



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Sistema FPGA controlável remotamente

Luan Talles da Silva Araújo

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia de Computação

Orientador
Prof. Dr. Daniel Chaves Café

Brasília
2019



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Sistema FPGA controlável remotamente

Luan Talles da Silva Araújo

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia de Computação

Prof. Dr. Daniel Chaves Café (Orientador)
ENE/UnB

Prof. Dr. Alexandre Nery Solon Prof. Dr. Daniel Mauricio Muñoz Arboleda
ENE/UnB FGA/UnB

Prof. Dr. José Edil Guimarães de Medeiros
Coordenador do Curso de Engenharia de Computação

Brasília, 28 de junho de 2019

Dedicatória

Dedico a todos os cotistas que não desistiram.

Agradecimentos

Meus sinceros agradecimentos ao meu orientador, professor Dr. Daniel Chaves Café, por aceitar ser meu orientador, por me ensinar bastante, pela amizade e por, mesmo com meus altos e baixos, não desistir de me orientar. Gostaria de agradecer também ao professor Dr. Daniel Mauricio Muñoz Arboleda pela sugestão do tema, pelas contribuições, principalmente, com o desenvolvimento do texto e por aceitar ser membro da minha banca. Meus agradecimentos também ao professor Dr. Alexandre Nery Solon pelas contribuições com o texto do projeto, por aceitar ser membro da minha banca e pela sugestão de publicar o projeto em alguma conferência.

A minha mãe, Zilda Pereira da Silva, a pessoa mais guerreira que eu conheço, por sempre me apoiar, acreditar em mim e possibilitar tudo que venho conquistando e vou conquistar. A minha irmã, Larissa S. Araujo, pelo companheirismo, preocupação, incentivo e por ser meu ponto de inspiração. Ao meu pai, Carlos A. de Araujo, por seu companheirismo e preocupação. A minha namorada, Luiza A. Moura, pelo apoio e companheirismo, que perdura por mais de cinco anos. A todos os meus familiares e parentes, em especial as minhas tias Dilma, Luzia, Edna, Eva, Dilça, Sirlene e Divina e meus tios Adão e Flávio, que foram muito importantes na criação e amadurecimento não só meu, mas de todos os jovens da minha família; a minha vó Amélia pelo carinho e cuidados nas longas férias e aos meus avós falecidos que, com certeza me apoiariam. Amo todos vocês.

Gostaria de agradecer também aos meus amigos: João Henrique, Pedro de Lyra, Helton Isamu, Victor Unoske, meus irmãos de coração, Rafael Franco, William Tavares, Pedro Victor, Paulo Shinji, Reinaldo Itaborai, Wanderson Almeida, Ena de Almeida, Marlla Alves, Davi Rodrigues, Maria Julia, Beatriz Alencar, Priscilla Rodrigues, Maria da Luz (1974-2014), Marcelo Tutida, Luana, Laryssa, Arthur e Thiago; meu grande amigo João da Silva pelas caronas "carrão", ao Reibe por me incentivar com a matemática, aos meus amigos do INEI COC, aos meus amigos da Escola Fundamental Alvacir Vite Rossi, em especial ao Mateus Alves; e a todos os meus amigos de Buritinópolis, em especial ao Rodrigo Jesus (1993-2015), a todos os meus amigos da Escola Classe 403 Norte, a todos os meus amigos da QMS 23, aos meus colegas do Centro Acadêmico de Engenharia de Computação, do time de futebol do CAEC, do Dibre, do Gol++, do CDF UnB,

da escolinha do Flamengo, do Sodeso, do Santos, meu primeiro técnico (seu Valdenor), ao pessoal da farmácia com quem interagi por meio da Luiza e muitos outros que não caberiam nessa folha.

Aos professores e funcionários das instituições de minha graduação, ensino médio, fundamental, pré-escolar e maternal, por contribuírem com minha formação tanto pessoal bem como intelectual. Sem eles tudo seria mais difícil.

À Universidade de Brasília, pelas diversas oportunidade que me proporcionou.

Por último, e mais importante, a Deus, por tudo nessa vida.

Resumo

As Field Programmable Gate Arrays são dispositivos extremamente importantes para o desenvolvimento da eletrônica digital, pois permitem a construção e a prototipação de diversos circuitos digitais de lógica simples ou complexa. Entretanto, possuem um alto custo de aquisição. Esse fator, no meio acadêmico, é extremamente sensível, pois não é qualquer pesquisador/aluno que consegue adquirir um *hardware* desses. Além disso, quando adquiridos, não são colocados à disposição de qualquer usuário, justamente pelo seu alto valor. Do ponto de vista de ensino, a demanda de uso de FPGAs é grande. Alunos querem poder prototipar seus experimentos em horários livres, fora da sala de aula. Do ponto de vista da pesquisa, ser capaz de utilizar dispositivos à distancia, permite colaboração entre pesquisadores de diferentes instituições. Diante desses problemas, um estudo foi feito e uma solução elaborada: desenvolver um sistema em que as FPGAs são controladas por uma aplicação *web*, tornando disponíveis equipamentos programáveis de alto custo de aquisição à alunos e pesquisadores, de maneira prática, dispensando a necessidade de presença física dos usuários no laboratório, processos burocráticos de empréstimos de equipamentos e configurações de *drivers* e *softwares*. Além disso, aumenta-se a disponibilidade dos equipamentos, fornecendo serviço, em feriados, finais de semana, madrugadas e outros horários pouco acessíveis, em plenitude, à comunidade acadêmica.

Palavras-chave: FPGA, dispositivos remotos, aplicação, pesquisa, ensino

Abstract

The Field Programmable Gate Arrays are extremely important to the development of digital electronics, once they allow building and prototyping diverse digital circuits with simple and complex logics. However, its acquisition cost is high. In the academic setting, this is an extremely sensitive factor, because not every student or researcher can acquire one of them. Besides that, when acquired, they are not available to all users, precisely because it's high cost. From a teaching point of view, the demand for using FPGAs is high. Students want to prototype their experiments during their free time, out of classroom. From a researching point of view, being able to use remote devices allows that researchers from different institutions collaborate. Regarding these problems, a study and a solution were elaborated: developing a system in which FPGAs are controlled through an application, making available high cost programmable equipment's, in a practical manner, preventing the need of physical presence of the users in the laboratory, bureaucratic processes for borrowing equipment's and the configuration of drivers and software's. Besides that, it increases the equipment's availability, providing service during holidays and weekends, late night and other times less accessible to the academic community.

Keywords: FPGA, remote devices, application, research, teaching

Sumário

1	Introdução	1
1.1	Introdução	1
1.1.1	Objetivo geral	4
1.1.2	Objetivos específicos	5
2	Revisão Bibliográfica	7
2.1	Ruby on Rails	7
2.1.1	Ruby	8
2.1.2	MVC – Model, View and Controller	8
2.1.3	<i>Frameworks</i> do Rails	9
2.2	Dispositivos Lógicos Programáveis	10
2.2.1	FPGA	11
2.3	Microcontroladores	13
2.4	Tecnologias utilizadas	15
3	Projeto	17
3.1	Metodologia de desenvolvimento	17
3.1.1	Desenvolvimento da interface de usuário	17
3.1.2	Desenvolvimento do servidor, <i>scripts</i> do controle do microcontrolador e da FPGA	21
3.1.3	Firmware do microcontrolador	23
3.2	Metodologia de teste	24
4	Resultados	27
4.1	Resultados	27
5	Conclusão	29
5.1	Conclusão	29
5.1.1	Trabalhos futuros	30

Lista de Figuras

1.1	Diagrama conceitual do sistema.	4
2.1	Arquitetura MVC.	9
2.2	PLD como uma “caixa preta”.	11
2.3	Arquitetura interna de um FPGA, formada por CLBs, IOBs e matrizes de comutação [29].	13
2.4	Lookup table (LUT) de duas entradas.[11].	14
2.5	Arquitetura interna de um bloco lógico configurável (CLB).[11].	14
2.6	Informações do ambiente de desenvolvimento da aplicação.	16
3.1	Diagrama Entidades Relacionamento do projeto. Representação gráfica em notação tradicional.	18
3.2	Diagrama Casos de Uso do projeto	19
3.3	Interface de usuário do sistema.	21
3.4	Firmware do microcontrolador.	24

Lista de Tabelas

Lista de Abreviaturas e Siglas

- ASIC** Application Specific Integrated Circuits. 11
- CI** Integrated Circuit. 13
- CLB** Configurable Logic Block. 12
- CLP** Controlador Lógico Programável. 11
- CLPD** Complex Programmable Logic Device. 11
- CoC** Convention over Configuration. 7
- CPU** Central Processing Unit. 15
- DRY** Don't Repeat Yourself. 7
- FPGA** Field Programmable Gate Array. vii, 2–4, 11, 12, 16, 24, 25, 27–30
- GUI** Graphical User Interface. 3
- HTML** Hypertext Markup Language. 10
- IOB** Input/Output Block. 12
- LCD** Liquid Crystal Display. 30
- LDAP** Lightweight Directory Access Protocol. 30
- LUT** LookUp Table. 12
- MIPS** Microprocessor *without interlocked pipeline stages*. 2
- MVC** Model-View-Controller. xi, 8–10, 21–23
- ORM** Object-*relational mapping*. 15, 21, 22

PAL Programmable Array Logic. 11

PLA Programmable Logic Array. 11

PLD Programmable Logic Device. xi, 1, 2, 10, 11

PROM Programmable Read-Only Memory. 10

RAM Random Access Memory. 12

ROM Read-Only Memory. 10, 12

RoR Ruby on Rails. 7, 9, 15

SGBD Sistema Gerenciador de Banco de Dados. 7

ULA Unidade Lógica e Aritmética. 2

UML Unified Modeling Language. 18

USB Universal Serial Bus. 5, 13, 29

WAN Wide Area Network. 26, 30

Capítulo 1

Introdução

Este documento apresenta a metodologia, a implementação e os resultados esperados do projeto

1.1 Introdução

A tecnologia vem evoluindo em um ritmo bastante acelerado e tem se mostrado bastante importante no desenvolvimento da sociedade. Inteligência artificial, *internet* das coisas (*IOT*), carros autônomos, etc. já são realidade. É bastante comum nos depararmos com objetos do presente que no passado não fazíamos ideia de que elas existiriam. Uma desses objetos são dispositivos de lógica programável.

Antigamente, os circuitos digitais integrados eram formados a partir da combinação de circuitos com funções fixas. Era necessário a fabricação do circuito integrado de interesse do usuário, ao final de todo um projeto, geralmente bastante custoso, pouco suscetíveis a mudanças e com grandes ciclos de duração.

À medida que a tecnologia evoluiu, surgiram os PLDs (Programmable Logic Device) e essa ideia de ter necessariamente de fabricar o circuito integrado alterou um pouco. Com isso, passou a ser possível o usuário possuir um dispositivo com funções *booleanas* programáveis.

