1 byte]:** Pode assumir três valores:

  1. UI (*Unnumbered Information* - Informação não-numerada): usado na transmissão de dados;
  2. XID (*eXchange IDentification* - Identificação de Troca): usado para que o transmissor e o receptor troquem dados de identificação entre si. Este valor pode ser um comando, ou seja, quando o transmissor envia sua identidade e pede a do receptor, ou uma resposta, que é quando o receptor envia sua identidade ao transmissor, obedecendo ao comando;
  3. Teste: transmissor envia um dado e o receptor o devolve, a fim de testar a comunicação.

- **Código [3 bytes]:** Estes bytes fazem parte do campo SNAP e são baseados no padrão IEEE para protocolos. O campo Código é usado para identificar o fabricante do protocolo no IEEE.

- **Tipo [2 bytes]:** Estes bytes também fazem parte do campo SNAP. O campo Tipo é o código dado pelo fabricante ao protocolo.

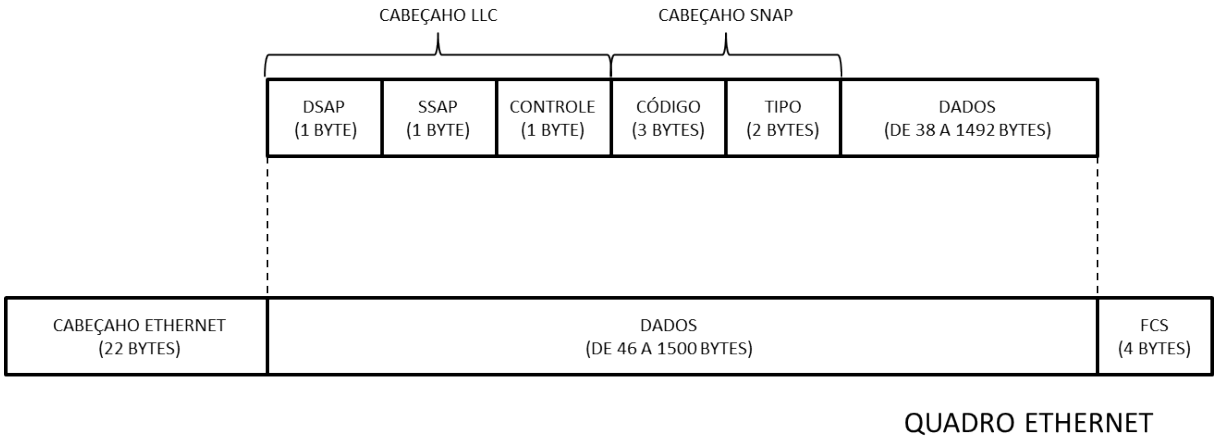
A Figura 29 ilustra como esses cabeçalhos são adicionados ao quadro Ethernet. Os campos do quadro Ethernet serão detalhados na subseção 3.5.2.2.

3.5.2.2 Controle de Acesso ao Meio (MAC, IEEE 802.3)

A camada de MAC tem a função de gerar o quadro Ethernet, acrescentando um cabeçalho aos dados recebidos da camada de LLC, o qual contém as informações de quem está enviando e quem deve receber o quadro. Em seguida, essa camada envia o quadro para a camada Física, que irá fazer a transmissão desse quadro para o cabeamento da rede (TORRES, 2009).

Além disso, a camada de MAC também tem a responsabilidade de verificar se o meio físico está livre para uso, através de um protocolo chamado CSMA/CD (*Carrier Sense Multiple Access with Collision Detection* - Acesso Múltiplo e Sensoreamento de Portadora com Detecção de Colisão) (TORRES, 2009).

Figura 29 – Estrutura do quadro Ethernet com a camada de LLC.



Fonte: Adaptado de Torres (2009).

Como todas as máquinas estão ligadas no mesmo cabo, é necessário que apenas uma delas transmita por vez. Basicamente, o protocolo CSMA/CD verifica se há alguma portadora de transmissão (*Carrier Sense*) no cabo e, caso não tenha, significa que ele está livre para uso. Dessa forma, a informação pode ser transmitida (TORRES, 2009).

No entanto, este é um protocolo de múltiplo acesso (*Multiple Access*), o que quer dizer que ele não gera prioridade. Assim, mais de uma máquina pode verificar o cabo ao mesmo tempo, perceber que ele está livre e tentar fazer uma transmissão. Quando isso ocorre, há uma colisão e nenhuma das máquinas consegue transmitir os dados (TORRES, 2009).

Ao ocorrer uma colisão, todas as placas de rede param de transmitir, esperam um intervalo de tempo aleatório e tentam retransmissão, que provavelmente será possível, já que cada transmissor vai esperar um tempo diferente. Ocorrem, no máximo, 16 tentativas e, caso não consiga enviar ainda, é gerada uma mensagem de erro (TORRES, 2009).

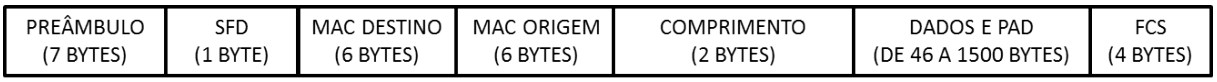
Quando um quadro é transmitido, todas as placas de rede ligadas àquele fio recebem as informações. Por isso, cada placa possui um endereço MAC diferente, gravado em uma memória ROM (*Read Only Memory* - Memória Somente Leitura), para que possa ser enviado pelo quadro Ethernet o endereço da placa de destino. Assim, somente a placa correta irá pegar os dados, enquanto as outras vão ignorá-los (TORRES, 2009).

Teoricamente, nenhum endereço MAC se repete no mundo, graças à padronização feita pelo IEEE. Esse endereço é composto de seis bytes: os três primeiros identificam o fabricante da placa e os três últimos são atribuidos pelo próprio fabricante (TANENBAUM; WETHERALL, 2011).

A camada de Controle de Acesso ao Meio utiliza os endereços MAC das placas envolvidas na transmissão para gerar o quadro Ethernet. Além disso, outros campos são

inseridos no quadro, como mostra a Figura 30 (TORRES, 2009).

Figura 30 – Estrutura do quadro Ethernet.



Fonte: Adaptado de Torres (2009).

De acordo com Torres (2009), Tanenbaum e Wetherall (2011), as funções de cada quadro são as seguintes:

- **Preâmbulo:** marca o início do quadro, com sete bytes 10101010. Juntamente com o SFD, forma um padrão de sincronismo, para que o receptor saiba estar diante do início de um quadro.
- **SFD (*Start of Frame Delimiter* - Delimitador de Início de Quadro):** é um byte 10101011, para indicar que o preâmbulo terminou e a informação começará a ser passada.
- **Endereço MAC de destino:** Endereço de seis bytes que identifica a placa que deve receber a transmissão.
- **Endereço MAC de origem:** Número de seis bytes que indica a placa que está enviando o sinal.
- **Comprimento:** informa quantos bytes há no campo de dados, já que essa quantidade é variável. Para isso, são necessários dois bytes.
- **Dados:** campo que contém os dados enviados pela camada de LLC. Além dos dados vindos das camadas de alto nível, contém os cabeçalhos de controle da camada de LLC. Esse campo deve ter, no mínimo, 46 bytes e, no máximo, 1500 bytes.
- **Pad ou Preenchimento:** caso a camada de LLC envie menos de 46 bytes de dados para a camada MAC, devem ser inseridos dados chamados pad para preencher o campo de dados, de forma que ele atinja seu número mínimo de bytes.
- **FCS (*Frame Check Sequence* - Sequência de Verificação de Quadros):** possui informações para o controle de correção de erros, contendo quatro bytes.

3.5.2.3 Camada Física

A camada física da arquitetura Ethernet é a que trata da transmissão dos quadros gerados pela camada de MAC. Ela é responsável por codificar os bits em sinais elétricos para a transmissão. Além disso, ela determina que tipo de cabo será utilizado: coaxial, par trançado ou fibra óptica (TORRES, 2009).

Fazem parte da camada física as placas de rede. De acordo com o modelo da placa, determina-se qual conector e cabo deve ser utilizado. No projeto dessa camada, deve-se pensar: o tipo de cabeamento; os tipos de comunicação, ou seja, se será no modo *half-duplex*, ou seja, com fluxo de dados unidirecional e o envio e recebimento não sendo realizados ao mesmo tempo, ou *full-duplex*, no qual o fluxo de dados é bidirecional e o envio e recebimento pode ser feito ao mesmo tempo; se a topologia será linear ou em estrela; e qual será a taxa de transferência e a codificação utilizada (TORRES, 2009).

De acordo com Torres (2009), para cada uma das quatro taxas de transferência máximas utilizada, é usado um padrão de codificação diferente:

- **10 Mbps (Ethernet padrão):** Codificação Manchester para todos os tipos de cabos.
- **100 Mbps (*Fast* Ethernet):** Codificação 4B/5B para todos os tipos de cabos.
- **1 Gbps (Gigabit Ethernet):** Codificação 4D-PAM-5 em cabos do tipo par trançado e codificação 8B/10B em fibras ópticas.
- **10 Gbps (10G Ethernet):** Codificação 64B/66B com fibras ópticas e DSQ128/PAM-16 com uso de cabos do tipo par trançado.

3.5.3 Protocolo TCP/IP

O protocolo Ethernet trata as camadas um e dois do modelo OSI. As camadas três a sete necessitam de protocolos de alto nível, os quais são implementados em *software*. Para isso, o protocolo mais utilizado no mundo é o TCP/IP, que, na verdade, é uma pilha de protocolos, sendo que cada protocolo atua em uma de suas quatro camadas, que são baseadas no modelo OSI (TORRES, 2009). A Figura 31 compara a arquitetura do TCP/IP com o modelo OSI.

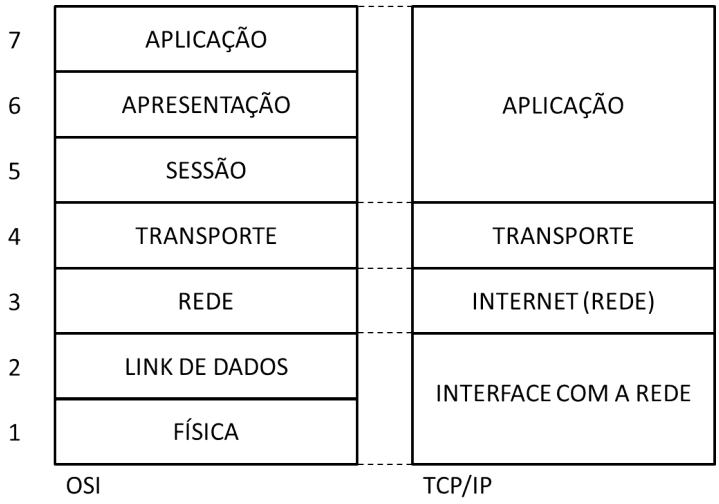
De acordo com Torres (2009), a camada de Interface com a Rede define-se pela arquitetura de rede de baixo nível sendo usada. O modelo utilizado é o padrão Ethernet, apresentado na Subseção 3.5.2.

A seguir, serão descritas as outras camadas do modelo TCP/IP.

3.5.3.1 Camada de Aplicação do TCP/IP.

A camada de aplicação contém todos os protocolos de nível mais alto. Dentre eles, estão o protocolo de transferência de arquivos (FTP - *File Transfer Protocol* - Protocolo de Transferência de Arquivos), o protocolo de terminal virtual (TELNET) e o protocolo de correio eletrônico (SMTP - *Simple Mail Transfer Protocol* - Protocolo de transferência de correio simples) (TANENBAUM; WETHERALL, 2011).

Figura 31 – Arquitetura do TCP/IP.



Fonte: Adaptado de Torres (2009).

Com o passar dos anos, outros protocolos foram incluídos, tais como, o DNS (*Domain Name Service* - Serviço de Nome de Domínio), que mapeia os nomes dos *hosts* para seus endereços da camada de rede, o HTTP (*Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto), que é usado para buscar páginas na *World Wide Web*, e o RTP (*Real-Time Transport Protocol* - Protocolo de Transporte em Tempo Real), para enviar mídia em tempo real (TANENBAUM; WETHERALL, 2011).

As camadas de Apresentação e Sessão do modelo OSI não estão presentes na arquitetura TCP/IP, pois os protocolos da camada de Aplicação tratam o que deveria ser feito nessas outras camadas (TANENBAUM; WETHERALL, 2011).

3.5.3.2 Camada de Transporte do TCP/IP.

Esta camada tem por objetivo permitir que as máquinas de origem e destino mantenham uma conversação. Para isso, foram definidos dois protocolos. O primeiro, é o protocolo de controle de transmissão, conhecido por TCP. Ele permite a entrega sem erros de um fluxo de bytes vindos de uma determinada máquina. O TCP fragmenta esses bytes em mensagens discretas e as passa para a camada Internet. No receptor, ele monta as mensagens recebidas. Além disso, ele cuida do controle de fluxo entre um transmissor rápido e um receptor lento (TANENBAUM; WETHERALL, 2011).

O segundo protocolo, chamado de protocolo de datagrama do usuário, ou UDP (*User Datagram Protocol* - Protocolo de datagrama do usuário), não possui conexões. É utilizado para aplicações que não necessitam da sequência ou do controle de fluxo do TCP, mas desejam oferecer seu próprio controle. É muito utilizado para consultas isoladas, com solicitação e resposta, do tipo cliente-servidor, e também em aplicações onde é necessária uma entrega imediata (TANENBAUM; WETHERALL, 2011).

3.5.3.3 Camada Internet (Rede) do TCP/IP.

O termo “internet”, nesse caso, significa “rede interligada”. Portanto, essa camada integra toda a arquitetura, mantendo-a unida. Sua função é permitir que as máquinas mandem pacotes para qualquer rede e garantir que eles chegarão ao destino correto (TANENBAUM; WETHERALL, 2011).

Essa camada define um formato de pacote oficial e um protocolo chamado IP, e mais um protocolo que o acompanha, denominado ICMP (*Internet Control Message Protocol* - Protocolo de Mensagens de Controle de Internet). A tarefa da camada internet, conforme afirmam Tanenbaum e Wetherall (2011, p. 29), se resume a “entregar pacotes IP onde eles são necessários”.

3.6 IMPLEMENTAÇÃO DO TCP/IP EM C

O protocolo TCP/IP pode ser implementado em diversas linguagens de programação. Na linguagem C, uma das mais conhecidas por profissionais da eletrônica devido ao seu uso em microcontroladores, existem algumas bibliotecas para auxiliar na implementação de protocolos de rede. No caso de TCP/IP, a comunicação é feita através de *sockets*.

De acordo com Tanenbaum e Wetherall (2011), um *socket* fornece um ponto de comunicação por meio de redes de computadores. Dessa forma, para que duas máquinas troquem informações via rede, é necessário um par de *sockets*. Cada *socket* é identificado pelo endereço IP e porta de rede (por exemplo, 192.168.0.1:80).

Toda conexão em um determinado *host* de rede deve ser única e isto é obtido através do uso das portas. Assim, cada conexão de rede ativa tem uma porta única, onde dois ou mais processos em execução não podem receber o mesmo número de porta, já que esta é utilizada para identificar qual processo deve receber a informação enviada via rede (TANENBAUM; WETHERALL, 2011).

Dentre os vários tipos de *sockets*, o utilizado pelo protocolo TCP/IP é o “SOCK STREAM”, no qual os pacotes são enviados de forma sequencial e seguem em duas vias, permitindo ler e gravar (LEON, 2018).

Para se declarar um *socket*, o processo é feito como qualquer variável, pois ele é do tipo `int`. Em seguida, defini-se uma estrutura usada para conexões na internet, chamada `sockaddr_in`, a qual possui família do endereço, número da porta, IP do *host* e uma variável para zerar a estrutura (LEON, 2018). A família mais usada é a “AF_INET” (LEON, 2018).

A função `socket` é utilizada para construir os *sockets*, recebendo como parâmetros a família, o tipo de *socket* e o número do protocolo utilizado, que para o *Internet Protocol* é 0. A conexão é estabelecida através da função `connect`, que recebe por parâmetro o

socket criado, sua estrutura e seu tamanho (LEON, 2018).

O que foi descrito até aqui é a implementação do cliente. Para o servidor, a metodologia é semelhante, mas não se utiliza a função `connect`, pois quem abre a conexão é sempre o cliente. Diferentemente do cliente, o servidor possui a função `bind`, que serve para associar uma porta da máquina local a um *socket*. Os parâmetros são o *socket*, a estrutura e o seu tamanho (LEON, 2018).

A função `bind` é utilizada junto com a função `listen`. Esta última tem a tarefa esperar uma determinada conexão de um *socket*, sendo utilizada para setar o número de conexões que serão permitidas para determinado serviço. Os parâmetros são o *socket* e o número de conexões que serão permitidas para o serviço, geralmente de 5 a 10. As conexões futuras ficarão esperando até que o servidor aceite, o que é realizado pela função `accept` (LEON, 2018). Para permitir mais de uma conexão, se faz necessário declarar outro *socket*, que será responsável pelas futuras requisições (LEON, 2018).

De acordo com Leon (2018) função `accept` recebe por parâmetro o *socket* do servidor, o *socket* do cliente e o tamanho do *socket* do cliente.

Para enviar e receber dados, utilizam-se as funções `send` e `recv`, respectivamente. A função `send` recebe o *socket* local, um ponteiro para a mensagem a ser enviada, o tamanho da mensagem e umas *flags*, que são parâmetros adicionais opcionais. A função `recv` recebe os mesmos parâmetros, exceto pelo *buffer* da mensagem, que é o local onde será armazenada a mensagem recebida (LEON, 2018).

Para esperar a conexão de vários clientes, o servidor pode ser escrito utilizando *threads*, de forma que cada nova conexão aceitada cria uma *thread* para tratá-la, enquanto a principal fica aguardando novas conexões.

3.7 BARRAMENTOS

No projeto de automação residencial, são necessários barramentos para comunicar os controladores, quando há mais de um, e para interligar os sensores e atuadores ao sistema. A forma mais comum de fazer a comunicação por padrão Ethernet é com cabo de par trançado. Para enviar e receber os sinais dos elementos, é possível fazer com cabo *flat*, o qual muitos dispositivos controladores possuem o conector necessário ligado às portas GPIO. No entanto, para agrupar esses os cabos e fazê-los servir no conector, se faz necessário alguma interface, que pode ser feita com trilhas em placas de circuito impresso (PCI). Finalmente, por se tratar de uma residência, são necessárias as instalações elétricas.

A seguir, são apresentados esses tipos de barramentos.

3.7.1 Par Trançado

O cabo de par trançado é composto por pares de fios, que são enrolados em espiral para reduzir o ruído e manter constantes as propriedades elétricas do meio por toda a sua extensão, através do efeito de cancelamento. Existem os cabos que possuem blindagem especial (STP - *Shielded Twisted Pair* - Par Trançado Blindado) e os que não possuem (UTP - *Unshielded Twisted Pair* - Par Trançado Não-Blindado) (NOGUEIRA; MELCHIORS, 1996).

Os cabos STP possuem uma malha blindada global e outra interna envolvendo cada par trançado. Essa blindagem deixa o cabo com maior imunidade às interferências externas, como ondas eletromagnéticas e radiofrequência. No entanto, possuem a desvantagem de serem mais espessos, dificultando as instalações que requerem vários cabos passando pelo mesmo eletroduto (NOGUEIRA; MELCHIORS, 1996).

Os cabos UTP não apresentam tanta imunidade quanto aos STP, mas conseguem demonstrar bom desempenho. Além disso, são mais baratos e fáceis de serem instalados. Se o local onde for utilizado não houver tanta interferência ou a aplicação não necessitar de tanta precisão, esses cabos são bem adequados (NOGUEIRA; MELCHIORS, 1996).

3.7.2 GPIO

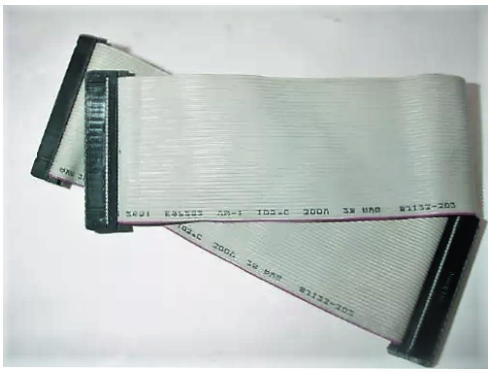
Os cabos *flat* são conectados a vários pinos de propósito geral, os GPIOs. Enquanto alguns pinos fornecem tensões fixas ou são de aterramento, outros podem fornecer ou receber níveis lógicos alto ou baixo, dependendo do que foi projetado. Dessa forma, esse tipo de barramento pode ser usado como saída para acionar os atuadores e como entrada para os sinais dos sensores (TERASIC TECNOLOGIES INC., 2017).

Por conter vários pinos, é possível empregar GPIO para interligar diversos elementos, podendo ser utilizado parte dos pinos como entrada e outra parte como saída. No kit DE10-Nano, há disponível dois conectores de 40 pinos para serem usados como GPIO. Na Figura 7, na subseção 3.1.5, os conectores do kit são apresentados (TERASIC TECNOLOGIES INC., 2017). A Figura 32 mostra uma foto do cabo *flat* de 40 pinos.

3.7.3 Trilhas

As trilhas são os barramentos presentes nas placas de circuito impresso, em geral, feitas com cobre. As PCIs, por sua vez, são placas que recebem um circuito impresso, previamente desenhado em um *software* adequado, através de algum processo de transferência, normalmente térmica ou por fotografia. São fabricadas com cobre em toda a sua superfície, mas, após o processo de corrosão, fica metal apenas onde havia trilhas desenhadas. As placas mais utilizadas são as de fenolite e as de fibra de vidro (MEHL, 2011).

