CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LABORATÓRIO REMOTO PARA PROTOTIPAÇÃO DE CIRCUITOS DIGITAIS UTILIZANDO KITS DE DESENVOLVIMENTO FPGAS XILINX

CRISTIANO VICENTE

MARÍLIA
2016

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LABORATÓRIO REMOTO PARA PROTOTIPAÇÃO DE CIRCUITOS DIGITAIS UTILIZANDO KITS DE DESENVOLVIMENTO FPGAS XILINX

Trabalho de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília - UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Fábio Dacêncio Pereira

MARÍLIA
2016

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Cristiano Vicente

LABORATÓRIO REMOTO PARA PROTOTIPAÇÃO DE CIRCUITOS DIGITAIS UTILIZANDO KITS DE DESENVOLVIMENTO FPGAS XILINX.

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 10 (dez)

Orientador:Fábio Dacêncio Pereira_____

1°.Examinador:Emerson Alberto Marconato_____

2°.Examinador:Fabio Lucio Meira_____

Marília, 05 de dezembro de 2016.

VICENTE, Cristiano. **Laboratório remoto para prototipação de circuitos digitais utilizando kits de desenvolvimento FPGAs Xilinx**. 2016. 71f. Trabalho de curso. (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2016.

Resumo

O Laboratório Remoto, é uma plataforma web para auxiliar na prototipação de circuitos digitais utilizando kits de desenvolvimento FPGAs remotamente, onde o usuário pode enviar o seu projeto sintetizado pela ferramenta da Xilinx para o laboratório remoto online o projeto enviado é programado no FPGA do kit de desenvolvimento no servidor, que retorna o resultado visual do dispositivo físico e também permite o seu controle através da interface da plataforma na web, com objetivo de facilitar o acesso aos dispositivos FPGAs e prover um laboratório remoto interativo de baixo custo, acessível, de fácil manutenção e implantação, para fins educacionais, para alunos e professores de graduações (e.g. Engenharia Elétrica e Ciência da Computação), que fazem uso da tecnologia FPGA no desenvolvimento prático de disciplinas e projetos de circuitos digitais, assim como alunos e professores engajados em pesquisas na área de desenvolvimento de hardware, que não possuam à sua disponibilidade um kit de desenvolvimento FPGA para testar seus projetos e verificar seus resultados.

Palavras-chave: FPGA, Laboratório Remoto, Programação Remota.

VICENTE, Cristiano. **Laboratório remoto para prototipação de circuitos digitais utilizando kits de desenvolvimento FPGAs Xilinx**. 2016. 71f. Trabalho de curso. (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2016.

Abstract

The Remote Lab is a web platform to assist in the prototyping of digital circuits remotely with FPGAs development boards, where the users may send their project synthesized by Xilinx's tool to the online remote lab, the project sent is programmed in the FPGA development board by the server, which returns the visual result of the physical device and also allow its control through the web platform's interface, in order to facilitate access to FPGAs devices, providing a low-cost and accessible, easy-to-maintain and inexpensive interactive remote lab for educational purposes, for students and teachers from courses (e.g. Electrical Engineering and Computer Science), which use FPGA technology in the development of practical learning and projects of digital circuits, as well as students and teachers engaged in research of hardware development, which might not have at their disposal a FPGA development kit to test their projects and check their results.

Keywords: *FPGA, Remote Lab, Remote Programing*

Lista de Figuras

Lista de Tabelas

Lista de siglas

AJAX .....Asynchronous Javascript and XML

API .....Application Programming Interface

ASCII .....American Standard Code for Information Interchange

CI .....Circuito Integrado

CSS .....Cascading Style Sheets

E/S .....Entrada e Saída

FPGA .....Field Programmable Gate Arrays

HTML .....Hypertext Markup Language

IDE .....Integrated Development Environment

JSON .....JavaScritp Object Notation

LED .....Light Emissor Diode

ORM .....Object-relational mapping

RISC .....Reduced Instruction Set Computer

UART .....Universal Asynchronous Receiver/Transmitter

UCF .....User Constraints File

USB .....Universal Serial Bus

VCS .....Version Control System

VHDL .....VHSIC Hardware Description Language

VHSIC .....Very High Speed Integrated Circuit

Sumário

Introdução

Um laboratório tem um papel fundamental para o desenvolvimento do aprendizado em diversas disciplinas, onde pode ser demonstrado, simulado e provado em prática, diversos conceitos e teorias da ciência, por meio de ferramentas específicas para cada área de conhecimento enriquecendo o aprendizado e também auxiliando na pesquisa e busca por novos conhecimentos, assim também para o desenvolvimento de novas tecnologias.

A definição de laboratório é um local ou sala especial de trabalho, experimentação e investigações científicas, equipada com aparelhagem específica para pesquisa e experimentos. (cf. Michaelis, Dicionário Brasileiro da Língua Portuguesa, 2016). Este trabalho aborda um laboratório diferenciado de sua definição comum, no conceito de local ou sala física, mas como uma plataforma acessível através da web com equipamentos específicos para as disciplinas de circuitos digitais e arquitetura de computadores.

O autor J. S. Pastor et al. (2004), também propôs a utilização de acesso remoto para aprendizado na disciplina de Arquitetura de Computadores em seu trabalho (*A Remote Laboratory for Debugging FPGA*) e também cita a importância da prática de estudos em laboratórios.

> É comumente aceito que experimentos em laboratórios são partes essências para o aprendizado nos cursos de Engenharia Elétrica e Ciência da Computação. Princípios básicos sobre Arquitetura de Computadores podem ser estudados com exemplos reais. (J. S. Pastor et al., 2004, tradução nossa).

Objetivo

Em suma, o objetivo deste trabalho é a criação de um laboratório remoto, em forma de uma plataforma web, para auxiliar na programação de circuitos digitais utilizando kits de desenvolvimento FPGAs de forma remota, através da plataforma web. Este projeto, visa um desenvolvimento de um laboratório de baixo custo e fácil manutenção, acessível e de fácil implantação, que permita ao usuário da plataforma ou laboratório remoto, poder enviar seu projeto VHDL já sintetizado pela ferramenta da Xilinx, o arquivo bit, para o laboratório remoto online, onde é programado no FPGA do kit de desenvolvimento no servidor e que também permite o controle do FPGA através da própria plataforma online.

Metodologia

Os métodos para o desenvolvimento deste projeto foram organizados em:

- Levantamento bibliográfico, pesquisa de trabalhos correlatos e tecnologias: Foi realizado pesquisas sobre laboratório de acesso remoto e laboratórios de acesso remoto para circuitos digitais com FPGA, também foi realizado pesquisas de tecnologias a serem utilizadas para a comunicação do FPGA com o servidor utilizando a porta serial e tecnologias para transmissão de vídeo.
- Levantamento de tecnologias a serem utilizadas para o desenvolvimento do projeto: Foi realizado um estudo sobre as tecnologias web, para escolha da tecnologia a ser utilizada no desenvolvimento do servidor, que foi definido o Node.js para o servidor, também foi realizado um estudo específico nas bibliotecas e métodos nativos da tecnologia escolhida para o servidor, para o correto desenvolvimento do módulo de comunicação entre o FPGA e o servidor.
- Implementação inicial do servidor e do lado cliente com controle de acesso: Foi implementado a estrutura básica do servidor com login e cadastro de usuários para o laboratório.
- Desenvolvimento da comunicação entre o servidor e o kit de desenvolvimento FPGA: foi desenvolvido um módulo VHDL, um módulo de comunicação serial no servidor e o protocolo de comunicação, utilizando o módulo *serialport* para Node.js e os estudos e análises levantados sobre o funcionamento do módulo *serialport* e o método nativo Buffer do Node.js para trabalhar com dados binários.
- Desenvolvimento da plataforma de controle, programação e visualização do Laboratório Remoto lado servidor e cliente: foi desenvolvido o streaming de vídeo utilizando câmeras IP, a interface de controle do FPGA foi implementada em cima do módulo de comunicação que foi desenvolvido e a implementação do reconhecimento e programação do kit de desenvolvimento utilizando a ferramenta iMPACT da Xilinx e pôr fim a interface de usuário para o lado cliente do laboratório.

Materiais

Este item lista os materiais utilizados para a implementação do projeto:

- Banco de dados MySQL.
- Bootstrap e JQuery, framework e biblioteca *frontend*.
- Computador tipo laptop com sistema operacional Windows 10, utilizado no desenvolvimento de todo o projeto.
- Digilent Nexys3, Kit de desenvolvimento FPGA.
- Express.js, framework *backend* para Node.js
- Ferramenta iMPACT, ferramenta do pacote de *software* da Xilinx, utilizado para reconhecer o kit de desenvolvimento FPGA conectado e fazer sua programação.
- IDE de desenvolvimento IntelliJ IDEA Ultimate 2016.2, utilizado no desenvolvimento do lado servidor e cliente.
- IDE de desenvolvimento Xilinx ISE Design Suite 14.5, utilizado para o desenvolvimento do módulo VHDL.
- Módulos SerialPort e Request, para Node.js
- Módulos VHDL, receptor e transmissor, utilizados no módulo UART de Merrick (2016).
- Node.js e seus módulos nativos.
- ORM Sequelize.
- Smartphone Android com aplicativo IPCam, que simula uma câmera IP utilizando a câmera do smartphone.

Organização do trabalho

Este trabalho está organizado em cinco capítulos, sendo:

1- Contextualização: Neste capítulo é feito a contextualização e fundamentação teórica das tecnologias utilizadas no projeto.

2- Trabalhos correlatos: Neste capítulo foi descrito os trabalhos correlatos que propõem o desenvolvimento de um laboratório de acesso remoto e relatam seus benefícios.

3- Arquitetura: Este capítulo aborda a estrutura e a arquitetura do projeto e como funciona os componentes e módulos do Laboratório Remoto e as suas comunicações.

4- Implementação e implantação: Este capítulo aborda o processo de implementação e implantação do projeto e as tecnologias e ferramentas utilizadas.
5- Conclusão: Este capítulo relata os resultados atingidos, as dificuldades no desenvolvimento do projeto, suas limitações e oportunidades para trabalhos futuros.

1. Contextualização

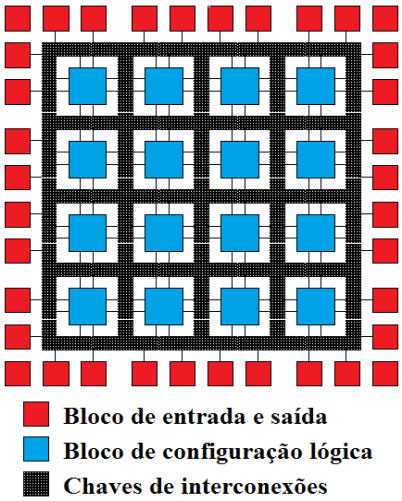
Este capítulo tem como objetivo apresentar e contextualizar as tecnologias e ferramentas utilizadas no projeto e também apresentar brevemente tecnologias similares ou concorrentes às utilizadas no projeto quando disponíveis, sem se aprofundar muito em suas fundamentações teóricas e técnicas, são abordados os seguintes itens:

- Tecnologia FPGA e os principais fabricantes de FPGA.
- Modelos de FPGAs, ferramentas e softwares do fabricante Xilinx.
- Kit de desenvolvimento FPGA
- Linguagens de descrição de Hardware (HDL)
- Comunicação serial UART.
- Tecnologias de desenvolvimento web e servidor, linguagem de programação, *frameworks*, persistência dedados e dispositivos de transmissão de vídeo. (Câmera IP).

1.1. Tecnologia FPGA

O FPGA (*Field Programmable Gate Arrays*), é um circuito integrado composto de uma matriz de blocos lógicos configuráveis conectados através de interconexões programáveis, este circuito integrado é bastante flexível e pode ser programado e reprogramado, possibilitando criar um número ilimitado de circuitos para diversas aplicação ou funções desejadas.

Figura 1 – Arquitetura básica de um FPGA



Fonte: Tecnologia de FPGA, MEDEIROS, D. S., 2016

Na Figura 1 a arquitetura básica, onde pode-se ver uma matriz de blocos ou células lógicas que são conectadas através do barramento de interconexões e por fim são ligados aos blocos de entradas e saídas. As células lógicas comportam lógicas simples e conectadas através do barramento de interconexões, formam lógicas complexas, podendo simular vários outros circuitos de apoio e até mesmo um processador simples, assim sendo uma ferramenta flexível e muito poderosa, para auxiliar na simulação em *hardware* e desenvolvimento de protótipos de outros circuitos integrados.

Segundo Maxfield, Clive (2004), no início da década de 90, os FPGAs eram utilizados principalmente nos sistemas de redes e de telecomunicações e logo ao fim da década de 90 já estavam sendo utilizados nos setores automotivos e em diversas aplicações industriais.

Atualmente os FPGAs, são utilizados nas mais diversas áreas, citando algumas, tais como; Aeroespacial, Segurança, Computação de Alta Performance, Processamento de Áudio, Vídeo e Imagens, equipamentos na área de Medicina, pesquisas Cientificas e Acadêmicas, entre outras diversas áreas no desenvolvimento de protótipos de CIs, devido a sua flexibilidade de desenvolvimento e baixo custo.

A maior parte do custo de um processador, micro controlador ou um chip qualquer está em seu desenvolvimento e esta tecnologia trouxe um avanço considerável para o desenvolvimento de protótipos de circuitos integrados para produção em massa e até mesmo a criação de chips únicos, utilizando o próprio FPGA como produto final, não sendo necessário passar por todo o processo de produção para um único chip.