Esses dispositivos revolucionaram a indústria da época, com essa ideia bastante promissora. Os ciclos de projetos de circuitos agora poderiam ser menores. E, ainda, seria possível usar esses dispositivos para prototipação dos circuitos integrados, criando protótipos de circuitos de lógica fixa, e, posteriormente, testando-os e fabricando-os, o que representou um ganho muito grande na indústria de hardware, tornando-os mais suscetíveis a mudanças.

Os anos foram passando e essa tecnologia foi se desenvolvendo, tornando-se mais complexa, robusta e com menor custo. Vários tipos de PLDs foram criadas, das quais cabe

destacar o subgrupo das FPGAs (Field Programmable Gate Array), que são os tipos PLDs mais conhecidos no mercado de *hardware*.

O contrapeso dessa tecnologia é seu alto custo de obtenção, principalmente para usuários comuns. Não é qualquer pessoa que consegue obter uma FPGA para consumo próprio. Apesar de ter diminuído, o custo dessa tecnologia ainda é relativamente alto, por isso ainda é inviável para esse tipo de usuário. Geralmente, as pessoas que têm acesso a essa tecnologia o têm por meio de laboratórios acadêmicos a que estão inseridas.

A FPGA é extremamente importante para o desenvolvimento da eletrônica digital. No meio acadêmico, as FPGAs são bastante utilizadas pelos alunos cujos currículos possuem disciplinas de eletrônica digital. Isso porque a FPGA possibilita a construção de circuitos digitais de forma simples, através de linguagens de descrição de *hardware* ou ferramentas de modelagem de circuitos, presentes na maioria dos *softwares* das fabricantes de FPGAs. Diversas bibliotecas com componentes de hardware são disponibilizadas na *internet*, aumentando a abstração de diversos projetos complexos. Tarefas bastante complexas, como criar uma arquitetura de computador MIPS 32 bits, se tornaram muito mais simples de serem implementadas em uma FPGA em relação ao caso de ter de implementá-las com componentes de *hardware*, como ULAs, multiplexadores etc.

As instituições de ensino disponibilizam esses dispositivos programáveis. Porém, nos casos de dispositivos mais poderosos, estes são encontrados em pequenas quantidades. Isso acaba limitando o desenvolvimento de projetos. Os alunos não conseguem comprar esses dispositivos por conta própria e também encontram dificuldade de acesso pela quantidade de oferta/demanda associado ao horário disponível para eles ao longo do dia na instituição.

O cenário ideal que possibilitaria um melhor desenvolvimento da eletrônica digital seria um que tornasse esses dispositivos disponíveis o tempo todo, sem a necessidade de configurações e com bastante oferta de dispositivos. Assim, os usuários detentores de projetos poderiam desenvolvê-los quando quisessem, sem se preocuparem com fatores elencados, o que ajudaria no desenvolvimento da área como um todo.

Algumas soluções, disponíveis principalmente na área de ensino, já foram propostas e implementadas, conforme indica a literatura. Elas foram desenvolvidas de forma a tornar alguns dos problemas das FPGAs para alunos com currículos de circuitos digitais.

Nesse sentido, Bascil, Yazici e Temurtas [19] criaram um protótipo de laboratório de sistemas digitais baseado em FPGA acessível de forma remota pelos usuários, no caso, estudantes de graduação de engenharia elétrica e eletrônica. O protótipo é implementado usando uma placa de desenvolvimento DE2 70 da Altera, ligada a uma plataforma C# Microsoft Studio 2008 através de uma porta serial RS 232. Os usuários utilizam a conexão remota do servidor Windows XP para interagir com o laboratório. Um *software* proprietário da Altera, o Quartus II Web Edition Software, é utilizado pelos usuários

para dar entradas e receber as saídas da FPGA, além da saída enviada pela *webcam*. Ao final, é sinalizado que o laboratório pode ser desenvolvido com trabalhos adicionais, ter o número de plataformas multiplicado e poder ser estabelecido em um servidor através de uma página *web* para facilitar o acesso.

El-Medany [22] descreve a implantação de um laboratório FPGA remoto com 20 bancadas de desenvolvimento disponíveis. Cada bancada é implementada usando uma placa de desenvolvimento Spartan 3E da Xilinx, conectada a um computador Windows XP através de uma porta USB. Os usuários utilizam a conexão remota do servidor Windows XP para interagir com o laboratório, utilizando também da porta paralela do computador para testar o *design* gravado por eles nas placas. Um *software* proprietário da Xilinx, o Xilinx ISE Foundation, é utilizado pelos usuários para implementação e simulação do circuito desejado. As entradas e saídas da FPGA são disponibilizadas através de uma GUI implementada em Visual Basic nas telas de seus computadores pessoais. El-Medany [22] sugere que o laboratório possa se desenvolver usando uma página *web* para controlar as FPGAs, ao invés de usar acesso remoto aos computadores conectados às FPGAs, e também com acréscimo de *webcams* para enriquecer os *outputs* dos usuários em visualizar as saídas das placas em seus computadores pessoais.

Hashemian e Riddley [24] implementaram o que eles consideraram ser uma nova técnica de ensino que permite um instrutor levar à construção de *hardware*, teste e experimentos de laboratório para a sala de aula através da *internet*. Uma unidade experimental, denominada e-Lab, foi produzida para viabilizar o objetivo do trabalho idealizado pelos autores. Essa unidade experimental consiste em uma plataforma bastante semelhante a uma das vinte unidades desenvolvida por El-Medany [22], diferindo-se apenas pela GUI das duas plataformas e pela presença de uma *webcam* transmitindo, em tempo real, a placa Spartan 3E para o usuário que está controlando o servidor remotamente e pela presença também de uma unidade de aquisição de dados NI PCI-6025E/CB-100. Diferente do trabalho de El-Medany [22] em que foi desenvolvida uma GUI em Visual Basic, a plataforma e-Lab utiliza a GUI LabView da National Instruments. Os autores colocam como trabalhos futuros a possibilidade de expandir o número de bancadas e-Lab, além de possibilitar a adição de outros equipamentos de *hardware*, outras funções e tarefas.

Cardenas[13] desenvolveu, como requisito para conclusão de um estágio realizado por ele, em conjunto com o professor Dr. Daniel Chaves Café, uma Interface Gráfica em linguagem de programação Python capaz de enviar comandos seriais a um microcontrolador da família MSP430, da Texas Instruments, por uma conexão USB, permitindo que um usuário possa controlar uma FPGA que esteja conectada ao microcontrolador, através de seus pinos de entrada e de pinos de saída do microcontrolador. Os usuários podem escolher pela interface construída, a porta serial à qual desejam interagir e as métricas da

comunicação (*baud rate*, *data bits*, *stop bits*, *parity*, etc.) estabelecida com o microcontrolador. Podem interagir com a solução por meio de alguns dos componentes, desenhados em tela, com correspondente em *hardware*, da placa de desenvolvimento FPGA utilizada pelo autor (Basys3 com FPGA Artix-7). Cardenas aponta, em seu modelo conceitual, que o trabalho foi modelado para ser desenvolvido por duas pessoas: uma com conhecimento em criação de páginas *web* e conectividade entre computadores pela *internet* e outro com experiência em microcontroladores e FPGAs. Ele se encarregou da segunda parte citada e que, por falta de expertise, na primeira, deixou-a para outra pessoa implementar como trabalho futuro.

Todos esses trabalhos buscaram resolver os problemas apresentados pela baixa disponibilidade de FPGAs no meio acadêmico. Eles também serviram como inspiração para desenvolver o sistema objeto desse estudo.

1.1.1 Objetivo geral

O objetivo geral do projeto é entregar o sistema proposto como trabalho futuro por Bascil, Yazici e Temurtas [19] e El-Medany [22], que consiste em desenvolver um sistema em que as FPGAs são controladas por uma aplicação *web*, e não por meio de acesso remoto aos computadores conectados às FPGAs. Em outros termos, o projeto tem por objetivo tornar disponíveis, inclusive em feriados, finais de semana e madrugadas, equipamentos programáveis de alto custo de aquisição à comunidade acadêmica, de maneira prática, pela *internet*, através de uma aplicação *web*, dispensando a necessidade de presença física no laboratório pelos usuários, processos burocráticos de empréstimos de equipamentos e configurações de *drivers* e *softwares* dos equipamentos em casos de empréstimos.

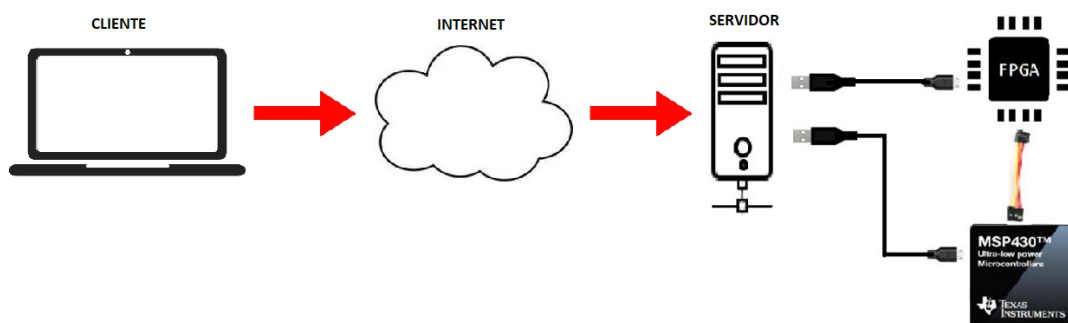


Figura 1.1: Diagrama conceitual do sistema.

A figura 1.1 acima, apresenta o diagrama conceitual do sistema, que representa o que se deseja entregar como produto. Do lado esquerdo da imagem, tem-se o usuário do sistema, representado pelo notebook cliente, o qual interage com o servidor, representado

pelo servidor, através da internet, representado pela nuvem, podendo gravar a FPGA conectada ao servidor pelo barramento USB e controlá-la por meio do microcontrolador, também conectado ao servidor pelo barramento USB.

Por se tratar de um sistema grande e complexo, a depender do grau de conhecimento do autor no domínio do problema, o projeto foi segmentado em três partes, ditas subprodutos entregáveis, que foram desenvolvidas como objetivos específicos apresentados na seção a seguir. Esses subprodutos são: 1) desenvolvimento da interface de usuário (frontend), 2) desenvolvimento do servidor, scripts de controle do microcontrolador e da FPGA (backend) e 3) firmware do microcontrolador que entende comandos do servidor e traduz em valores lógicos para a FPGA. Essa abordagem foi seguida pois permite aumentar o detalhamento do projeto, diminuir o impacto das mudanças e aumentar a precisão quanto ao detalhamento dos recursos e prazos necessários, facilitando a forma de lidar com o problema.