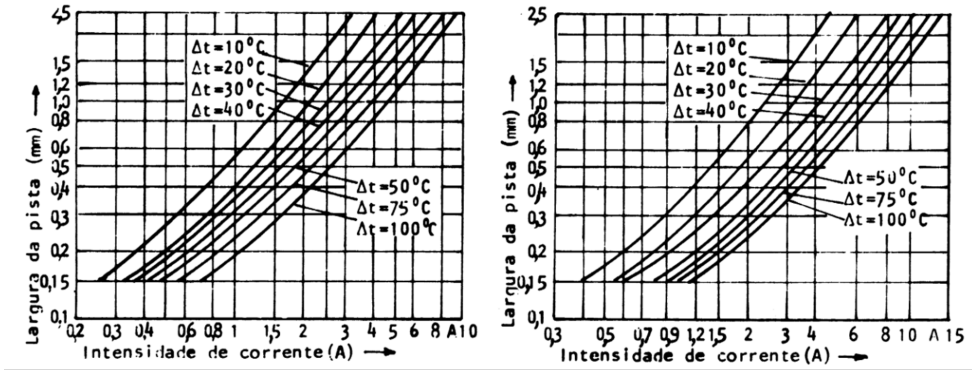
Figura 32 – Cabo *flat* de 40 pinos.



Fonte: Autoria Própria.

Independente do procedimento de transferência utilizado, as trilhas precisam seguir algumas padrões de dimensões, como espessura e espaçamento, o que depende da corrente, tensão, espessura do cobre e temperatura no local de cada trilha. A norma ABNT NBR 8188/89 apresenta um ábaco com as dimensões que devem ser utilizadas. A Figura 33 mostra esse gráfico para as espessuras, enquanto a Figura 34 traz o ábaco para o espaçamento entre as trilhas (MEHL, 2011).

Figura 33 – Gráficos existentes na NBR 8188/89, para dimensionamento da corrente máxima que pode fluir nas trilhas de um circuito impresso.



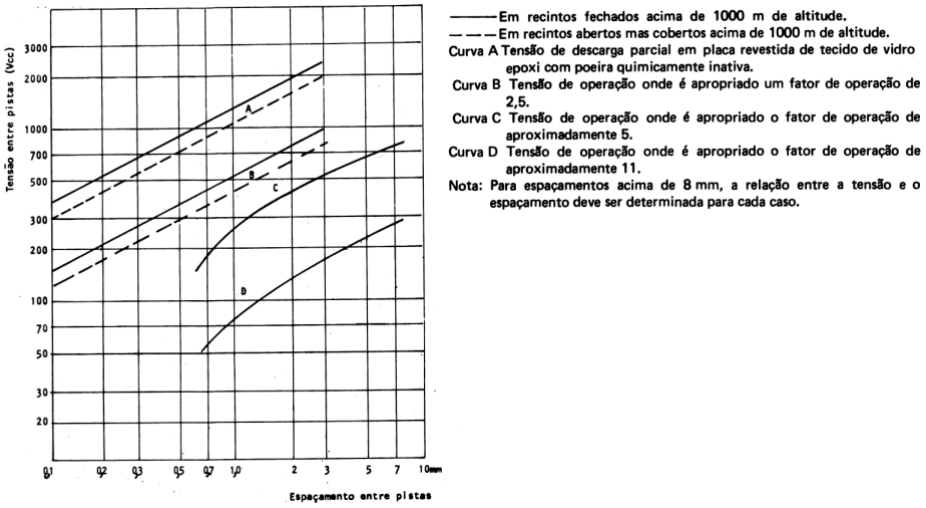
Fonte: Adaptado de Mehl (2011).

As medidas podem ser representadas em milímetros (mm) ou milésimos de polegada (mil), onde 1mil$= 0,0254$mm. É muito comum a utilização das medidas imperiais (polegadas) nos *softwares*, pois normalmente são desenvolvidos por empresas americanas.

3.7.4 Instalações Elétricas

Existem diversos critérios para fazer o dimensionamento de cabos para instalação elétricas de uma residência de forma segura. A norma ABNT NBR 5410 trata especificamente sobre isso. Um dos principais critérios é a dimensão da espessura dos condutores. A

Figura 34 – Gráfico que permite estabelecer o espaçamento entre duas trilhas contíguas, em função da tensão existente entre elas.



Fonte: Adaptado de Mehl (2011).

tabela 47 apresentada na norma informa quais são os critérios para o dimensionamento, aqui ilustrada pela Figura 35.

Apesar da tabela apresentar as dimensões em função do tipo de circuito, isso é devido à corrente que normalmente são utilizadas nesses locais (ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2004).

Figura 35 – Dimensões para seções mínimas dos condutores em uma instalação elétrica.

Tabela 47 — Seção mínima dos condutores[1)]

Tipo de linha		Utilização do circuito	Seção mínima do condutor mm² - material
Instalações fixas em geral	Condutores e cabos isolados	Circuitos de iluminação	1,5 Cu 16 Al
		Circuitos de força[2)]	2,5 Cu 16 Al
		Circuitos de sinalização e circuitos de controle	0,5 Cu[3)]
	Condutores nus	Circuitos de força	10Cu 16 Al
		Circuitos de sinalização e circuitos de controle	4 Cu
Linhas flexíveis com cabos isolados		Para um equipamento específico	Como especificado na norma do equipamento
		Para qualquer outra aplicação	0,75 Cu[4)]
		Circuitos a extrabaixa tensão para aplicações especiais	0,75 Cu

[1)] Seções mínimas ditadas por razões mecânicas
[2)] Os circuitos de tomadas de corrente são considerados circuitos de força.
[3)] Em circuitos de sinalização e controle destinados a equipamentos eletrônicos é admitida uma seção mínima de 0,1 mm².
[4)] Em cabos multipolares flexíveis contendo sete ou mais veias é admitida uma seção mínima de 0,1 mm².

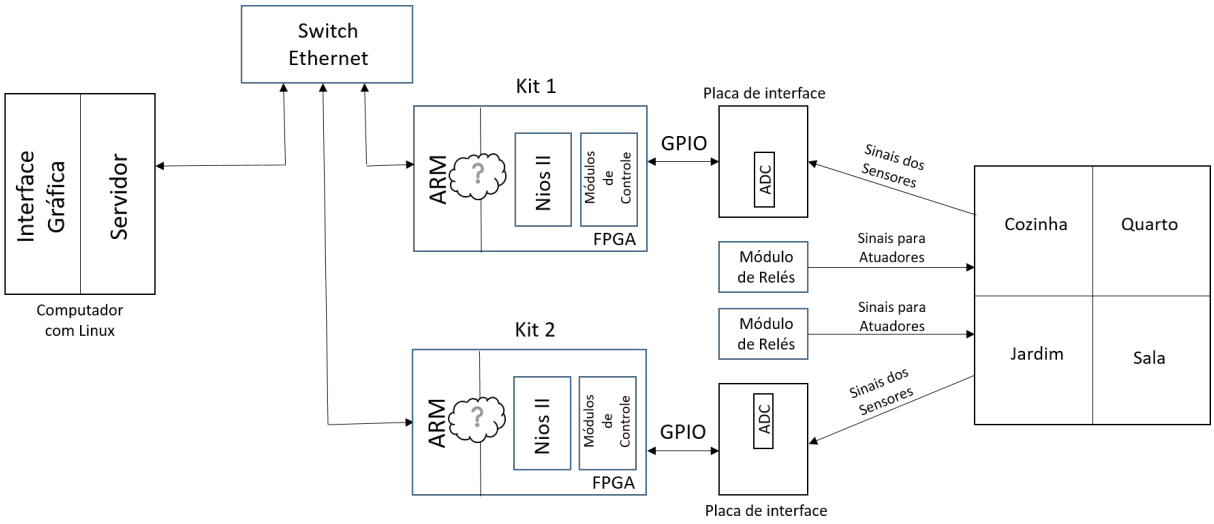
Fonte: ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2004.

4 METODOLOGIA

4.1 DIAGRAMA DE BLOCOS

A fim de deixar explícito do que se trata o projeto, é apresentado um diagrama de blocos do sistema completo, conforme Figura 36. O sistema possui uma interface gráfica em uma das pontas e uma maquete de uma casa na outra.

Figura 36 – Diagrama de blocos geral.



Fonte: Autoria Própria.

O sistema é dividido em duas plantas, sendo responsabilidade da planta 1 tratar do quarto e da cozinha, enquanto a planta 2 controla os outros dois cômodos. A maquete também contém sensores de presença na sala e cozinha, e sensores de luminosidade e temperatura em todos os cômodos. Como atuadores, há uma lâmpada e uma tomada para ligar ventilador em todos os cômodos, exceto jardim, o qual possui uma fita de LED e uma sirene de alarme.

No quarto, o usuário tem a possibilidade de manipular o estado do ventilador e da lâmpada de duas formas distintas: através de uma opção na interface gráfica ou por um interruptor, o qual alterna o estado atual, enquanto, no *software*, há a possibilidade de escolher entre ligado ou desligado.

Na sala e na cozinha, as opções aumentam. Além das possibilidades presentes no quarto, o usuário pode acionar o modo automático. Neste modo, as lâmpadas irão acender apenas se tiver presença detectada pelo sensor e o ventilador irá ligar se tiver presença e a temperatura estiver acima do limite superior configurado na interface gráfica. Na interface, o usuário estabelece a temperatura na qual deseja que o ventilador ligue (limite superior)

e que ele desligue (limite inferior).

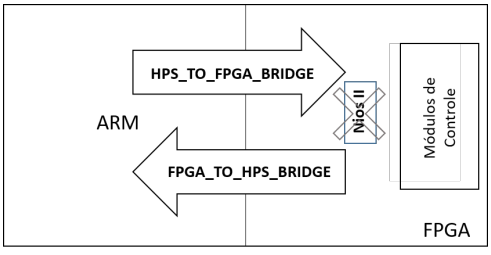
O jardim possui um sistema de iluminação e outro de alarme. O sistema de iluminação possui a opção manual e a automática. A manual é semelhante às lâmpadas dos outros cômodos. A opção automática depende do sensor de luminosidade e das configurações de usuário. Se a luminosidade estiver abaixo do limite inferior, o LED acende. Caso esteja acima do limite superior, o LED apaga.

O alarme possui apenas uma sirene como atuador, para oferecer segurança contra invasão e incêndio. O usuário tem a possibilidade de acionar ou desligar, tanto por *software*, como por *hardware*. Na segunda opção, diferentemente dos outros atuadores da casa, o interruptor não alterna o estado, mas valida uma senha colocada em chaves de duas posições, formando uma combinação com números binários, com as opções de ligar e desligar. Para a proteção contra incêndios, o alarme não precisa estar acionado, ele irá disparar sempre que algum dos sensores de temperatura da casa detectar mais que 50 graus Celsius. Mas se ele estiver acionado, o disparo irá ocorrer também quando algum sensor de presença detectar movimento.

Os kits de FPGA trocam informações através do servidor, por meio do protocolo Ethernet e TCP/IP. Além disso, a interface gráfica também usa essa rede para se comunicar com as FPGAs. Cada FPGA contém um bloco de módulo de controle e outro do Nios II, para auxiliar na comunicação. Elas enviam sinais para a casa e recebem informações de lá através das placas de interface e módulos de relés.

O sistema ainda não está completo neste trabalho. Como está indicado no diagrama, o kit recebe as informações da rede através do ARM. No entanto, é necessário haver uma comunicação entre o ARM e a FPGA, que pode ser implementadas de várias formas, sendo a mais indicada o uso das pontes internas, como apresenta a Figura 37. Com essa implementação, o Nios II poderia ser retirado do projeto e somente o ARM faria o papel de intermediador.

Figura 37 – Bloco para tornar o sistema completo.



Fonte: Autoria Própria.

4.2 MÓDULOS DE CONTROLE

Os módulos de controle foram implementados totalmente em descrição *hardware*, utilizando VHDL para e construção da lógica e Verilog para a interligação entre os sinais do Nios II, do módulo de controle e das saídas. O projeto foi feito baseado no kit de desenvolvimento DE10-Nano.

As lógicas de controle foram desenvolvidas em dois projetos separados, um para a FPGA1 e o outro para a FPGA2. Após a síntese, o diagrama de blocos do *hardware* contido na FPGA1 foi o apresentado na Figura 38.

Como é possível observar, há dois módulos maiores, chamados de `MODULE_FPGA1` e `NIOS_FPGA1`. O Nios é o bloco criado pelo Qsys. Ele contém o Nios II propriamente dito, memória e as portas PIO, que são as que se comunicam com a GPIO e com o módulo de controle.

Além disso, há um bloco chamado `RELOGIO`, que serve para fornecer o *clock* de 640 kHz para o ADC. A partir do *clock* de 50 MHz que tem interno ao kit DE10-Nano, um novo *clock* é gerado, alternando-se os pulsos de uma saída conforme um contador é incrementado até um certo valor.

Os outros 4 blocos, chamados `REPIQUE`, são para fazer o tratamento de *debouncing* dos interruptores. Eles comparam o sinal da entrada e da saída. Se elas forem diferentes, começam incrementar um contador, que conta a cada ciclo de *clock*. Se a saída permanecer diferente da entrada após esse tempo, o bloco trava a saída com o valor recebido da entrada, pois o *bouncing* ja terminou. As entradas do sistema que dependem do interruptor recebem esse sinal tratado.

Com a Figura 39, é possível ver quais sinais entram e saem dos sub-módulos de controle de cada cômodo. e os sub-sistemas que controlam cada um deles.

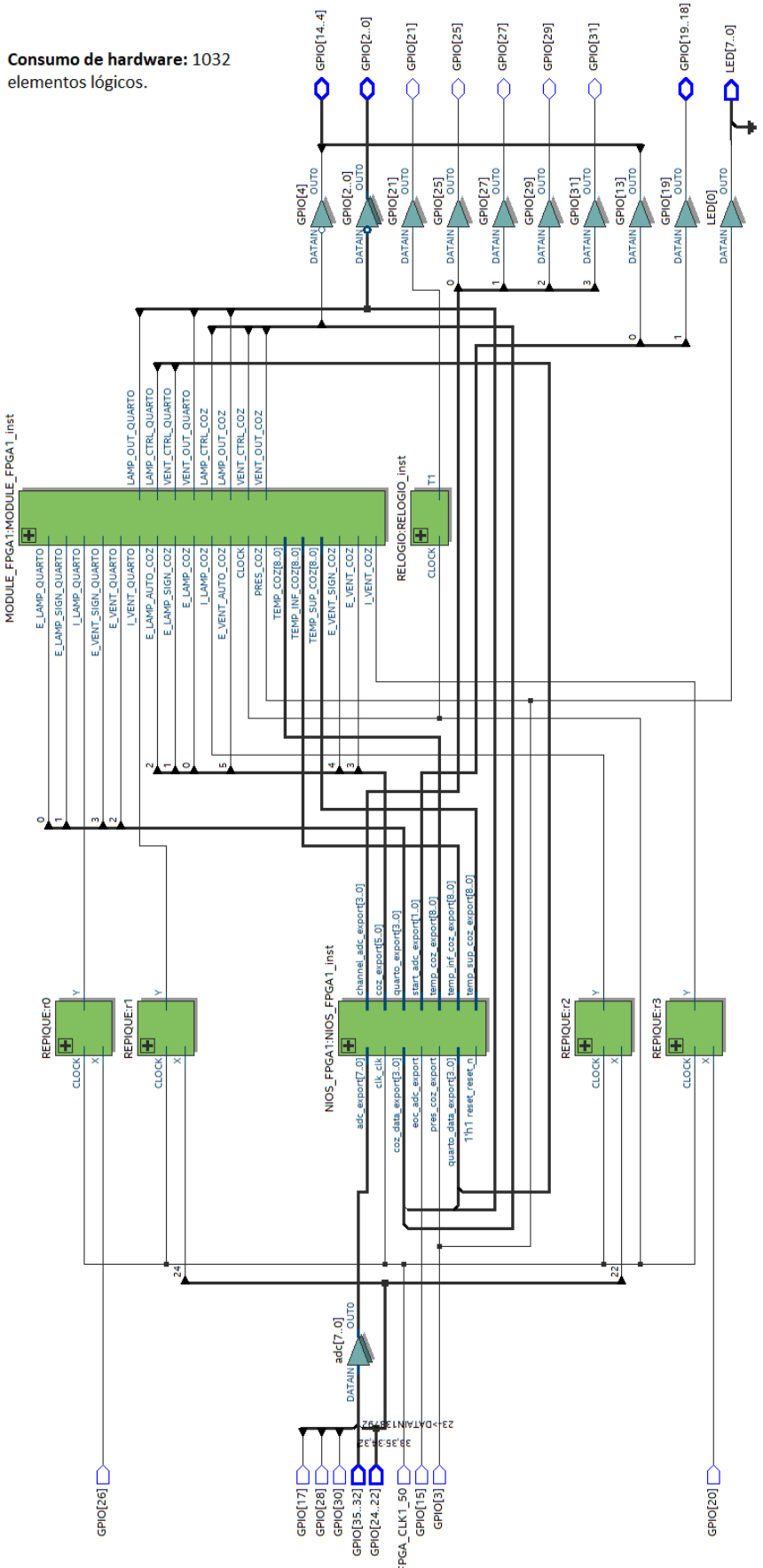
Na FPGA2, o diagrama de *hardware* é bem semelhante, conforme apresenta a Figura 40, aumentando um pouco a quantidade de sinais. Isso é por causa do alarme, que utiliza também as temperaturas de todos os cômodos para fazer o controle. Os outros blocos apresentam o mesmo funcionamento e implementação já descritos.

Apesar da imagem parecer complexa, se analisar cada bloco separadamente o entendimento torna-se mais simples, pois são vários blocos com lógicas mais simples que se repetem, como foi descrito para a FPGA1.

A FPGA2 é responsável pelo jardim e pela sala, como é possível ver na Figura 41. Além disso, pode-se atentar que a sala possui os mesmos blocos que a cozinha, pois a lógica de tratamento é igual. Já o jardim, além dos sistema automático, há também um bloco responsável pelo alarme.

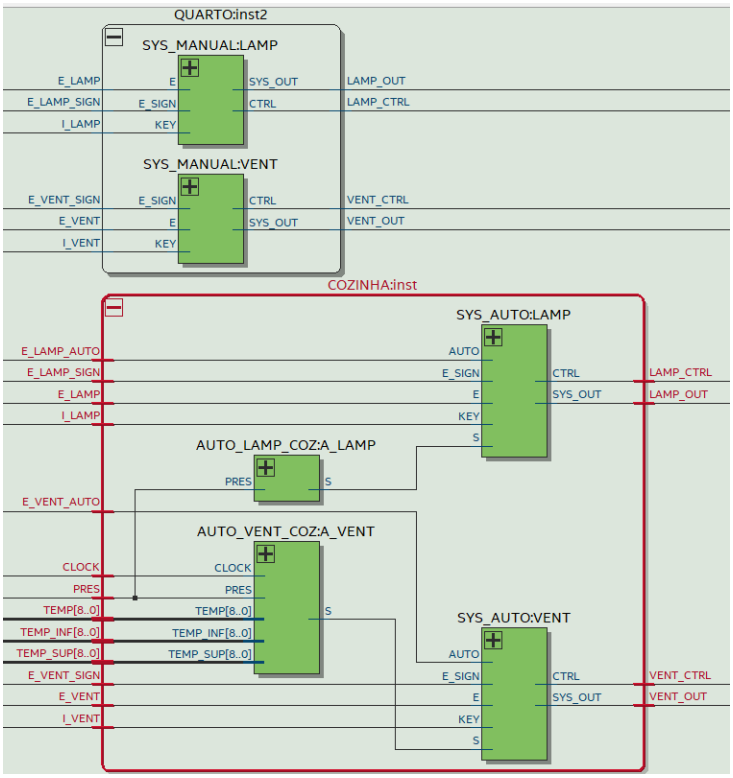
É possível observar que cada atuador é composto por módulos semelhantes.

Figura 38 – Visão geral do sistema implementado na FPGA1.



Fonte: Autoria Própria.

Figura 39 – Módulo da parte de *hardware* da FPGA1, contendo os sub-módulos de controle do quarto e da cozinha.



Fonte: Autoria Própria.