1.2. Principais fabricantes

O primeiro FPGA comercial foi lançado em 1985 pela Xilinx, (*Funding Universe* apud *International Directory of Company Histories*, Vol. 16. St. James Press, 1997), o atual líder do mercado de FPGAs seguido pela sua maior concorrente, a Altera, que tiveram um grande crescimento na década de 80, pela enorme demanda do mercado pela tecnologia do FPGA.

Segundo informações publicadas no site *Electronic Engineering Journal* por Morris, Kevin (2014), os dois fabricantes, Xilinx e Altera controlavam aproximadamente 90% do mercado de FPGAs em 2014, sendo a Xilinx com a maior parcela, tendo sozinha aproximadamente 45% a 50% do mercado, os outros 10% do mercado são compartilhados entre; Achronix, Atmel, e2v, Lattice Semiconductor, MicroSemi *(*Actel*)*, Tabula e demais fabricantes menores.

No ano de 2016 foi publicado um novo artigo no site *Electronic Engineering Journal*, também por Morris, Kevin (2016), "Xilinx vs Intel", no qual relata a aquisição do fabricante Altera pela Intel, que é o maior fabricante de semicondutores do mundo, segundo relatório publicado no site ICinsights (2016), Morris conclui em seu artigo, que apesar de a Xilinx possuir uma liderança significativa no mercado de FPGAs no momento, a Xilinx é uma empresa muito menor do que a agora combinada Intel/Altera, porém ainda dominante até que esta grande aquisição se estabilize.

1.3. Xilinx

Para o desenvolvimento deste trabalho, será utilizado um kit de desenvolvimento com FPGA do fabricante Xilinx e seus softwares, as informações a seguir terão maior enfoque nos dispositivos FPGAs e softwares da Xilinx.

1.3.1. Famílias de FPGAs

Como os FPGAs são utilizados hoje em dia nas mais diversas áreas da tecnologia, eles são divididos em famílias ou séries, existem FPGAs desde uso geral há fins específicos.

Tabela 1 – Famílias de FPGAs Xilinx

Portfólio da Família de FPGAs Xilinx				Família Adicional		
45nm	**28nm**	**20nm**	**16nm**	**Automotiva**	**Defesa Militar**	**Aeroespacial**
Spartan-6	Artix-7	Kintex UltraScale	Kintex UltraScale+	XA Artix-7	Virtex-7Q	Virtex-5QV
				XA Spartan-6	Kintex-7Q	
Artix-7	Kintex-7	Virtex UltraScale	Virtex UltraScale+	XA Spartan-3A	Artix-7Q	Virtex-4QV
				XA Spartan-3A DSP	Virtex-6Q	
Zinq	Virtex-7			XA Spartan-3E	Spartan-6Q	
					Virtex-5Q	

Fonte: Xilinx Inc, All Programmable FPGAs and 3D ICs, 2016

Atualmente a Xilinx conta com 5 modelos base e uma família especial adicional, assim como ilustrado na Tabela 1, (*Spartan 6, Artix 7, Zynq, Kintex 7 e Virtex 7*), divididos em grupos ou famílias de acordo com suas escalas, que são os; *Low-End 45nm, 7 Series 28nm, UlraScale 20nm* e *UlraScale+ 16nm*, e a família adicional para fins mais específicos, que são as séries Automotiva, Defesa (militar e comercial) e aeroespacial.

Para fins de conhecimento, a Altera, maior concorrente direto da Xilinx, conta também com uma família de FPGAs; *Stratix Series, Arria Series, Cyclone Series* e *MAX Series*, cada família com diversos modelos de uso geral e outros voltados para fins específicos assim como a Xilinx.

1.3.2. Softwares e ferramentas de desenvolvimento para FPGA

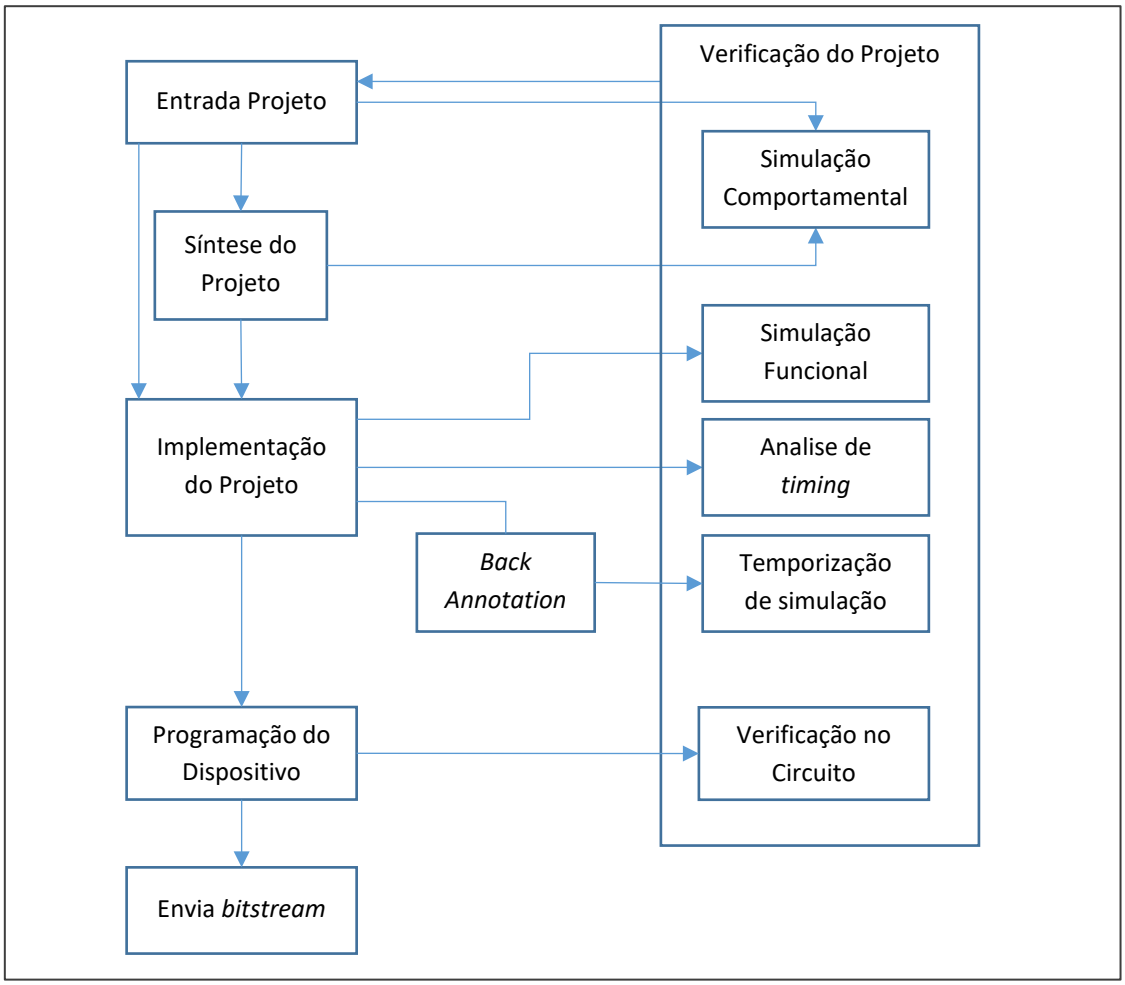
Além do mercado de semicondutores, a Xilinx conta também com softwares específicos, para auxiliar no desenvolvimento com os dispositivos FPGAs, o seu atual pacote para desenvolvimento é o *Vivado Design Suite*, este voltado para nova família de FPGAs da série 7, tais como; Artix 7, Kintex 7 e Virtex 7 e o ISE Design Suite voltados para dispositivos da série 6 como o Spartan 6 e FPGAs das gerações anteriores a este.

No fluxograma da Figura 2, o fluxo de projeto segue com a entrada do projeto que são modelos de descrição de hardware, que podem ser descritos nas linguagens Verilog, VHDL, UCF ou ABEL, após a entrada, segue para verificação e síntese do projeto, onde é feito a checagem de sintaxe, simulação comportamental e mapeamento da tecnologia, na implementação do projeto é feita a tradução do projeto para o dispositivo, o mapeamento para os recursos disponíveis do dispositivo, criação das rotas e geração do arquivo de programação *bitstream* para o dispositivo alvo. (Xilinx Inc., 2016).

Após gerado o arquivo de programação *bitstream*, é feita a programação do dispositivo através do iMPACT, é uma ferramenta presente nos pacotes de desenvolvimento da Xilinx, apresentado em batch ou interface de usuário, o iMPACT permite a configuração direta dos dispositivos FPGAs da Xilinx que estão conectados ao PC, faz a verificação de compatibilidade do projeto com o dispositivo, executa configuração inicial e envio do *bitstream* para o dispositivo e depuração de erros (Xilinx Inc., 2016).

O iMPACT será uma das ferramentas utilizadas para concepção deste trabalho, o *webservice* do Laboratório Remoto, chamará o iMPACT via batch que executará a verificação dos dispositivos FPGAs conectados no servidor e fazer sua configuração e programação.

Figura 2 – Diagrama de fluxo de projeto FPGA.



Fonte: Xilinx Inc, FPGA Design Flow Overview, 2016

1.4. Kit de desenvolvimento FPGA

Os FPGAs consistem apenas no circuito integrado, que podem ser acoplados em placas de desenvolvimento *protoboards* ou diretamente a um equipamento que fará uso do chip.

Para auxiliar no desenvolvimento e melhorar a produtividade, a Xilinx e algumas empresas comercializam o FPGA já acoplado a um kit de desenvolvimento, com vários componentes acoplados, como *displays*, LEDs, botões e vários tipos de entradas e saídas, como áudio, vídeo, USB, VGA, HDMI, RJ45, etc, facilitando os testes e simulações de aplicações desenvolvidas.

1.4.1. Kits de desenvolvimento Xilinx.

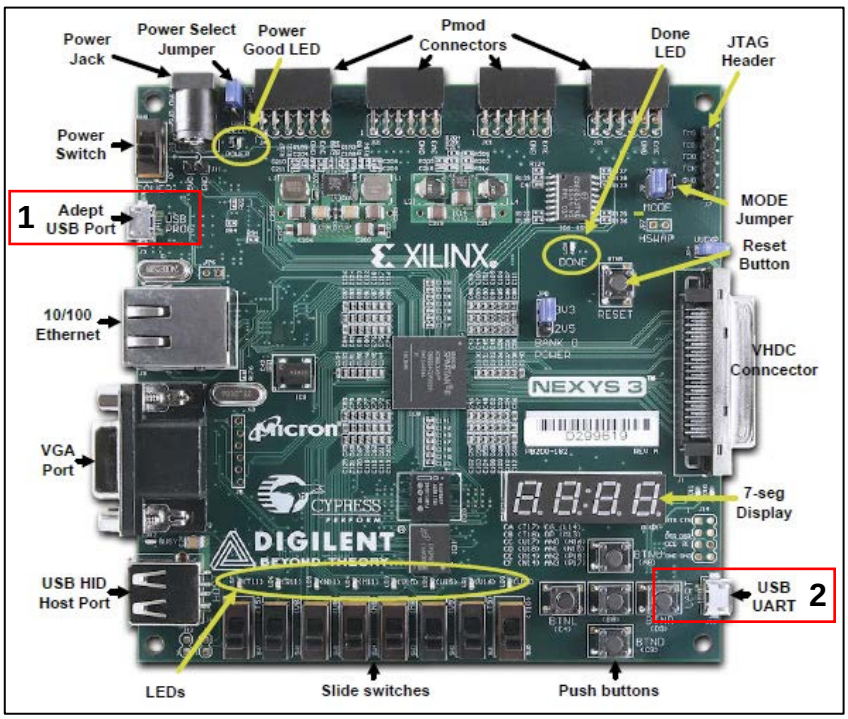
A Xilinx fornece uma grande quantidade de kits de desenvolvimento para diversas categorias e áreas, como kits para iniciantes, projetos de baixo custo e projetos de alto custo e computação de alto desempenho.

Assim como possuem FPGAs para diversas áreas, também cotam com kits de desenvolvimento para essas áreas como; médica, industrial, automotiva, aeroespacial, defesa, áudio e vídeo, telecomunicações e etc. os kits de desenvolvimento para as áreas específicas possuem componentes, módulos e acessórios específicos voltados para a área designada.

1.4.2. Kits de desenvolvimento Digilent Inc.

A Digilent, é uma empresa de produtos para Engenharia Elétrica, servindo estudantes, universidades e empresas, sua missão é fazer com que as tecnologias de engenharia e design elétricos sejam compreensíveis e acessíveis a todos, fornecendo ferramentas educacionais relevantes para a indústria e universidades (Digilent Inc, 2016).

Figura 3 – Kit de desenvolvimento Digilent Nexys3



Fonte: Digilent Inc, 2016

Para o desenvolvimento deste trabalho, será utilizado um Kit de desenvolvimento *Nexys3*, fabricado pela Digilent Inc, utilizando o FPGA Spartan-6 LX16 fabricado pela Xilinx, que vem acoplado a placa do kit, este projeto é compatível não somente com este kit de desenvolvimento, mas também com todos os kits de desenvolvimento para FPGAs da 6ª geração, suportados pela ferramenta iMPACT do pacote de *softwares* ISE Design Suite.

Na Figura 3 é ilustrado uma visão da placa do kit de desenvolvimento Nexys3 e seus componentes de entrada e saída e este kit conta também com 48Mbytes de memória não volátil, no centro da placa encontra-se o CI do FPGA, um Xilinx Spartan-6 LX16 de 324 pinos, que opera a 100MHz, que é conectado diretamente a maior parte dos componentes da placa.