1.1.2 Objetivos específicos

Para entregar o objetivo específico 1, desenvolvimento da interface de usuário do sistema, foi necessário realizar, como primeira atividade entregável, uma modelagem conceitual das entidades que se relacionam no sistema. Isso gerou como resultado um diagrama de entidades relacionamento (DER). A partir desse diagrama e, utilizando dos requisitos funcionais coletados na etapa de modelagem de requisitos (atividade 2), foi gerado um diagrama (atividade 3), contendo os casos de uso do sistema, ou seja, a descrição narrativa dos eventos dos atores no sistema. Após isso, foram feitos os esboços da interface que o usuário terá contato, conhecidos como *wireframes* (atividade 4), contemplando os conteúdos e os casos de uso do sistema. Por fim, utilizando dos *wireframes*, foram implementadas as telas finais da interface de usuário (atividade 5). Uma tela que cabe destacar, é a de uso da FPGA, que contempla o *layout* de uma placa de desenvolvimento FPGA, em que foram associadas ações de teclas e ações do *mouse*/de toque na tela para os botões de pressionamento e *switchs* da placa, ações essas que serão utilizadas, posteriormente, para enviar dados para o microcontrolador.

Para entregar o objetivo específico 2, desenvolvimento do servidor, scripts de controle do microcontrolador e da FPGA, foram realizadas cinco atividades. A primeira consistiu em criar as entidades do sistema, listadas no DER, conhecidas como modelos. A segunda atividade foi criar as relações entre as entidades. Em seguida, essas entidades e relações foram migradas para o banco de dados escolhido (atividade 3), de forma a persistir os atributos dessas entidades quando elas forem instanciadas. A quarta atividade consistiu em criar os controladores de ações e os caminhos que esses controladores devem seguir para realizar as ações descritas nos casos de uso. A última atividade consistiu em concretizar

os casos de uso do sistema, ou seja, implementar as ações possíveis de serem realizadas no sistema, como fazer login, visualizar bancadas, reservar assento, instanciar e enviar comandos seriais para o microcontrolador, etc.

Para entregar o objetivo específico 3, *firmware* do microcontrolador que entende comandos do servidor e traduz em valores lógicos para a FPGA, foi realizada uma única atividade, em que foi criado um *firmware* para o microcontrolador escolhido, capaz de receber dados da porta serial, identificar de que dados se trata e traduzi-lo em valores lógicos para a FPGA. Os dados recebidos na porta serial são traduzidos em ação semelhante à realizada na FPGA virtual, a qual está disponível na tela de uso da FPGA, por exemplo, ao clicar em um *switch* na placa virtual, levantado-o ou abaixando-o, isso será traduzido na mesma ação no *hardware* da FPGA controlada pelo microcontrolador.

Capítulo 2

Revisão Bibliográfica

Este capítulo descreve uma breve fundamentação teórica relativa aos conceitos mais utilizados no projeto.

2.1 Ruby on Rails

O Ruby on Rails, também conhecido como Rails ou RoR, é um moderno *framework* livre de licença e de código aberto, usado para implementar aplicações *web*. Ele é inteiramente implementado numa única linguagem de programação, a Ruby. Foi criado por David Heinemeier Hansson, em 2003, e é considerado um *framework* de *frameworks*[18], que serão apresentados na subseção 2.1.3.

Esse *framework* é caracterizado, dentre outras coisas, pela facilidade de utilização e aprendizagem, o que o torna uma ferramenta produtiva e que mantém baixa a curva de aprendizagem. Ele segue dois conceitos com intenção de aumentar a produtividade no desenvolvimento de aplicações, o conceitoDRY (*Don't Repeat Yourself*) — “Cada parte do conhecimento deve ter uma representação única, não ambígua e definitiva dentro do sistema” — e o conceitoCoC (*Convention over Configuration*) — “Assumir valores padrão onde existe uma convenção”[26]. Outros atributos desse *framework* são: uso de metodologia ágil de desenvolvimento de software; utilização do modelo de arquitetura de software MVC; e boa integração do *framework* com SGBDs bastante utilizados no mercado, como MySQL e PostgreSQL, o que proporciona boa otimização, velocidade e restrição de integridade dos dados de uma aplicação implementada em Rails[23].

Segundo o site oficial do *framework*, o Rails foi projetado para ser uma solução de desenvolvimento completa, transparente e uniforme. Ele pode ser instalado e utilizado em diversos sistemas operacionais modernos (Distribuições Linux, Microsoft Windows e Mac OS X).

2.1.1 Ruby

Como foi apresentado, Rails é um *framework* implementado inteiramente em Ruby, que é uma linguagem de *script* - escrita em C e baseada em Perl, Python e Smalltalk -, geralmente usada para programação orientada a objetos, mas que é multiparadigma. Possui tipagem forte, dinâmica e implícita, um gerenciador de memória automático, é multiplataforma, diferencia letras maiúsculas de minúsculas e não permite herança múltipla[25].

A linguagem de programação Ruby foi criada, no início da década de 90, no Japão, por Yukihiro Matsumoto, sendo na época conhecido somente por lá e em alguns lugares dos Estados Unidos, mas que rapidamente ganhou popularidade no mundo inteiro por causa do Rails.

A filosofia de Ruby é ser uma linguagem feita por pessoas para pessoas, focada na simplicidade, colaboração e humanização. Esse foco nas pessoas está bem evidente na famosa frase do criador da linguagem "*We are the masters, they are the slaves*", que, em tradução livre, significa "Nós [humanos] somos os mestres, eles [computadores] são os escravos". Dessa frase, pode-se inferir que a linguagem deve ser simples de ler e compreender pelos usuários e deixar que os computadores façam a parte deles.

A proposta da linguagem é ser uma linguagem legível, simples (do ponto de vista de facilidade de escrever), reusável, confiável, de fácil aprendizagem e portátil.

2.1.2 MVC – Model, View and Controller

O MVC é um padrão de arquitetura de *software*, em que a aplicação/projeto é separado em 3 camadas. São elas: camada de manipulação dos dados(*model*), camada de interação do usuário(*view*) e camada de controle(*controller*)[26].

A ideia central desse padrão é organizar o projeto a ser desenvolvido de forma que tudo tenha seu lugar, e cada camada tenha sua própria responsabilidade, separando as regras de negócio e as informações da interface de interação do usuário. A intenção do padrão é funcionar como o conceito de encapsulamento de programação orientada a objetos, em que os detalhes internos dos métodos de uma classe são ocultos para os objetos. Assim, a principal ideia do padrão MVC é a separação dos conceitos e do código[23].

As três camadas da arquitetura MVC estão descritas a seguir:

1. *Model*: camada do padrão responsável por gerenciar os elementos de dados, reportar seu estado e responder às instruções de mudança de estado. Essa camada pode ser vista também como um banco de dados.[20].
2. *View*: camada que gerencia a interface de interação dos sistema (tela do celular, computador, etc.) e apresenta as informações aos usuários. Essa camada não sabe

nada sobre a lógica de controle, mas somente recebe instruções do controle, informações do modelo e as apresenta para o usuário. Uma outra função dela é ficar reportando seu estado para as outras camadas[20].

3. *Controller*: camada do padrão responsável por identificar as entradas do usuário e mapear essas entradas em comandos que serão enviados para o modelo e/ou para a interface do usuário, para realizar a alteração apropriada[20].

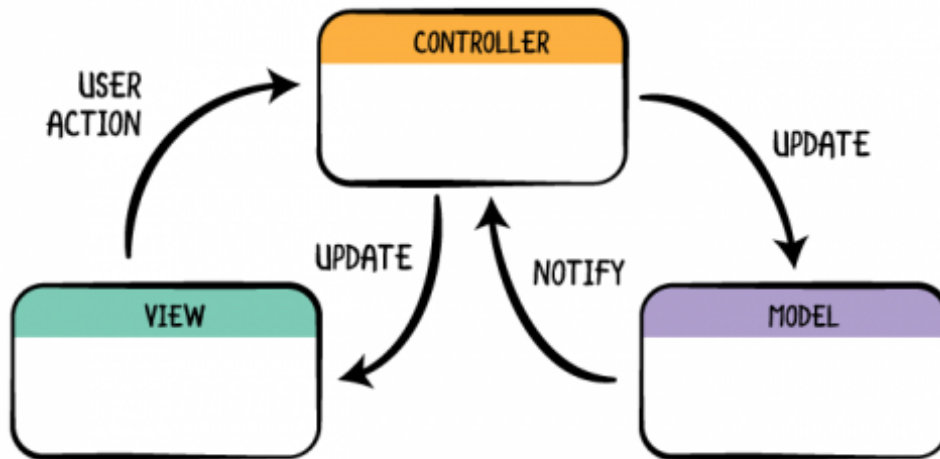


Figura 2.1: Arquitetura MVC.

Algumas das vantagens em utilizar a arquitetura MVC são[23]:

- capacidade de divisão de tarefas e desenvolvimento simultâneo no projeto;
- facilidade na implementação de camadas de segurança;
- aumento da escalabilidade dos sistemas devido ao baixo acoplamento e alta coesão da separação de responsabilidades;
- redução de esforço na manutenção do software, pois a separação de responsabilidades garante que as modificações realizadas numa camada não vão afetar as demais;
- bom reaproveitamento de código e regras.

2.1.3 *Frameworks* do Rails

Quando o *framework* Rails é instalado na máquina, uma série de outros *frameworks* que compõem o RoR também são instalados, são eles Action Pack, que se decompõe em Action View e Action Controller, Active Record, Action Mailer, Active Support e Active

Resource (que substituiu o Active Web Service, presente nas versões anteriores ao Rails 2.0) [17].

A seguir são apresentados os *frameworks* que compõem o meta-*framework* Rails:

- *Action Pack* – o *framework* composto pelo Action View e o Action Controller. É responsável pela manipulação e resposta de solicitações da web [2].
- *Action View* – o *framework* que gera os documentos HTML recebidos como resposta de uma solicitação à aplicação pelo usuário. É a camada V da arquitetura MVC [2].
- *Action Controller* – o *framework* responsável por manipular os dados advindos do *action model* e requisições feitas pelo *action view*. É a camada C da arquitetura MVC [2].
- *Active Record* – *framework* que, no Rails, é responsável pela interação entre a aplicação e o banco de dados e pela abstração dos dados. É a camada M da arquitetura MVC [5].
- *Action Mailer* – *framework* que, no Rails, é responsável pelo serviços de recebimento e entrega de e-mails da aplicação [1].
- *Active Support* – *framework* que coleciona diversas classes utilitárias e extensões de bibliotecas padrões consideradas importantes para o Rails [7].
- *Active Resource* – *framework* responsável por conectar objetos de negócio e *web services* RESTful, através de mapeamentos objeto-relacional [6].

2.2 Dispositivos Lógicos Programáveis

Os dispositivos lógicos programáveis (PLDs) são circuitos digitais integrados (operam com valores discretos) que possuem diversas portas lógicas e chaves programáveis. As funções lógicas são determinadas pelo usuário através da modelagem de circuitos com linguagens como VHDL ou Verilog. É característica de alguns PLDs serem reconfiguráveis. Na figura 2.2, tem-se o esquema de representação dos PLDs como uma “caixa preta”.