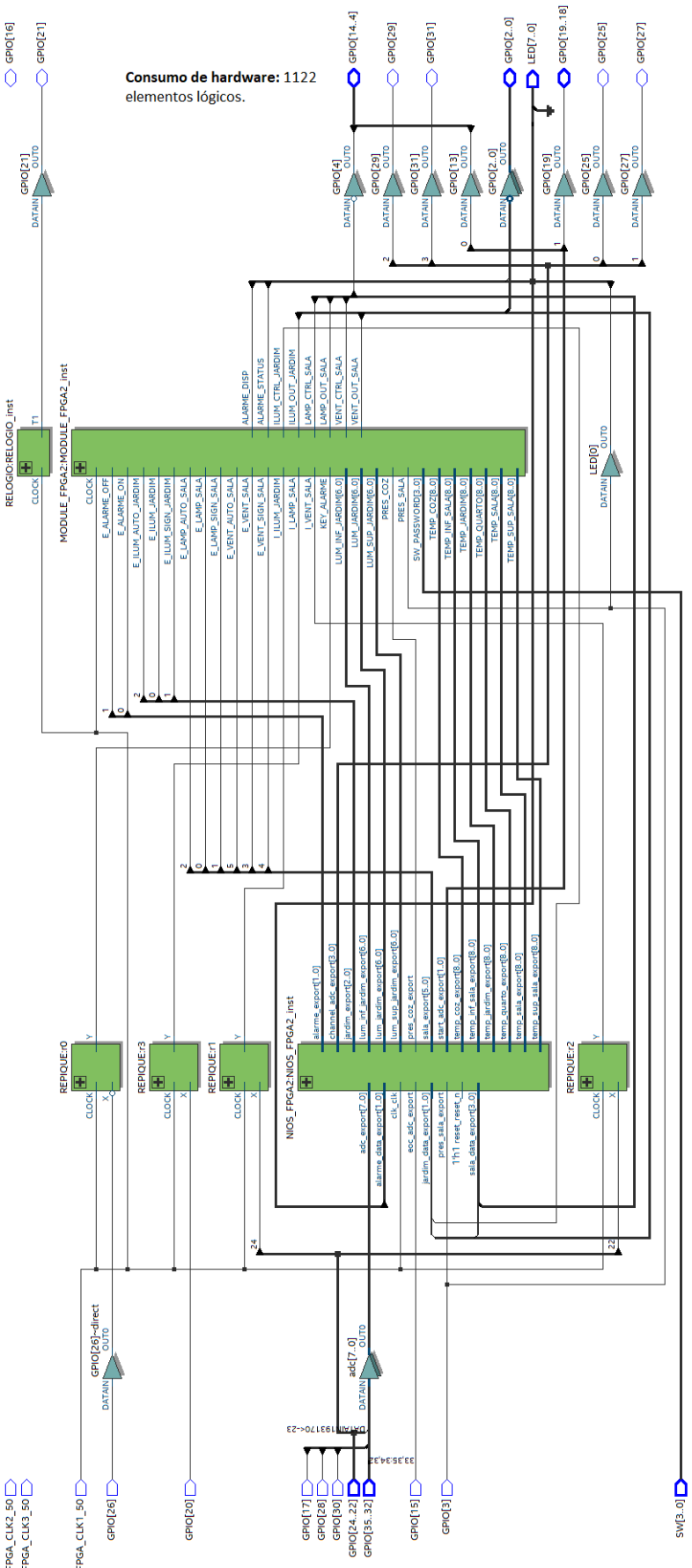
Foram desenvolvidos dois módulos genéricos, chamados de `SYS_MANUAL` e `SYS_AUTO`. O `SYS_MANUAL` é o sistema de controle dos atuadores que só possuem acionamento manual, ou seja, que não dependem de sensores. Já o `SYS_AUTO`, além de permitir o acionamento manual, ele também tem a opção de utilizar sensores para acionar seu respectivo atuador, isto é, o modo automático. Com exceção do alarme, todos os atuadores são controlados a partir de algum desses dois módulos.

Mais detalhes sobre a implementação desses módulos e das lógicas de funcionamento do controle de cada cômodo são apresentados no Apêndice A.

O controle do quarto é composto por dois blocos `SYS_MANUAL`, um para o ventilador e outro para a lâmpada. Dessa forma, o acionamento desses dois atuadores só pode ser feito manualmente, por *software* ou *hardware*.

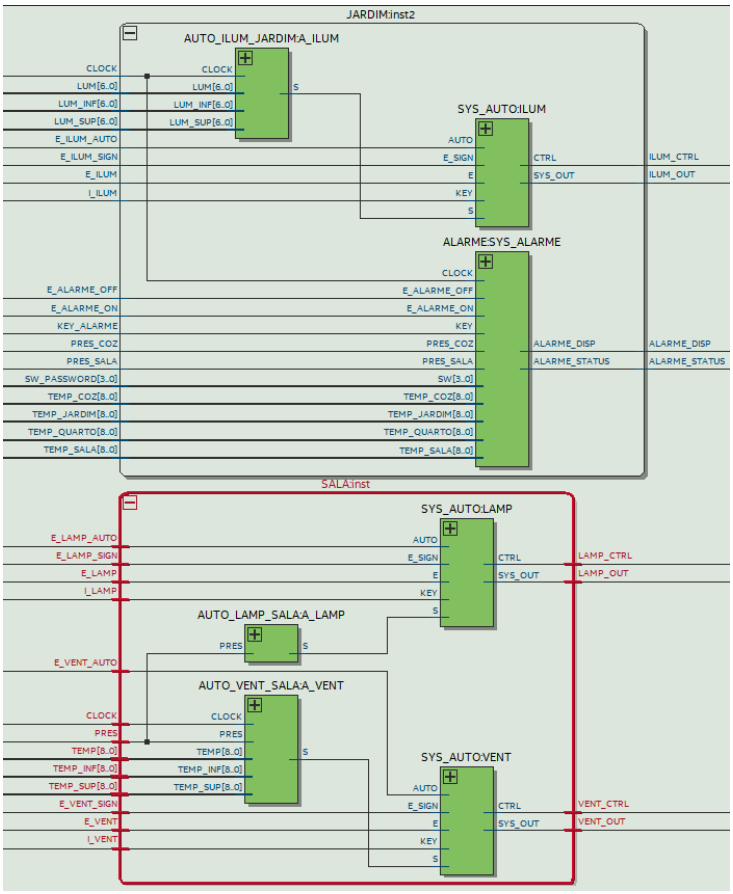
Os atuadores estão conectados a sinais semelhantes. Por exemplo, a entrada `E_LAMP_QUARTO` recebe o sinal correspondente ao acionamento da lâmpada do quarto via *software* que, dentro do bloco `QUARTO`, é chamado apenas de `E_LAMP`, para diferenciar-se do mesmo sinal correspondente ao ventilador, `E_VENT`. Este sinal, apesar de apenas trocar de nome quando adentra os blocos, está ligado diretamente à entrada `E` de um dos blocos `SYS_MANUAL`, que é o responsável por controlar a lâmpada do quarto. A saída deste bloco, que será denominada `LAMP_OUT_QUARTO`, se conectará ao opto-acoplador, o qual está ligado

Figura 40 – Visão geral do sistema implementado na FPGA2.



Fonte: Autoria Própria.

Figura 41 – Módulo da parte de *hardware* da FPGA2, contendo os sub-módulos de controle do jardim e da sala.



Fonte: Autoria Própria.

ao relé que contém os fios da instalação dessa lâmpada na maquete. Como o opto-acoplador aciona com nível lógico baixo, este sinal passa antes por uma porta NOT. A mesma lógica de nomeação vale para os outros sinais.

Na cozinha e na sala, há mais sinais de entrada do que no quarto, além de mais blocos. Isso se deve ao fato de que os atuadores da cozinha possuem recursos para serem acionados automaticamente, por isso o uso do `SYS_AUTO`. A lógica da sala é exatamente a mesma da cozinha. A diferença está apenas nos nomes dos sinais e no fato de que o controle é feito por outra FPGA. Por conta disso, as saídas são conectadas em outra placa de interface, a qual está ligada à sala e ao jardim.

O jardim possui três blocos: `SYS_ALARME`, `SYS_AUTO` e seu auxiliar `AUTO_ILUM_JARDIM`, pois possui uma fita de LED e uma sirene de alarme como atuadores. Diferente dos outros cômodos, o modo automático da iluminação do jardim depende do sensor de luminosidade.

Além disso, há um sistema de alarme simples, para demonstrar o quesito segurança presente na automação residencial. Para este sistema, o alarme pode ser acionado de duas formas: por *software*, escolhendo a opção “Acionar”, ou por *hardware*, colocando a senha

"1111" nas chaves SW e, em seguida, pressionando o interruptor correspondente. Para desligar, o procedimento é o mesmo, mas a senha padrão é "1010". Ainda existe uma "trava", que inibe o disparo do alarme, mas não o desliga. Para acionar a trava, basta colocar a senha "1100", sem a necessidade de apertar o botão. O alarme dispara se houver detecção de movimento em qualquer um dos dois sensores e ele estiver no estado "Acionado", ou quando algum dos sensores de temperatura detecta um valor elevado (acima de 50 graus Celsius).

4.3 *FIRMWARES* DO NIOS II

Na seção 4.2, foram apresentados os diagramas de blocos internos às FPGAs, os quais continham dois sistemas principais: o módulo de controle e o processador *softcore* Nios II. No diagrama de blocos geral, é possível ver que o sistema foi projetado para que as FPGAs se comunicassem com um computador, contendo a implementação de um servidor e uma interface gráfica. Um dos objetivos do Nios II é auxiliar o módulo de controle nessa comunicação, fazendo uma interface entre as duas partes. Outra tarefa é preparar os sinais recebidos dos sensores para serem enviados aos módulos de controle, já que o processador contém unidade de operações matemáticas.

Assim como o *hardware* possui um projeto para cada FPGA utilizada, a programação do Nios II também foi feita em dois projetos separados, utilizando a linguagem de programação C. Cada projeto possui duas partes, uma para a execução de suas tarefas e outra para realizar as funções de cliente, enviando e recebendo dados da rede.

Como essa comunicação não foi implementada totalmente, a parte do cliente foi substituída por um simulador do servidor, com uma função para mostrar os dados que seriam recebidos pelo computador e outra para coletar os dados que seriam enviados do computador à FPGA. Tanto a coleta de dados, como a apresentação deles são feitas através de um terminal de texto, o que foi apresentado na subseção 3.2.2 como sendo um dos métodos para a interação entre o usuário e o sistema.

A parte de execução das tarefas para auxiliar no controle é dividida em várias outras funções. Por haver muitas variáveis e vários locais diferentes para serem utilizadas, aquelas que são recebidas ou enviadas pela linha de comunicação (o simulador, neste caso) foram declaradas como globais. Em ambas as FPGAs, a execução do código começa com a preparação dos dados a serem enviados para o servidor, pois na arquitetura servidor-cliente o cliente sempre envia primeiro. Essa função é chamada `data_send`.

Nesse lugar, com auxilio da função `sprintf`, que converte variáveis de número inteiro para texto, informações como temperatura, luminosidade, presença, estados dos atuadores e quem realizou o último comando são armazenadas em uma única variável de texto. Além disso, um identificador é enviado junto, para que o servidor saiba quem é o

cliente que está se comunicando com ele. Essas variáveis citadas são apenas usadas por essa função, mas são alteradas em outras que serão explicadas a seguir.

Após a preparação dos dados, aconteceria o envio dessa variável de texto para o servidor. Ao invés disso, foram inseridas as funções de simulação, que representam funções presentes no servidor. Na visão do servidor, este começaria recebendo os dados enviados. A fim de demonstrar a correta preparação desses dados, eles recebidos por outras variáveis, através da função `sscanf`, que converte uma variável de texto para outras de outros tipos, como números inteiros. Essas novas variáveis, representadas com a letra `l` ao final para se diferenciarem das que pertencem apenas ao cliente, são apresentadas no terminal. Por exemplo, se a lâmpada da sala estiver acesa, aparecerá `lamp_sala_status_l = 1` no terminal de texto.

A próxima etapa seria o envio de dados do servidor ao cliente, contendo variáveis com os valores obtidos a partir da interface gráfica. No Nios II, isso é simulado através de perguntas feitas ao usuário. Primeiramente, o programa pergunta se ele deseja alterar o estado de algum atuador de determinado cômodo. Se a resposta for "não", ele avança para o próximo cômodo. Se for "sim", o programa dará a opção de ligar ou desligar manualmente ou ativar o modo automático. Isso foi implementado dessa forma para representar as opções que os usuários teriam ao usar a interface gráfica: primeiro, deveriam pensar qual cômodo realizar alterações, depois realizá-las. A diferença é que, com a interface gráfica, não seria necessário sempre seguir a mesma ordem, já que todas as opções aparecem ao mesmo tempo.

Ainda nessa função, os dados são preparados para serem enviados, o que seria feito através da função `data_send` presente no servidor. O procedimento é o mesmo já explicado, através da função `sprintf`. As variáveis enviadas são `e_(atuador)_(cômodo)_l`, `e_(atuador)_(comodo)_sign_l` e limites de temperatura e luminosidade, esta última apenas na FPGA2. As letras `e` representam que o dado foi enviado pela comunição Ethernet e a letra `l` ainda representa que a variável faz parte do simulador.

Na sequência, haveria a função do protocolo TCP/IP, para fazer a transmissão, o que não está presente no Nios II. Em seguida, os dados são recebidos pelo cliente. Esta parte é puramente do cliente, não fazendo mais parte da simulação. Com a função `data_recv`, os dados recebidos em uma variável de texto são convertidos e distribuídos para variáveis do tipo inteiro, as que serão utilizadas pelas outras funções. Essa conversão e distribuição é feita através da função `sscanf`.

Continuando a execução, começam as funções dos cômodos. Ao contrário do *hardware*, aqui não há lógica de controle, apenas conversão de variáveis para sinais. Para os sinais de acionamentos ou modo automático, que dependem de ser 0 ou 1 para desligado ou ligado, o programa verifica seu estado e atribui nível alto ou baixo para o bit correspondente à PIO daquela variável. Por exemplo, o quarto possui um PIO com 4 bits, um par para a

lâmpada e outro para o ventilador. Cada par contém um bit de acionamento e outro de sinalização. Esses bits são os sinais recebidos pelos módulos de controle explicados na **??**.

Para as variáveis de acionamento, o programa faz uma leitura da PIO com auxílio da função `IORD_ALTERA_AVALON_PIO_DATA`, aplica uma máscara para não alterar o estado dos outros bits e substitui o bit correspondente por 0 ou 1, dependendo o que estiver presente na variável, através do comando `IOWR_ALTERA_AVALON_PIO_DATA`. A forma mais simples de se fazer essa manipulação de bits é através de operadores lógicos, como AND e OR.

Já os bits de sinalização precisam ser em pulsos para seguir o protocolo do módulo de controle. Como o pulso não pode ser transmitido por Ethernet, a interface gráfica (e o simulador) fazem com que a variável `sign` alterne seu estado entre 0 e 1, sempre que o usuário optar por fazer alguma alteração no cômodo. O *firmware* monitora essa variável e, a cada iteração, ele salva seu estado em uma variável auxiliar, que no próximo ciclo será o estado anterior da variável recebida. No início do ciclo, ele compara a nova leitura com a anterior. Caso sejam diferentes, o procedimento a ser seguido é gerar um pulso, o que é feito aplicando-se 1 à PIO, esperando um tempo curto (1 μs) e retornando a 0. Se forem diferentes, o estado do sinal é mantido em nível baixo.

Mesmo que o usuário opte por desligar uma lâmpada que já está desligada, a sinalização de que teve comando na interface gráfica é feita. O procedimento a ser seguido é definido pelo módulo de controle, o *firmware* apenas repassa as informações.

Após atribuir todos os sinais, o *firmware* faz a leitura da PIO correspondente ao sinais de saída dos cômodos, direcionando os bits para as variáveis correspondentes. As funções de deslocamento de bits auxiliam nesse procedimento.

Por último, são calculadas temperatura e luminosidade. O procedimento possui funções próprias. Essas funções chamam a rotina que faz a manipulação do ADC e retorna o valor lido. A partir desse valor retornado, o cálculo é feito. A luminosidade utiliza a Equação 2 e a temperatura é calculada com a Equação 3 e a Equação 4. O cálculo da tensão recebida é feito através da Equação 5,

$$A0 = code \times \frac{VCC}{255.0} \tag{5}$$

onde VCC é a tensão de referência e alimentação do ADC e sensores e *code* é o código recebido do ADC.

Essa tensão $A0$ é substituída na Equação 3 e o resultado desse cálculo é utilizado na determinação da temperatura, através da Equação 4. Como o cálculo é feito em Kelvin, a transformação para outra escala pode ser feita. No caso de graus Celsius, basta subtrair 273 da temperatura em Kelvin.

A função que manipula o ADC segue o procedimento descrito no gráfico da Figura 71. Para facilitar a programação, foram criadas três PIOs para o ADC: controle dos canais, controles de início e ativação da saída e bits de recebimento dos valores lidos. Essas PIOs estão diretamente associadas às GPIOs correspondentes aos pinos do ADC. O procedimento é realizado manipulando-se os bits das duas primeiras PIOs e obtendo-se os valores da terceira. Para identificar qual sensor e cômodo chamou a função, foram criadas constantes, que são recebidas por parâmetro. De acordo com quem chamou a funções, são selecionados canais diferentes de leitura no ADC.

Todo esse algorítimo descrito é executado em um laço de repetição infinito. Com o sistema completo funcionando corretamente, a todo momento as variáveis são alteradas, enviadas e recebidas, funcionando como uma interface entre computador e módulo de controle.

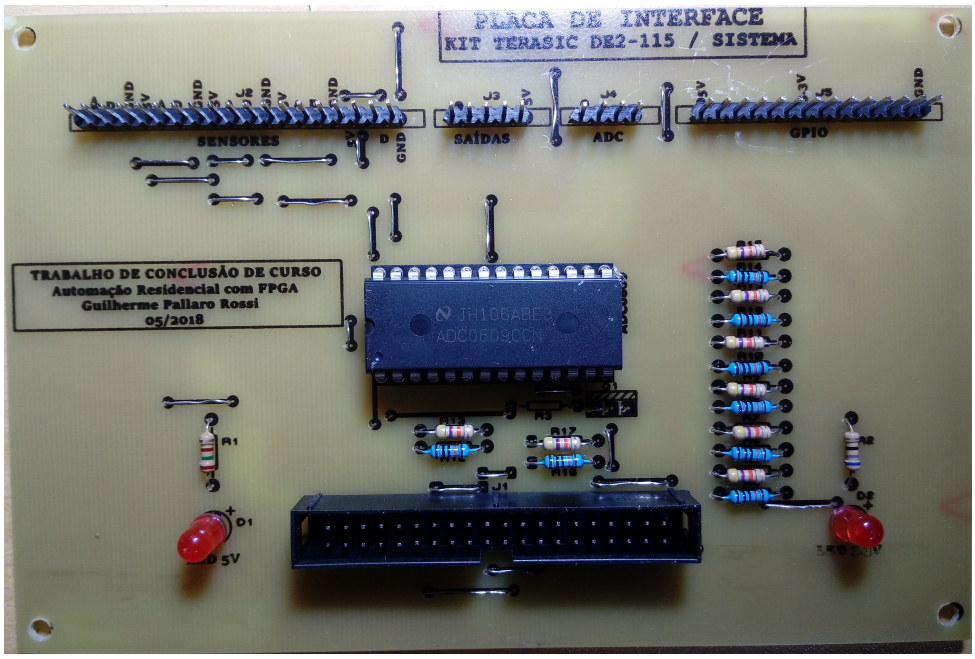
4.4 PLACAS DE INTERFACE

As placas de interface foram necessárias para unir o kit de desenvolvimento da FPGA com as plantas. Como os sensores, o ADC e os módulos de relés trabalham com 5 V e as GPIOs 3,3 V, foram necessários alguns resistores para abaixar a tensão entre os canais de saída do ADC e as entradas da GPIO. Para acionar os módulos, 3,3V foram suficientes, pois, como os módulos são acionados em nível baixo, o nível alto seria para os LEDs dos opto-acopladores não acionarem, o que foi suficiente com essa tensão.

Além disso, as placas continham as trilhas para conectar os pinos do conector da GPIO para outros pinos compatíveis com os elementos. Já que nem todos os canais do ADC e pinos GPIO seriam utilizados, foram deixados alguns pinos extras para futuras expansões e instalações de outras interfaces, como interruptores. Caso isso seja feito, deve-se atualizar o *hardware* contido na FPGA.

A placa final pode ser vista na Figura 42, que mostra a placa utilizada com a FPGA1. Mais informações sobre o projeto e a confecção podem ser vistas no Apêndice B. Apesar de estar indicado na serigrafia que ela é para o kit DE2-115, a placa também serviu para o kit DE10-Nano, pois a pinagem do conector GPIO é igual. Isso aconteceu pois, inicialmente, a ideia era que o projeto fosse baseado neste outro kit. A mudança ocorreu quando as placas já estavam fabricadas, fazendo com que fosse necessário utilizar os pinos GPIO sobressalentes. O kit DE2-115 já possui quantidade suficiente de chaves *push-button*, ao contrário do DE10-Nano. Isso mostrou que as placas são genéricas e podem atuar como periféricos auxiliares para kits de FPGA que possuem o conector GPIO de 40 pinos, desde que os sinais de alimentação estejam nos mesmos pinos.

Figura 42 – Aparência final da placa de interface.



Fonte: Autoria Própria.

4.5 INSTALAÇÕES ELÉTRICAS

A instalação elétrica foi feita com a colocação de pontos de lâmpada e tomadas para os ventiladores. Cada ponto recebia o fio neutro da rede e um retorno dos relés. Os relés foram conectados à fase e aos fios de retorno. A fase foi conectada a um interruptor, que serviu de chave geral.

As lâmpadas utilizadas para demonstração foram incandescentes e, para os ventiladores, foram deixadas tomadas. Os testes foram realizados colocando-se luminárias ligadas às tomadas, representando os ventiladores, pois era o que havia disponível.

No jardim, foi utilizada uma fita de LED para representar a iluminação. Ela foi ligada em uma tomada através de uma fonte. O mesmo foi feito para a sirene. Essas tomadas foram conectadas ao neutro e aos relés.

Os sensores foram fixados com fita dupla face, com cabos de 4 fios conectando-os às placas de interface.

Em uma *protoboard*, foi montado um circuito com quatro chaves *push-button*, representando os interruptores. O circuito continha resistores de *pull-up*, com valores de 100 kΩ cada, e foi ligado aos pinos remanescentes da placa de interface. Como só haviam quatro botões e um kit de FPGA disponíveis no dia do teste, cada planta foi testada separadamente, sendo necessária a religação dos fios dos botões.

A Figura 43 apresenta o aparato de testes, com os elementos de cada cômodo.

Figura 43 – Aparato de testes.

Fonte: Autoria Própria.

4.6 INTERFACE GRÁFICA DESENVOLVIDA

A interface gráfica foi desenvolvida em Linux, com linguagem de programação C, utilizando a biblioteca GTK como base e o *software* Glade para o desenho do *layout*. Primeiramente, declarou-se as variáveis e associo-se os sinais aos botões, conforme deve ser feito com a GTK.

A função ociosa foi utilizada para apresentar os valores de temperatura, luminosidade, presença, estado dos atuadores e quem realizou o último comando. A função de *timeout* foi utilizada para contar o tempo do relógio e apresentá-lo na janela.