Na Figura 3, destacados por retângulos numerados estão as duas portas USB que são conectadas diretamente na máquina do servidor, sem a necessidade de qualquer outro hardware adicional, a porta *Adept USB Port/USB Prog* (retângulo **1**) é a porta utilizada para fazer a identificação e programação do kit de desenvolvimento, a outra porta (retângulo **2**), *USB UART* é utilizada para se comunicar com o servidor através um módulo implementado em VHDL que faz a comunicação de entrada e saída entre FPGA e o servidor do laboratório.

1.5. Linguagens de descrição de hardware (HDL)

As HDLs, linguagens de descrição de hardware, são linguagem semelhantes às linguagens de programação, porém são orientadas a descrição da estrutura e do comportamento do hardware, tendo grande vantagem em comparação à esquemática do projeto de um hardware, pois podem representar diretamente equações booleanas, tabelas verdade e operações complexas, de forma fácil de ser interpretado. (Midorikawa, Edson T., 2001).

1.5.1. Linguagem VHDL

O nome VHDL (***V**HSIC **H**ardware **D**escription **L**anguage*), e o termo VHSIC (***V**ery **H**igh **S**peed **I**ntegrated **C**ircuit*), em tradução livre sendo “linguagem de descrição de hardware para circuitos integrados de velocidade muito alta”.

A linguagem VHDL foi originalmente desenvolvida em 1980, por empresas contratadas pelo Departamento de Defesa dos Estados Unidos, com o propósito de criar uma HDL

padronizada, no ano de 1987, o Departamento de Defesa designou que todos os circuitos eletrônicos digitais fossem descritos em VHDL, (PEREIRA, Fábio Dacêncio, et al, 2003).

O *Institute of Electrical and Electronics Engineers* (IEEE), ratificou o primeiro padrão do VHDL, o (Padrão 1076-1987 ou VHDL 87), que foi atualizado em 1993: (Padrão 1076-1993 ou VHDL 93) e sua última atualização foi em 2008, atual (Padrão 1076-2008), (*IEEE Standard VHDL Language Reference Manual*, 2009).

A estrutura mínima de um projeto descrito em VHDL requer uma declaração de entidade (*entity*) e uma arquitetura (*arquiteture*). A *entity* é a parte principal do projeto que define os aspectos externos, descreve as entradas e saídas, largura de barramento, funções e como ocorre a transferência de informações. O nome de uma *entity* deve identificar o projeto, podendo usar letras e números, porém deve iniciar por uma letra. A *arquiteture* define os aspectos internos, onde são descritas todas as lógicas operacionais, atribuições, comparações e como se relacionam com outros sinais internos.

1.5.2. Linguagem VERILOG

A linguagem Verilog foi introduzida em 1985 pela *Gateway Design Automation* e em 1989, a Gateway foi comprada pela empresa *Cadence Design Systems*, que tornou a linguagem de domínio público em maio de 1990 com a formação da *Open Verilog International* (OVI).

O Verilog foi ratificado pelo IEEE, sendo Verilog-95 (padrão IEEE 1364-1995), Verilog 2001 (IEEE 1364-2001) e Verilog 2005 (IEEE 1364-2005). O Verilog tem uma grande semelhança com a linguagem de programação C. (Midorikawa, Edson T., 2001, p. 3).

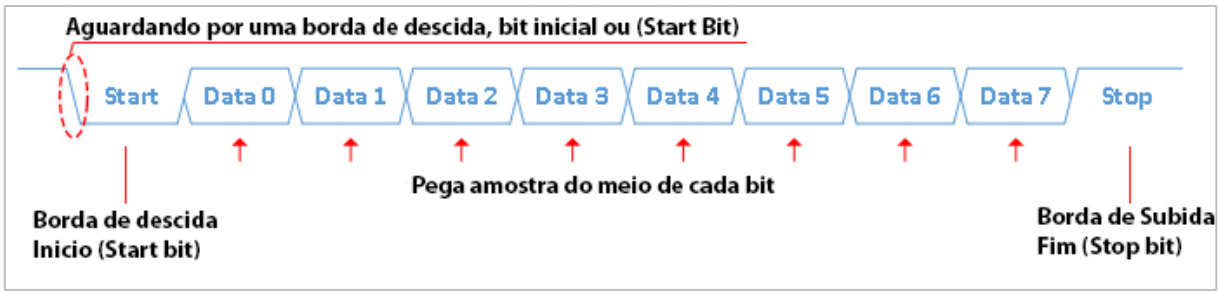
1.6. Sobre UART

O significado de UART é *Universal Asynchronous Transmitter/Receiver*, Transmissor e Receptor Universal Assíncrono, é um componente de hardware tipicamente encontrado em dispositivos com portas seriais assíncronas, utilizado para converter dados entre comunicações seriais e paralelas. Alguns exemplos de portas controladas pelas UARTs; Porta Serial RS232, eram padrão em PCs e outros dispositivos antes de as USB se tornarem comuns; Portas seriais que são conectados ao PC por um módulo conversor USB (AXELSON, J., 2007);

> Portas que se conectam a um PC através de um conversor USB que usa um driver que atribui uma porta COM para o dispositivo. Os conversores estão disponíveis como módulos e como chips para incorporação em circuitos. Um conversor pode converter entre USB e RS-232, RS-485, TTL serial, ou mesmo uma interface paralela. (AXELSON, J., 2007)

Esta última citação de AXELSON, J., é a forma em que o UART é utilizado neste projeto, um módulo UART é implementando no FPGA e conectado através da porta USB UART do kit de desenvolvimento à porta USB do PC servidor, que converte os dados seriais transmitidos pela USB em *full-duplex*, (transmite nas duas direções, cada um com sua linha, receptor e transmissor ou Rx/Tx).

Figura 4 – UART, fluxo de dados serial



Fonte: (NANDLAND, 2016)

Na Figura 4 é apresentado o fluxo de dados do receptor serial UART, de característica assíncrona, este fica em estado de espera até que recebe o bit inicial ou *Start bit*, para iniciar a detecção de um byte em uma cadeia de 10 bits, 1 bit inicial, 8 bits de dados e 1 bit final, ao fim da detecção este tem em sua saída, e.g. Out[7,0] a cadeia inteira de bits que formam o byte enviado, que podem ser lidos agora em paralelo.

O envio através do transmissor é feito de forma similar, este quando recebe uma cadeia de bits a serem enviados, primeiro este emite o *start bit* ou bit inicial e depois transfere bit a bit a cadeia a ser enviada finalizando com 1 *stop bit*.

1.7. Tecnologias web, cliente/servidor e interface gráfica.

As informações deste tópico descrevem as tecnologias web utilizadas no desenvolvimento do lado servidor e cliente. O lado servidor deste trabalho foi desenvolvido utilizando a linguagem de programação JavaScript para Node.js com *framework* Express e lado

cliente ou interface gráfica, é uma aplicação desenvolvido em forma de website utilizando as atuais tecnologias web, que serão descritos nos sub tópicos seguintes.

1.7.1. Lado servidor com Node.js

Node.js é um ambiente de E/S não bloqueante assíncrono e orientado a eventos, construído sobre o motor V8 de JavaScript do Google. Node.js foi projetado para criar aplicativos de rede escaláveis para muitas conexões que podem ser manipuladas simultaneamente.

Node.js é semelhante a projetos e influenciado por sistemas como *Ruby's Event Machine* ou *Python's Twisted*. O Node.js leva o modelo de evento um pouco mais, ele apresenta um *loop* de evento como uma construção em tempo de execução em vez de como uma biblioteca. Em outros sistemas há sempre uma chamada de bloqueio para iniciar o ciclo de eventos. Normalmente, o comportamento é definido através de *callbacks*, que são funções passadas como parâmetro de outra função quando um evento acontece, no início de um script e no final, inicia um servidor através de uma chamada de bloqueio como *EventMachine::run()*.

No Node.js não existe tal chamada de inicie-o-laço-de-evento, o Node.js simplesmente entra no *loop* de eventos depois de executar o script de entrada e sai do *loop* de eventos quando não há mais *callbacks* a executar, esse comportamento é como o JavaScript do navegador - o *loop* de eventos está oculto do usuário. O HTTP é uma entidade de primeira classe no Node, projetado com *streaming* e baixa latência em mente, isso torna o Node.js bem adaptado para a fundação de uma biblioteca ou *framework* da Web.

O ecossistema de pacotes do Node.js, “npm”, é o maior ecossistema de bibliotecas de código aberto do mundo. (Node.js Foundation, 2016, tradução nossa).

1.7.2. Google motor V8

Como parte fundamental da tecnologia de construção do Node.js este item descreve um pouco desta tecnologia. O Goolge V8 é um interpretador de JavaScript de alta performance de código fonte aberto, escrito em C++ e utilizado principalmente no Google Chrome, navegador de código aberto do Google.

O V8 implementa o ECMAScript conforme especificado em ECMA-262 que define e padroniza linguagens de programação de propósito geral, o V8 roda nos sistemas, Microsoft Windows XP ou mais recente, Mac OS X 10.5+ e sistemas Linux que utilizam processadores do tipo IA-32, ARM ou MIPS e pode ser executado autônomo, ou pode ser incorporado em um aplicativo C++, assim como o Node.js.

O motor V8 compila e executa código-fonte em JavaScript, manipula alocação de memória para os objetos e o coletor de lixo desaloca os objetos que ele não precisa mais. O coletor de lixo *stop-the-world*, geracional e preciso do V8 é uma das chaves do desempenho do V8 (Google Developers, 2016, tradução nossa).

1.7.3. Framework Express.js para Node.js

O Express é um projeto da Fundação Node.js, um *framework* JavaScript para Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicações web e móvel que roda no lado servidor (StrongLoop, Inc e IBM, 2016).

Este framework é utilizado no servidor do Laboratório Remoto para a organização e construção de parte do lado cliente, ele controla as rotas “URL” de acesso e a renderização de parte das páginas web servidos no lado cliente.

1.7.4. Linguagem de programação JavaScript

A JavaScript ou JS é uma linguagem de programação leve, interpretada e orientada a objetos com funções de primeira classe, [a linguagem JavaScript foi criada por Brendan Eich enquanto trabalhava na Netscape, e desde então foi atualizada para conformar-se ao padrão ECMA-262 Edition 5 e suas versões mais recentes], conhecida como a linguagem de *scripting* para páginas Web, mas também utilizada em muitos ambientes fora dos navegadores. (Mozilla Developer Network, 2016).

Outra aplicação comum para JavaScript é usá-lo como uma linguagem de *scripting* para o lado servidor (da Web). Um servidor web de JavaScript expõe objetos de *host* que representam uma solicitação HTTP e objetos de resposta, que são então manipulados por um programa em JavaScript que gera páginas web dinamicamente.

O Node.js é um exemplo popular deste tipo de uso, o Javascript é uma linguagem de *scripting* baseada em protótipos, multi-paradigma e dinâmica, suportando os estilos orientado a objetos, imperativo e funcional. (Mozilla Developer Network, 2016).

1.7.5. Lado cliente e interface gráfica.

O lado cliente do Laboratório Remoto utiliza as tecnologias web atuais como, HTML5 que é a linguagem de marcação de hipertexto para apresentar e estruturar o conteúdo na web, CSS3 é a linguagem que define os estilos e dão aparência para as páginas web e a linguagem de programação JavaScritp, descrito no item 1.7.4, para controlar elementos da aplicação no lado cliente e sua comunicação com o servidor utilizando a técnica Ajax.

O lado cliente foi desenvolvido com o *framework* Bootstrap para parte visual, que é um *framework* para desenvolvimento *front-end* (lado cliente de aplicações web), de projetos responsivo para dispositivos móveis e na web. Este *framework* tem como dependência a biblioteca JQuery, que é uma biblioteca JavaScript de código aberto que conta com uma grande gama de recursos para facilitar a manipulação de páginas web. (Bootsrap, JavaScript Overview, 2016).

1.7.6. Câmeras IP

Como foi proposto para este trabalho a visualização do Kit de desenvolvimento do FPGA por *streaming* de vídeo através da interface gráfica do usuário, foi adotado então a utilização de Câmeras IP para a transmissão de vídeo.

A câmera IP é um dispositivo de hardware para transmissão de vídeo através de uma rede IP, como LAN, Intranet ou Internet. Usando simplesmente um navegador web e uma conexão de Internet, pode-se convenientemente ter acesso ao vídeo de uma câmera IP, em alguns casos, até áudio, de qualquer local que esteja.

Os modelos atuais são compatíveis com as tecnologias Ethernet e Wi-Fi e são separadas em categorias como Pan/Tilt/Zoom que permite ao usuário mudar o ângulo das câmeras, habilitar áudio, controlar uso de luz infravermelha para uso noturno, entre outros.

É possível ainda instalar a câmera IP com os convencionais fios (cabo de rede) para streaming de áudio e vídeo ou utilizar tecnologia Wireless com criptografia. A vantagem é que

o sistema pode ser facilmente ampliado, com a utilização de hub’s, que custam bem menos do que uma placa de captura de imagem das câmeras analógicas. (Sitehosting, 2016).

1.7.7. Persistência de dados

As informações enviadas para o servidor como, contas de usuários, informações de configurações e informações de armazenamento do arquivo bit do projeto sintetizado pelo usuário, são persistidas pelo servidor utilizando um banco de dados relacional.

O banco de dados adotado para este projeto é o *MySQL Community Server*, “é o banco de dados de código aberto mais popular do mundo, que é suportado por uma comunidade ativa de desenvolvedores e entusiastas de código aberto”, segundo (MySQL, 2016).

2. Trabalhos correlatos

Laboratórios de acesso remoto para circuitos digitais utilizando dispositivos FPGAs através de uma conexão web, foram propostos por outras pesquisas anteriores, com o objetivo de maximizarem o tempo de uso e a facilidade de acesso aos laboratórios convencionais, dentre os trabalhos propostos há o trabalho do autor Trevelyan, J. para área de Engenharia Mecatrônica, onde o autor descreve as vantagens de um laboratório remoto e reforça a sua importância.