Siqueira [27] define que toda função lógica pode ser derivada na forma de soma de produtos (Mintermos) e que essa característica foi quem possibilitou a criação dos dispositivos lógicos programáveis. Ele ainda afirma que os primeiros PLDs surgiram na década de 70 com o surgimento das memórias ROM programáveis (PROM's). Adiante o surgimento das PROMs, foram implementados dispositivos com arranjos lógicos programáveis (arranjos de Es e OUs lógicos) que implementam a soma de produtos, que é especificada pelo número de entradas, produtos e somas de saídas. Esses dispositivos são conhecidos como

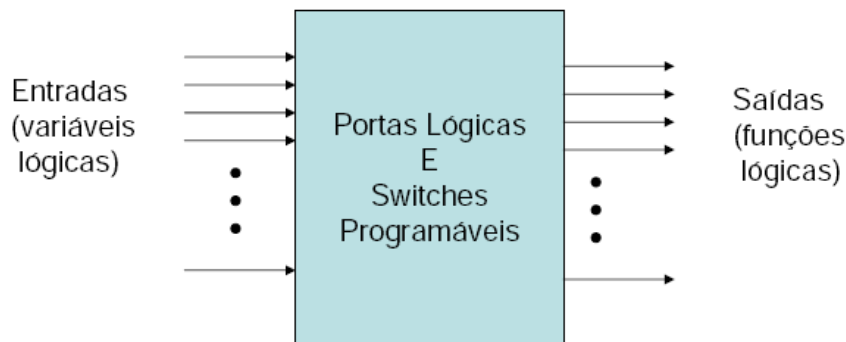


Figura 2.2: PLD como uma “caixa preta”.

PLAs (Programmable Logic Array). Em seguida, outros dispositivos — bem parecidos com as PLAs, mas que diferem pelo fato de os arranjos lógicos OU serem fixos —, fazem com que as somas de produtos sejam especificadas somente pelo número de entradas e produtos, e o tempo de propagação dos sinais no CI sejam reduzidos. Surgiram assim, as PALs (Programmable Array Logic). Esses dispositivos compõem o grupo de PLDs que implementam programação em arranjos lógicos, conhecidos por SPLDs [27] [21].

O outro grupo dentro dos PLDs é o que tange a programação em arranjo de portas, conhecido por HCPLDs. Esse grupo incorpora dispositivos que possuem mais de 600 portas lógicas, em que contamos com os CLPDs e FPGAs. Os CLPs são circuitos que integram diversos PLDs de arranjos lógicos programáveis dentro do mesmo CI [29] [21]. Implementa lógica reconfigurável do *hardware*, com a presença de *flip flops*. Difere-se da FPGA, que é da mesma categoria, a menos de capacidade lógica. A FPGA, que também implementa lógica reconfigurável, é mais fácil de ser reutilizado e possui capacidade lógica maior que as CLPDs. A FPGA é parte do objeto de estudo e, por isso, será mostrado com mais detalhes na subseção a seguir.

2.2.1 FPGA

O Field Programmable Gate Array (FPGA), criado pela empresa estadunidense Xilinx Inc. em 1985, é um dispositivo semicondutor lógico programável, que era, inicialmente, bastante usado para prototipagem de ASICs (Application Specific Integrated Circuits), graças à característica reconfigurável do *hardware*, e, atualmente, usado também para implementação de diversos circuitos digitais complexos, como processadores, decodificadores, controladores. Possui capacidade lógica extensa, com a presença de milhares de portas lógicas ou blocos lógicos integrados no dispositivo.

O FPGA possui a característica de ser um dispositivo programável de acordo com a necessidade do usuário. Essa característica é o que faz, em parte, o dispositivo ter esse

nome. Um dispositivo é dito programável em campo se a sua lógica de *hardware* pode ser modificada sem que ele precise ser desmontado ou retornado ao fabricante[28]. Daí que surge o termo Field Programmable. A outra parte se dá pela presença de milhares de portas lógicas ou blocos lógicos.

A programação do FPGA é feita com o auxílio de uma memória externa não volátil, como uma ROM programável, pois os FPGAs são fabricados com configuração volátil, em que os dados são perdidos após cessar a alimentação do dispositivo. Essa característica dos FPGAs, que também é encontrada nas memórias de acesso randômico (RAM), possibilita a programação dos FPGAs mesmo eles estando em plena operação, o que o faz interessante para gerar circuitos que demandam reconfiguração dinâmica.

Segundo Zelenovsky [29], o FPGA consiste basicamente em uma composição de blocos funcionais de pequenos circuitos digitais compactamente integrados. Ele também afirma que esses pequenos circuitos são compostos por algumas portas lógicas e *flip flops* tipo D, com alguns sinais de interação e que as conexões entre as saídas e entradas dos blocos funcionais são programáveis por meio de um protocolo simples e de fácil implementação.

O FPGA é composto por três conjuntos de componentes de elementos de configuração: os blocos de entrada e saída (I/O *blocks* – IOB), as matrizes de comutação (Switch Matrix) e os blocos lógicos configuráveis (Configuration Logical Blocks – CLB). Há também a presença de pinos de interface com o mundo externo, como pode ser visto na figura 2.3, mas que não é considerada como componente do FPGA na literatura. A figura 2.3 ilustra a arquitetura de um FPGA e a disposição dos seus componentes.

Os blocos de entrada e saída (IOBs) são circuitos responsáveis pelo interfaceamento das saídas dos CLBs com o exterior do FPGA. São basicamente *buffers* bidirecionais, que se conectam aos pinos de interface do FPGA e determinam como esses pinos vão funcionar: entrada, saída, bidirecional ou coletor-aberto [29].

As matrizes de comutação, compostas pelas interconexões, são trilhas utilizadas para conectar as entradas e saídas dos CLBs e IOBs às redes apropriadas [29].

Os blocos lógicos configuráveis (CLB) são células básicas dispostas em um arranjo 2-D que são configurados e interligados de forma a se obter a lógica desejada. São formados por alguns *flip flops* tipo D, que são geralmente entre 2 a 4 e necessários para implementação de lógica sequencial, e lógica combinacional extra. A lógica combinacional é formada por tabelas verdades (*Look Up Table* - LUTs) da funcionalidade e outros componentes[29]. As figuras 2.4 e 2.5 ilustram a implementação de uma LUT e de um CLB, respectivamente.

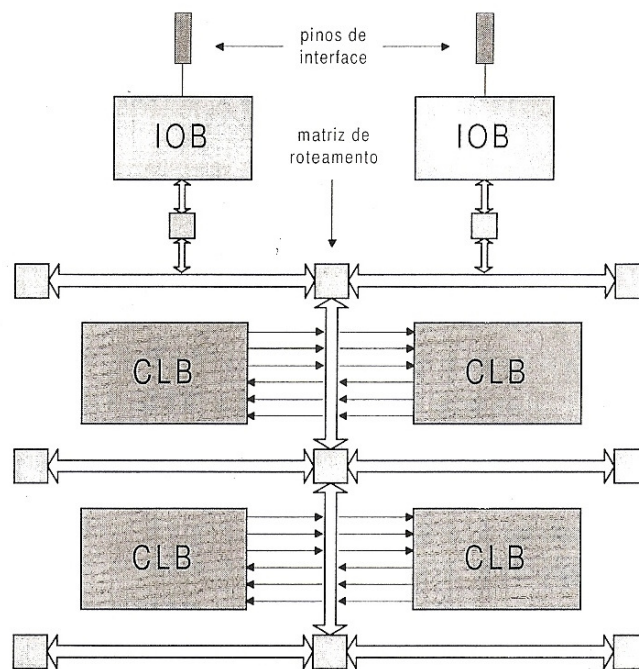


Figura 2.3: Arquitetura interna de um FPGA, formada por CLBs, IOBs e matrizes de comutação [29].

2.3 Microcontroladores

Microcontrolador é um microcomputador disponível em apenas um circuito integrado, composto internamente por um núcleo de processamento, memória de programa, memória de dados, pinos de entrada e saída, e, nos modelos mais atuais, interface USB.

Ele é um circuito integrado assim como um CI (Integrated Circuit) de portas lógicas (OR, AND etc.), mas de uso geral, que pode ser programado para desempenhar uma tarefa desejada, dentro das limitações do microcontrolador. Suas funções básicas são as de controle. Daí que deriva o nome desse tipo de circuito integrado. Difere de um CI de portas lógicas por seu caráter generalista, ou seja, não possui uma função específica, a princípio.

O primeiro microcontrolador comercial (TMS1000) [10] foi produzido em meados da década de 70, pela Texas Instruments, como uma solução para sistemas que exigiam atividades mais simples ou atividades específicas (sistemas embarcados) que combinasse, em uma única peça de silício, o equivalente a um microprocessador e seus periféricos, mas que fosse menor e tivesse um custo de produção mais baixo que estes.

Os famosos Raspberry Pi, computadores de placa única, tiveram seu conceito advindo de um microcontrolador, onde todos os componentes eletrônicos de um computador estão integrados a uma única placa de circuito impresso, ao passo que um microcontrolador

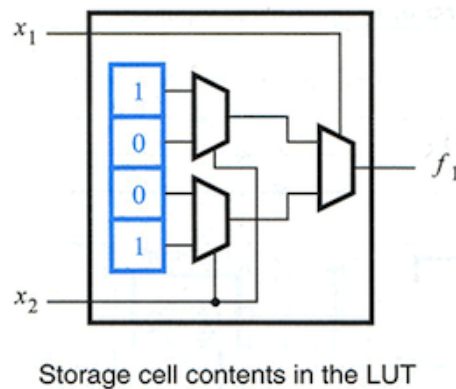
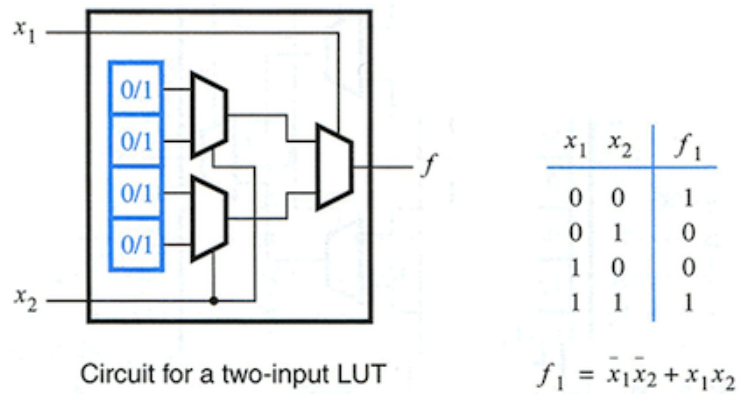


Figura 2.4: Lookup table (LUT) de duas entradas.[11].