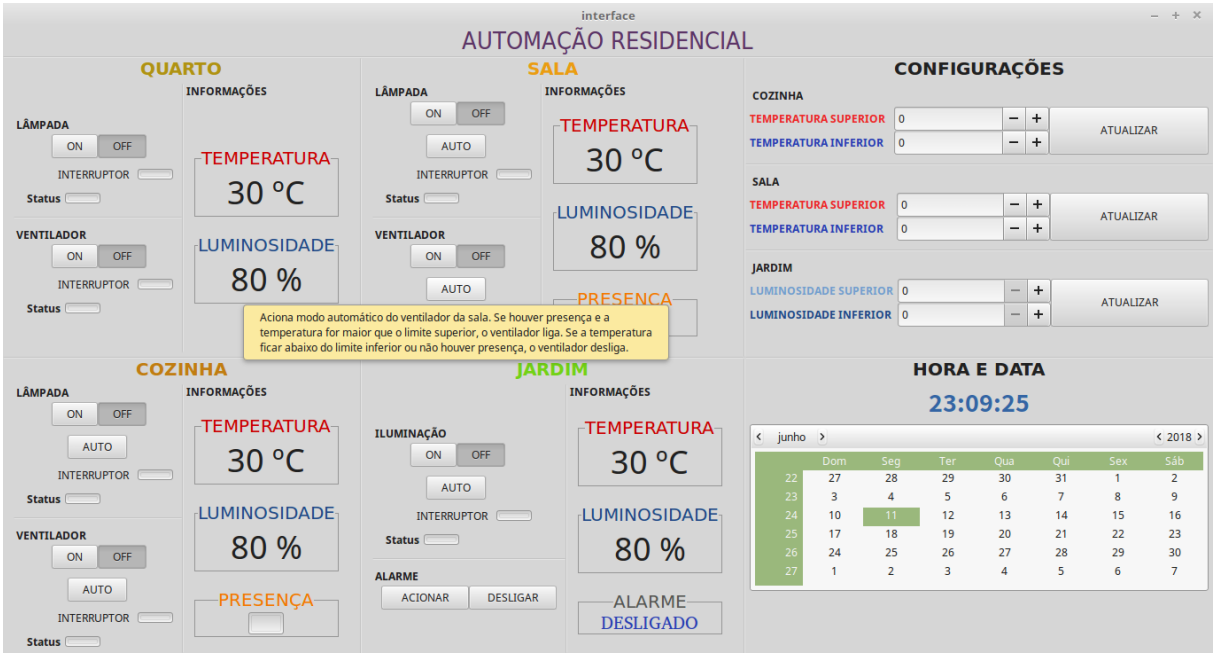
As funções dos botões realizaram a modificação das variáveis que seriam enviadas para as FPGAs. Além disso, para evitar que o usuário apertasse mais de um botão ao mesmo tempo, foi feito um tratamento, no qual um botão pressionado solta o outro. E, como pede o protocolo dos módulos de controle, a variável de sinalização era alternada entre 0 e 1 toda vez que um botão da interface era pressionado.

Junto à interface gráfica, foi desenvolvido o programa do servidor, para ser executado em outra *thread*. O programa enviava a variável de texto feita com a função `data_send`, como foi feito no Nios II, utilizando `sprintf`. Na função `data_recv`, foi utilizado o comando `sscanf`.

O calendário é tratado automaticamente pela GTK, utilizando a data presente no computador.

A interface desenvolvida é apresentada na Figura 44. Os valores mostrados nela não são provenientes de simulações, mas sim valores configurados como iniciais, até que se comece a leitura das variáveis.

Figura 44 – Interface gráfica desenvolvida.



Fonte: Autoria Própria.

4.7 COMUNICAÇÕES

O sistema foi projetado para que as partes se comunicassem através dos protocolos Ethernet e TCP/IP. Para isso ocorrer de forma correta, a FPGA deve estar preparada com o código de um cliente TCP/IP, o que não foi desenvolvido no Nios II, pois a porta Ethernet é conectada ao processador ARM integrado no kit. Uma alternativa a isso, seria implementar o protocolo no Linux e utilizar pontes comunicando o ARM e a FPGA.

As pontes não foram implementadas neste trabalho, mas a comunicação entre o servidor e o cliente sim. O procedimento utilizado foi a criação da função do cliente em um programa presente dentro do Linux da DE10-Nano. Esse *software* continha as mesmas variáveis que os *firmwares* implementados no Nios II, mas as informações que deveriam

ir da FPGA para o servidor eram digitadas pelo usuário, como estado dos atuadores, informações dos sensores e assim por diante. Os dados eram recebidos pela interface gráfica e apresentados. Na visão da interface, não importa quem alterou a variável na outra ponta da comunicação, apenas importa o valor que veio. Dessa forma, é possível demonstrar seu funcionamento, mesmo não tendo as pontes implementadas.

No sentido contrário, também foram realizados testes. Os botões eram apertados na interface, assim como os limites eram configurados, e os valores de suas variáveis eram mostrados no terminal de comando do Linux do kit.

4.8 PROCEDIMENTO DE TESTES

Para a realização dos testes e obtenção dos resultados, foi utilizado um kit DE10-Nano, as placas de interface, os interruptores montados na *protoboard*, os módulos de relé e a maquete. Com o aparato de testes montado, os resultados puderam ser obtidos através de observação dos atuadores acionando e também pela interface de texto implementada no Nios II.

Como foi utilizado apenas um kit, o teste foi realizado para uma planta por vez. Foram realizadas medições de temperatura e luminosidade de cada cômodo, acendendo a lâmpada e aproximando uma fonte de calor, para verificar variações nas condições. As leituras eram feitas no terminal de texto do Nios II.

Outro teste realizado foi do sensor de presença, um dos mais utilizados pelo sistema no todo, que tem sua saída conectada diretamente ao *hardware* da FPGA. O sensor de presença tem parte no acionamento automático da lâmpada e ventilador da cozinha e no disparo do alarme. Por isso, seu correto funcionamento é de extrema importância. Neste teste, o sensor foi configurado com o tempo mínimo de duração após cessar o movimento no cômodo (aproximadamente 2,5 s). Para simular presença, foi colocado a mão em seu alcance, retirando-a em seguida.

Além disso, foram testados os acionamentos por *hardware*, aplicando um pulso nos interruptores. Para verificar o resultado, foi observado na interface do Nios II os valores das variáveis correspondentes aos estados dos atuadores e os próprios atuadores, no aparato de testes. Assim, testou-se os circuitos de *pull-up* e parte dos módulos de controle implementados em *hardware* na FPGA, além da instalação elétrica.

O teste de acionamento por *software* também foi feito. Através das perguntas feitas pela interface, foram acionados os elementos. Para verificar e demonstrar os resultados, observou-se os LEDs indicadores presentes nos módulos de relé, além das lâmpadas acendendo e apagando. Esse teste foi realizado com o intuito de avaliar o funcionamento de outra parte dos módulos de controle, além da associação de sinais através das PIOs e GPIOs.

Os testes dos modos automáticos foram realizados de forma semelhante. Como não foi previsto o acionamento deste modo via *hardware*, os comandos eram enviados via *software* e verificados no aparato de testes.

O teste do alarme foi realizado, em primeiro momento, acionando o alarme via *hardware* e depois utilizando a trava, para demonstrar a lógica utilizada. Os acionamentos foram feitos através do sensor de presença da sala, pois, como não havia sido implementada toda a parte de comunicação, não houve a integração do sistema, de forma que não havia como uma FPGA conhecer as variáveis recebidas pela outra sem haver alterações no projeto. Para demonstrar o alarme disparando, ele foi acionado com os sensores de presença sendo ativados.

Ao final, para demonstrar o funcionamento da interface gráfica e comunicação Ethernet, foi utilizado o Linux da FPGA para apresentar as variáveis recebidas pela comunicação e simular o estado dos sensores e atuadores para enviá-los à interface.

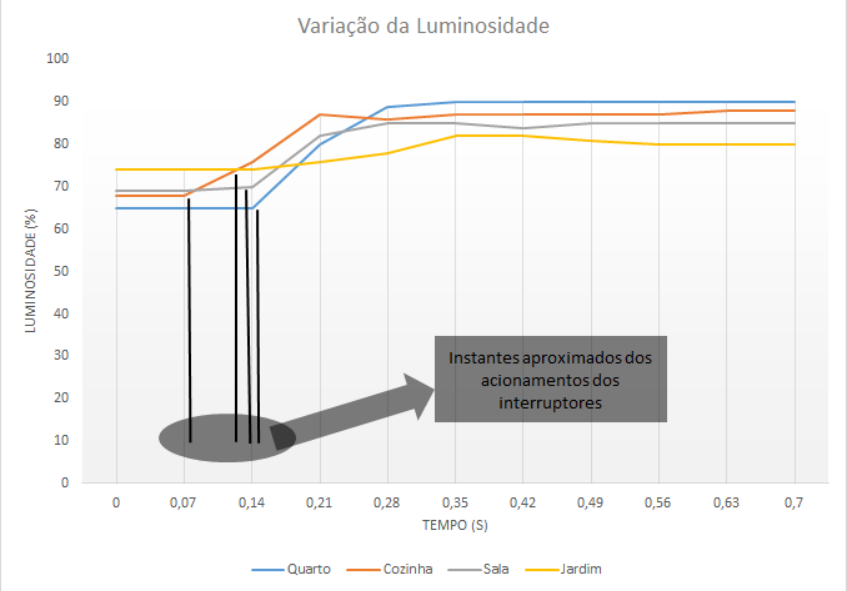
5 RESULTADOS E DISCUSSÕES

O primeiro teste realizado foi o dos sensores, feito de forma separada para cada FPGA. Os resultados foram agrupados em gráficos. Em seguida, realizou-se o teste dos atuadores e do sistema de alarme. Os resultados eram vistos na prática, com alguns deles sendo apresentados em fotos. Por fim, realizou-se o teste da interface gráfica e da comunicação de rede.

5.1 TESTE DOS SENSORES

O primeiro teste realizado foi a medição das luminosidades. Para isso, as medidas foram obtidas com o terminal do Nios II, primeiro para a FPGA 1 e depois para a FPGA 2. Com os registros, foi possível construir um gráfico indicando as leituras. Os intervalos de tempo foram aproximados, pois dependiam da velocidade de processamento e do atraso do terminal, mas foi possível ter uma base do funcionamento. Os níveis de luminosidade foram variados com o acionamento das lâmpadas e foi monitorado instantes antes e depois. A Figura 45 apresenta esses resultados. As diferenças se devem ao fato de que o teste não foi realizado ao mesmo tempo e os sensores estavam posicionados em locais diferentes.

Figura 45 – Leitura dos sensores de luminosidade.

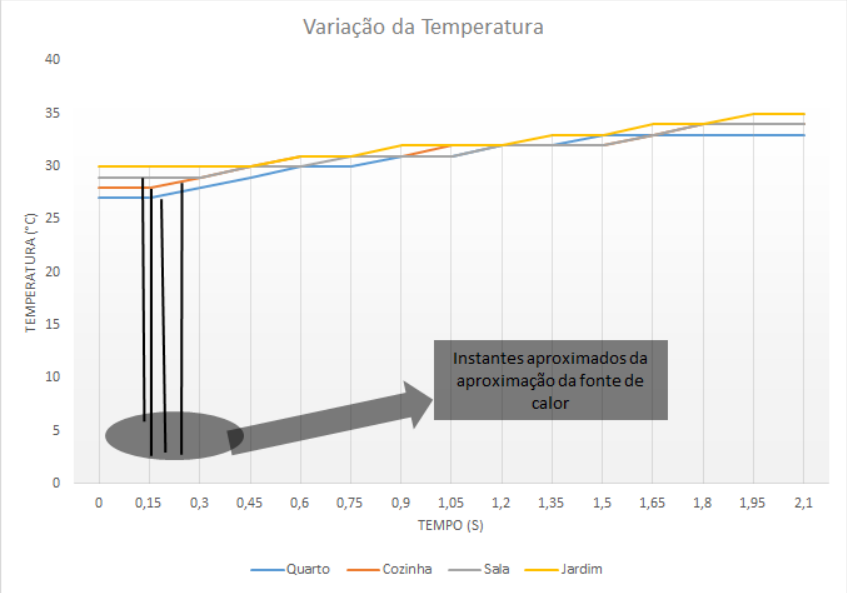


Fonte: Autoria Própria.

A resposta foi imediata, com as luminosidades aumentando no momento em que as lâmpadas acendiam. O sensor de luminosidade do jardim, por exemplo, é essencial para o acionamento automático da iluminação externa, por isso seu funcionamento deve ser garantido.

O próxima teste foi o aumento da temperatura, através da aproximação de uma fonte de calor. Novamente, o intervalo de tempo foi aproximado e os testes não foram realizados ao mesmo tempo. A Figura 46 ilustra as medições das temperaturas durante o aquecimento dos ambientes.

Figura 46 – Leitura dos sensores de temperatura.



Fonte: Autoria Própria.

Esses resultados dependiam de várias partes para ser obtidos, tais como: sensores, barramentos, placa de interface e ADC, direcionamento das GPIOs de entrada e saída para o processador (feito em Verilog) e *firmware* com algoritmos para o controle e a leitura do ADC, e o cálculo de temperatura e porcentagem de luminosidade. Conforme visto, todo esse conjunto se mostrou funcional. Além disso, a parte de acionamento das lâmpadas foi feita de forma independente das leituras dos sensores, mesmo ambas as funções sendo controladas pelo mesmo dispositivo.

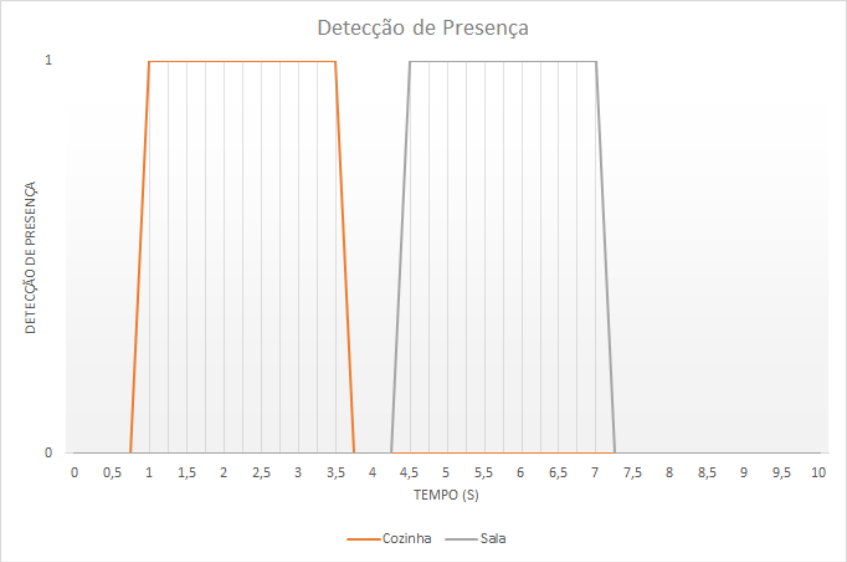
A Figura 47 demonstra a detecção de movimento feita pelos sensores de presença da cozinha e da sala. Para o teste, foi inserida e retirada a mão da área de alcance dos sensores e o número 1 representa “movimento detectado”, enquanto 0 representa “sem movimento”.

Assim, conclui-se a verificação de resultados de toda a parte de aquisição de dados, tanto por *firmware*, com a parte dos sensores de luminosidade e temperatura, como por *hardware*, com o sensor de presença.

5.2 TESTE DOS ATUADORES

A etapa seguinte foi o acionamento das lâmpadas e ventiladores, que já havia sido realizados para o teste dos sensores. Assim como o sensor de presença, os estados de saída

Figura 47 – Leitura dos sensores de presença.



Fonte: Autoria Própria.

são passados ao processador por meio de PIOs. O sistema prevê acionamento por *software* e por *hardware*. Inicialmente, o teste foi feito pelo *hardware*, apertando-se os botões para o acionamento dos atuadores.

Essa etapa também dependia de várias partes, tais como: ligação correta das chaves, *hardware* de controle (em VHDL) implementado corretamente, placa de interface, módulo de relés e instalação elétrica da maquete. Os comandos não foram realizados de forma aleatória, mostrando que a ordem não interferia no funcionamento, já que o *hardware* foi desenvolvido para que trabalhassem de forma independente. Essa é uma vantagem da FPGA, pois um sistema independente continuará respondendo de forma instantânea mesmo em aplicações mais complexas e com um número maior de elementos para acionar.

Após o teste dos acionamentos por *hardware*, foram feitos os acionamentos por *software*, através do *firmware* desenvolvido para demonstração. Uma simulação para a FPGA 1 é apresentada na Figura 48.

Nesse teste, foram acionados todos os elementos, como é possível ver nas respostas das perguntas e também no retorno ao final. A fim de demonstrar de forma mais clara, são apresentadas algumas fotos da maquete. A Figura 49 mostra o quarto e a cozinha com as luzes acesas. A Figura 50 mostra todos os relés acionados, inclusive os das tomadas dos ventiladores.

No teste do modo automático, os acionamentos ocorriam após a detecção de movimento no sensor de presença. As temperaturas limites inferior e superior também influenciaram no acionamento do ventilador da cozinha, conforme previsto no *hardware*, ou seja, acima da superior, o ventilador ligava (desde que haja detecção de movimento) e,

Figura 48 – Simulação de alteração dos dados pelo usuário a partir da interface no computador, para a FPGA1.

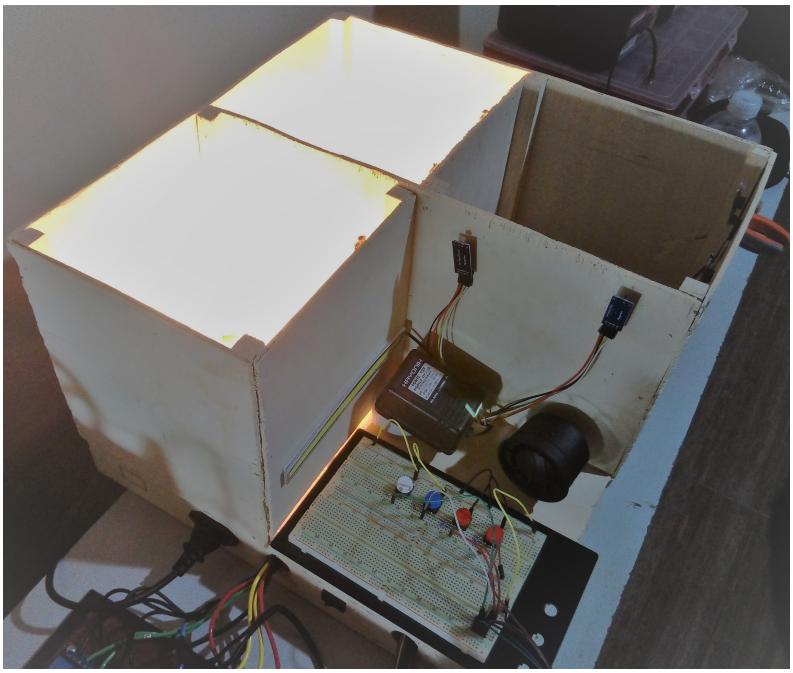
```
lamp_quarto_status_1 = 0       vent_quarto_status_1 = 0
lamp_coz_status_1 = 0          vent_coz_status_1 = 0

***VALORES QUE DEVEM SER ENVIADOS PELO SERVIDOR***
**Nas opçoes, digite 0 para OFF e 1 para ON**
Deseja alterar algum estado da lampada do quarto? (1)sim (0)nao
1
Escolha 1 em apenas uma das opções
e_lamp_quarto_1 =
1
Deseja alterar algum estado do ventilador do quarto? (1)sim (0)nao
1
Escolha 1 em apenas uma das opções
e_vent_quarto_1 =
1
Deseja alterar algum estado da lampada da cozinha? (1)sim (0)nao
1
Escolha 1 em apenas uma das opções
e_lamp_coz_1 =
1
e_lamp_coz_auto_1 =
0
Deseja alterar algum estado do ventilador da cozinha? (1)sim (0)nao
1
Escolha 1 em apenas uma das opções
e_vent_coz_1 =
1

e_vent_coz_auto_1 =
0
lamp_quarto_status_1 = 1       vent_quarto_status_1 = 1
lamp_coz_status_1 = 1          vent_coz_status_1 = 1

***VALORES QUE DEVEM SER ENVIADOS PELO SERVIDOR***
**Nas opçoes, digite 0 para OFF e 1 para ON**
Deseja alterar algum estado da lampada do quarto? (1)sim (0)nao
```

Fonte: Autoria Própria.

abaixo da inferior, ele desligava. Para a FPGA2, os resultados foram semelhantes.