Segundo Trevelyan, J. (2004 apud (Goldberg 1999, Taylor e Trevelyan 1995, Henry 2003)), laboratórios de acesso remoto usando conexões de Internet, existem em forma de protótipo desde meados de 1990. Trevelyan, J. et al, em outubro de 1994, iniciaram na Universidade da Austrália Ocidental um laboratório online, o *Australia's Telerobot on the Web.*

O autor apresenta em seu artigo uma revisão das principais experiências que eles aprenderam desde 1994 a 2004 e descreve brevemente o *Telelabs*, um *framework* de custo efetivo que provê uma série extensível de laboratórios online que podem ser mantidos por um orçamento normal de operação. O laboratório de Trevelyan, J. et al entrou em funcionamento em 2002 e em 2003 continha 5 diferentes módulos, incluindo o seu primeiro laboratório online, o *Telerobot*.

A partir de 2002, Trevelyan, J. et al concentrou seu projeto na atividade de aprendizado efetivo para estudantes usando o laboratório online e avaliação da efetividade do aprendizado. O autor reporta em sua publicação os resultados dos ensaios com o laboratório remoto online como parte de um tutorial de exercícios semanais para graduandos do curto se engenharia Mecatrônica.

Os ensaios revelaram que os estudantes que usaram o laboratório de acesso remoto, operaram equipamentos por muito mais tempo do que em aula de laboratório convencional e como resultado tiveram o aprendizado melhorado consideravelmente. Isto permite aos estudantes tempo necessário para explorar as diferenças entre a teoria e o comportamento real dos equipamentos por conta própria.

Com a carga horaria obrigatória das aulas normais de laboratório, em maior parte tendem a seguirem uma sequência pré-determinada de medições, cálculos e escrita de relatórios, sem necessariamente entenderem o que eles estão fazendo. O estilo de interação com os equipamentos usando o laboratório de acesso remoto é muito mais representativo de um cenário típico industrial.

O autor Pastor, J. S. et al (2004), apresenta em seu artigo um framework para testar protótipos de microprocessadores. Um microprocessador RISC é projetado pelos estudantes utilizando a linguagem VHDL e adaptado para ser implementado em um dispositivo FPGA.

O correto comportamento do microprocessador projetado é verificado executando um programa de testes escrito e compilado pelos alunos para este microprocessador. Usando um cliente web, os usuários enviam um arquivo com o programa de testes e o projeto para um laboratório remoto, que são carregados em um dispositivo FPGA real. Um conjunto de ferramentas para depuração da execução remota dos testes foram desenvolvidos usando uma interface gráfica similar as demais ferramentas de depuração.

Grupos de estudantes do curso de Arquitetura de Computadores participaram deste experimento. As boas opiniões recebidas dos alunos, sugerem a incorporação dos experimentos do laboratório remoto no próximo curso regular.

Hashemian, R. e Riddley, J. (2007) propôs o FPGA e-Lab, uma técnica de acesso remoto ao laboratório para projetos e testes com FPGA, em seu artigo o autor ilustra o objetivo e importância do seu trabalho para educação em engenharia, com o enriquecimento da experiência de uso do laboratório permitindo aos estudantes mais tempo de uso do laboratório e também melhorando o aprendizado com experimentos complementares.

A técnica de acesso remoto proposta por R. Hashemian e J. Riddley é feita através do Assistente de Área de Trabalho Remota do Microsoft Windows XP, que permite o acesso remoto ao computador do laboratório que estará rodando uma aplicação desktop que mostra a imagem de um kit de desenvolvimento *Xilinx Spartan-3E Starter Kit*, por streaming de vídeo e permite o controle do dispositivo FPGA que está ligado ao computador do laboratório via uma interface gráfica interativa *LabView* e outros hardwares bem como porta serial RS-232 e cabo USB.

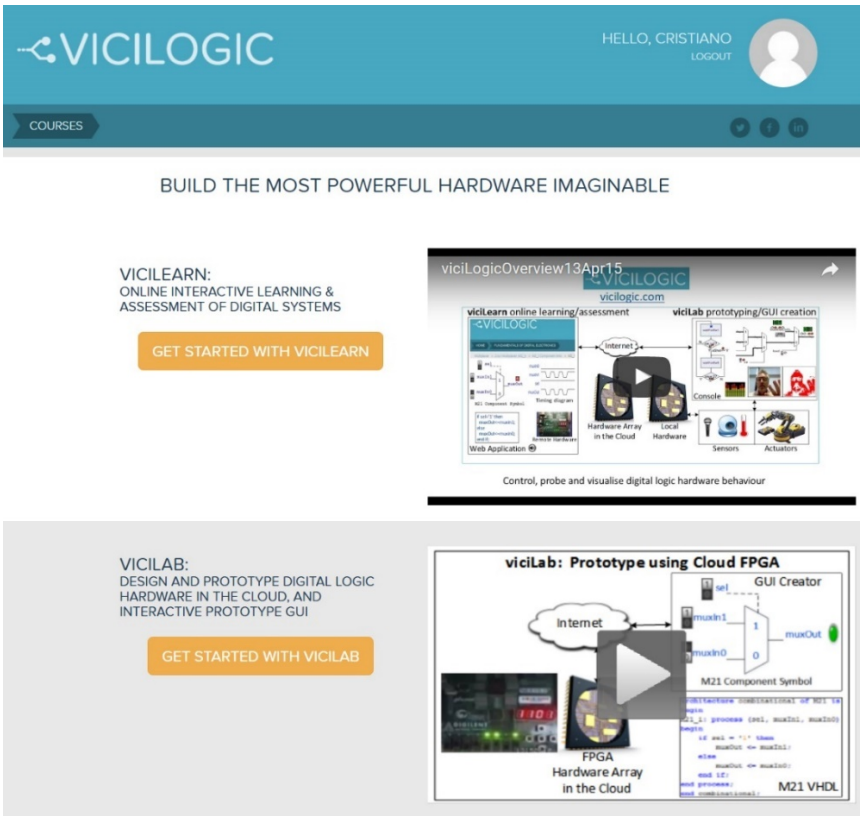
O autor Morgan, F. (2011), apresenta em seu artigo o projeto de um laboratório remoto para melhorar o aprendizado em sistemas digitais para os cursos de engenharia elétrica e eletrônica utilizando FPGAs. Morgan, F. também cita os benefícios descritos pelo autor Trevelyan, J. e ressalta a melhora do aprendizado pela aplicação da prática de “aprender fazendo”, custos reduzidos, compartilhamento de recursos, flexibilidade e conveniência do tempo de uso do laboratório e melhoras da interação com o estudante permitindo novas técnicas pedagógicas. Morgan, F., até o ano de 2016 publicou 3 artigos sobre o laboratório remoto, em

seu primeiro artigo ele descreve os elementos do laboratório remoto e demonstra aplicações de utilização para apoiar o aprendizado.

O laboratório descrito por Morgan, F. em seu artigo, é composto basicamente por um módulo com um kit de desenvolvimento FPGA (Digilent Nexys 2) e uma webcam, ligados a um servidor USB e este conectado a um servidor web, sua arquitetura é expansível e suporta uma certa quantidade de pares de dispositivos FPGAs com sua respectiva webcam e servidor USB, o servidor web controla o acesso alocando uma sessão para cada usuário, permitindo o envio do projeto, a visualização e interação.

Morgan em seu terceiro artigo em 2014, definiu o seu laboratório remoto como RFL (Remote FPGA Lab), sendo uma aplicação baseada na web, com controles interativos para manuseamento de *hardwares* de circuitos lógicos configuráveis na nuvem em tempo real e oferecendo suporte a uma abordagem de aprendizado de sistemas digitais, antes disponível no website <http://remoteFPGA.com>, que agora é redirecionado para: <http://vicilogic.com/>, referente ao mesmo projeto de Morgan, F. que foi renomeado para *Vicilogic* (VICILOGIC, 2016).

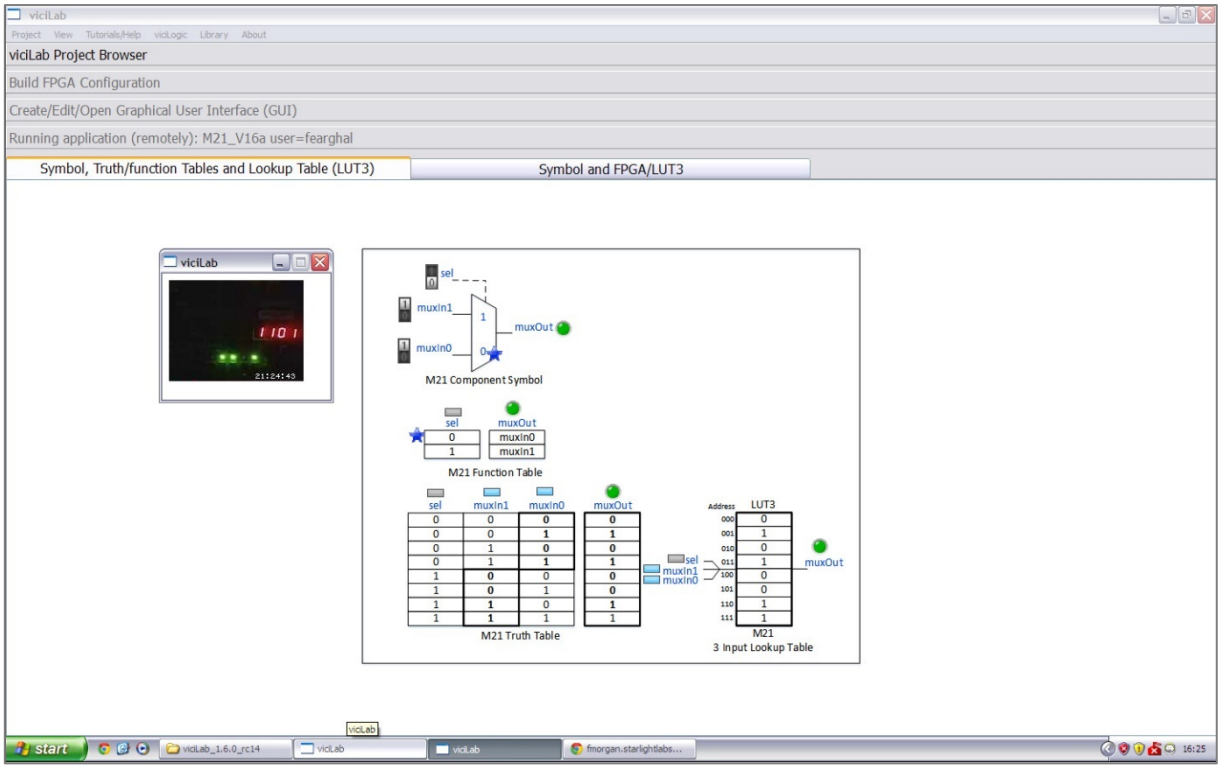
Figura 5 – Laboratório remoto *Vicilogic* do autor Morgan, F.



Fonte: (Próprio autor, 2016)

O laboratório *Vicilogic*, ilustrado na Figura 5, possui dois módulos, o *Vicilern*, que conta com um curso online interativo com vídeos e animações, que faz a introdução no aprendizado de sistemas digitais e o módulo *Vicilab*, que conta com uma ferramenta desktop para o sistema operacional Microsoft Windows, onde o usuário tem acesso a um kit de desenvolvimento FPGA real, através da aplicação desktop remotamente, disponível para download no módulo *Vicilab*, a Figura 6 ilustra a ferramenta desktop *viciLab*, onde é possível ver uma janela sobreposta com a visualização por vídeo do kit de desenvolvimento FPGA e no centro da tela os controles, que possibilitam controlar o FPGA remotamente.

Figura 6 – Aplicação desktop *viciLab* do autor Morgan, F.



Fonte: (Próprio autor, 2016)

Os trabalhos correlatos descritos neste item reforçam a importância de um laboratório e o seu tempo de uso para o desenvolvimento do aprendizado prático, através de laboratórios de acesso remoto.

Do primeiro trabalho do autor Trevelyan, J. (2004), a ideia de um laboratório de orçamento normal ou baixo de operação, foi levado em consideração no desenvolvimento deste projeto, resultando em um laboratório remoto simplificado, de baixo orçamento e de fácil manutenção e acesso.

Do primeiro trabalho, autor Trevelyan, J. (2004), segundo trabalho de J. S. Pastor et al (2004) e o terceiro trabalho dos autores Hashemian, R. e Riddley, J. (2007), descrito neste capítulo, reforçam a eficácia da utilização do laboratório de acesso remoto e os seus resultados para o aprendizado prático.