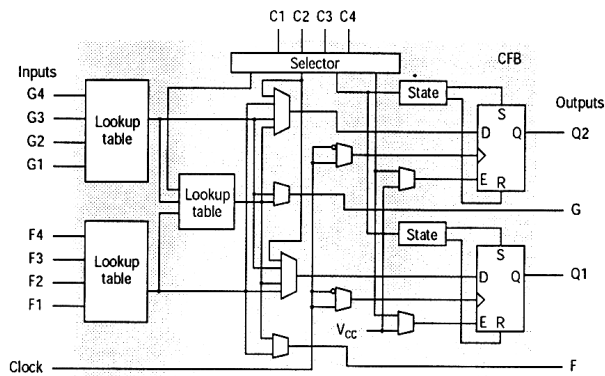


Figura 2.5: Arquitetura interna de um bloco lógico configurável (CLB).[11].

integra todos os componentes em um único *chip*. Uma comparação rápida entre as duas tecnologias, ou até mesmo entre um computador pessoal ou de propósito geral e um microcontrolador, aponta que o segundo possui limitações em relação ao primeiro nos quesitos de frequência de *clock*, tamanhos das memórias de programa e de dados, mas que, em geral, possui custo de produção mais baixo, menor consumo energético e dimensões

menores, possibilitando, assim, diversas aplicações.

No geral, um microcontrolador não precisa ter um conjunto de instruções poderoso e extenso, tampouco ser muito rápido, além de não precisar de instruções que tratem dados do tipo ponto flutuante e nem trabalhar com vetores/strings e métodos de endereçamento, que são aspectos fundamentais para suportar sistemas operacionais completos. Por esses motivos, não se têm computadores pessoais nos quais a CPU seja um microcontrolador. Microcontroladores são encontrados também em periféricos, como teclados, mouses, monitores, placas de vídeo etc. É encontrado também em geladeiras, telefones celulares, televisores, impressoras.

Na área de educação, o microcontrolador tem grande destaque por compor as plataformas de prototipagem eletrônica Arduino, por serem o principal componente dessas plataformas, que são bastante populares entre entusiastas e estudantes de eletrônica, por causa de sua ampla documentação, comunidade de usuários numerosa e proativa, além de serem de fácil aprendizagem. Servem também como interface entre computadores e outros tipos de hardware na área de eletrônica, onde se mostra bastante aplicável.

2.4 Tecnologias utilizadas

O projeto foi desenvolvido usando diversas tecnologias atuais, visando um desenvolvimento rápido e prezando pela manutenibilidade, escalabilidade, confiabilidade, legibilidade e disponibilidade do *software*, algumas das características de softwares que afetam no sucesso da solução.

O Ruby on Rails foi escolhido dentre diversas possibilidades de *frameworks* para desenvolvimento, manutenção e implantação de *software*. As principais características que justificam a escolha do Rails são uma maior familiaridade com a ferramenta, o fato desse *framework* estar bastante difundido entre os desenvolvedores na internet, e se mostrar bastante confiável, de fácil uso e capaz de fornecer as características de software almejadas no projeto.

Outro justificava de usar o Ruby on Rails é alta produtividade que ele fornece ao disponibilizar a integração de outros *frameworks* indispensáveis para produção de aplicações *web*, como *frameworks* de banco de dados, web services, páginas *web* etc. Essa combinação poderosa de *frameworks* permite a construção de uma aplicação de maneira bastante rápida. Além disso, Ruby é uma linguagem orientada a objetos, que é bastante adequada para o problema atacado e por um dentre os vários *frameworks* do RoR possuir suporte para ORM, capaz de diminuir a impedância entre o tipo de linguagem de programação e um banco de dados relacional, também adequado para o problema.

A aplicação foi desenvolvida em sistema operacional Linux 18.04.1 LTS 64 bits, seguindo a documentação oficial da ferramenta, disponível em [12]. A versão do *framework* utilizado no projeto foi Rails 5.2.1, usando *ruby* 2.5.1p57 (2018-03-29 *revision* 63029). A imagem 2.6 apresenta as informações do ambiente de desenvolvimento da aplicação.

```
Luan@Luan-MS-7A39:~$ ruby -v
ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]
Luan@Luan-MS-7A39:~$ rails -v
Rails 5.2.1
Luan@Luan-MS-7A39:~$ cat /etc/*-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"
NAME="Ubuntu"
VERSION="18.04.1 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.1 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
Luan@Luan-MS-7A39:~$
```

Figura 2.6: Informações do ambiente de desenvolvimento da aplicação.

A ferramenta de modelagem de dados utilizada para gerar o diagrama de entidade relacionamento dos objetos da aplicação foi o brModelo, versão 3.3, uma "ferramenta de código aberto e totalmente gratuita voltada para ensino de modelagem de banco de dados relacionais", disponível em [8].

O microcontrolador escolhido no projeto foi um MSP430F5529LP devido a sua alta disponibilidade em laboratório, documentação abrangente, muitos exemplos disponíveis na internet, facilidade de manipulação e por ser o mesmo modelo usado por outros alunos do mesmo grupo de trabalho no departamento. Além disso, esse equipamento permite uma comunicação serial com um computador através de uma porta USB, o que facilita no desenvolvimento, por não haver necessidade de implementar um protocolo de comunicação pra uso do hardware.

O placa de desenvolvimento FPGA usada para implementar a solução foi uma Basys3 da Xilinx. A justificativa para o uso desse modelo foi, além da alta demanda em laboratório, a documentação abrangente, alta difusão no meio acadêmico, facilidade de manipulação, software de análise e síntese já pré instalados no laboratório desenvolvido o projeto, tutoriais de uso de fácil acesso no laboratório etc.

Capítulo 3

Projeto

Este capítulo visa a apresentar a implementação do projeto.

3.1 Metodologia de desenvolvimento

Devido ao fato de o projeto se tratar da construção de um sistema grande, ele foi segmentado em três partes, ditas subprodutos entregáveis, para facilitar a forma de lidar com a construção, conforme mencionado no Objetivo Geral. Esses subprodutos são retomados aqui para terem seu desenvolvimento detalhado. São eles: 1) desenvolvimento da interface de usuário (*frontend*), 2) desenvolvimento do servidor, *scripts* de controle do microcontrolador e da FPGA (*backend*) e 3) *firmware* do microcontrolador que entende comandos do servidor e traduz em valores lógicos para a FPGA.

3.1.1 Desenvolvimento da interface de usuário

O primeiro objetivo específico foi desenvolvido como requisito parcial para conclusão da disciplina "Projeto Final em Engenharia de Computação 1". Para entregar esse subproduto, foi desenvolvida a interface de usuário da aplicação *web* pretendida. Foram seguidos diversos passos, em atividades, até a conclusão dessa entrega. O *link* público para acesso do que foi implementado está disponível em [14].

A primeira atividade da entrega consistiu em realizar uma modelagem conceitual do sistema. Foram analisadas as entidades participantes das relações e seus atributos, o que deu origem a um Modelo Entidade Relacionamento (MER). Em seguida, foi feita uma modelagem diagramática dessas entidades e suas relações, gerando como resultado uma representação gráfica do modelo MER, conhecido como Diagrama Entidade Relacionamento (DER), que pode ser visto na imagem 3.1 a seguir.

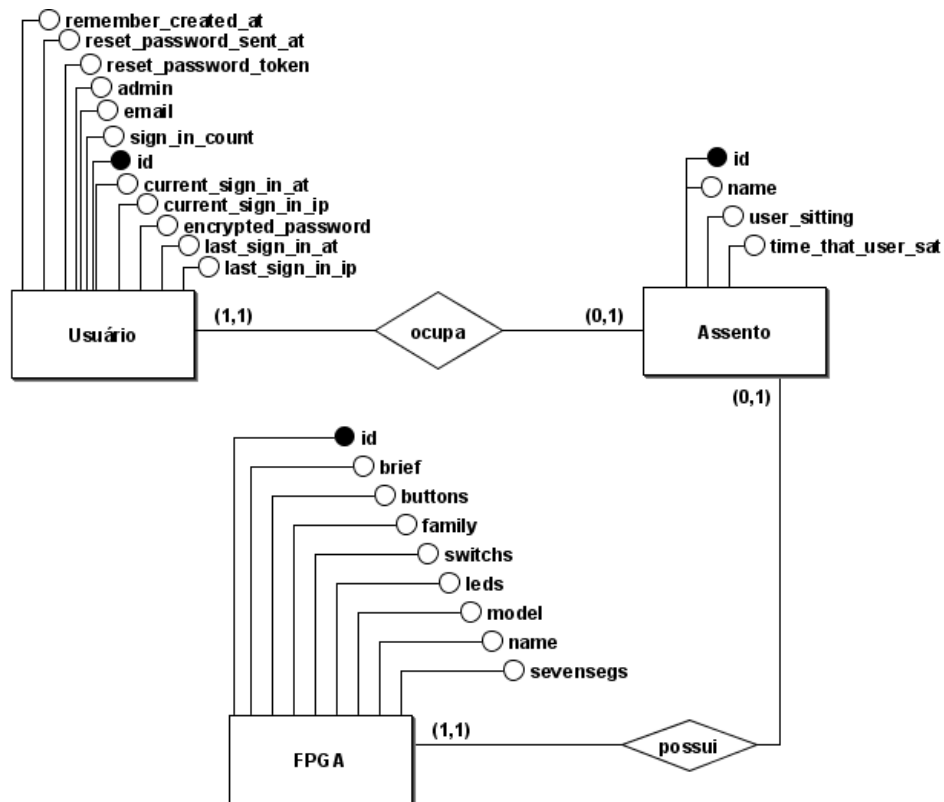


Figura 3.1: Diagrama Entidades Relacionamento do projeto. Representação gráfica em notação tradicional.

O DER é uma representação gráfica do MER. Em notação tradicional, os retângulos representam as entidades; os losangos, as relações entre as entidades; as bolas pretas, os atributos que são chave primária (atributos identificadores); as bolas brancas, os atributos que não são chave primária; os números entre parênteses, as cardinalidades (grau) das relações e as linhas, as participações das entidades nas relações.

Na sequência, foi realizada a segunda atividade da entrega, que consistiu em levantar os requisitos funcionais do sistema. Um esboço gráfico em papel foi realizado, listando todas as funcionalidades do sistema necessárias para atender o objetivo do projeto.

A terceira atividade consistiu em gerar uma representação visual dos artefatos e funcionalidades do sistema, partindo das funcionalidades listadas na etapa de levantamento dos requisitos funcionais. Isso foi feito para facilitar a compreensão e pré-implementação do sistema. O produto dessa atividade é o diagrama de casos de uso (padrão UML) do projeto. Os atores, os casos de uso e as relações do sistema foram representadas no diagrama, que pode ser visualizado na imagem 3.2. Os bonecos representam os atores; as elipses, os casos de uso e as linhas, os relacionamentos.