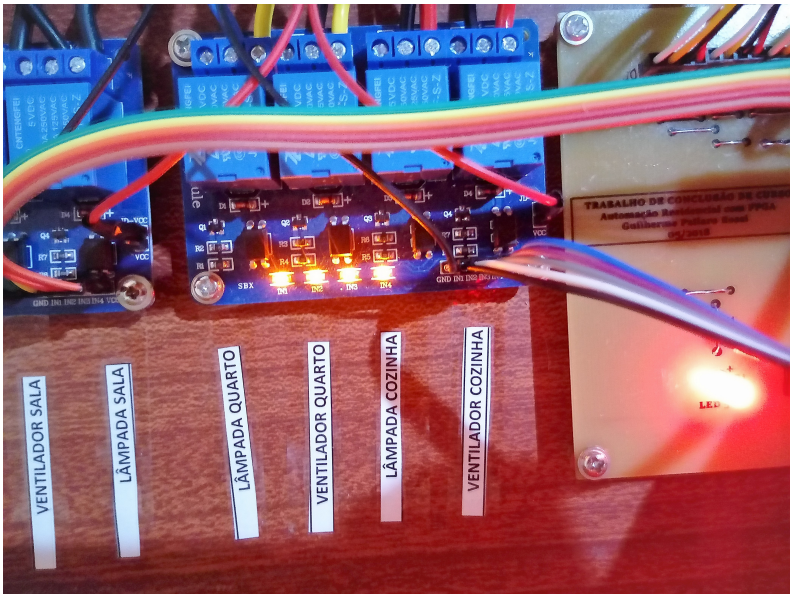
Com esses resultados, foi possível verificar que o *hardware* de controle foi implementado corretamente e a lógica utilizada pode ser empregada em sistemas maiores, basta ir replicando os blocos do código, conforme aumentam-se os elementos. Além disso, confirma-se mais uma vez que a placa de interface é funcional. Com isso, é possível dizer que o projeto dessa placa pode ser usado para outros fins, além de automação residencial. Para as duas plantas, elas se mostraram eficientes e trabalharam em harmonia com o *kit* da FPGA, através da GPIO, podendo ser utilizadas como um periférico genérico, já que contém várias entradas e saídas, além do ADC.

Figura 49 – Lâmpadas do quarto e da cozinha acesas.



Fonte: Autoria Própria.

Figura 50 – Relés acionados, indicado pelos LEDs acesos, conforme resultado da simulação.



Fonte: Autoria Própria.

A Figura 51 apresenta fotos das placas de interface ligadas, apenas para ilustrar seu funcionamento.

Apesar de ser apenas uma casa, ela está dividida em duas plantas, e a mesma lógica implementada nas duas centrais mostra que o projeto de uma automação com arquitetura descentralizada é bem mais simples do que na centralizada, devido ao reaproveitamento

Figura 51 – Placas de interface em funcionamento.

Fonte: Autoria Própria.

de código, poupando trabalho do projetista. Se o circuito fosse criado na mesma FPGA, também iria funcionar, mas o *hardware* gerado seria muito mais complexo e o projeto começaria a ficar sobrecarregado com o aumento do número de elementos lógicos. A síntese do *hardware* não é tão trivial para o processador do computador de projetos, fazendo com que haja um tempo elevado na compilação do projeto. Dividindo em dois ou mais projetos menores, esse problema é amenizado. Dentre outras, essa é uma grande vantagem em dividir o sistema em várias centrais, quando se trabalha com FPGA.

5.3 TESTE DO ALARME

O primeiro teste com o alarme foi a demonstração de seu acionamento por *hardware*. Acionando-o com a senha e o interruptor, o estado alterou para `Acionado`. Quando foi realizado um movimento no cômodo, pôde-se notar que o alarme dispara.

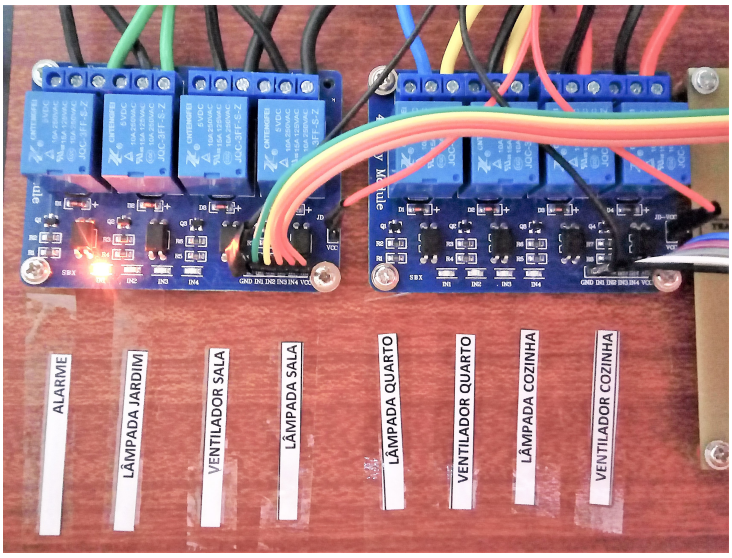
Para demonstrar o alarme em disparo, a Figura 52 apresenta uma foto do relé que aciona o alarme ligado, indicado através do LED.

Também foi realizado um ensaio mostrando o alarme sendo desligado pela senha e o outro pela trava. Pôde-se perceber que, mesmo ainda havendo presença na sala, o disparo não ocorreu depois que o estado se alterou para `Desligado`.

Ao desligar o alarme por trava, ele não disparou ao detectar presença, mas continuou acionado. A trava foi criada para facilitar no desligamento rápido do alarme, no caso de um disparo acidental, por exemplo, e evitar barulho excessivo. Ela possui uma senha mais simples e rápida de executar.

Sem realizar a integração, não foi possível apresentar todos os resultados do alarme. No entanto, com esse fato, foi possível ver uma vantagem da arquitetura descentralizada,

Figura 52 – Relé do alarme acionado.



Fonte: Autoria Própria.

pois mesmo sem a interligação dos sistemas, o alarme ainda funcionou com os elementos de sua região. É como se houvesse uma falha em alguma parte, isso não comprometeria o sistema todo, garantindo mais segurança ao usuário.

Os resultados obtidos com a parte de controle mostram que o sistema atendeu às expectativas, pois respondeu conforme foi projetado. Com o que foi apresentado até aqui, foi possível observar que o sistema funcionou bem sem a parte de comunicação, podendo ser integrado a outros projetos, como interfaces gráficas mais elaboradas, maior número de centrais de comando, mais elementos sensores e atuadores e outras possibilidades de expansão.

5.4 TESTE DA INTERFACE GRÁFICA E COMUNICAÇÃO DE REDE

O teste foi feito com as perguntas do simulador do cliente sendo respondidas, conforme Figura 53. Além disso, os botões que foram apertados na interface durante a simulação tiveram seus resultados mostrados pelo Linux do kit DE10-Nano.

Após enviar os comandos via rede, a interface foi atualizada automaticamente, conforme valores digitados. Os resultados são apresentados na Figura 54. Assim, foi possível perceber que a comunicação foi estabelecida corretamente, bem como o funcionamento da interface gráfica, mostrando que ela pode ser utilizada por este sistema se toda a integração for concluída.

Figura 53 – Simulação de alteração das variáveis pela FPGA.

```
COM3 - PuTTY
root@de10-nano:~# ./fpga1


***DIGITE OS DADOS QUE DEVEM SER GERADOS PELO HARDWARE DE CONTROLE***

lamp_quarto_status [OFF(0) ON(1)]:
1
lamp_quarto_ctrl [O ultimo comando foi da interface grafica (0) O ultimo comando foi do botao de controle(1)]:
1
vent_quarto_status [OFF(0) ON(1)]:
1
vent_quarto_ctrl [O ultimo comando foi da interface grafica (0) O ultimo comando foi do botao de controle(1)]:
0
temp_quarto (em Celsius):
26
lum_quarto (em porcentagem):
60
lamp_coz_status [OFF(0) ON(1)]:
0
lamp_coz_ctrl [O ultimo comando foi da interface grafica (0) O ultimo comando foi do botao de controle(1)]:
0
vent_coz_status [OFF(0) ON(1)]:
1
vent_coz_ctrl [O ultimo comando foi da interface grafica (0) O ultimo comando foi do botao de controle(1)]:
0
pres_coz [Sem presenca (0) Com presenca(1)]:
1
temp_coz (em Celsius):
25
lum_coz (em porcentagem):
58


INFORMACOES RECEBIDAS PELO CONTROLE DO QUARTO

e_lamp_quarto = 0
e_lamp_quarto_sign = 0
e_vent_quarto = 1
e_vent_quarto_sign = 1


INFORMACOES RECEBIDAS PELO CONTROLE DA COZINHA

e_lamp_coz = 0
e_lamp_coz_sign = 0
e_vent_coz = 0
e_vent_coz_auto = 1
e_vent_coz_sign = 1

temp_coz_sup: 25
temp_coz_inf: 20


***DIGITE OS DADOS QUE DEVEM SER GERADOS PELO HARDWARE DE CONTROLE***

lamp_quarto_status [OFF(0) ON(1)]:
```

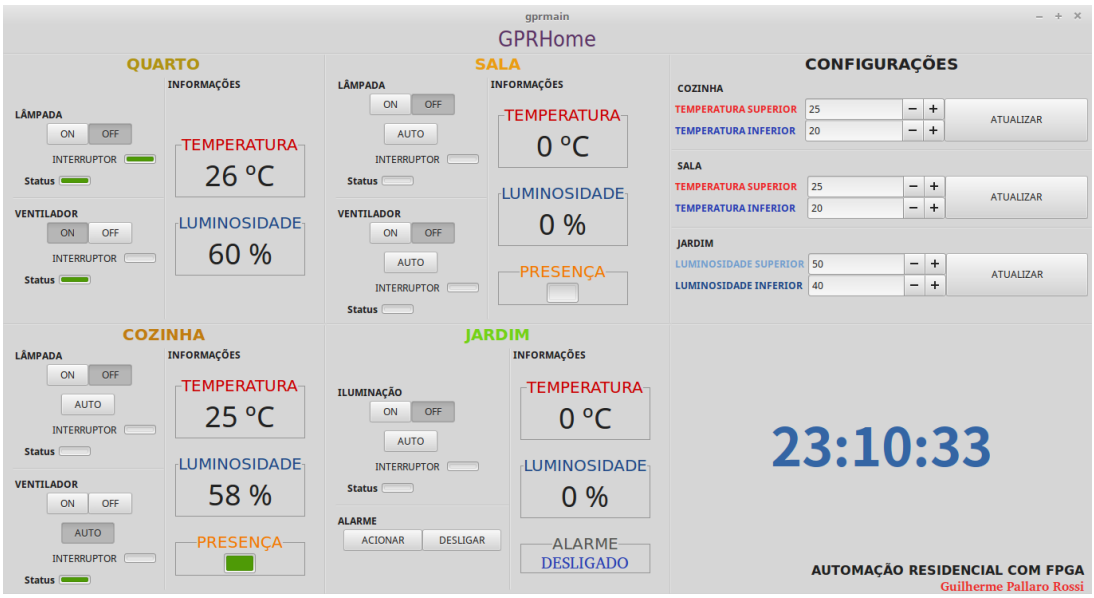
Fonte: Autoria Própria.

5.5 RESULTADOS INDIRETOS

Com a realização dos testes e do trabalho no geral, foi possível perceber alguns resultados não apresentados, mas que foram importantes. Nota-se, nesse sistema, que o paralelismo criado com a parte dos módulos de controle, puramente feitos em *hardware*, e o uso de várias FPGAs formam uma arquitetura de alta velocidade, dedicada a uma aplicação específica. O uso de FPGAs em projetos complexos pode se tornar viável, pois utilizar controladores mais simples, como os microcontroladores que processam de forma sequencial, necessitaria de mais processamento conforme aumentasse o sistema, elevando o custo. No caso das FPGAs, por ser tudo realizado em paralelo, haveria rápida execução de comandos sem necessitar de um processador melhor.

Além disso, como foi feito nesse projeto, podem ser utilizados *hardware* e *software*

Figura 54 – Interface gráfica após a realização dos testes.



Fonte: Autoria Própria.

trabalhando em conjuntos, deixando o *hardware* para as tarefas que necessitam de mais paralelismo enquanto o *software* faria as tarefas de apoio, como cálculos que necessitam de pouco processamento manipulação de variáveis intermediárias.

A FPGA pode ser uma ferramenta utilizada para aplicações que necessitam de aceleração de *hardware*, por conta do seu paralelismo. Normalmente, utilizam-se *hardwares* prontos para isso, como em placas de vídeo, mas a FPGA, por possuir *hardwares* customizáveis, poderia tornar mais viáveis e eficientes essas soluções.

Esta é uma ferramenta apropriada para quando se necessita de computação intensiva, pois sua flexibilidade a permite se adequar para a aplicação, aumentando a velocidade e capacidade de processamento e oferecendo arquiteturas sob demanda.

6 CONCLUSÃO

O trabalho, de forma geral, apresentou bons resultados. Através do sistema desenvolvido, foi possível observar que a FPGA é uma solução para atender sistemas de automação residencial com arquiteturas descentralizadas. O custo de implantação com esta ferramenta pode ser elevado, dependendo o modelo escolhido, mas existem as FPGAs de baixo custo que podem também ser utilizadas, mesmo possuindo menor capacidade. A vantagem técnica que a FPGA tem sobre os outros controladores é o *hardware* customizável, tornando o sistema mais enxuto, de forma que é possível aproveitar mais de seus recursos. Além disso, o *hardware* também possui mais velocidade de execução do que um *software*. Portanto, para aplicações com mais elementos, a FPGA é uma boa alternativa.

Conforme previsto nos objetivos, os *hardware* desenvolvidos para os módulos de controle apresentaram bom desempenho ao executar comandos vindos de duas entradas de controle. Um módulo não interferiu no funcionamento do outro, mostrando o paralelismo esperado da FPGA.

A interface gráfica também se mostrou funcional. Apesar de não ter sido interligada com o restante do sistema, os testes realizados com as simulações provaram que, quando houver a interligação, o sistema funcionará. A comunicação entre computador e FPGA apresentou bons resultados, pois os dados eram passados corretamente de um lado para o outro.

Os *firmwares* auxiliares foram essenciais para verificar as medições das temperaturas. Também verificou-se que os comandos eram recebidos corretamente pelos módulos de *hardware*, conforme resultados das simulações.

Todos os testes só foram possíveis de ser realizados devido às placas de interface. Como obteve-se bons resultados, conclui-se que as placas funcionaram corretamente. Além disso, verificou-se que elas podem ser utilizadas como periféricos genéricos para kits de FPGA.

A maquete desenvolvida como aparato de testes, apesar de ter sido feita apenas para essa aplicação, foi essencial para a realização do trabalho, pois, com ela, os outros resultados puderam ser obtidos, atendendo seus objetivos.

A busca por novos métodos para controladores de automação residencial, como foi realizado neste projeto, também é de grande importância para a comunidade externa à universidade, pois o aumento de soluções e formas de se criarem estruturas para essa área traz a tendência de melhorias na qualidade dos serviços prestados e diminuição dos custos para o consumidor final.

Os resultados obtidos também mostraram que os a solução utilizada pode ser implantada em áreas e sistemas mais complexos do que uma automação residencial com poucos elementos. A demonstração em um sistema simples mostrou que a solução proposta é funcional e tem capacidade de ser suplantada.

É notável que o projeto ainda tem pontos a serem melhorados, que seriam interessantes de se realizar em trabalhos futuros. Sugere-se que seja implementada de forma completa a comunicação entre os sub-sistemas, com a utilização das pontes apresentadas ou com outras soluções. Além disso, a interface gráfica desenvolvida está limitada àquela topologia da residência, então poderia ser criada, baseando-se na que foi implementada, uma interface mais genérica, que permita a inserção de novos cômodos e elementos, além de oferecer mais opções ao usuário. Outra sugestão é implementar o servidor e a interface internos à um dos kits, eliminando a necessidade do computador.

REFERÊNCIAS

ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 5410: Instalações elétricas de baixa tensão**. [S.l.], 2004.

ABREU, E. R. de; VALIM, P. R. O. Domótica: Controle de automação residencial utilizando celulares com bluetooth. **VIII Simpósio de Excelência em Gestão e Tecnologia**, 2011.

ACCARDI, A.; DODONOV, E. Automação residencial: Elementos básicos, arquiteturas, setores, aplicações e protocolos. **Revista TIS**, v. 1, n. 2, 2012.

ADAFRUIT INDUSTRIES. **PIR Motion Sensor**. 2014. Disponível em: <https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/overview>. Acesso em: 20 jul. 2018.

ALMEIDA, R. **Leitura de chaves mecânicas e o processo de debounce**. 2014. Disponível em: <https://www.embarcados.com.br/leitura-de-chaves-debounce/>. Acesso em: 25 jul. 2018.

ALVAREZ, D. F. d. S.; ANTUNES, F. I. **Automação residencial utilizando bluetooth, ethernet e smartphone**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2015.

AMETHERM. **NTC Thermistors Steinhart and Hart Equation**. 2013. Disponível em: <https://www.ametherm.com/thermistor/ntc-thermistors-steinhart-and-hart-equation>. Acesso em: 30 mar. 2018.

ATHOS ELECTRONICS. **Portas Lógicas: Entendendo a Eletrônica Digital**. 2018. Disponível em: <https://athoselectronics.com/portas-logicas-eletronica-digital/>. Acesso em: 26 jul. 2018.

AUTOMALABS. **Módulo de Relês**. 2012. Disponível em: <http://www.automalabs.com.br/modulo-de-reles/>. Acesso em: 02 jul. 2018.

BOLZANI, C. A. M. **Desenvolvimento de um simulador de controle de dispositivos residenciais inteligentes: uma introdução aos sistemas domóticos**. Dissertação (Mestrado) — EPUSP, 2004.

BRAGA, N. C. **Como funciona a lâmpada incandescente (ART443b)**. 1996. Disponível em: <http://www.newtoncbraga.com.br/index.php/como-funciona/3215-art443b>. Acesso em: 03 jul. 2018.

BRAGA, N. C. **Como funcionam os Conversores A/D - parte 1 (ART224)**. 2015. Disponível em: <http://www.newtoncbraga.com.br/index.php/como-funciona/1508-conversores-ad>. Acesso em: 12 jul. 2018.

CLIMABRISA. **Entenda como funciona um ventilador e um exaustor**. 2018. Disponível em: <http://blog.climabrisa.com.br/2018/02/16/entenda-como-funciona-um-ventilador-e-um-exaustor/>. Acesso em: 03 jul. 2018.

CUNHA, M. **Resistores Pull-Up e Pull-Down**. 2018. Disponível em: <http://www.marciocunha.eti.br/2016/01/resistores-pull-up-e-pull-down.html>. Acesso em: 28 jul. 2018.

ECOCASA. **Iluminação - Tipos de Lâmpadas**. 2014. Disponível em: <https://www.ecocasa.pt/energia_content.php?id=1>. Acesso em: 03 jul. 2018.

GNOME DEVELOPER. **Timeouts, IO and Idle Functions**. 2014. Disponível em: <https://developer.gnome.org/gtk-tutorial/stable/c1759.html>. Acesso em: 05 maio 2018.

GONZAGA, D. **Circuito de interface para microcontroladores**. 2015. Disponível em: <https://www.embarcados.com.br/circuito-de-interface-para-microcontroladores/>. Acesso em: 02 jul. 2018.

HONGFA RELAY. **Datasheet: JQC-3FF**. [S.l.], 2018.

INTEL FPGA. **Nios II Gen2 Processor Reference Guide**. [S.l.], 2016.

JÚNIOR, J. S. da S. **Lâmpadas fluorescentes**. 2018. Disponível em: <https://mundoeducacao.bol.uol.com.br/fisica/lampadas-fluorescentes.htm>. Acesso em: 03 jul. 2018.

LEITE, J. C. **Design de Interfaces de Usuário**. 2000. Disponível em: <https://www.dimap.ufrn.br/~jair/ES/c6.html>. Acesso em: 11 maio 2017.

LEON, N. **Programa**. 2018.

LOMBARDI, R. R. R. Controle remoto infravermelho para automação. **Centro Universitário de Brasília**, 2006.

MACÊDO, D. **Arquitetura: Von Neumann vs Harvard**. 2012. Disponível em: <http://www.diegomacedo.com.br/arquitetura-von-neumann-vs-harvard/>. Acesso em: 13 maio 2017.

MAGALHÃES, A. **Processador Nios II**: Introdução. 2015. Disponível em: <https://agetechnology.wordpress.com/2015/12/16/processador-nios-ii-introducao/>. Acesso em: 13 maio 2017.

MALVINO, A. P. **Eletrônica**. 4. ed. São Paulo: Pearson, 1996. v. 1. ISBN 85-346-0378-2.

MATTEDE, H. **Como funciona uma lâmpada incandescente?** 2018. Disponível em: <https://www.mundodaeletrica.com.br/como-funciona-uma-lampada-incandescente/>. Acesso em: 03 jul. 2018.

MEHL, E. L. de M. Conceitos fundamentais sobre placas de circuito impresso. **Universidade Federal do Paraná**, 2011.

MUNDO DA ELÉTRICA. **Como funciona um relé? O que é um relé?** 2018. Disponível em: <https://www.mundodaeletrica.com.br/como-funciona-um-rele-o-que-e-um-rele/>. Acesso em: 02 jul. 2018.

MURATORI, J. R.; BÓ, P. H. D. Automação residencial: histórico, definições e conceitos. **O Setor Elétrico**, São Paulo, p. 70, 2011.

MURATORI, J. R.; BÓ, P. H. D. **Automação Residencial: Conceitos e Aplicações**. 2. ed. Belo Horizonte: Educere, 2014.

NOGUEIRA, M. L. B.; MELCHIORS, C. **Um pequeno estudo sobre par trançado.** 1996. Disponível em: <http://penta2.ufrgs.br/rc952/Cristina/utpatual.html>. Acesso em: 11 maio 2017.

NOVA ELETRÔNICA. **Como funciona a lâmpada fluorescente?** 2018. Disponível em: <http://blog.novaeletronica.com.br/como-funciona-a-lampada-fluorescente/>. Acesso em: 03 jul. 2018.

OLIVEIRA, R. A. de; SILVA, A. P. B. da. William herschel, os raios invisíveis e as primeiras ideias sobre radiaçao infravermelha. **Revista Brasileira de Ensino de Física**, v. 36, n. 4, p. 4603, 2014.

PALNITKAR, S. **Verilog HDL: A guide to Digital Design and Synthesis**. 1. ed. [S.l.]: Prentice Hall PTR, 1996. v. 1. ISBN 9780134516752.

PEDRONI, V. A. **Eletrônica Digital Moderna e VHDL**. Rio de Janeiro: Elsevier, 2010. ISBN 9788535234657.