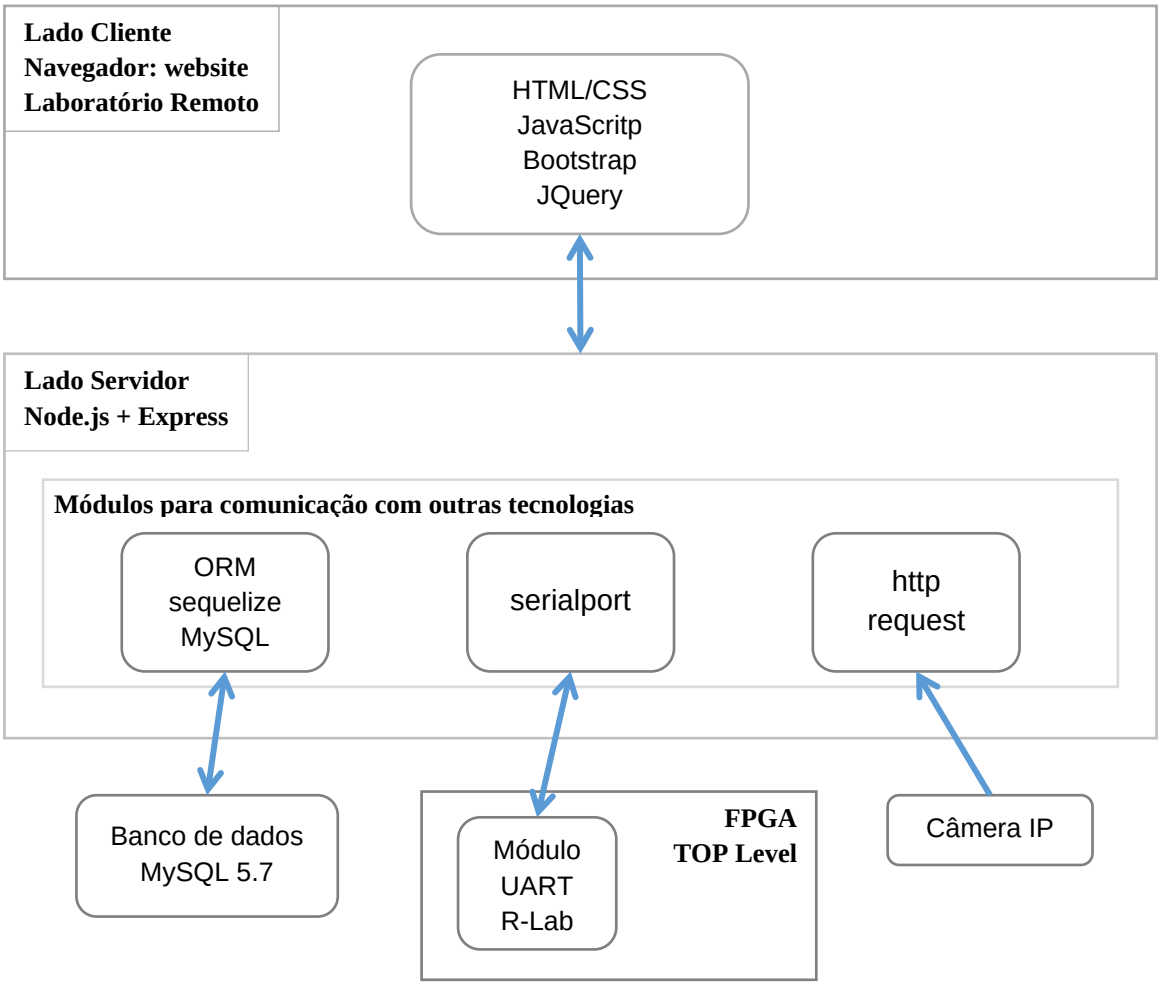
O quarto trabalho do autor Morgan, F. (2011), apresentado neste item, é o mais similar a este projeto, que também abordou o laboratório de acesso remoto baseado em um website, porém a visualização, controle e envio de projetos é feito através de uma aplicação desktop para o sistema operacional Windows, sendo neste ponto o maior diferencial deste projeto, no qual a visualização, controle e envio é feito diretamente no website do projeto desenvolvido, conforme descritos no item 3.1 e os seus sub itens, sem a necessidade de baixar qualquer *software* adicional e independente do sistema operacional utilizado o tornando mais acessível e simples de utilizar.

3. Arquitetura

Este capítulo aborda a estrutura e a arquitetura do projeto e como funciona os componentes e módulos do Laboratório Remoto e as suas comunicações.

O Laboratório Remoto, como citado anteriormente na introdução, é uma plataforma web desenvolvido no modelo de cliente servidor, a Figura 7 ilustra uma visão geral da arquitetura do Laboratório Remoto.

Figura 7 – Visão geral da arquitetura do Laboratório Remoto.

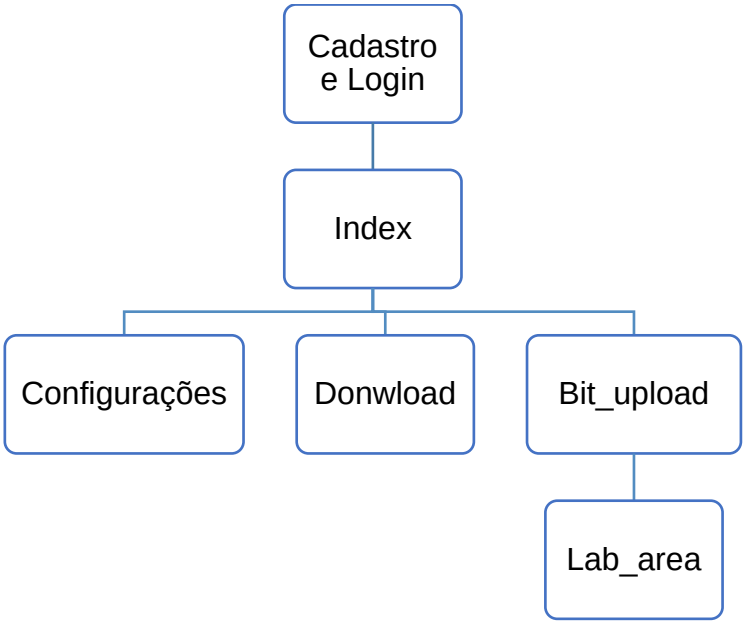


Fonte: (Próprio autor, 2016)

3.1. Estrutura do lado cliente.

As interfaces do lado cliente do Laboratório Remoto foram desenvolvidas utilizando HTML5/CSS, Bootstrap e JQuery, como citado no item 1.7.5. O *framework* Bootstrap de CSS, provê o visual das interfaces através da utilização de suas classes CSS e animações através da biblioteca JQuery, o *framework* possui visual limpo leve e simples que torna a navegação agradável aos usuários. Para a manipulação dos elementos do lado cliente das interfaces não foi utilizado nenhum *framework*, sendo utilizado apenas JavaScript puro.

Figura 8 - Mapa das *view*/interfaces lado cliente.



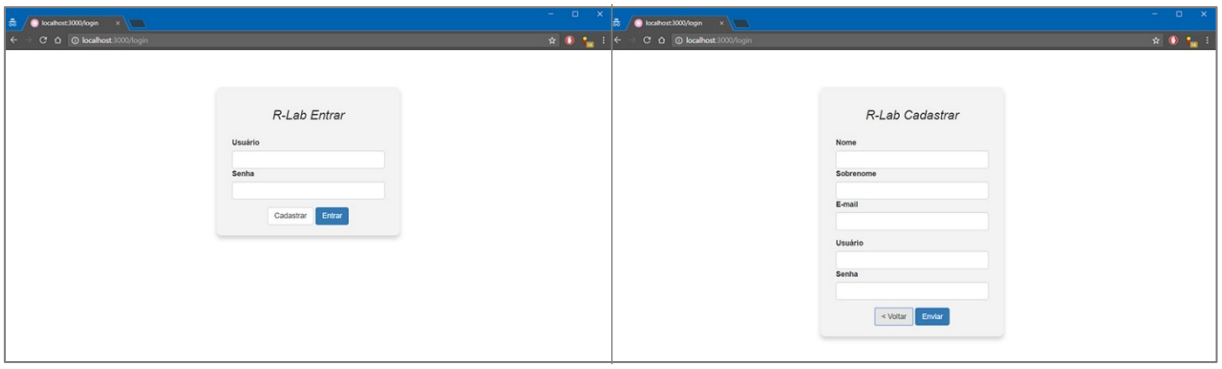
Fonte: (Próprio autor, 2016).

Na Figura 8 é apresentado a estrutura das interfaces de usuário, a interface do Laboratório Remoto é composta basicamente por 6 tela ou *views*, que são descritos a seguir.

3.1.1. Tela de cadastro e login

Tela inicial, onde o usuário faz o seu cadastro ou login caso já tenha cadastro, o usuário sempre é redirecionado para esta tela caso não tenha efetuado login. A Figura 9, mostra uma visualização da *view* de login e cadastro, ambos login e cadastro fazem parte da mesma *view*, são alternados pelo JavaScript no lado cliente.

Figura 9 – Tela de Login e cadastro.

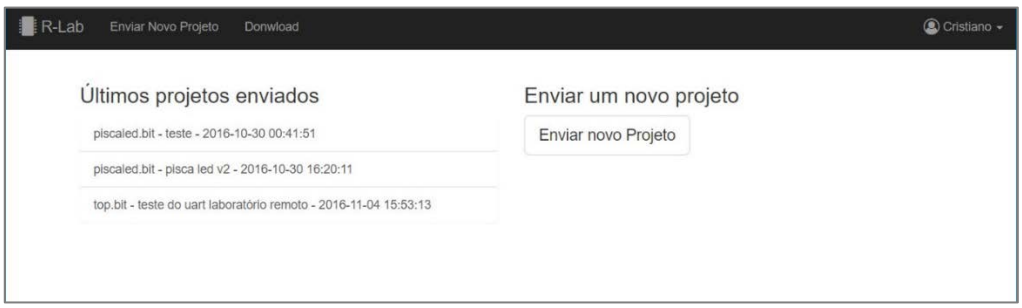


Fonte: (Próprio autor, 2016).

3.1.2. Tela inicial index

Página inicial do laboratório (Figura 10) esta página dá acesso aos outros recursos do laboratório, Configuração, Download e a tela de upload do arquivo bit, nesta tela o usuário tem a visualização dos últimos projetos, arquivos *.bit, enviados com sucesso, é possível utiliza-lo novamente caso queira repetir a simulação do mesmo.

Figura 10 – Tela inicial do laboratório.



Fonte: (Próprio autor, 2016)

3.1.3. Tela de configurações:

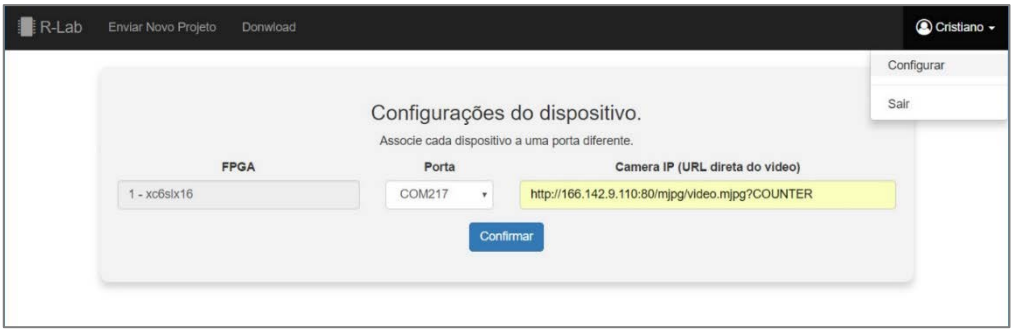
Esta interface é onde é feito a configuração dos kits de desenvolvimento FPGAs conectados ao servido, esta configuração só é necessária ser executada apenas uma vez, ou sempre que for adicionado um novo kit de desenvolvimento ao servidor, esta tela pode ser acessível a partir do menu com o nome de usuário caso este seja administrado, no canto superior

direito da barra de navegação conforme mostra a Figura 11. Nesta tela é listado os nomes/modelo dos FPGAs detectados pelo servidor e todas as postar seriais detectadas ativas com algum dispositivo conectado.

Na Figura 11 é ilustrado que um dispositivo foi detectado no servido, para a configuração basta selecionar uma porta para o dispositivo, esta porta é a utilizada para controlar o projeto desenvolvido para o FPGA, através da porta UART, como ela não é detectada automaticamente, é necessário saber qual a porta correta para associar ao Kit do FPGA, é obrigatório associar ao menos uma porta a um kit de desenvolvimento FPGA para prosseguir com a configuração.

No último campo é possível adicionar o link ou IP da câmera IP que é carregado quando a simulação estiver em execução, este não é um item obrigatório, caso não adicionado a simulação funciona normalmente, porém sem visualização, este item pode ser adicionado posteriormente voltando ao menu de configurações.

Figura 11 – Tela de configurações

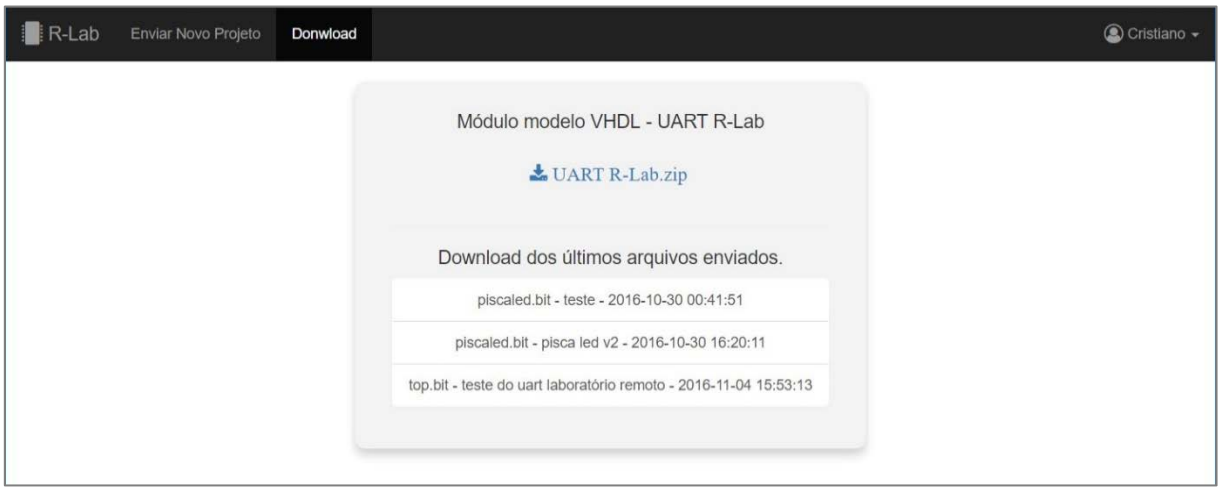


Fonte: (Próprio autor, 2016).