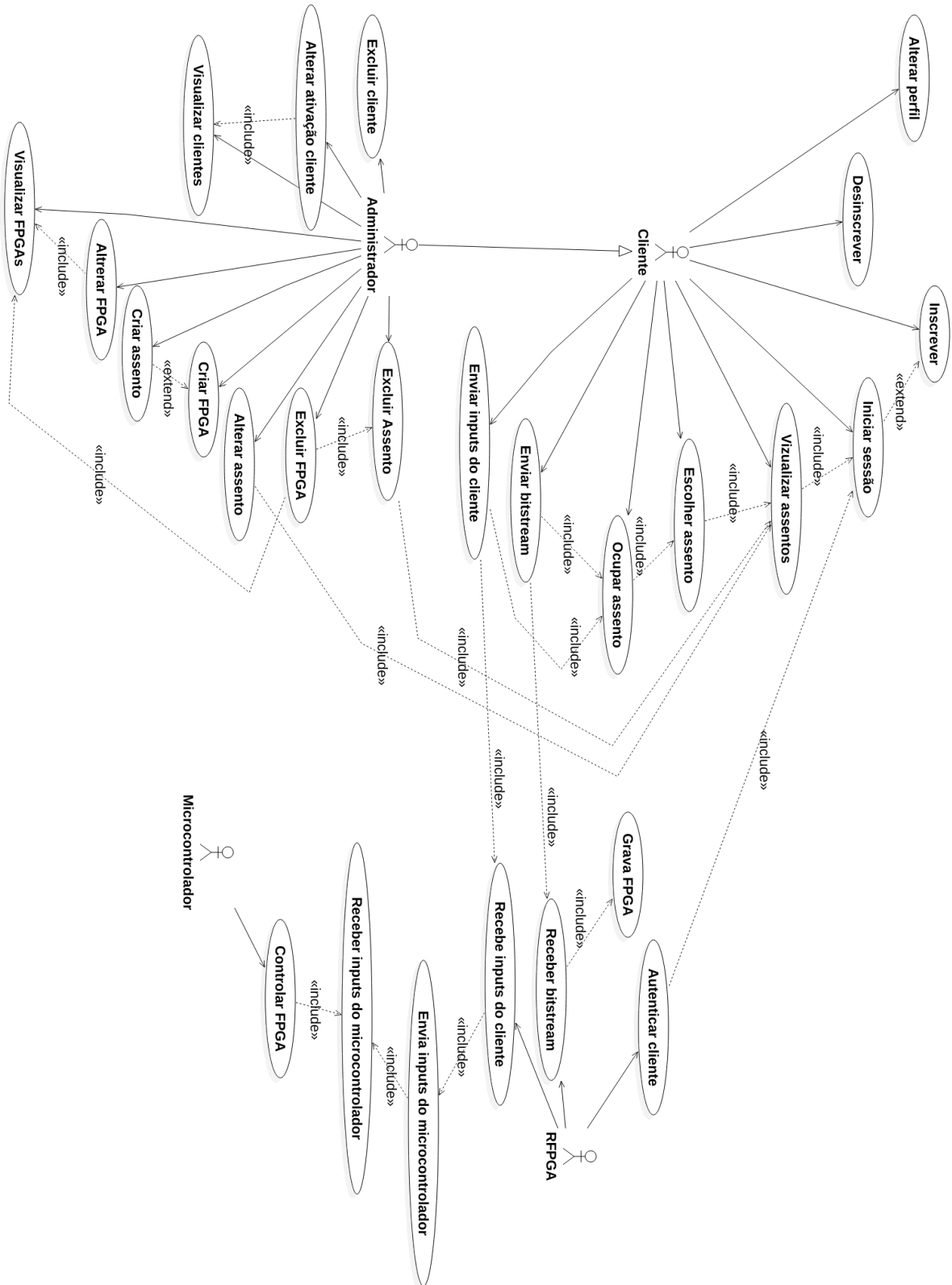


Figura 3.2: Diagrama Casos de Uso do projeto .

A quarta atividade consistiu em utilizar uma ferramenta chamada MockFlow para implementar os esboços das interfaces que os usuários terão contato, conhecidos como *wireframes*. Os *wireframes* contêm os esboços dos conteúdos das páginas da aplicação, as quais estão abstraindo, e os casos de uso do sistema. A ferramenta possui diversas versões, sendo que apenas uma é gratuita. Essa versão é um pouco limitada em relação às outras, pois permite a criação de apenas três páginas ou três *wireframes*. Por isso, cada página teve de ser apagada para ser reutilizada diversas vezes. Assim, apenas as 3 últimas páginas implementadas na conta criada para utilizar a versão gratuita estão persistindo. Antes de reutilizar cada página, foi realizada uma impressão do *wireframe* implementado, de forma a guardar o esboço feito para as próximas atividades dependentes desses esboços.

Para realizar a quinta atividade do subproduto 1, foi necessário primeiro carregar a estrutura inicial da aplicação. Só após a correta instalação do interpretador Ruby e das bibliotecas (gemas) necessárias para o funcionamento do *framework* Rails, essa estrutura foi carregada. Para isso, foi dado, em terminal, o comando "rails new RFPGA". O Rails, então, realizou o mecanismo de inversão de controle e criou a estrutura inicial da aplicação. Os detalhes da implementação estão disponíveis em [14].

Após a criação da estrutura inicial da aplicação, deu-se início à quinta atividade: implementar as telas finais da interface de usuário, utilizando os esboços criados anteriormente. As telas foram criadas utilizando as linguagens HTML, CSS e JavaScript. A primeira linguagem é utilizada para marcar os elementos presentes nas páginas; a segunda para formatar a apresentação desses elementos nas páginas e a terceira para realizar ações sobre os elementos marcados e formatados. Nessa atividade, foi implementada a tela de uso da FPGA, que contempla o *layout* de uma placa de desenvolvimento FPGA, em que foram associadas ações de teclas e ações do *mouse*/de toque na tela para os botões de pressionamento e *switchs* da placa, ações essas que serão utilizadas, posteriormente, para enviar dados para o microcontrolador. Os componentes da página de uso da FPGA foram marcados com a linguagem HTML; o aspecto visual dos componentes e do layout da FPGA, formatados pelo CSS e as ações listadas, implementadas em JavaScript. A figura 3.3 ilustra a tela de uso da FPGA descrita.

Cada componente de entrada e saída da placa desenhada na interface acima possui um correspondente em hardware na placa de desenvolvimento física. A imagem é meramente ilustrativa, não correspondendo necessariamente com a placa real. Observa-se que as duas primeiras chaves da imagem estão levantadas, o que significa que a sua equivalente, no hardware físico está levantada também, ou seja, possui nível lógico 1.

Para as páginas onde havia formulários, foram criados *templates*, que, em seguida, foram carregados nelas. Isso foi feito para evitar ao máximo a repetição de código e padronizar os formulários. Os detalhes da implementação estão disponíveis em [14].

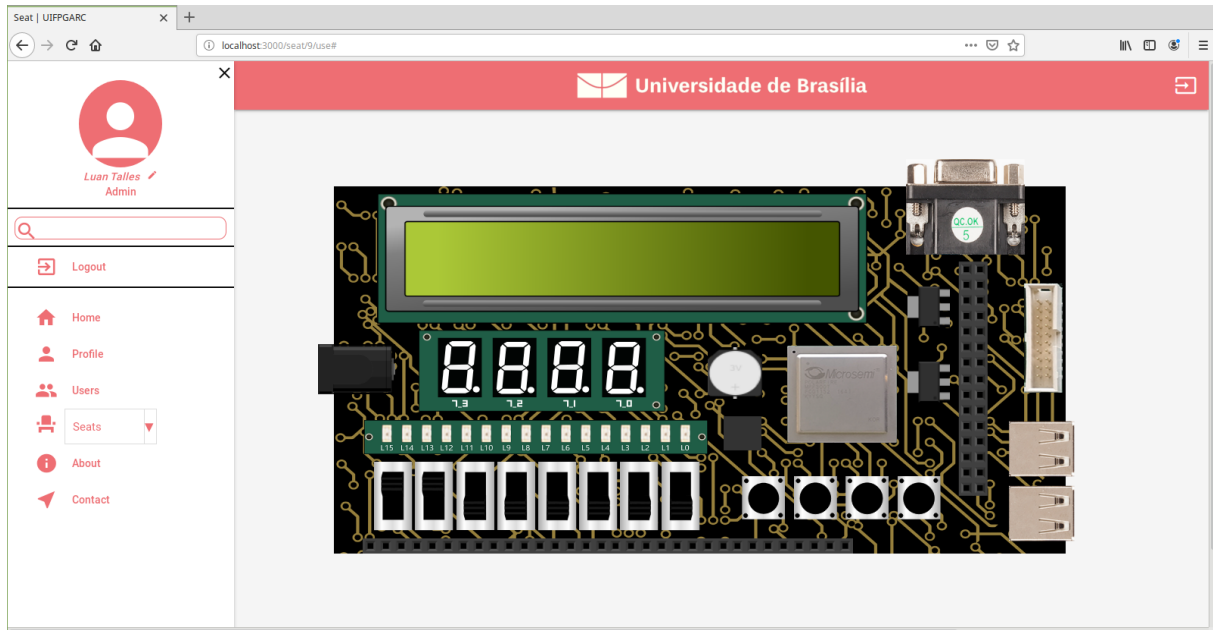


Figura 3.3: Interface de usuário do sistema.

Após o termino dessa etapa, pode-se dizer que foi implementada a parte V da arquitetura MVC. A partir de então, teríamos a estrutura de visualização do projeto, parte com a qual o usuário interage.

3.1.2 Desenvolvimento do servidor, *scripts* do controle do microcontrolador e da FPGA

Terminado e entregue o primeiro subproduto, deu-se início ao segundo objetivo específico, que também foi requisito parcial para conclusão da disciplina "Projeto Final em Engenharia de Computação 1".

A primeira atividade do segundo objetivo específico consistiu em criar as entidades do sistema, listadas no DER (figura 3.1), em que um comando foi passado, em terminal, várias vezes (uma vez para cada entidade criada) para o *framework* Rails, que se encarregou de criar as classes e a camada de persistência do projeto. Os detalhes da implementação estão disponíveis em [14].

O comando mencionado tem a estrutura sintática "rails generate model NAME [field[:type][:index] field[:type][:index]] [options]"[16].

Esse comando é responsável por criar as entidades, e também suas respectivas associações num banco de dados relacional. Essas associações são criadas graças ao fato do Active Record do Rails conseguir realizar a técnica conhecida como ORM, capaz de mapear as classes e objetos do sistema com tabelas e tuplas de bancos de dados relacionais.

Dando prosseguimento ao projeto, deu-se início a segunda atividade, em que foram criadas as relações entre as entidades do sistema. Isso foi feito adicionando algumas palavras reservadas dentro dos modelos previamente criados. Essas palavras reservadas, que denotam o tipo de relação estabelecida, podem ser: *belongs_to*, *has_one*, *has_many*, *has_many :through*, *has_one :through* e *has_and_belongs_to_many*[4]. O significado de cada palavra tem a ver com a cardinalidade da relação. Uma relação do tipo *has_many* (tem muitos) indica que o modelo em que foi inserido essa palavra reservada pode se relacionar com muitas (n) entidades modelos do tipo marcado após o comando.