PIBER, P. R. V. O uso da aritmética distribuída na implementação de filtros fir no processador niosii. **UFRS**, 2017.

PRODUÇÃO VIRTUAL. **RISC x CISC**. 2017. Disponível em: <http://producao.virtual.ufpb.br/books/edusantana/introducao-a-arquitetura-de-computadores-livro/livro/livro.chunked/ch04s04.html>. Acesso em: 13 maio 2017.

PROGRAMMER'S NOTES. **GTK+ 3 C Program using Glade 3**. 2015. Disponível em: <https://prognotes.net/2015/06/gtk-3-c-program-using-glade-3/>. Acesso em: 28 abr. 2018.

PUHLMANN, H. F. W. **Trazendo o mundo real para dentro do processador - Conversor A/D**. 2015. Disponível em: <https://www.embarcados.com.br/conversor-a-d/>. Acesso em: 12 jul. 2018.

RIBEIRO, D. **Internet a cabo ou Wi-Fi?**: Saiba qual é a mais indicada para você. 2014. Disponível em: <http://www.techtudo.com.br/dicas-e-tutoriais/noticia/2014/02/internet-a-cabo-ou-wi-fi-saiba-qual-e-a-mais-indicada-para-voce.html>. Acesso em: 2 maio 2017.

ROVERI, M. R. Automação residencial. **Faculdade Politec**, 2012.

SHARP. **Datasheet: PC817**. [S.l.], 2018.

SILVA, R. P. M. da. **Processador softcore Altera Nios II**. 2014. Disponível em: <https://www.embarcados.com.br/altera-nios-ii/>. Acesso em: 12 maio 2017.

SOUZA, F. **Acionamento de uma lâmpada com Arduino**. 2014. Disponível em: <https://www.embarcados.com.br/acionamento-de-uma-lampada-com-arduino/>. Acesso em: https://www.embarcados.com.br/acionamento-de-uma-lampada-com-arduino/.

TANENBAUM, A. S.; WETHERALL, D. **Redes de Computadores**. 5. ed. São Paulo: Pearson, 2011. ISBN 9788576059240.

TECNOHOLD. **Central de Alarme Contra Incêndio.** 2018. Disponível em: <http://www.tecnohold.com.br/informacoes/central-alarme-incendio.php>. Acesso em: 03 jul. 2018.

TECNOLOGÍA. **Reles**. 2018. Disponível em: <http://www.areatecnologia.com/electricidad/rele.html>. Acesso em: 02 jul. 2018.

TERASIC TECNOLOGIES INC. **My First Nios II**. [S.l.], 2013.

TERASIC TECNOLOGIES INC. **User Manual: Altera DE10-Nano**. [S.l.], 2017.

TEXAS INSTRUMENTS. **Datasheet: ADC0808/ADC0809 8-Bit uP Compatible A/D Converters with 8-Channel Multiplexer**. [S.l.], 2013.

TEZA, V. R. **Alguns aspectos sobre a automação residencial: domótica**. Dissertação (Mestrado) — Universidade Federal de Santa Catarina - UFSC, 2002.

THOMAZINI, D.; ALBUQUERQUE, P. U. B. D. **Sensores Industriais**: Fundamentos e aplicações. 8. ed. São Paulo: Érica, 2005. ISBN 9788536500713.

TOCCI, R.; WIDMER, N. S.; MOSS, G. L. **Sistemas Digitais. Princípios e Aplicações**. São Paulo: Pearson, 2011. ISBN 9788576059226.

TORRES, G. **Redes De Computadores**: versão revisada e atualizada. Rio de Janeiro: Novaterra, 2009. ISBN 9788561893057.

VANZETTI, R. **Practical applications of infrared techniques**: A new tool in a new dimension for problem solving. [S.l.]: Wiley-Interscience, 1972.

VERGANI, L. **Descubra as principais soluções de sistema de segurança**. 2018. Disponível em: <http://www.onixsecurity.com.br/blog/descubra-as-principais-solucoes-de-sistema-de-seguranca/>. Acesso em: 03 jul. 2018.

APÊNDICE A – IMPLEMENTAÇÃO DOS HARDWARES DE CONTROLE

O sistema de controle desenvolvidos para as FPGAs são compostos por módulos genéricos, cuja lógica atende a maioria dos atuadores, e pelo agrupamento desses módulos com outros auxiliares para criar a lógica de cada cômodo. Além disso, há um sistema diferenciado para o alarme.

A.1 MÓDULOS GENÉRICOS

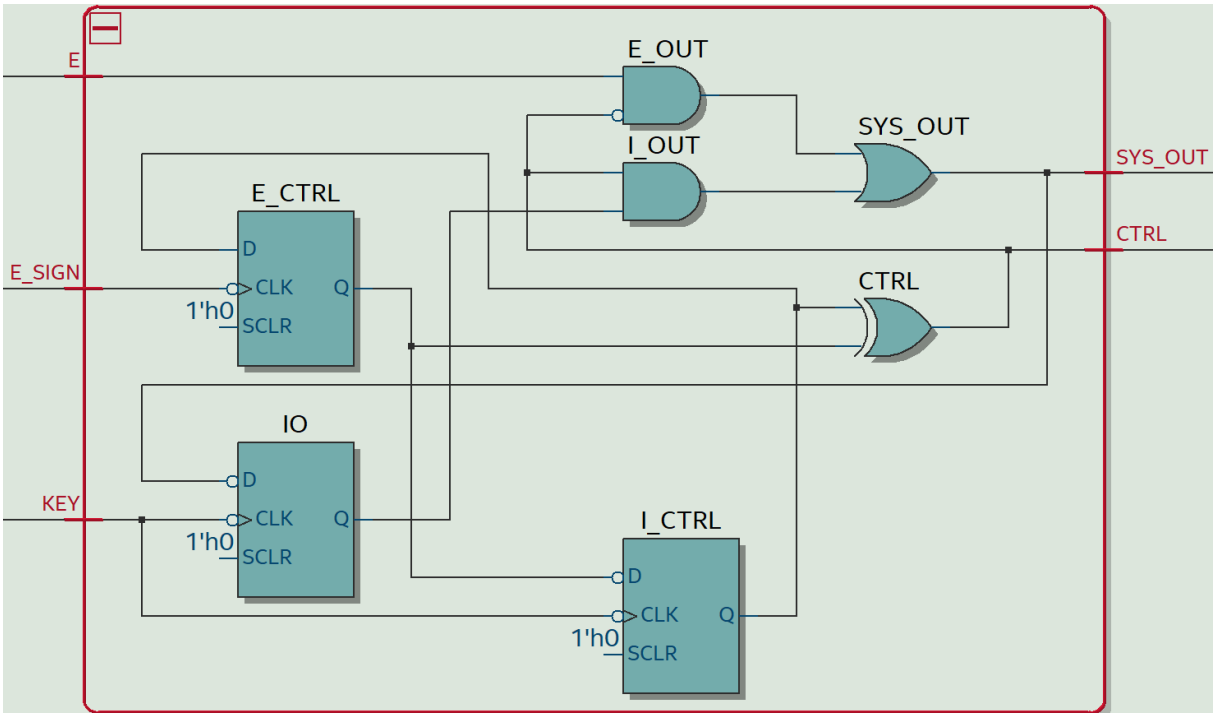
A maioria dos atuadores são controlados pelos blocos `SYS_MANUAL` ou `SYS_AUTO`. O sistema manual determina a saída a partir de duas possíveis entradas de usuário, que pode escolher acionar via *software*, através da interface gráfica, ou via *hardware*, através de chaves. O acionamento pelo computador é feito com a escolha de dois estados, ligado ou desligado. A saída também será ligada ou desligada, conforme a escolha, independente do estado anterior. Já o acionamento por interruptores foi projetado para chaves do tipo *push button*, que mudam de estado apenas enquanto estão pressionadas. Toda vez que a chave recebe um pulso, o estado do atuador é alternado, independente de qual parte fez o último acionamento. O sistema automático possui a mesma implementação do manual, acrescentando-se a opção do modo automático. Quando este recurso está ativado (apenas via *software*), o estado do atuador depende de uma lógica feita com os sinais dos sensores.

A Figura 55 apresenta o circuito lógico do `SYS_MANUAL`. Na imagem, observa-se que o bloco possui três entradas e duas saídas. A saída `SYS_OUT` está ligada diretamente no atuador, enquanto a saída `CTRL` está ligada ao *software*, para indicar quem fez o último acionamento. O protocolo de saída segue a seguinte lógica: para `SYS_OUT`, 1 é ligado e 0, desligado; para `CTRL`, 1 indica que o último acionamento foi feito por *hardware* e 0, por *software*.

A entrada `E` é um sinal vindo do *software*, no qual nível lógico alto é para acionar e baixo para desligar. A entrada `KEY` é o sinal vindo da chave, que será recebido na entrada de *clock* de dois *flip-flops* tipo D, `IO` e `I_CTRL`. Isso significa o bloco detectará apenas a borda, no caso do circuito, a de descida.

A entrada `E_SIGN` faz parte do protocolo e é utilizada para sinalizar que houve um novo comando recebido via *software*. Seu sinal deve ser um pulso, assim como `KEY`, o qual tem que ser gerado pela parte de *software*. Ao pressionar um botão na interface gráfica, uma variável deve alternar entre 1 e 0, independente de qual botão foi pressionado. Essa variável é enviada ao processador paralelo à FPGA, no caso o NIOS II. O *firmware*, executado em laço de repetição infinito, recebe essa variável e armazena seu estado. Na próxima iteração,

Figura 55 – Sistema de controle manual SYS_MANUAL.



Fonte: Autoria Própria.

ele verifica se a variável recebida é diferente da anterior que está armazenada. Caso seja, ele gera um pulso na PIO que está conectada ao E_SIGN.

O *flip-flop* E_CTRL recebe esse sinal em sua entrada de *clock*. A sua entrada D está conectada à saída do I_CTRL. Isso significa que, toda vez que tiver uma borda de descida em E_SIGN, a saída de E_CTRL será igual à saída de I_CTRL. Como ambas saídas estão conectadas à porta XOR, o sinal CTRL será 0, ou seja, como já dito, acionamento por *software*. Com CTRL sendo 0, a porta AND IOUT, que tem uma de suas entradas conectadas à saída da XOR, terá 0 na saída, independente do estado de IO, passando esse 0 à porta OR seguinte. Assim, a saída SYS_OUT depende do sinal vindo da AND E_OUT. Essa porta tem o sinal CTRL ligada em uma de suas entradas através de uma porta NOT. Como esse sinal é 0, a AND tem 1 nessa entrada, o que faz ela depender apenas do sinal vindo de E. Portanto, se E for 1, a saída será 1, caso contrário, a saída será 0.

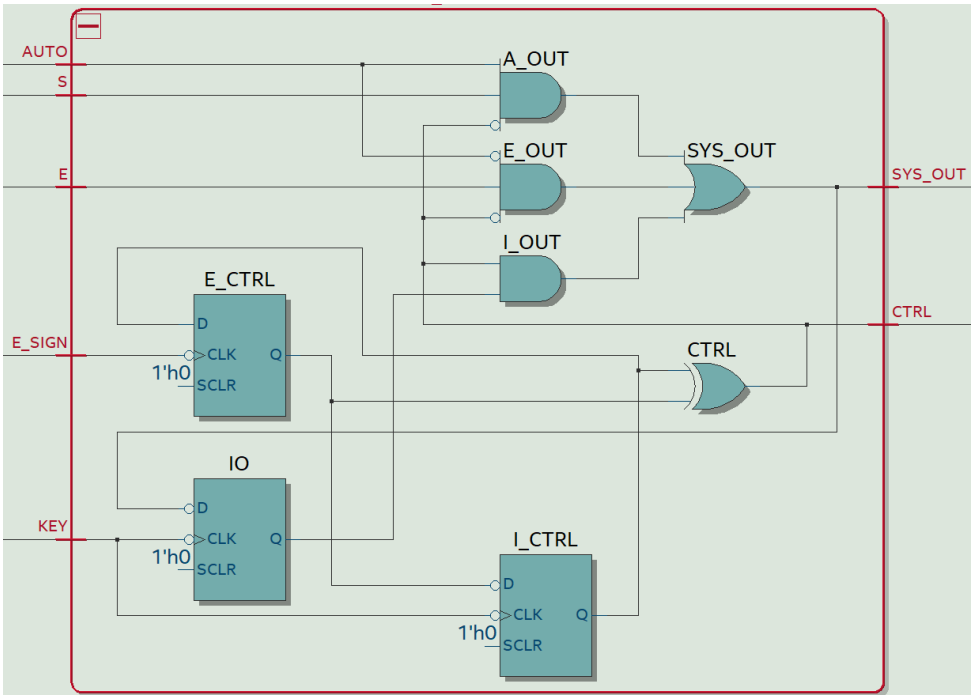
Caso seja feito o acionamento por *hardware*, a borda de descida de KEY fará com que o *flip-flop* I_CTRL tenha o estado complementar de E_CTRL, já que este está conectado à porta D de I_CTRL através de uma porta NOT. Como as saídas dos dois *flip-flops* estão conectadas à porta XOR e os sinais são complementares, a saída dessa porta será 1, fazendo com que E_OUT sempre seja 0, I_OUT dependa do sinal vindo de IO e SYS_OUT dependa apenas de I_OUT. Por sua vez, a saída de IO será o complemento de SYS_OUT, já que este está ligado à porta D do *flip-flop* através de uma porta NOT.

Esse módulo garante a independência entre os dois modos de acionamento, pois

um automaticamente inibe a ação do outro sempre que é requisitado.

A Figura 56 apresenta o circuito lógico do `SYS_AUTO`. Esse módulo possui as mesmas entradas e saídas que o `SYS_MANUAL` e mais as entradas `AUTO` e `S`. `AUTO` é o sinal que vem do *software*, sendo 1 para modo automático ligado e 0 para desligado. `S` é o sinal que vem de um bloco auxiliar que faz o tratamento da lógica que depende de sensores, o que não pode ser generalizado, pois cada aplicação requer uma lógica de acionamento automático diferente. Independente disso, S deve ser 1 para ativar o atuador e 0 para desativá-lo.

Figura 56 – Sistema de controle manual e automático `SYS_AUTO`.



Fonte: Autoria Própria.

A lógica de `CTRL`, para selecionar quem vai assumir o controle, permanece a mesma que no sistema manual, pois, independente do acionamento ser por `E` ou por `AUTO`, ambos são via *software* e gerarão o pulso no `E_SIGN`. A lógica para acionamento por *hardware* também permanece a mesma. O que muda é a separação dos sinais que sempre estão sendo recebidos do *software*. Esse controle é feito pelo próprio sinal `AUTO`. Quando ele é 0, a porta AND `A_OUT` será 0, enquanto `E_OUT` dependerá do sinal E e do sinal `CTRL`. Quando o usuário ativa o modo automático, a porta `E_OUT` será 0 e `A_OUT` dependerá de `CTRL` (que é 0, transformado em 1 pela NOT) e `S`. Assim, a saída fica dependente apenas de `S`. Dessa forma, o bloco auxiliar assume o controle da saída. Além disso, se porventura `E` e `AUTO` forem 1, não haverá disputa, pois `AUTO` tem prioridade, já que é ele que faz o controle.

A.2 QUARTO

A Figura 57 apresenta o código em VHDL do bloco do quarto. O código basicamente cria dois componentes do `SYS_MANUAL` e faz a associação de sinais.

Figura 57 – Código em VHDL do bloco do quarto.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY QUARTO IS

    PORT (
        E_LAMP, E_VENT              :  IN STD_LOGIC;
        E_LAMP_SIGN, E_VENT_SIGN    :  IN STD_LOGIC;
        I_LAMP, I_VENT              :  IN STD_LOGIC;
        LAMP_OUT, VENT_OUT          :  BUFFER STD_LOGIC;
        LAMP_CTRL, VENT_CTRL        :  BUFFER STD_LOGIC
    );
END ENTITY;

ARCHITECTURE QUARTO OF QUARTO IS

COMPONENT SYS_MANUAL IS
    PORT (
        E         :  IN STD_LOGIC;
        E_SIGN    :  IN STD_LOGIC;
        KEY       :  IN STD_LOGIC;
        SYS_OUT   :  BUFFER STD_LOGIC;
        CTRL      :  BUFFER STD_LOGIC
    );
END COMPONENT;

BEGIN
    LAMP : SYS_MANUAL PORT MAP (E_LAMP, E_LAMP_SIGN, I_LAMP, LAMP_OUT, LAMP_CTRL);
    VENT : SYS_MANUAL PORT MAP (E_VENT, E_VENT_SIGN, I_VENT, VENT_OUT, VENT_CTRL);
END ARCHITECTURE;
```

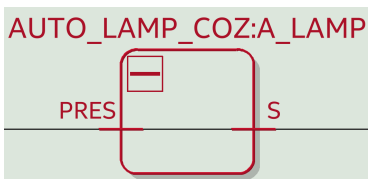
Fonte: Autoria Própria.

A.3 COZINHA E SALA

O bloco do sistema automático precisa de um bloco auxiliar que crie a lógica de automatização. No controle da cozinha, os blocos são `AUTO_LAMP_COZ` e `AUTO_VENT_COZ`, enquanto na sala os blocos são `AUTO_LAMP_SALA` e `AUTO_VENT_SALA`.

Para a lâmpada, o acionamento automático depende apenas do estado do sensor de presença, como é possível ver na Figura 58. O sinal do sensor está ligado diretamente à entrada `S` do sistema automático.

Figura 58 – Bloco auxiliar do acionamento automático da lâmpada da cozinha.



Fonte: Autoria Própria.

Já para o ventilador há, além do sinal do sensor de presença, os sinais de temperatura medida e os limites superiores e inferiores, que são definidos pelo usuário na interface. Para o tratamento, também é necessário o sinal de *clock*, gerado pelo oscilador interno ao kit de desenvolvimento.

A Figura 59 apresenta o código para o controle automático do ventilador da cozinha. A lógica utilizada é a seguinte: se a temperatura medida for maior que o limite superior e houver detecção de movimento, o ventilador liga, e se for menor que o limite inferior ou não houver presença, ele desliga. Essa diferença entre os limites é necessária, na lógica utilizada, para que não ocorram oscilações que aconteceriam se fosse configurado apenas um limite e a temperatura pairasse sobre esse valor. O sinal de presença, por sua vez, é utilizado para melhorar a eficiência energética, pois não tem necessidade do ventilador permanecer ligado se não tiver alguém no cômodo.

Figura 59 – Código para o controle automático do ventilador da cozinha.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY AUTO_VENT_COZ IS
6
7      PORT (
8          PRES                        :  IN STD_LOGIC;
9          CLOCK                       :  IN STD_LOGIC;
10         TEMP_INF, TEMP_SUP, TEMP    :  IN INTEGER RANGE 218 TO 398;
11         S                           :  OUT STD_LOGIC
12     );
13 END ENTITY;
14
15 ARCHITECTURE AUTO_VENT_COZ OF AUTO_VENT_COZ IS
16     SIGNAL TEMP_CTRL : STD_LOGIC;
17 BEGIN
18
19     PROCESS
20     BEGIN
21
22         WAIT UNTIL CLOCK'EVENT AND CLOCK = '1';
23
24         IF (TEMP > TEMP_SUP) THEN
25             TEMP_CTRL <= '1';
26         ELSIF (TEMP < TEMP_INF) THEN
27             TEMP_CTRL <= '0';
28         ELSE
29             TEMP_CTRL <= TEMP_CTRL;
30         END IF;
31
32     END PROCESS;
33
34     S <= PRES AND TEMP_CTRL;
35
36 END ARCHITECTURE;
```

Fonte: Autoria Própria.

Apesar de ter sido ilustrado apenas o controle da cozinha, a mesma lógica vale para a sala.

A.4 JARDIM

A Figura 60 apresenta a lógica de controle do modo automático da iluminação do jardim, o qual depende do sensor de luminosidade.