3.1.4. Tela download

Nesta tela, Figura 12, está disponível para download o módulo VHDL UART R-Lab e um exemplo de utilização, este módulo é o responsável pela comunicação entre o FPGA e o servidor, este módulo é implementado no lado do FPGA, para o usuário utilizar no seu projeto, caso o usuário do laboratório opte por não utilizar este módulo, ele pode enviar normalmente seu arquivo bit este é programado no kit do FPGA do servidor, mas não é possível controla-lo, tendo apenas a visualização. Fica disponível para download também nesta página os últimos 10 arquivos bit que o usuário enviou com sucesso para o servidor.

Figura 12 – Tela de downloads.

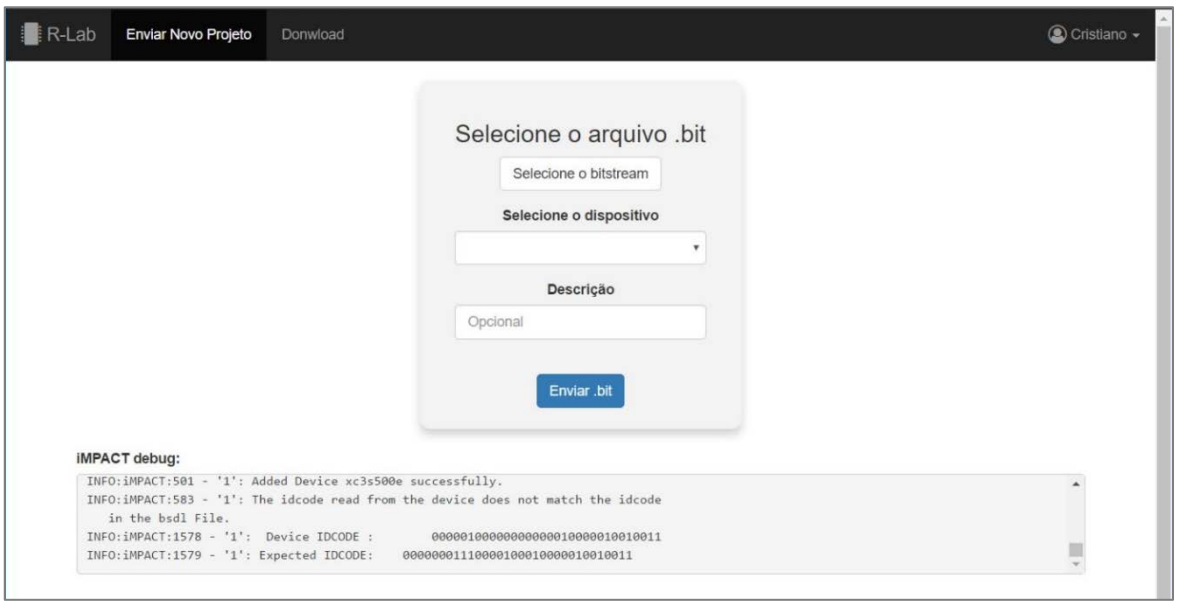


Fonte: (Próprio autor, 2016).

3.1.5. Tela bit_upload:

Esta é a interface onde o usuário efetua o upload do arquivo .bit, que é gerado pelo sintetizador da Xilinx a ferramenta ISE Design Suite, descrito no item 1.3.2. Tem um campo/botão para selecionar o arquivo em disco, um campo para selecionar o modelo do FPGA disponível e um campo de descrição opcional, após o envio caso nenhum erro ocorra o usuário é redirecionado diretamente para a tela Lab_area onde ele pode visualizar e controlar o FPGA.

Figura 13 – Tela de upload do arquivo bit



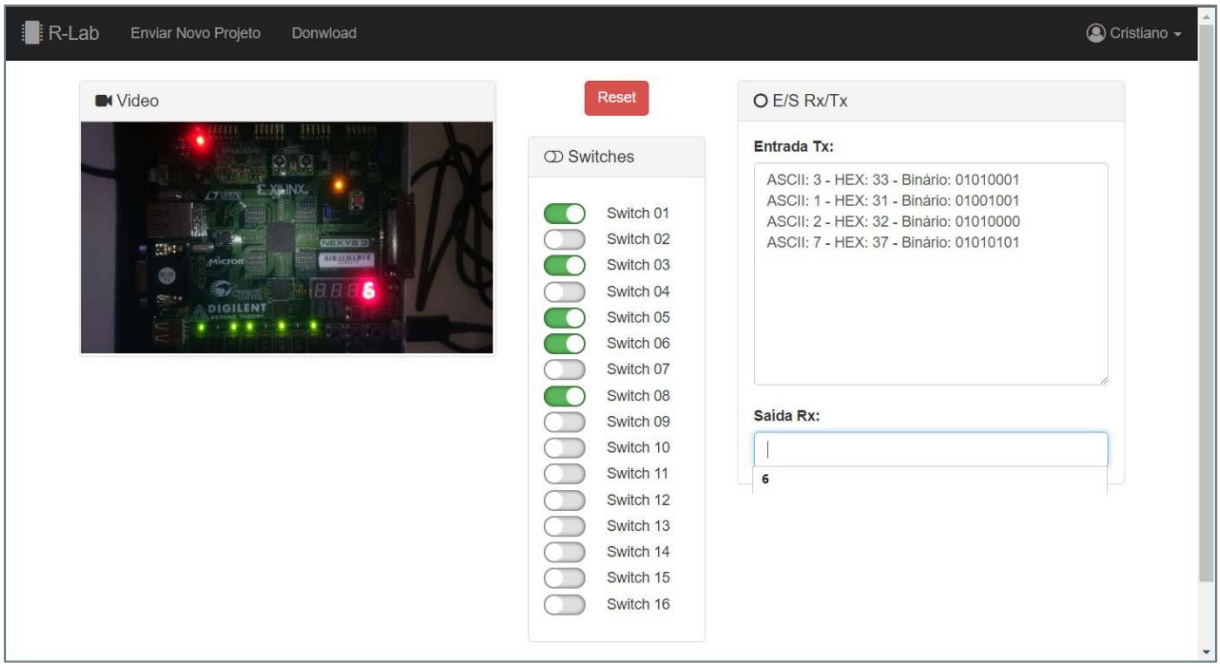
Fonte: (Próprio autor, 2016).

A Figura 13 ilustra a tela Bit upload com uma mensagem de erro do iMPACT, onde foi feita a tentativa de envio de um arquivo bit gerado para um modelo de FPGA diferente do disponível no kit ligado ao servidor, neste caso o usuário não é redirecionado. O funcionamento no lado servidor é descrito no item 3.2.5.

3.1.6. Tela de simulação, controle e visualização, lab_area

A tela lab_area é a tela onde pode ser visualizado o resultado do projeto que foi enviado, em execução, caso o projeto contenha o módulo fornecido na tela de download para fazer a comunicação com o servidor, é possível controlar a aplicação em execução da forma em que o usuário projetou os comandos recebidos pelo módulo UART. A Figura 14 ilustra o exemplo de teste que é distribuído junto do módulo UART R-Lab em execução.

Figura 14 – Tela de simulação, controle e visualização do Kit FPGA.



Fonte: (Próprio autor, 2016).

No exemplo de teste, os 8 LEDs da parte inferior do kit de desenvolvimento estão ligados nos 8 primeiros *switchs*/chaves, as chaves foram ativadas e os LEDs correspondentes da direita para esquerda foram ligados e no painel de E/S à direita, mostra a entrada que veio do transmissor do módulo UART, no exemplo executado em seu *TOP level*, está implementado

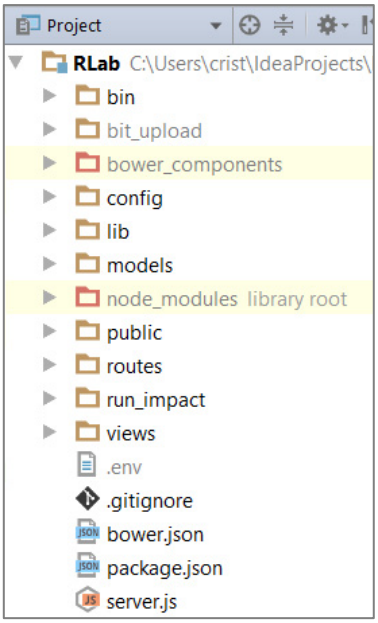
soma + 1 para entrada recebido e envia de volta, foram enviados os números “2”, “0”, “1” e “6”, a caixa de texto “Entrada Tx:” mostra o resultado recebido de volta do FPGA em 3 formas, ASCII, Hexadecimal e binário e o resultado em ASCII pode-se notar que está na sequencia os dígitos “3”, “1”, “2” e “7”, todos incrementados em + 1 como implementado no *TOP level* de teste.

3.2. Arquitetura do lado servidor.

O servidor é a parte principal do laboratório, onde está concentrado toda a aplicação e é por onde tramita todas as comunicações do cliente, kit de desenvolvimento FPGA, banco de dados e câmeras, o servidor do Laboratório Remoto foi desenvolvido utilizando como base principal o Node.js com o framework Express.js, descritos nos itens, 1.7.1 e 1.7.3.

Este tópico aborda a estrutura do servidor e seus módulos, a Figura 15 ilustra a estrutura completa de diretórios do servidor, parte dessa estrutura é gerado pelo *framework* Express descrito no item 3.2.2, os diretórios *config* e *models* fazem parte o ORM Sequelize, descritos no item 3.2.3, *bower_components* e *node_modules* são os diretórios das bibliotecas utilizadas no projeto e são descritos no item 3.2.1, o diretório *lib* contém bibliotecas adicionais desenvolvidas para o projeto, item 3.2.4 e por fim *bit_upload* e *run_impact* fazem parte do gerenciamento do arquivo .bit enviado pelo usuário do laboratório no item 3.2.5.

Figura 15 – Estrutura completa de diretórios do servidor.



Fonte: (Próprio autor, 2016).

3.2.1. Dependências

Este projeto utiliza dois gerenciadores de pacotes de dependência, o NPM que é o gerenciador de dependências do Node.js e o Bower, que é um gerenciador de pacotes para web lado cliente.

Para facilitar no desenvolvimento, *deploy*, distribuição e gerenciamento bibliotecas de uma aplicação, utiliza-se os gerenciadores de pacotes, eles são responsáveis por instalar todas as dependências nas versões descritas pelo desenvolvedor, necessárias para o correto funcionamento da aplicação.

Ainda na Figura 15, nota-se os arquivos JSON no diretório raiz do servidor, *bower.json* e *package.json*, estes são arquivos que contém as informações das dependências do laboratório.

A Figura 16 mostra o conteúdo de cada um dos arquivos JSON, ambos possuem o atributo "dependencies", onde estão listadas todas as dependências do Laboratório Remoto, seguidas de suas referentes versões na frente do nome, no *bower.json*, estão as dependências utilizadas no lado cliente, que são armazenadas no diretório "*bower_components*" Figura 15 e são servidor como os arquivos estáticos, igual ao diretório "*public*", descrito no item 3.2.2.

Figura 16 – Arquivos de dependências package.json e bower.json

package.json

```
{
  "name": "RLab",
  "description": "Laboratório re...",
  "version": "0.9.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "bcryptjs": "^2.3.0",
    "body-parser": "~1.15.1",
    "cookie-parser": "~1.4.3",
    "debug": "~2.2.0",
    "dotenv": "^2.0.0",
    "express": "~4.13.4",
    "fs-extra": "latest",
    "hbs": "~4.0.0",
    "jsonwebtoken": "^7.1.9",
    "morgan": "~1.7.0",
    "multer": "^1.2.0",
    "mysql": "^2.11.1",
    "request": "^2.78.0",
    "sequelize": "^3.24.3",
    "serialport": "^4.0.3",
    "serve-favicon": "~2.3.0"
  }
}
```

bower.json

```
{
  "name": "RLab",
  "description": "Laboratorio R…",
  "main": "",
  "authors": [
    "Vicente, C. <cristiano@live.jp>"
  ],
  "license": "MIT",
  "homepage": "",
  "private": true,
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "test",
    "tests"
  ],
  "dependencies": {
    "bootstrap": "^3.3.7",
    "font-awesome": "^4.6.3"
  }
}
```

Fonte: (Elaborado pelo Autor, 2016).

As dependências listadas no "*package.json*", são armazenadas no diretório "*node_modules*", este contém todas as dependências que funcionam no lado servidor incluindo as dependências de dependências, de forma compartilhada, caso uma biblioteca dependa de outra, se sua dependência já existir apenas um código é armazenado e compartilhado entre as bibliotecas (NPM, 2016).