Após o termino desta atividade, podemos dizer que foi implementada a parte M (dados e lógica de negócio) da arquitetura MVC, seguida no projeto e suportada pelo Rails. Os detalhes da implementação estão disponíveis em [14].

A próxima atividade consistiu em migrar a lógica mapeada nas etapas anteriores para a base de dados escolhida. Por padrão, ao iniciar uma aplicação no Rails, a base de dados do SGBD SQLite3 é instanciada. O projeto seguiu o padrão, então a base de dados utilizada é o SQLite3. Mais uma vez, o Rails utilizou do ORM do Active Record para diminuir a impedância entre o tipo de programação (orientada a objetos) do Ruby e o banco de dados relacional utilizado (SQLite3).

Em seguida, deu-se início a atividade de criar os controladores de ações e os caminhos que esses controladores devem seguir para realizar as ações (métodos das entidades descritas nos modelos) descritas nos casos de uso.

Para isso, um comando foi passado, em terminal, várias vezes (uma vez para cada controlador criado) para o *framework* Rails, que se encarregou de criar esses controladores. Os detalhes da implementação estão disponíveis em [14].

O comando mencionado tem a estrutura sintática "rails generate scaffold_controller NAME [action action] [options]"[16].

Esse comando é responsável por criar os controladores e, pelo fato da palavra *scaffold* ter sido passada junto com *controller*, criar os métodos Index, Show, Edit, Update, Create e Delete para o modelo passado como parâmetro em NAME, além de criar as rotas (caminhos) para chamar esses métodos criados.

Ainda durante a atividade de criar os controladores e rotas do sistema, uma gema chamada Devise[9] foi carregada ao projeto. Para isso, bastou adicionar sua assinatura no arquivo Gemfile, enviar os comandos "bundle install" e "rails generate devise:install" pelo terminal, na raiz do projeto, e modificar o arquivo "config/environments/development.rb" adicionando esse comando "config.action_mailer.default_url_options = host: 'localhost', port: 3000" no corpo do arquivo. Essa gema serve para ajudar no processo de autenticação dos usuários do sistema. Em seguida, outra gema foi adicionada, chamada CanCanCan[15], para auxiliar no processo de controle de acesso e autorização. Para uti-

lizar essa gema, bastou adicionar sua assinatura no arquivo Gemfile e enviar os comandos "bundle install" e "rails generate cancan:ability" pelo terminal, na raiz do projeto.

Por último, foram implementados todos os casos de uso do sistema, atendendo às restrições de cada ator, finalizando-se, assim, a parte V da arquitetura MVC. Durante essa etapa, foram criadas as demais rotas e métodos das entidades que não foram pré-criadas com o comando *scaffold*, como usar assento, gravar *firmware* da FPGA, liberar assento, etc. Para instanciar e enviar comandos seriais do servidor para o microcontrolador, foi necessário instalar a gema SerialPort[3] ao projeto, sendo necessário apenas adicionar sua assinatura no arquivo Gemfile e enviar os comandos "bundle install" pelo terminal, na raiz do projeto. Após isso, bastou instanciar a classe MSPService criada e utilizar os métodos dessa classe que possuem em seus corpos lógica da gema SerialPort, para enviar comandos via porta serial para o microcontrolador. A possibilidade de enviar comandos via porta serial foi muito útil para viabilizar a solução proposta. Os detalhes da implementação estão disponíveis em [14].

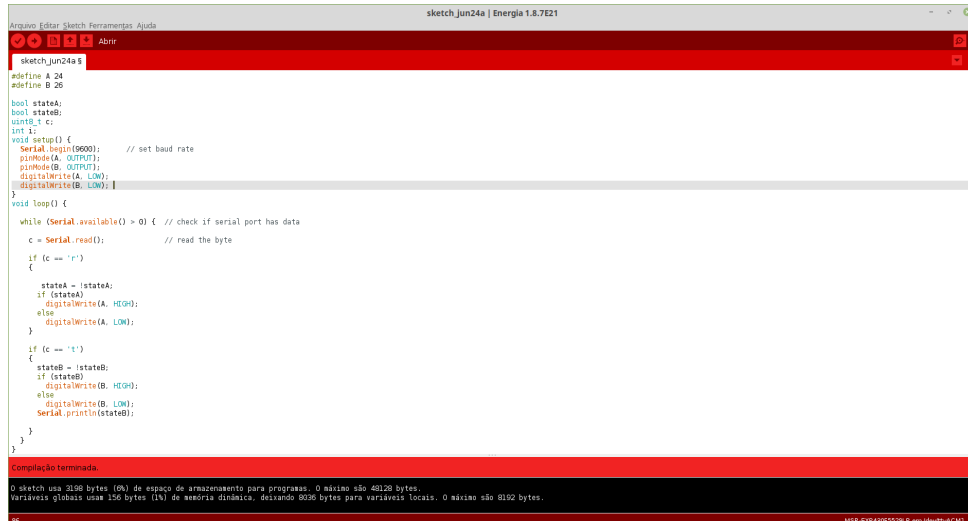
Todos os métodos da gema SerialPort podem ser visualizadas e utilizadas no console do Rails. Considerando que a gema foi instalada corretamente no projeto, basta inserir o comando "rails console" pelo terminal, na raiz do projeto e instanciar a classe SerialPort, passando todos os parâmetros adequados para o construtor dessa classe, que é possível utilizá-la diretamente no console do Rails. O "rails console" foi bastante útil no projeto, pois em cima dele que a classe MSPService foi construída. Todos os métodos dessa classe que possuem lógica da gema SerialPort, foram implementados a partir do console do Rails. O comando utilizado para instanciar a classe no console é "SerialPort("/dev/ttyACM1", 9600, 8, 1, SerialPort::NONE)".

Terminado e entregue o segundo subproduto, deu-se início ao terceiro e último objetivo específico do projeto, que foi requisito parcial para conclusão da disciplina "Projeto Final em Engenharia de Computação 2".

3.1.3 Firmware do microcontrolador

A primeira e única atividade dessa entrega consistiu em criar e gravar um *firmware* para o microcontrolador MSP430, capacitando-o de receber dados da porta serial, identificar de que dados se trata e traduzi-lo em valores lógicos para a FPGA. Os dados recebidos na porta serial são traduzidos em ação semelhante à realizada na FPGA virtual, a qual está disponível na tela de uso da FPGA, por exemplo, ao clicar em um *switch* na placa virtual, levantado-o ou abaixando-o, isso será traduzido na mesma ação no *hardware* da FPGA controlada pelo microcontrolador.

A figura 3.4 ilustra o *firmware* desenvolvido, de forma suficiente, para anteder aos testes que serão discutidos na próxima seção. O *firmware* completo está disponível em [14].



```
sketch_jun24a_5
#define A 24
#define B 25

bool stateA;
bool stateB;
uint8_t c;
int i;

void setup() {
  Serial.begin(9600); // set baud rate
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  digitalWrite(A, LOW);
  digitalWrite(B, LOW);
}

void loop() {
  while (Serial.available() > 0) { // check if serial port has data
    c = Serial.read(); // read the byte
    if (c == 'r')
    {
      stateA = !stateA;
      if (stateA)
        digitalWrite(A, HIGH);
      else
        digitalWrite(A, LOW);
    }
    if (c == 't')
    {
      stateB = !stateB;
      if (stateB)
        digitalWrite(B, HIGH);
      else
        digitalWrite(B, LOW);
      Serial.println(stateB);
    }
  }
}
```

Compilação terminada.

O sketch usa 2156 bytes (6%) do espaço de armazenamento para programas. O máximo são 49152 bytes.
Variáveis globais usam 136 bytes (1%) de memória dinâmica, deixando 8036 bytes para variáveis locais. O máximo são 8192 bytes.

Figura 3.4: Firmware do microcontrolador.

Após a realização da criação do *firmware* do microcontrolador, conectado à placa de desenvolvimento FPGA por meio de pinos de entrada da FPGA e pinos de saída do microcontrolador, deu-se início a última atividade desenvolvida, a de teste de usabilidade da solução. Essa atividade está descrita na próxima sessão.

Ao final dessa atividade foi levantada, ainda, a possibilidade de acréscimo de mais uma atividade, consistindo em acrescentar uma câmera de vídeo filmando em tempo real as saídas da FPGA. A justificativa para tal atividade seria permitir a visualização da FPGA utilizada pelo usuário na tela do sistema. A justificativa para não o fazer foi que a adição da câmera implicaria em atrasos das imagens do vídeo em relação as entradas do usuário, com a placa, por meio da interface virtual. As saídas da FPGA para o usuário no âmbito da interface implementada ficaram modelada como trabalho futuro e será discutida na sessão de Trabalhos Futuros do capítulo de Conclusão.

3.2 Metodologia de teste

Para validar os modelos, relações, controladores e rotas criadas no projeto, foram escritos um conjunto de testes de integração (*request*) e modelo (*model*) com o framework BDD (Behaviour-Driven Development) RSpec. Para utilizar essa gema no projeto, bastou adicionar sua assinatura no arquivo Gemfile, enviar os comandos "bundle install" e "rails generate rspec:install" pelo terminal, na raiz do projeto. Essa atividade está descrita após

todas as outras atividades fazendo parecer que ela foi realizada de uma única vez e após realizar todas as atividades descritas acima, porém o ciclo de construção da aplicação foi realizado seguindo o desenvolvimento orientado por comportamento, em que os testes de modelo foram realizados após a criação dos modelos, enquanto os testes de integração realizados após a criação dos controladores e instalação do devise no projeto. Os resultados desses testes serão discutidos no próximo capítulo.

Para realizar o teste de um modelo de forma unitária, é preciso enviar um comando pelo terminal, na raiz do projeto, com a sintaxe "bundle exec rspec spec/models/NAME_spec.rb", em que NAME é o nome do modelo testado. Para teste unitário de integração, a sintaxe é "bundle exec rspec spec/requests/NAME_spec.rb", em que NAME é o nome do modelo no plural, que é o mesmo do controlador desse modelo, já que por convenção no Rails o nome do controlador de um modelo é o nome desse modelo pluralizado. Para realizar todos os testes do sistema de uma vez, basta inserir o comando pelo terminal, na raiz do projeto, com sintaxe "bundle exec rspec".

Para validar a solução, foi gerado dois projetos, um de uma porta AND e outro de uma porta OR, com dois pinos de entrada e um de saída, em VHDL, com a utilização da ferramenta de síntese de hardware da placa de desenvolvimento escolhida, chamada Vivado. As etapas de análise e síntese para cada projeto foram implementadas, seguindo os passos descritos no manual da ferramenta. Em seguida, prosseguiu-se com o mapeamento dos pinos de entrada e saída da FPGA para cada projeto. Foram escolhidos dois de vários pinos da FPGA, disponíveis, como entrada, a serem conectados a dois pinos de saída do microcontrolador e um *led* do FPGA como pino de saída.