Figura 60 – Lógica de controle do modo automático da iluminação do jardim.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY AUTO_ILUM_JARDIM IS
6
7     PORT (
8        CLOCK                    :  IN STD_LOGIC;
9        LUM_INF, LUM_SUP, LUM    :  IN INTEGER RANGE 0 TO 100;
10       S                        :  OUT STD_LOGIC
11    );
12 END ENTITY;
13
14 ARCHITECTURE AUTO_ILUM_JARDIM OF AUTO_ILUM_JARDIM IS
15    SIGNAL LUM_CTRL : STD_LOGIC;
16 BEGIN
17
18    PROCESS
19    BEGIN
20
21       WAIT UNTIL CLOCK'EVENT AND CLOCK = '1';
22
23       IF (LUM > LUM_SUP) THEN
24          LUM_CTRL <= '0';
25       ELSIF (LUM < LUM_INF) THEN
26          LUM_CTRL <= '1';
27       ELSE
28          LUM_CTRL <= LUM_CTRL;
29       END IF;
30
31    END PROCESS;
32
33    S <= LUM_CTRL;
34
35 END ARCHITECTURE;
```

Fonte: Autoria Própria.

Assim como os ventiladores da cozinha e da sala dependem de limites de temperatura, a iluminação do jardim depende de limites de luminosidade. A lógica aqui é inversa: quando a luminosidade medida é maior que o limite superior, o LED se apaga e quando é menor que a inferior, ele se acende, não necessitando de sensor de presença. O LED foi utilizado por ser uma lâmpada de menor consumo, já que, se fosse em uma residência real, seria comum o jardim permanecer aceso durante a noite toda. O modo automático também traz a vantagem da eficiência energética para a automação residencial, pois, durante o dia, a lâmpada se desligaria sozinha.

A.4.1 Alarme

Baseando-se nos parâmetros do alarme, apresentados na Seção 4.2, foi desenvolvido um sistema em *hardware* para ser implementado junto aos módulos de controle. Por se tratar de um *hardware*, a ordem das linhas dentro de uma arquitetura de VHDL não importa, pois todas serão executadas ao mesmo tempo pela FPGA.

O código se inicia com a união dos sinais principais para gerar a saída. Para haver disparo, o alarme precisa estar acionado e deve haver presença em um dos dois cômodos, ou então alguma temperatura se elevar demais, que seria o caso de um incêndio. Tudo isso se a trava não estiver em nível lógico baixo, ou seja, acionada.

Em seguida, vem a parte de controle da trava, com um *flip-flop* dependente de um sinal de *clock*, para colocar 0 ao sinal de saída se as chaves estiverem na posição de trava ou 1, caso contrário.

O próximo trecho contém o monitoramento de temperatura, que coloca 1 na variável de saída caso os valores de algum dos sensores sem elevem demais.

Na sequência, se inicia a lógica de acionar o alarme manualmente. Diferentemente dos outros atuadores, o alarme não indica quem foi o último que acionou, se foi *hardware* ou *software*. Para ele, o que importa é se foi acionado ou não. O sistema foi projetado para que o *hardware* tivesse as mesmas opções que no *software*, mas disponde-se apenas de um interruptor e 4 chaves. No *software*, há dois botões, um para acionar e o outro para desligar. Já no *hardware*, há duas senhas com as mesmas funções do computador, que são validadas pelo interruptor.

Para isso, são gerados dois sinais diferentes, um para ligar o alarme e outro para desligá-lo. Sempre que uma ação é recebida para acionar ou desligar, o sinal correspondente vai para 1, gerando uma borda de subida. Caso não sejam colocadas as senhas corretas ou o botão não seja apertado, ambos os sinais continuam em 0, não gerando borda. Em seguida, o sinal de acionamento é atribuído a uma porta XOR, que recebe como entrada um sinal intermediário da parte de controle por *software* e outro do controle por *hardware*. Cada um desses sinais intermediário dependem de outros sinais, chamados `A, B, C, D`, que são aplicados a portas XOR para gerarem suas saídas. O algorítimo descrito até aqui é apresentado no código da Figura 61

Continuando o algorítimo, começam os blocos de verificação dos acionamentos ou desligamentos. Tanto a parte do *hardware* como a do *software* dependem das bordas de subida dos sinais gerados anteriormente para acontecer. Os sinais `A, B` são correspondentes ao *software*, enquanto `C, D` são para o *hardware*. Caso seja detectado que o usuário ligou o alarme, independente de como foi, o sistema deve forçar para que `E_ALARME` e `I_ALARME` sejam diferentes, para que a saída da XOR que une os dois sinais seja 1. Caso seja requisitado um desligamento, os dois sinais devem ser iguais, fornecendo 0 à saída da XOR. Essa lógica é obtida a partir da manipulação e monitoramento dos sinais `A,B,C,D` e os outros intermediários. Como não é possível dois *flip-flops* alterarem o mesmo sinal, essa lógica é necessária para que o sistema possa escolher o sinal de saída.

Observando os código da Figura 62 e também da Figura 63, é possível compreender a lógica descrita.

Figura 61 – Parte um do código do sistema de alarme.

```
19
20  BEGIN
21
22      ---------------------------------------------------------------------------------
23      --SAÍDA -> ALARME DISPARA SE TIVER PRESENÇA NA SALA OU COZINHA E ESTIVER ACIONADO,
24      ---OU ENTÃO SE A TEMPERATURA ESTIVER ALTA. TUDO ISSO SE A TRAVA ESTIVER DESACIONADA.
25      ---------------------------------------------------------------------------------
26      ALARME_DISP <= (((PRES_COZ OR PRES_SALA) AND ALARME) OR TEMP_HIGH) AND TRAVA;
27      ---------------------------------------------------------------------------------
28
29      PROCESS
30      BEGIN
31         WAIT UNTIL CLOCK'EVENT AND CLOCK = '1';
32
33         ------------------------------------------------------------
34         --CONTROLE DA TRAVA -> COM SENHA 1100, ALARME NUNCA DISPARA
35         ------------------------------------------------------------
36         IF (SW = "1100") THEN
37            TRAVA <= '0';
38         ELSE
39            TRAVA <= '1';
40         END IF;
41         ------------------------------------------------------------
42
43
44         ---------------------------------------------------------------------------------------
45         --MONITORAMENTO DE TEMPERATURA -> SE ALGUM SENSOR DETECTAR MAIS QUE 50ºC, ALARME DISPARA
46         ---------------------------------------------------------------------------------------
47         IF (  TEMP_QUARTO > 323 OR
48               TEMP_COZ > 323 OR
49               TEMP_SALA > 323 OR
50               TEMP_JARDIM > 323 ) THEN
51            TEMP_HIGH <= '1';
52         ELSE
53            TEMP_HIGH <= '0';
54         END IF;
55         ---------------------------------------------------------------------------------------
56
57
58         ---------------------------------------------------------------------------------------------
59         --ACIONAR OU DESLIGAR ALARME VIA HARDWARE -> SE A CHAVE FOR PRESSIONADA E A  SENHA FOR 1111,
60         ---ALARME ACIONA. SE A SENHA FOR 1010, ALARME DESLIGA.
61         ---------------------------------------------------------------------------------------------
62         IF (KEY = '1') THEN
63            IF (SW = "1111") THEN
64               I_ALARME_ON <= '1';
65            ELSIF (SW = "1010") THEN
66               I_ALARME_OFF <= '1';
67            ELSE
68               I_ALARME_ON <= '0';
69               I_ALARME_OFF <= '0';
70            END IF;
71         ELSE
72            I_ALARME_ON <= '0';
73            I_ALARME_OFF <= '0';
74         END IF;
75         ---------------------------------------------------------------------------------------------------
76      END PROCESS;
77
78      ---------------------------------------------------------------------------
79      --CONTROLE DO ESTADO DO ALARME -> DETECTAR SE ALARME FOI ATIVADO OU DESTIVADO.
80      ---DEPENDE DO SINAL DE SOFTWARE (E_ALARME) E DO SINAL DE HARDWARE (I_ALARME).
81      ---ALARME ESTARÁ ATIVADO APENAS QUANDO UM DOS DOIS SINAIS FOREM 1.
82      ---------------------------------------------------------------------------
83      ALARME <= E_ALARME XOR I_ALARME;
84      E_ALARME <= A XOR B;
85      I_ALARME <= C XOR D;
86      ---------------------------------------------------------------------------
```

Fonte: Autoria Própria.

Por fim, para indicar o estado do alarme, outra saída recebe este sinal que indica se o alarme está acionado ou desligado. Essa saída seria enviada à interface gráfica, para indicar no campo correspondente se o alarme está desligado ou acionado.

A saída `ALARME_DISP` indica se o alarme entrou em disparo. O mesmo sinal dessa saída vai para a GPIO e também a uma PIO correspondente, fazendo com que o alarme dispare e seja indicado no *software* que isso ocorreu.

Através deste sistema, é possível demonstrar a aplicação que a automação residencial tem na parte de segurança, pois são demonstradas duas formas: uma contra invasão e outra contra incêndios. O sistema é simples, pois possui apenas uma sirene, mas o sinal de

Figura 62 – Parte dois do código do sistema de alarme.

```
88    ----------------------------------------------------------------------
89    --ACIONAR POR SOFTWARE -> VERIFICA SE TEVE ACIONAMENTO POR SOFTWARE
90    ---SE TEVE PULSO PARA ACIONAMENTO, ALARME = 1;
91    ----------------------------------------------------------------------
92    PROCESS
93    BEGIN
94       WAIT UNTIL E_ALARME_ON'EVENT AND E_ALARME_ON = '1';
95
96       IF (I_ALARME = '0') THEN
97          IF (B = '0') THEN A <= '1';
98          ELSE A <= '0';
99          END IF;
100      ELSE
101         IF (B = '0') THEN A <= '0';
102         ELSE A <= '1';
103         END IF;
104      END IF;
105   END PROCESS;
106   ----------------------------------------------------------------------
107
108   ----------------------------------------------------------------------
109   --DESLIGAR POR SOFTWARE -> VERIFICA SE TEVE DESLIGAMENTO POR SOFTWARE
110   ---SE TEVE PULSO PARA DESLIGAMENTO, ALARME = 0;
111   ----------------------------------------------------------------------
112   PROCESS
113   BEGIN
114      WAIT UNTIL E_ALARME_OFF'EVENT AND E_ALARME_OFF = '1';
115
116      IF (I_ALARME = '0') THEN
117         IF (A = '0') THEN B <= '0';
118         ELSE B <= '1';
119         END IF;
120      ELSE
121         IF (A = '0') THEN B <= '1';
122         ELSE B <= '0';
123         END IF;
124      END IF;
125   END PROCESS;
126
```

Fonte: Autoria Própria.

disparo poderia ser utilizado pelo servidor para requisitar outras tarefas, como contatar polícia ou bombeiro, pensando no sistema sendo aplicado em uma casa de verdade.

Figura 63 – Parte três do código do sistema de alarme.