Segue uma breve descrição do que são as dependências listadas na Figura 16;

- **bcryptjs:** biblioteca de criptografia, utilizada para codificar a senha de usuário armazenado no banco de dados, para segurança e maior privacidade de senha.
- **body-parser:** dependência padrão do *framework* Express, serve para fazer o análise do corpo das requisições e adicionas as informações analisadas no *req.body*.
- **cookie-parser:** outra dependência padrão do Express, serve para fazer o análise do atributo Cookie do cabeçalho de requisição HTTP e adiciona as informações no *req.cookies*.
- **debug:** módulo utilizado pelo Express para registrar as correspondências de rodas e seus ciclos de requisição e resposta e também dos outros módulos que estão em uso. (EXPRESS, *Debugging Express*, 2016).
- **dotenv:** Este módulo carrega as informações do arquivo **.env**, que são variáveis de ambiente, do tipo porta http a ser utilizado, modo de execução, e.g. *production* ou *development*, o arquivo .env está presente no diretório raiz do servidor Figura 15.
- **express:** *framework* Express, descrito nos itens 1.7.3 e 3.2.2.
- **fs-extra:** Esta dependência complementa algumas funções na biblioteca padrão fs, *file system*, como a função copiar arquivo, ausente na biblioteca padrão, responsável por acessar e manipular arquivos do sistema.
- **hbs:** Módulo responsável por renderizar *views* html no lado servidor.
- **jsonwebsoken:** Módulo que gera e valida os *tokens* de acesso, utilizado pelo módulo *authentication*, descrito no item 3.2.4.
- **morgan:** Módulo padrão do Express, que registra as requisições http.
- **multer:** Módulo para manipulação de requisições do tipo *multipart/form-data*, são requisições utilizadas principalmente para upload de arquivos.
- **mysql:** Drive do banco de dados MySQL para o Node.js, utilizado pelo ORM Sequelize para realizar a conexão com o banco de dados.
- **request:** Módulo que faz requisições http, utilizado neste projeto para requisitar a url ou IP da Câmera IP, descrito no item 3.4.

- **sequelize:** ORM Sequelize, descrito no item 3.2.3.
- **serialport:** Módulo para acessar portas seriais para leitura e escrita.
- **serve-favico:** Módulo padrão do Express, apesar de opcional, ele vem junto das dependências do Express, ele serve o ícone para a página web que aparece na aba do navegador.
- **bootstrap:** *Framework* CSS que provê o visual das *view*, descrito nos itens 1.7.5 e 3.1.
- **font-awesome:** *Framework* CSS de ícones em forma de fonte.

A referência e documentação de todos os módulos descritos neste item podem ser encontradas no site do gerenciador de pacotes do Node.js NPM <https://www.npmjs.com/>.

3.2.2. Estrutura do express.js

A estrutura base de diretórios do servidor é gerado pelo *framework* Express.js, que conta com os pacotes/módulos necessários para prover recursos de uma aplicação web. A Figura 17 ilustra a estrutura gerada pelo express-generator, ferramenta do *framework* express.js. (EXPRESS, 2016).

Figura 17 – Estrutura de diretórios base gerado pelo express-generator.