Após a implementação da análise, síntese, mapeamento dos pinos de entrada e saída da FPGA e compilação do projeto, um arquivo binário foi gerado, para cada projeto. Esses arquivos gerados serão os usados para descrever o *hardware* desejado em FPGA. De um por vez, após clicar em usar a FPGA e ocupar o assento, o arquivo binário gerado em cada um dos projetos foi gravado na persistência do computador, ao clicar na caixa de seleção de arquivo, escolher o arquivo binário e clicar em enviar *bitstream*, e, em seguida, por meio de chamada de sistema operacional, carregado no FPGA.

A partir desse momento, foi possível conectar os pinos de saída do microcontrolador nos pinos de entrada do FPGA e testar o sistema. Um vídeo da ação de gravar a FPGA com os 2 arquivos binários descritos e usar a FPGA a partir da interface da aplicação *web* foi gravado e está disponível em [14]. Os resultados dessa atividade serão discutidos no próximo capítulo. A lógica descrita acima se estende para outras placas de desenvolvimento de FPGA que possuam ao menos dois pinos de entrada e ao menos um *led* capaz de ser alterado pelo usuário, para servir como saída.

Todos os conjuntos de testes realizados no projeto foram feitos em ambiente controlado.

Por motivos de logística, os testes foram realizados em redes locais, geralmente no próprio computador de desenvolvimento, através do endereço de *loopback* da máquina, na porta 3000. *A priori*, deveria ser testado, também, em redes WAN, para melhor validação dos resultados, porém, os computadores do laboratório 3 do módulo SG11 do Campus Darcy Ribeiro são bloqueados para uso fora da instituição, o que inviabiliza essa abordagem. Em alguns momentos, foram utilizadas máquinas diferentes da que estava rodando o servidor virtual em *loopback*, utilizando o endereço de rede IPv4 local dessa máquina na porta 3000.

Capítulo 4

Resultados

Este capítulo apresenta os resultados obtidos, juntamente com a interpretação dos mesmos.

4.1 Resultados

Os resultados obtidos nos testes de modelo e integração, descritos no capítulo 3 se mostraram satisfatórios. Todos os testes escritos resultaram em sucesso, refletindo que o código foi escrito sem erros ou com poucos erros.

O resultado da capacidade do sistema controlar os equipamentos através da rede foi satisfatório. Primeiro porque foi possível gravar e sintetizar um *hardware* FPGA através da rede; segundo, foi possível utilizar o *hardware*, também, através da rede. Esse resultado pode ser visualizado, também, no vídeo gravado para mostrar a implementação das portas AND e OR em FPGA, disponível em [14].

Os testes de projeto das portas AND e OR em *hardware* FPGA, descritos no capítulo de Projeto, serviram para validar a solução proposta e implementada no projeto, em que as ações realizadas na interface de uma página *web*, foram capazes de gerar o resultado esperado das portas lógicas AND e OR em *hardware* FPGA.

No caso da porta AND, quando os níveis lógicos das duas chaves de seleção estavam altas, a saída resultante foi o *LED* do FPGA estar também em nível lógico alto (*LED* aceso) e no restante dos casos, em que o nível lógico de alguma, ou de ambas as chaves, era baixo, teve-se como resultado que o *LED* do FPGA permaneceu apagado, ou seja, possuía nível lógico baixo. Isso mostra exatamente a tabela verdade de uma porta AND, em que a saída possui nível lógico 1 (alto) somente se todas as entradas possuírem nível lógico 1. Em qualquer outra combinação de níveis lógicos nas entradas, implica nível lógico 0 (baixo) na saída.

No caso da porta lógica OR, os resultados obtidos também refletiram a tabela verdade da porta lógica testada, onde bastou que uma das portas tivesse o nível lógico alto que o *LED* do FPGA ficou ligado. Somente no caso em que os níveis das duas chaves na interface da página *web* estavam baixo foi que deixou o *LED* apagado. O vídeo mostrando as duas implementações e os resultados delas está disponível em [14].

Capítulo 5

Conclusão

Este capítulo contém a conclusão referente aos resultados obtidos neste trabalho.

5.1 Conclusão

O objetivo principal do sistema foi atingido. Implementar uma aplicação *web* capaz de permitir a configuração e o uso de FPGAs através da *internet*. Os testes foram realizados em ambiente controlado, limitando-se a uma rede local e computadores conectados a essa rede local, mas os resultados obtidos foram satisfatórios. O sistema, atualmente, tem uma limitação de não retornar os dados provindo da FPGA para o servidor através do microcontrolador.

A escolha do Rails como *framework* de implementação da aplicação *web* se mostrou adequada para o projeto. Simplificou em algumas etapas o desenvolvimento, deu suporte para as tecnologias mais recentes de desenvolvimento de aplicações *web* como HTML5, CSS3, Javascript etc., e foi capaz de permitir o uso de FPGAs por vários computadores, em rede local, de forma simultânea, respeitando o controle de acesso às informações e o uso exclusivo de cada assento por um único usuário.

A escolha do MSP430 como microcontrolador responsável pela interconexão entre o servidor e a FPGA se mostrou adequado para o projeto. Foi possível enviar comandos seriais através da interface USB do microcontrolador e comandar o FPGA escolhido no projeto. O dispositivo concedeu todos os recursos necessários para implementação do projeto. Sua vasta documentação e programas de exemplos foram bastante produtivas na realização do projeto.

A escolha da placa de desenvolvimento Basys3, também, se mostrou adequada para o projeto. Foi possível implementar um *hardware* de porta lógica OR e AND na placa e utilizar as entradas para alterar os estados da lógica e a saída para validar o *hardware* pro-

jetado. Sua ampla documentação e exemplos disponíveis na internet foram importantes na criação das sínteses dos *hardwares* projetados.

5.1.1 Trabalhos futuros

Cabe mencionar que houve testes somente em console do Rails, de retorno de dados do microcontrolador para o servidor, e que estes retornaram com sucesso. Foi possível retornar *bytes* do microcontrolador para o servidor, o que será bastante útil para um dos trabalhos futuros: especializar o sistema, fazendo com que os *leds*, *display* de sete segmentos e *display* LCD, desenhados na página *web*, tenham o mesmo estado que o seu correspondente em *hardware*.

Fica como trabalho futuro, também, implementar o sistema em servidor, com endereço de IP fixo e desbloqueado para o mundo, capaz de permitir o completo uso da FPGA em rede WAN e Internet.

Por último, um terceiro trabalho futuro consiste em configurar o servidor, com protocolo LDAP, nas redes da instituição de ensino, com propósito de disponibilizar o projeto para uso da comunidade acadêmica.

Referências

- [1] “action mailer – easy email delivery and testing”. <https://github.com/rails/rails/blob/master/actionmailer/README.rdoc>. acessado em 14/02/2019. 10
- [2] “action pack – from request to response”. <https://github.com/rails/rails/blob/master/actionpack/README.rdoc>. acessado em 14/02/2019. 10
- [3] “action pack – from request to response”. <https://github.com/hparra/ruby-serialport>. acessado em 14/02/2019. 23
- [4] “active record associations”. https://guides.rubyonrails.org/v5.2/association_basics.html. acessado em 02/07/2019. 22
- [5] “active record – object-relational mapping in rails”. <https://github.com/rails/rails/blob/master/activerecord/README.rdoc>. acessado em 14/02/2019. 10
- [6] “active resource”. <https://github.com/rails/activeresource>. acessado em 14/02/2019. 10
- [7] “active support – utility classes and ruby extensions from rails”. <https://github.com/rails/rails/blob/master/activesupport/README.rdoc>. acessado em 14/02/2019. 10
- [8] “brmodelo”. <http://www.sis4.com/brModelo/>. acessado em 21/11/2018. 16
- [9] “flexible authentication solution for rails with warden”. <https://github.com/plataformatec/devise>. acessado em 02/07/2019. 22
- [10] “history of innovation”. <http://www.ti.com/corp/docs/company/history.html#1960s>. acessado em 14/02/2019. 13
- [11] “logic devices”. <http://beta.ivc.no/blog/2011/03/30/logic-devices/>. acessado em 24/09/2018. xi, 14
- [12] “rails guides”. <https://guides.rubyonrails.org/>. acessado em 25/11/2018. 16
- [13] “rapportdestage2a”. https://github.com/LuanTalles/RFPGA/blob/master/RapportDeStage2A_Version3.docx. acessado em 01/07/2019. 3
- [14] “remote fpga solution”. <https://github.com/LuanTalles/RFPGA>. acessado em 14/02/2019. 17, 20, 21, 22, 23, 24, 25, 27, 28

- [15] “the authorization gem for ruby on rails”. <https://github.com/CanCanCommunity/cancancan>. acessado em 02/07/2019. 22
- [16] “the rails command line”. https://guides.rubyonrails.org/command_line.html. acessado em 02/07/2019. 21, 22
- [17] “welcome to rails”. <https://github.com/rails/rails>. acessado em 14/02/2019. 10
- [18] M. Bächle e P. Kirchberg. Ruby on rails. *IEEE software*, 24(6), 2007. 7
- [19] M. S. Bascil, I. Yazici, e F. Temurtas. A fpga based remote accessible digital system laboratory prototype. In *e-Learning and e-Technologies in Education (ICEEE), 2012 International Conference on*, pages 49–53. IEEE, 2012. 2, 4
- [20] F. P. Conter e F. F. F. Peres. “utilização do framework “ruby on rails” no desenvolvimento de um módulo web para sistema de biblioteca”. *UNOPAR Científica Ciências Exatas e Tecnológicas*, 6(1), 2015. 8, 9
- [21] C. A. de Oliveira, J. A. de Aguiar, M. G. Said Fontanini, e M. Wendling. “dispositivos lógicos programáveis”. 11
- [22] W. M. El-Medany. Fpga remote laboratory for hardware e-learning courses. In *Computational Technologies in Electrical and Electronics Engineering, 2008. SIBIRCON 2008. IEEE Region 8 International Conference on*, pages 106–109. IEEE, 2008. 3, 4
- [23] V. B. Fuentes. “*Ruby on Rails: coloque sua aplicação web nos trilhos*”. Casa do Código, 2014. 7, 8, 9
- [24] R. Hashemian e J. Riddley. A method to design, construct and test digital hardware all in classroom environment. In *Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE’07. 37th Annual*, pages T3G–1. IEEE, 2007. 3
- [25] J. Mcgavren. “*Use a Cabeça!: Ruby*”. Alta Books, 2016. 8
- [26] S. Ruby. “*Agile Web Development with Rails 5*”. Pragmatic Bookshelf, 2016. 7, 8
- [27] To. M. Siqueira. “implementacao de um modem ofdm em fpga”. *Projeto de Graduação–Universidade Federal do Espírito Santo*, 2004. 10, 11
- [28] G.C.J. Wang, C.Y. Ho, W.J. Hwang, e W.W.F. Wang. “field-programmable lab-on-a-chip based on microelectrode array architecture”, April 1 2014. US Patent 8,685,325. 12
- [29] R. Zelenovsky e A. Mendonça. “*Eletrônica Digital: Curso Prático e Exercícios*”, 2^a Ed., MZ Editora LTDA., 2007. xi, 11, 12, 13