```
127     ---------------------------------------------------------------------
128     --ACIONAR POR HARDWARE -> VERIFICA SE TEVE ACIONAMENTO POR HARDWARE
129     ---SE TEVE PULSO PARA ACIONAMENTO, ALARME = 1;
130     ---------------------------------------------------------------------
131     PROCESS
132     BEGIN
133        WAIT UNTIL I_ALARME_ON'EVENT AND I_ALARME_ON = '1';
134
135        IF (E_ALARME = '0') THEN
136           IF (D = '0') THEN C <= '1';
137           ELSE C <= '0';
138           END IF;
139        ELSE
140           IF (D = '0') THEN C <= '0';
141           ELSE C <= '1';
142           END IF;
143        END IF;
144     END PROCESS;
145     ---------------------------------------------------------------------
146
147
148     ---------------------------------------------------------------------
149     --DESLIGAR POR HARDWARE -> VERIFICA SE TEVE DESLIGAMENTO POR HARDWARE
150     ---SE TEVE PULSO PARA DESLIGAMENTO, ALARME = 0;
151     ---------------------------------------------------------------------
152     PROCESS
153     BEGIN
154        WAIT UNTIL I_ALARME_OFF'EVENT AND I_ALARME_OFF = '1';
155
156        IF (E_ALARME = '0') THEN
157           IF (C = '0') THEN D <= '0';
158           ELSE D <= '1';
159           END IF;
160        ELSE
161           IF (C = '0') THEN D <= '1';
162           ELSE D <= '0';
163           END IF;
164        END IF;
165     END PROCESS;
166     ---------------------------------------------------------------------
167
168     ------------------------------------
169     --SAÍDA INDICANDO ESTADO DO ALARME
170     ------------------------------------
171     ALARME_STATUS <= ALARME;
172     ------------------------------------
173
174
175  END ARCHITECTURE;
```

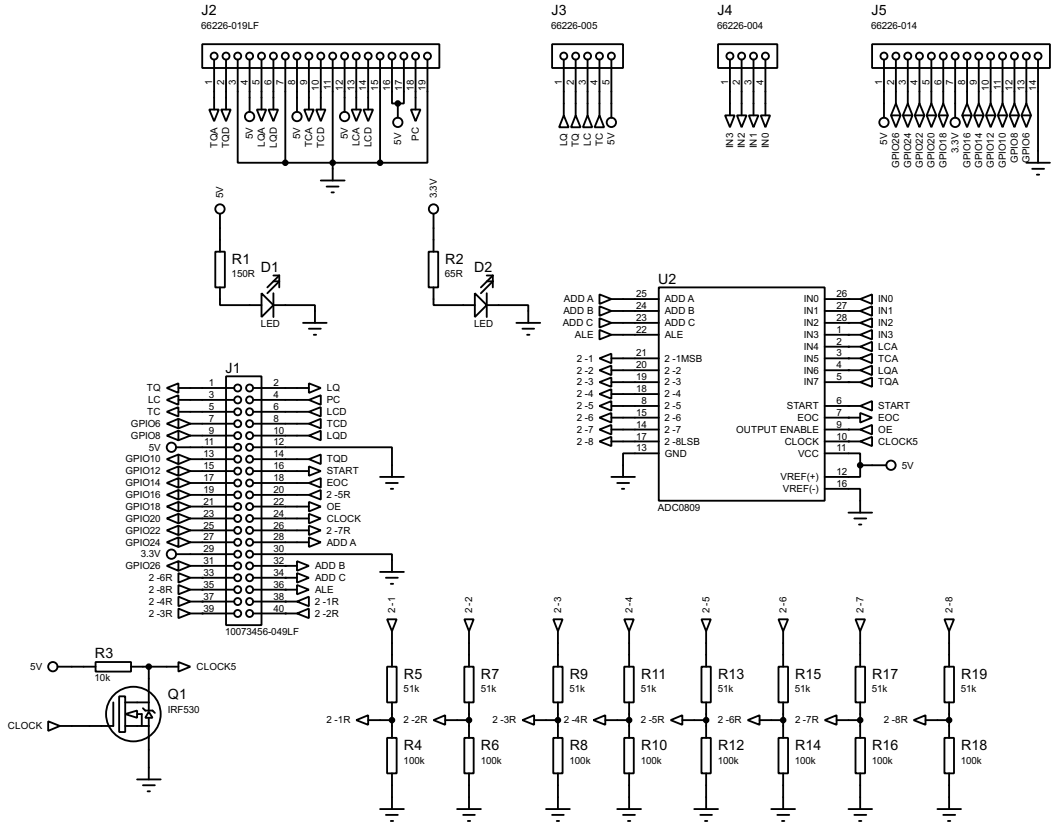
Fonte: Autoria Própria.

APÊNDICE B – DESENVOLVIMENTO DAS PLACAS DE INTERFACE

O desenvolvimento das placas foi feito em três etapas: a elaboração do esquemático, o desenho das trilhas e a confecção.

O desenho das trilhas, chamado *layout*, partiu do desenvolvimento do esquemático, feito através de um *software* específico para essa aplicação. O esquemático é apresentado na Figura 64. O *layout* foi realizado com auxílio de outro *software*. As dimensões utilizadas no desenho das trilhas foi além do mínimo estabelecido pela ABNT, com o intuito de facilitar na confecção da placa de forma artesanal.

Figura 64 – Esquemático das placas de interface.

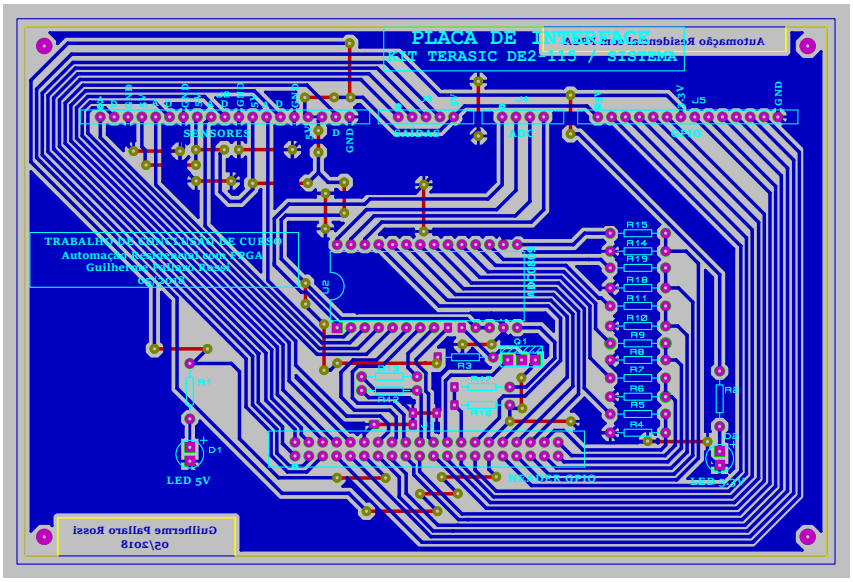


Fonte: Autoria Própria.

A Figura 65 mostra o projeto do *layout* da placa, contendo todas as camadas. As partes espelhadas representam o lado inferior, pois a imagem é vista a partir da parte superior. A confecção foi feita em placas de fibra de vidro revestida com cobre e contendo apenas uma face condutora. O processo de transferência foi o térmico, corroendo-se o cobre com ácido.

As fotos apresentadas na Figura 66 apresentam o processo de montagem das placas. Após a corrosão e serigrafia, os componentes foram adicionados e soldados. Os

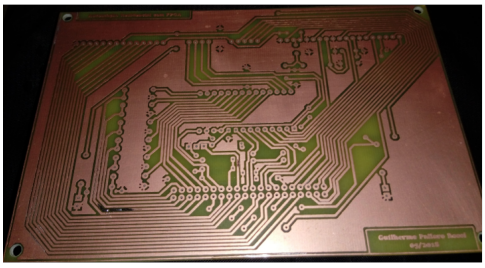
Figura 65 – *Layout* das placas de interface.



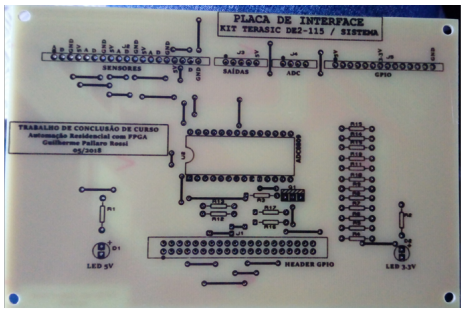
Fonte: Autoria Própria.

LEDs foram colocados para indicar que as placas estão energizadas. Um indica que tem 5 V e o outro 3,3 V. As fontes de alimentação foram obtidas dos pinos de tensão da GPIO.

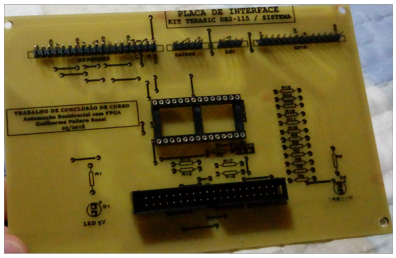
Figura 66 – Fabricação das placas de interface.



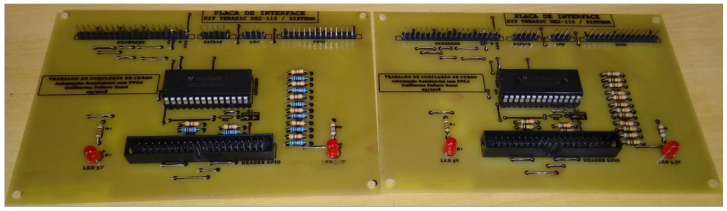
Parte inferior após corrosão.



Parte superior após serigrafia.



Montagem da placa.



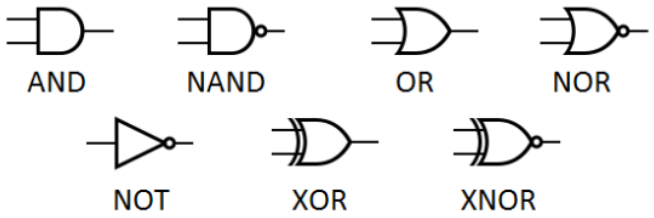
Fabricação concluída.

Fonte: Autoria Própria.

ANEXO A – ELEMENTOS DE CIRCUITOS DIGITAIS

Os circuitos digitais normalmente são compostos por portas lógicas. As principais são NOT, AND, OR, XOR e suas complementares NAND, NOR e XNOR. A Figura 67 apresenta os símbolos de cada uma das portas lógicas.

Figura 67 – Portas lógicas.



Fonte: ATHOS ELECTRONICS, 2018.

A porta NOT é uma inversora. Se o sinal que está na entrada for 1, a saída será 0 e vice-versa. O número 1 representa nível lógico alto, enquanto 0 representa nível lógico baixo. Na prática, são níveis de tensões diferentes (ATHOS ELECTRONICS, 2018).

A porta AND faz uma análise das entradas para determinar a saída. A saída só será 1 se todas as entradas forem 1. Caso contrário, havendo pelo menos uma entrada em nível baixo, a saída será 0 (ATHOS ELECTRONICS, 2018).

A porta OR terá saída 1 se ao menos uma das entradas for 1. Ela só será 0 se todas as entradas forem nulas (ATHOS ELECTRONICS, 2018).

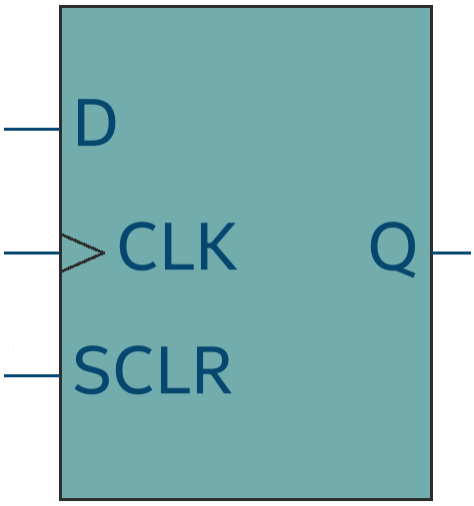
A porta XOR é chamada de OU EXCLUSIVO. Isso significa que a saída será alta se tiver 1 em apenas uma entrada. Se duas ou mais entradas estiverem altas, a saída será 0 (ATHOS ELECTRONICS, 2018).

As portas NAND, NOR e XNOR são nada menos que a porta correspondente com uma inversora NOT na frente. Por exemplo, a saída da NAND será 0 se qualquer uma das entradas for 1, funcionando de forma contrária à AND. O mesmo vale para as outras, que possuem correspondência com a OR e a XOR, respectivamente (ATHOS ELECTRONICS, 2018).

Também é comum o uso d elementos chamados *flip-flops*, que são circuitos construídos a partir das portas lógicas primitivas. O mais comum é o *flip-flop* tipo D, apresentado na Figura 68.Este possui três entradas e uma saída. As entradas são D, CLK e CLR, esta última para desativar a saída, que normalmente fica conectada em nível alto para que se tenha uso constante do elemento. A saída será igual à entrada D sempre que

houver uma borda de subida em CLK. Para utilizar borda de descida, pode-se acrescentar uma NOT na entrada CLK (PEDRONI, 2010).

Figura 68 – *Flip-flop* tipo D.



Fonte: Autoria Própria.

Os *flip-flops* são utilizados para criar dependência em circuitos lógicos, pois fazem um bloco de circuito ser acionado a partir da mudança de estado de outros sinais.

ANEXO B – COEFICIENTES DE STEINHART & HART

A equação de Steinhart & Hart, para calcular o valor da temperatura medida por um termistor, é descrita por:

$$T = \frac{1}{A + B + \ln(R) + C[\ln(R)]^3}$$

Para encontrar a fórmula dos coeficientes, basta resolver um sistema linear com 3 variáveis. Para isso, deve-se conhecer três valores de temperatura e suas respectivas resistências. O ideal é que se escolha temperaturas espaçadas uniformemente e com pelo menos 10 Kelvin de diferença (AMETHERM, 2013).

A seguir, será apresentado, de forma resumida, o desenvolvimento das equações para se chegar no cálculo das constantes, baseando-se em Ametherm (2013).

$$\frac{1}{T_1} = A + B\ln(R_1) + C[\ln(R_1)]^3$$

$$\frac{1}{T_2} = A + B\ln(R_2) + C[\ln(R_2)]^3$$

$$\frac{1}{T_3} = A + B\ln(R_3) + C[\ln(R_3)]^3$$

$$L_1 = \ln(R_1),\ L_2 = \ln(R_2),\ L_3 = \ln(R_3)$$

$$Y_1 = \frac{1}{T_1},\ Y_2 = \frac{1}{T_2},\ Y_3 = \frac{1}{T_3}$$

$$\gamma_2 = \frac{Y_3 - Y_2}{L_2 - L_1},\ \gamma_3 = Y_3 - Y_1 L_3 - L_1$$

$$\boxed{C = \left(\frac{\gamma_3 - \gamma_2}{L_3 - L_2}\right)(L_1 + L_2 + L_3)^{-1}}$$

$$\boxed{B = \gamma_2 - C\left({L_1}^2 + L_1 L_2 + {L_2}^2\right)}$$

$$\boxed{A = Y_1 - L_1\left(B + C{L_1}^2\right)}$$

ANEXO C – DEFINIÇÃO DE LÂMPADAS, VENTILADORES E TOMADAS

Dentre os diversos atuadores que podem ser utilizados na automação residencial, os mais comuns são lâmpadas e ventiladores. Para tratar de questões de segurança, utiliza-se os alarmes, que possuem uma série de sistemas diferentes.

C.1 LÂMPADAS

Uma lâmpada é um dispositivo que atua na luminosidade de um ambiente. De acordo com Ecocasa (2014), são cinco os principais tipos de lâmpada utilizadas em residências: lâmpadas fluorescentes compactas, lâmpadas fluorescentes tubulares, LED, lâmpadas de halogêneo e lâmpadas incandescentes. Dentre essas, as mais utilizadas são as incandescentes, as fluorescentes e as de LED. Cada tipo de lâmpada possui funcionamento e características diferentes, como eficiência energética e durabilidade.

A lâmpada com o princípio de funcionamento mais simples é a do tipo incandescente. Funcionam como resistores, nas quais uma diferença de potencial aplicada sobre um filamento resistivo gera luminosidade e, consequentemente, calor. Thomas Edison (1847-1931) foi o cientista que levou crédito pela invenção, por descobrir as melhores ligas metálicas resistentes ao calor. O filamento utilizado é de tungstênio, que fica dentro de um bulbo com uma atmosfera sem oxigênio. Ao aplicar tensão sobre a lâmpada, o calor gerado é suficiente para produzir luz. No entanto, esse tipo de lâmpada é o de mais baixa eficiência, pois perde muita energia em calor (90% a 95% calor e 5% a 10% em luminosidade). Na Europa, por exemplo, elas foram proibidas de serem comercializadas entre 2009 e 2012, devido à sua baixa eficiência (BRAGA, 1996; MATTEDE, 2018; ECOCASA, 2014).

A lâmpada fluorescente é composta por um tubo de vidro revestido com fósforo branco, composto de vapor de mercúrio e argônio sob baixa pressão em seu interior, filamentos e eletrodos revestidos de óxido. Ao serem submetidos a uma corrente elétrica, os filamentos se aquecem, liberando elétrons, que entram em contato com os gases no interior do tubo. Com o choque dos elétrons, o gás no interior da lâmpada é ionizado, ou seja, a corrente elétrica excita o vapor de mercúrio, fazendo-o produzir luz ultravioleta. Essa radiação é convertida em luz brilhante e visível quando entra em contato com o fósforo que reveste o interior da lâmpada. Esse tipo de lâmpada possui algumas vantagens, como eficiência energética maior do que as incandescentes, baixo custo de produção, longa vida, boa temperatura de cor e boa difusão da luz, mas também algumas desvantagens, como oscilações de alta frequência que podem causar incômodo e provocar interferência de rádio em aparelhos eletrônicos (JÚNIOR, 2018; NOVA ELETRÔNICA, 2018).

As lâmpadas mais eficientes são as de LED. Para entender seu funcionamento, basta compreender como os LEDs trabalham. O LED é basicamente um diodo que, quando é diretamente polarizado, os elétrons livres cruzam a junção *pn* e caem nas lacunas. Como esses elétrons vão de um nível de energia mais alto para um mais baixo, eles irradiam energia. Nos diodos comuns, essa energia é dissipada em forma de calor, mas, nos LEDs, ela produz luz. Enquanto os diodos comuns são feitos de silício, um material opaco que não permite a passagem de luz, os LEDs são feitos de elementos como gálio, arsênico e fósforo, o que permite a emissão de luz. De acordo com o material utilizado, o LED emite uma cor diferente (MALVINO, 1996, p. 161). A desvantagem das lâmpadas de LED é que elas são mais caras, mas, por serem mais econômicas, menos perigosas e mais duráveis, elas estão substituindo as lâmpadas incandescentes e fluorescentes (NOVA ELETRÔNICA, 2018).

C.2 VENTILADOR

Na automação residencial, variáveis importantes a serem tratadas são a temperatura e o calor. Existem diversas formas de fazer isso, como o uso de aparelhos de ar condicionado, exaustores, climatizadores, aquecedores, ventiladores, entre outros.

O método mais simples e econômico é o uso de ventiladores. Os ventiladores possuem pás que, ao se movimentarem, deslocam o ar e produzem vento. Este vento entra em contato com a pele e gera uma troca de calor, o que promove a sensação de refrescamento, pois faz o suor evaporar da pele, e esta é a principal maneira de eliminar o calor do corpo (CLIMABRISA, 2018).

Os ventiladores são encontrados em vários tamanhos, materiais e tipos, como os de mesa, teto, parede e chão. Por terem baixo custo e facilidade de instalação, são amplamente comercializados e utilizados. No entanto, é preciso levar em conta de que eles não diminuem a temperatura ambiente, apenas amenizam o calor. Mas estima-se que os ventiladores podem reduzir a sensação térmica do ambiente em até três graus (CLIMABRISA, 2018).

C.3 ALARME

Na automação residencial, o quesito segurança também é muito importante. Sistemas de proteção contra roubos, incêndios, inundações ou qualquer outro perigo são essenciais para quem quer se sentir mais seguro em sua residência.

Existem diversos tipos de sistemas de segurança. Eles variam desde os mais simples, que funcionam apenas com sensores, central de controle e sirene, até os mais complexos, com câmeras, cerca elétrica, entre outros (VERGANI, 2018).

De acordo com Vergani (2018), os sistemas de alarme são bons sistemas contra roubos, sendo compostos por três estruturas: sensores de movimento, central de controle

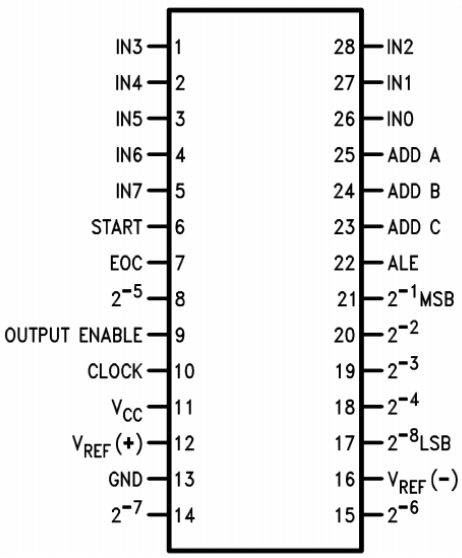
e o próprio aparelho de alarme. A vantagem desses sistemas é que são mais simples e econômicos. O som emitido pela sirene alerta os moradores, além de poder repelir o intruso.

Mas não é só contra roubos que o sistema de alarme pode ser utilizado. De acordo com Tecnohold (2018), existem os alarmes de incêndio, que são constituídos, normalmente, por detectores de fumaça, acionador, alarme e sinalizador. Esse tipo de alarme detecta um incêndio, aciona uma sirene e até mesmo válvulas para liberarem água, assim como podem notificar o corpo de bombeiros, evitando que o estrago seja grande ou, pelo menos, aumentando as chances de sobrevivência das pessoas.

ANEXO D – DESCRITIVO FUNCIONAL DO ADC0809

O ADC0809 é um conversor analógico-digital fabricado pela Texas Instruments. A Figura 69 apresenta a pinagem do ADC0809, cujas funções dos pinos são descritas no Quadro 1.

Figura 69 – Pinagem do ADC0809.



Fonte: TEXAS INSTRUMENTS, 2013.

O CI requisita uma tensão de $4,5$ a $5,25$ V para alimentação, entradas de controle com tensões acima de $VCC - 1,5$ V para nível lógico alto e até $1,5$V para nível lógico baixo, tensões de referência que podem variar dentro da faixa da tensão de alimentação e *clock* de 10 kHz a 1280 kHz. Os níveis de tensão de saída são equivalentes aos de alimentação e o sinal analógico nas entradas podem variar dentro da faixa de referência (TEXAS INSTRUMENTS, 2013).

O ADC0809 é constituído de três seções maiores em sua construção: a rede de 256 resistores em escada, o registrador de aproximação sucessiva e o comparador. Os resistores das extremidades da rede possuem valores diferentes dos demais, para que a saída seja simétrica em relação aos pontos mínimo e máximo da escala. O registrador de aproximação sucessiva (SAR - *Successive Approximation Register*) realiza oito iterações para aproximar a tensão de entrada. A Figura 70 ilustra a rede dos 256 resistores em escada, com as chaves controladas pelo SAR (TEXAS INSTRUMENTS, 2013).

O comparador, por sua vez, é responsável pela precisão final do conversor. É também o seu desvio que tem a maior influência na repetibilidade do dispositivo. O conversor possui um comparador do tipo *chopper-stabilized*, que realiza a técnica de

Quadro 1 – Funções dos pinos do ADC0809.

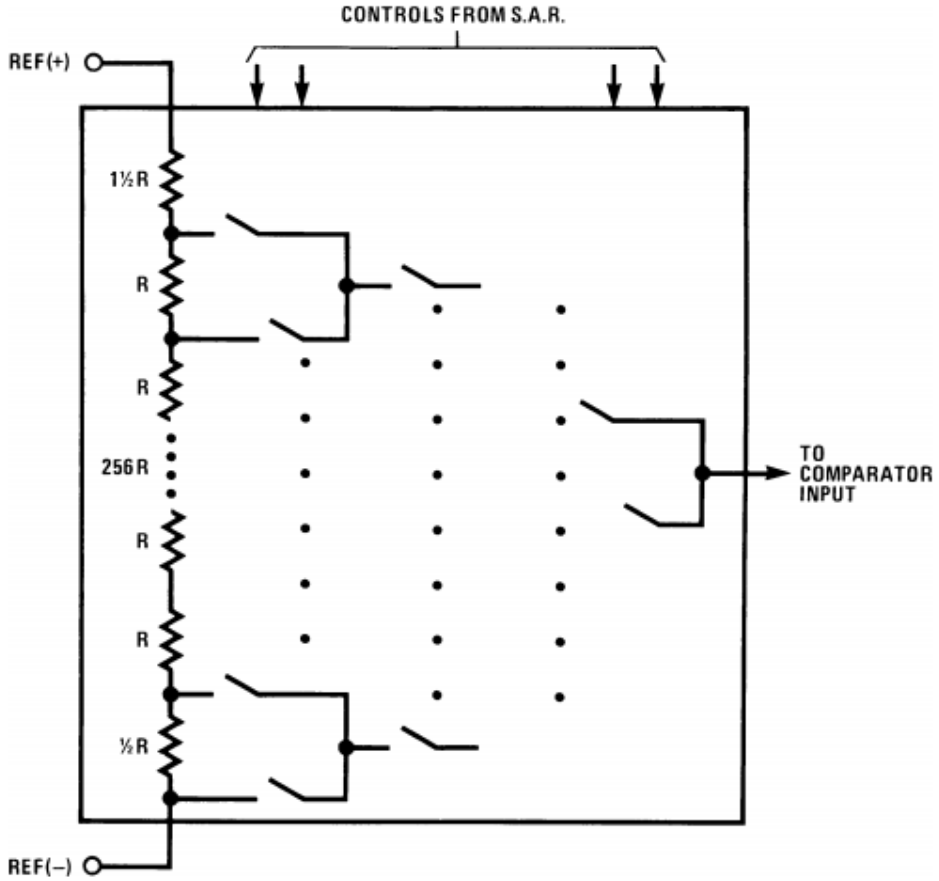
Pino	**Nome**	**Função**
1	IN3	Canal de entrada 3
2	IN4	Canal de entrada 4
3	IN5	Canal de entrada 5
4	IN6	Canal de entrada 6
5	IN7	Canal de entrada 7
6	START	Inicia conversão
7	EOC	*End of Conversion* - Indica fim da conversão
8	2^{-5}	Bit 3 de saída
9	OUTPUT ENABLE	Habilita saídas
10	CLOCK	Entrada de clock
11	VCC	Alimentação
12	$V_{REF}(+)$	Tensão maior de referência
13	GND	Terra
14	2^{-7}	Bit 1 de saída
15	2^{-6}	Bit 2 de saída
16	$V_{REF}(-)$	Tensão menor de referência
17	2^{-8} LSB	Bit 0 de saída - LSB (*Least Significant Bit*) - Bit menos significante.
18	2^{-4}	Bit 4 de saída
19	2^{-3}	Bit 5 de saída
20	2^{-2}	Bit 6 de saída
21	2^{-1} MSB	Bit 7 de saída - MSB (*Most Significant Bit*) - Bit mais significante
22	ALE	*Address Latch Enable* - Habilita canal de entrada selecionado
23	ADD C	Linha de endereço C - MSB
24	ADD B	Linha de endereço B
25	ADD A	Linha de endereço A - LSB
26	IN0	Canal de entrada 0
27	IN1	Canal de entrada 1
28	IN2	Canal de entrada 2

Fonte: Autoria própria.

converter o sinal de entrada em corrente contínua (CC) para um sinal de corrente alternada (CA) e, então, amplificá-lo através de um amplificador CA de alto ganho, limitando o componente de desvio do amplificador, já que o desvio é um componente CC que não é passado pelo amplificador CA. Isso faz com que o ADC seja insensível à temperatura, desvios e outros erros de *offsets* na entrada (TEXAS INSTRUMENTS, 2013).

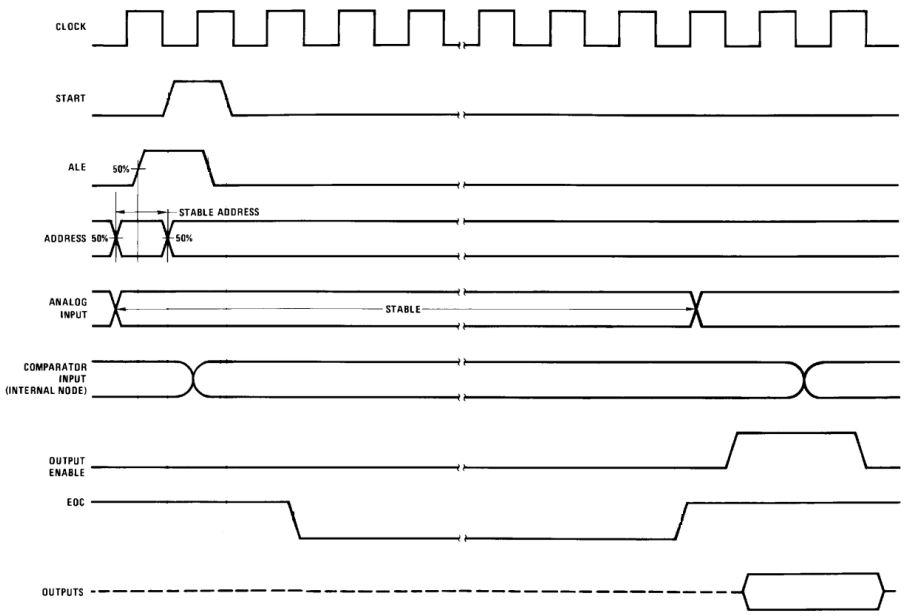
A conversão segue o gráfico da Figura 71. Após o pulso no START, o sinal no canal determinado pelo ADDRESS e ativado pelo ALE é lido e convertido. Ao final da conversão, a saída EOC gera um pulso indicando o término. Assim, as saídas podem ser ativadas para leitura pelo controlador (TEXAS INSTRUMENTS, 2013)

Figura 70 – Rede de 256 resistores em escada.



Fonte: TEXAS INSTRUMENTS, 2013.

Figura 71 – Diagrama de tempo do funcionamento do ADC0809.



Fonte: Adaptado de Texas Instruments (2013).

ANEXO E – MODELO OSI

O modelo OSI é uma arquitetura composta de sete camadas, conforme apresentado na Figura 72. Em teoria, cada camada é de responsabilidade de um protocolo, mas, na prática, a maioria das pilhas de protocolos não segue esse modelo exatamente (TORRES, 2009).

Figura 72 – Modelo OSI de protocolos.

Fonte: Adaptado de Torres (2009).

Torres (2009) divide as camadas em três grupos. As camadas de 1 a 3 formam o grupo de rede, que é responsável pela transmissão e recepção dos dados através da rede e, por isso, são camadas de baixo nível. A camada 4, que forma o grupo de transporte, tem responsabilidade de tratar os dados recebidos pela rede para repassá-los às camadas de aplicação de uma forma compreensível. O grupo de aplicação, que é composto pelas camadas de 5 a 7, é responsável por colocar os dados recebidos em um padrão que seja compreensível pelo programa que fará seu uso.

A seguir, são descritas as funções de cada camada do modelo OSI:

- **Aplicação:** A camada de aplicação realiza a interface entre a arquitetura de rede e o aplicativo que pediu ou receberá a informação (TORRES, 2009).
- **Apresentação:** Esta camada está relacionada à sintaxe e à semântica das informações transmitidas. Para que seja possível a comunicação entre computadores com diferentes representações internas dos dados, ocorre uma tradução para uma codificação padrão que será usada durante a conexão. Esta tradução é responsabilidade da camada de apresentação (TANENBAUM; WETHERALL, 2011).
- **Sessão:** A camada de sessão permite que dois computadores diferentes estabeleçam uma sessão de comunicação entre as aplicações. Nesta sessão, as aplicações

determinam a forma que a transmissão será feita e coloca marcações nos dados, para determinar um ponto de reinício no caso de falhas, que será a partir da última marcação recebida pelo computador receptor (TORRES, 2009).

Além disso, Tanenbaum e Wetherall (2011) afirmam que as sessões de comunicação também oferecem serviços como controle de diálogo, determinando quem deve transmitir em cada momento, e gerenciamento de *tokens*, impedindo que duas partes executem a mesma operação crítica ao mesmo tempo.

- **Transporte:** A camada de transporte tem a função de dividir em vários pacotes os dados recebidos pela camada de sessão, para depois repassá-los à camada de rede. No receptor, sua função é receber os pacotes passados pela camada de rede e remontar o dado original, para enviá-lo à camada de sessão. Além disso, ela deve verificar para qual protocolo operando acima dela os dados devem ser entregues, porque é possível ter vários protocolos operando simultaneamente (TORRES, 2009).

- **Rede:** A camada de rede faz o endereçamento lógico dos pacotes de dados e também a tradução de endereços lógicos em físicos. Além disso, trata a prioridade da entrega de determinados pacotes de dados (TANENBAUM; WETHERALL, 2011).

  Os pacotes de dados precisam usar endereçamentos físicos e lógicos. O endereçamento lógico é independente da arquitetura utilizada, o que permite que a transmissão seja feita através de diferentes arquiteturas de rede. O endereçamento físico varia conforme a arquitetura, e é responsabilidade da camada de rede tratar isso (TORRES, 2009).

- **Link de Dados:** A camada de link de dados recebe os pacotes da camada de rede e os transforma em quadros que serão enviados pela rede. Esta camada adiciona informações como o endereço de origem e o de destino, dados de controle, os próprios dados e o *checksum*, ou soma de verificação, que é um código usado para averiguar a integridade dos dados transmitidos. Caso o tamanho do quadro usado pela rede seja menor do que o pacote de dados, o pacote será dividido em quadros menores, quantos se fizerem necessários, para que ocorra a transmissão completa (TORRES, 2009).

  Esta camada também é usada para verificar se o meio físico está disponível e pode ser usado. Além disso, por ser uma camada de baixo nível, a camada de link de dados é controlada por *hardware*, assim como a camada física, enquanto que as camadas superiores são controladas por *software* (TORRES, 2009).

- **Física:** Esta camada recebe os quadros enviados pela camada de link de dados e os converte em sinais compatíveis com o meio físico em que o sinal será transmitido. Portanto, a camada física especifica a forma com que os 0s e 1s dos quadros serão enviados ou recebidos pela rede, apesar dela não interpretar o significados dos dados.