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade

7 directories, 9 files
```

Fonte: (EXPRESS, 2016).

A estrutura gerada pelo express-generator, Figura 17, contém 7 diretórios base e 9 arquivos, sendo os principais descritos a seguir;

- **app.js:** Contém todas as configurações iniciais do servidor, como declarações de rotas e extensões utilizadas. O app.js neste projeto foi renomeado para server,js.
- **www:** É o script de inicialização do servidor, pode ser visto descrito no package.json na Figura 16 no atributo "*scripts: start*", ele importa o app.js que carrega todas as configurações do servidor, o www é executado/interpretado pelo processo node.exe.
- **package.json:** Este é o arquivo json do gerenciador de pacotes NPM do Node.js, visto na Figura 16, qual contem meta-dados referentes ao projeto, como descrição, dependências e configurações, (NODEJITSU, *What is the file `package.json`?*, 2016).
- **public**: Diretório contém os arquivos estáticos do lado cliente da aplicação web, como JavaScript, imagens e CSS.
- **routes**: Neste diretório ficam os códigos responsáveis por gerenciar os recursos servidos de cada rota da aplicação web, quando uma rota é chamada e.g. http://dominio.com/**users** a rota users.js é requisitado no servidor, está trata a requisição e responde com um recurso, que pode ser texto simples, html ou arquivos.
- **views:** Contém as views, interfaces a serem servidas para o cliente, podem ser em html, ou html pré renderizado no lado servidor, o express-generator conta com 7 opções (ejs, hbs, hjs, jade, pug, twig, vash), de motores de renderização html para o lado servidor, (EXPRESS, 2016).

3.2.3. ORM Sequelize, models e banco de dados

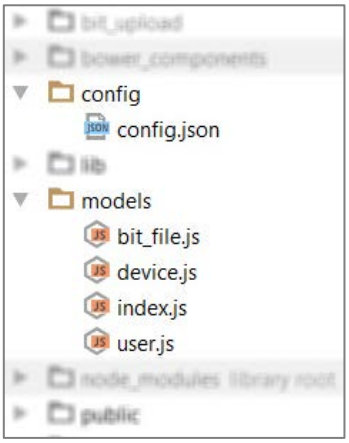
O framework Express não conta com um gerenciador de bando de dados ou ORM, sendo necessário criar classes para comunicação com o banco de dados ou utilizar módulo que faça as comunicações com o bando. Este projeto utiliza o Sequelize, um ORM para o Node.js que suporta os bancos de dados PostgreSQL, MySQL, MariaDB, SQLite e MSSQL (SEQUELIZE, 2016).

Ilustrado na Figura 18, estão 2 diretórios do ORM Sequelize, o item **config/config.json** possui as informações de conexão com o banco de dados, como "username", "password", "database", "host" e "dialect", dispostas em 3 objetos "development", "test" e "production",

permitindo utilizar 3 diferentes configurações para cada ambiente, de desenvolvimento, teste e produção.

Ainda na Figura 18, há o diretório das **models**, este possui as classes referentes as tabelas do banco de dados, neste diretório há 3 classes, sendo elas “user.js”, “bit_file.js” e “device.js”, o arquivo “index.js” não é uma *model*, dentro da estrutura de módulos do Node.js, é possível importar outras classes, e.g. utilizando o método require('./../models/user');, está importando a classe *user.js* do diretório *models*, porém pode-se importar o diretório todo como um pacote de classes caso este tenha um *index.js* referenciando seu conteúdo, e.g. require('./../models');, está requisição importar todas as classes do diretório *models* que estão especificadas no *index.js*.

Figura 18 – Diretórios do ORM Sequelize.

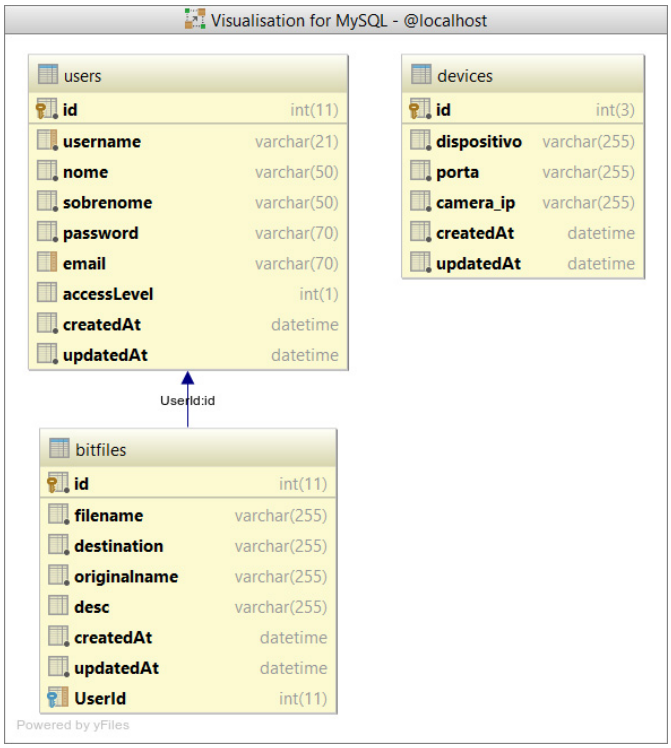


Fonte: (Próprio autor, 2016).

O ORM Sequelize possui um método “sequelize.sync()”, para sincronização das models no banco de dados, ele cria as tabelas do banco de dados, no esquema especificado no arquivo de configurações **config/config.json,** atributo “database”, conforme modelado nas classes do diretório “models”, este método possui 1 parâmetro, atributo “force”, que pode ser verdadeiro ou falso, quando verdadeiro sempre que executado ele executa um DROP em todas as tabelas do esquema configurado e as recria novamente, utilizado no desenvolvimento e na modelagem inicial das models para fins de teste, quando configurado para falso, o método “sequelize.sync()”, apenas verifica a existência das tabelas, se as mesmas não existirem ele as cria, este último assim utilizado para *deploy* da aplicação.

Este projeto implementa o método em sua inicialização em modo falso, sendo necessário apenas criar o esquema no banco de dados e configurar o arquivo **config/config.json**. Na Figura 19 é ilustrado o esquema do banco de dados gerado pelo Sequelize.

Figura 19 – Modelo do banco de dados.



Fonte: (Elaborado pelo Autor, 2016).

Nas classes de modelo do Sequelize é possível descrever não somente os atributos da entidade, mas também suas relações, as chaves, indexes, tipos de dado e métodos. Tomando como exemplo a Figura 19, mostra a tabela *users* e *bitfiles* relacionadas, na *model* da classe *user* é declarado esta relação e.g. “User.hasMany(models.BitFile);” e na model *bitfile* e.g. “BitFile.belongsTo(models.User);”.

O ORM possui um conjunto completo de métodos para *querying* para todas as principais sintaxes SQL, permitindo executar todas as chamadas ao bando através de métodos das classes sem a necessidade projetar as *strings* SQL, e.g. “User.findAll()”, busca todos os usuários, “User.create();”, insere um novo usuário. Conforme descrito na documentação do Sequelize. (SEQUELIZE, 2016).

3.2.4. Bibliotecas desenvolvidas para o projeto

Para o projeto do Laboratório Remoto foram desenvolvidos três módulos complementares, que são, “*authentication*”, “*device_identify*” e “*serial_uart*”, destacados na Figura 20 no diretório “lib”, o módulo *authentication* foi desenvolvido para fazer a autenticação

do login do usuário no laboratório e controlar sua seção, pois o *framework* Express não conta com um módulo para autenticação.

Os seus métodos estão divididos em arquivos separados no diretório *authentication* e são relacionados pelos arquivos *index.js*, bastando importar o diretório para ter todos os métodos como fora descrito no item 3.2.3, a forma de importar todas classes do diretório *models*, o módulo de autenticação gera um *token* que é adicionado no *cookie* do navegador, para verificar e identificar o usuário e a seção em todas as requisições, permitindo fazer o controle de acesso as rotas.

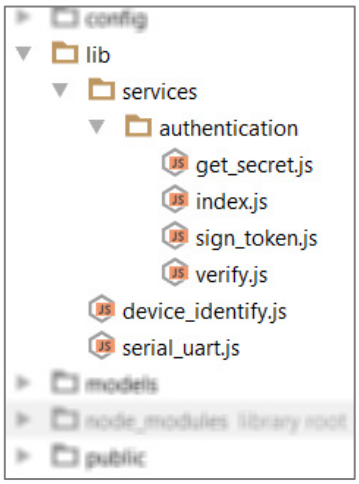
O módulo *"device_identify"*, faz a identificação do modelo do FPGA do kit de desenvolvimento e das portas seriais ativas no sistema, o kit de desenvolvimento é conectado ao servidor utilizando duas portas USB, como fora descrito no item 1.4.2, na Figura 3 destacado por dois retângulos numerados, estão as duas portas USB do kit utilizado no desenvolvimento deste projeto.

A porta com nome Adept USB é identificada utilizando a ferramenta iMPACT em modo batch, "linha de comando", o módulo *"device_identify"*, utiliza o método *execFile()* da biblioteca nativa do Node.js *"child_process"*, que permite o servidor executar um processo filho de um arquivo executável (NODE.JS, Child Process, 2016), assim este módulo pode instanciar um processo do iMPACT, e.g. **exec('impact', ['-batch ./run_impact/identify.cmd'])**, este comando executa o iMPACT em modo -batch e passa o arquivo **identify.cmd,** que contém a sequência de comandos para identificar os FPGAs conectados na máquina, a sequência de comandos executado utilizando o **identify.cmd** são;

- **setmode -bscan**; seleciona o modo de configuração para buscar dispositivos.
- **setCable -p auto**; configura a velocidade e a porta para modo automático.
- **identify**; identifica cada um dos dispositivos listados pelo *bscan*.
- **info**; imprime as informações dos dispositivos, este é retornado em forma de *string* pelo método em execução.
- **quit**; encerra a execução do iMPACT. (Xilinx Inc, 2016)

O módulo de identificação faz o parse das informações retornados pelo iMPACT, criando uma lista de dispositivos numerados e com o nome do modelo do FPGA, sendo o número da lista o parâmetro **-p** do comando **setCable** que é utilizado posteriormente para selecionar o FPGA a ser programado. Os dispositivos reconhecidos, são listados na tela de configurações para serem associados a uma porta serial manualmente, pois a ferramenta iMPACT não detecta as portas seriais, estas são detectadas pela biblioteca *"SerialPort"*.

Figura 20 – Bibliotecas desenvolvidas para o projeto.



Fonte: (Elaborado pelo Autor, 2016).

O terceiro módulo destacado na Figura 20, é o *"serial_uart"*, este módulo possui cinco métodos, sendo um de configuração, um de escrita na porta serial, um de leitura da porta e dois para tratamento de concorrência de acesso ao dispositivo FPGA, o módulo de leitura também utiliza a biblioteca *"SerialPort"*, que é responsável por abrir a conexão serial com a porta UART do kit de desenvolvimento.

Após a configuração de dispositivo e porta, como descrito no item 3.1.3, é chamado o método **serial_uart.setup_uart();** o qual faz uma verificação no banco de dados e carrega as configurações das portas que foram persistidas após a configuração feita no item 3.1.3, abrindo as portas seriais que foram associadas, para cada porta serial aberta é atribuído uma instância global da classe *"SerialPort"*, esta instância possui os métodos para escrita e leitura da porta e também a variável de bloqueio de acesso concorrente ao dispositivo FPGA.

O método *setup_uart()*, também é executado sempre que o servidor for inicializado, reabrindo as portas que estão nas configurações que foram armazenadas no banco de dados caso estas existam, o método de leitura **serial_uart.read_tx_buffer ();** consome a buffer de entrada de dados enviados pelo FPGA, esta *buffer* é preenchida por um *event emitter*.

> Grande parte da API núcleo do Node.js é construída em torno de uma arquitetura idiomática assíncrona orientada a eventos em que certos tipos de objetos (chamados *"emitters"* ou emissores) que periodicamente emitem eventos nomeados que fazem com que os objetos Funcionais ("*listeners*") sejam chamados. (Node.js, 2016).

O *listener* que recebe os dados da porta serial, chama uma função nomeada SerialPort**.on('data')** toda vez que a porta serial instanciada receber algum dado vindo do

FPGA, assim preenchendo uma buffer que é consumida pela rota da *view Lab_area* chamado por Ajax, preenchendo o campo **Entrada Tx**, visto na Figura 14 e descrito no item 3.1.6.

Do método de escrita do módulo **serial_uart.writer();** este faz a escrita na porta serial, ele pode receber como parâmetro, os tipos primários *string*, *number* ou objeto *array* dos mesmos tipos primários e também pode receber *Buffer*, que é uma classe que faz parte da API do Node.js qual permite ler e manipular dados binários, de acordo como a documentação:

> Antes da introdução do TypedArray no ECMAScript 2015 (ES6), a linguagem JavaScript não tinha nenhum mecanismo para ler ou manipular fluxos de dados binários. A classe Buffer foi introduzida como parte da API Node.js para tornar possível a interação com fluxos octetos no contexto de coisas como fluxos TCP e operações do sistema de arquivos. Agora que TypedArray foi adicionado no ES6, a classe Buffer implementa a Uint8Array API de uma forma que é mais otimizado e adequado para casos de uso do Node.js. (Node.js, 2016).

Todos os parâmetros recebidos pelo método **serial_uart.writer();**, são verificados caso não sejam do tipo buffer, são convertidos para *buffer* de bytes de 8 bits cada para serem enviados byte a byte, a conversão é feita utilizando a classe Buffer utilizando o parâmetro *"binary"*, e.g. **new Buffer('a', 'binary')**, que converte os caracteres em sequência de um byte por caractere.

Os métodos para tratamento de concorrência de acesso ao kit de desenvolvimento são os *set_busy()* e *is_busy()*, o método *set_busy()* recebe como parâmetro o id e um valor booleano, para ser atribuído a variável da instância global referente ao dispositivo que é marcado como ocupado ou livre de acordo com o valor booleano recebido, este método também atribui um objeto de tempo "Date()", para servir de referência de quanto tempo o dispositivo está marcado como ocupado, um evento de intervalo *setInterval* é utilizado para verificar a cada um minuto se o dispositivo ainda está em uso, caso não esteja em uso o dispositivo é liberado novamente, o método *is_busy()* recebe como parâmetro o id do dispositivo FPGA e retorna um valor booleano indicando se o dispositivo está ocupado ou livre para uso.

Os métodos de tratamento de concorrência são utilizados pela rota *bit_upload*, para identificar se o dispositivo FPGA está sendo utilizado por outro usuário no momento ou se está livre e também atribuí como ocupado o dispositivo quando é enviado um arquivo .bit para o servidor e este programado com sucesso, também é utilizado pela rota *lab_area* que atualiza o tempo da variável de ocupação a cada segundo, indicando que o dispositivo FPGA continua em uso, quando este parar de ser atualizado o evento *setInterval* atribuí à instância do dispositivo FPGA como desocupado após passado um minuto.

3.2.5. Gerenciamento do envio do arquivo bit, rota “Bit_upload”

Este item aborda como é feito o gerenciamento do arquivo bit enviado com sucesso pelo usuário.

O arquivo .bit, é o arquivo de programação de um FPGA, ele é enviado para o dispositivo pela ferramenta iMPACT, descritos no item 1.3.2, este arquivo é gerenciado no lado do servidor diretamente no controlador de roda da *view bit_upload*, item 3.1.5.

Quando o usuário faz upload do arquivo, na *view bit_upload*, a requisição recebida é analisada pelo módulo *multer*, responsável pela manipulação de requisições *multipart/form-data*, utilizado no upload de arquivos.

O arquivo recebido é enviado para um subdiretório nomeado de acordo com o id do usuário que fez o upload para o diretório *bit_upload* e uma cópia do mesmo que é utilizado para programar o FPGA é enviada para o diretório *run_impact* e renomeada para um padrão, **main** + “id do kit de desenvolvimento” + **.bit**, e.g. se o id for 1, o nome do arquivo fica: **main1.bit**.

Um arquivo de comando .cmd com o id do kit é escrito no diretório *run_impact*, nomeado com o mesmo padrão **set** + “id do kit de desenvolvimento” + **.cmd**, e.g. se id for 1, então o arquivo será **set1.cmd**, este arquivo é executado pelo *batch* do iMPACT, ele contém os seguintes comandos, “supondo-se que o id do kit seja **1**, como nos exemplos anteriores”:

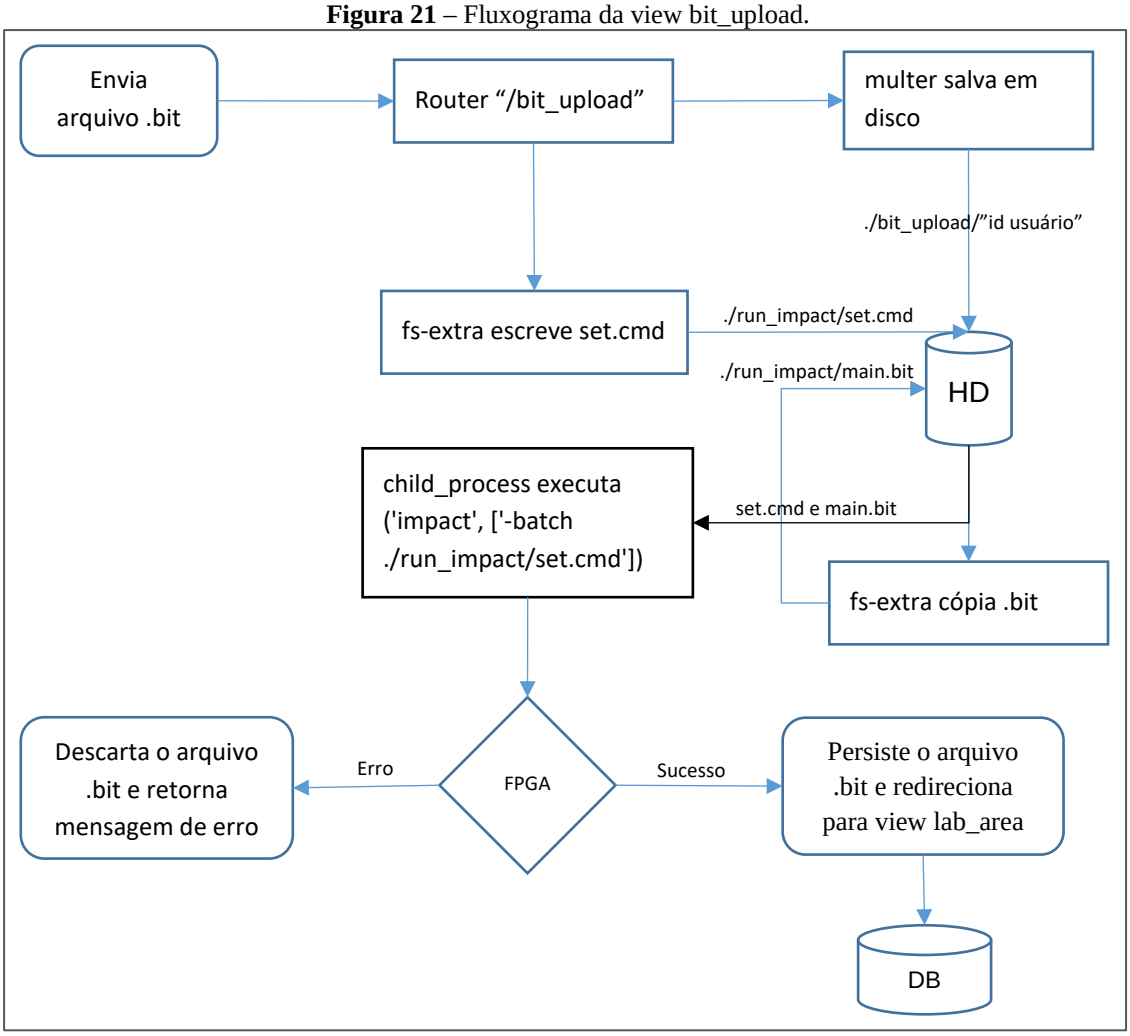
- *setmode -bscan*; seleciona o modo de configuração para buscar dispositivos.
- *setCable -p auto*; configura a velocidade e a porta para modo automático.
- *identify*; identifica cada um dos dispositivos listados pelo *bscan*.
- *assignfile -p **1** -file ./run_impact/main**1**.bit*: atribui um arquivo .bit ao kit de id **1**.
- *program -p **1***:
- *quit*: encerra a execução do iMPACT. (Xilinx Inc, 2016)

Parte deste procedimento é descrito no item 2.3.4, onde foi utilizado apenas para identificar os kits de desenvolvimento, nesta parte agora é atribuído o arquivo .bit e executado o comando *program* para programar o FPGA do kit. Nota-se o id do dispositivo nos nomes dos arquivos a serem utilizados para programação e nas linhas de comando do .cmd, este padrão foi definido para organizar cada arquivo para o seu definido kit de desenvolvimento.

No caso de existir um arquivo de mesmo nome este é sobrescrito sem problemas, pois só há um conjunto de arquivos para cada sessão de uso do kit de desenvolvimento, sendo os arquivos anteriores descartados/sobrescritos com exceção dos arquivos .bit persistido no

diretório *bit_upload*, estes são mantidos se a programação ocorrer com sucesso e podem ser baixados de volta ou utilizados para repetir a simulação no kit de desenvolvimento.

Figura 21 – Fluxograma da view bit_upload.



Fonte: (Elaborado pelo Autor, 2016).

O fluxograma (Figura 21) do processo descrito neste item até a execução da programação no FPGA, que é feito pelo iMPACT que é instanciado como processo filho pelo módulo nativo do Node.js *child_process*, também descrito no item 3.2.4, utilizado no processo de identificação dos kits conectados, para a programação é executado o seguinte comando, *exec('impact', ['-batch ./run_impact/set.cmd'],...* sendo o *exec* uma instância do método *execFile()* do *child_process*, este chama o iMPACT em modo *batch* "linha de comando", passando o arquivo de comandos *set.cmd* que executa os procedimentos de programação do dispositivo.

É importante notar que o iMPACT está sendo chamado diretamente pelo nome do seu executável 'impact' no exemplo dado, para chamar um arquivo executável utilizando uma

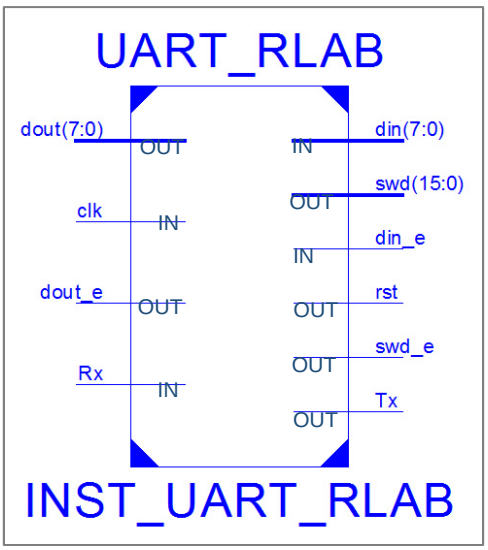
chamada direta pelo seu nome, este deve ter o caminho de seu executável definido na variável de ambiente PATH do sistema operacional.

3.3. Conexão com o kit de desenvolvimento FPGA

Este item aborda o módulo VHDL, desenvolvido para permitir a comunicação do FPGA do kit de desenvolvimento e o servidor utilizando a comunicação serial através da porta USB.

O módulo responsável por esta comunicação é o UART, que converte os dados entre as comunicações seriais e paralelas, descrito no item 1.6. Na Figura 22 é ilustrado o componente principal encapsulado em seu *top level*, com as suas entradas e saídas, que é utilizado neste projeto.

Figura 22 – Componente UART_RLAB.



Fonte: (Elaborado pelo Autor, 2016).

Segue o detalhamento do componente UART ilustrado na Figura 22, o componente possui 10 pinos, 4 entradas e 6 saídas, sendo 3 deles barramentos;

- **clk:** entrada do sinal de *clock* do componente.
- **rst:** implementado como saída, este é ativado quando recebido determinado dado pelo receptor, envia uma borda de subida para o *top level* que implementar este componente, indicando estado de reset
- **Rx:** entrada do receptor, ligado a uma máquina de estados, este fica em espera aguardando um *pull down* bit de início ou *star bit* para começar a receber a cadeia de bits que formam um byte e enviar para do seu componente.

- **Tx:** saída que transmite os dados recebidos no barramento **din** do componente UART_RLAB, quando habilitado o **din_e**, (pino *enable*, que habilita a leitura da entrada **din**), o dado presente na entrada é transmitido serialmente bit a bit pela saída **Tx**.
- **din_e:** entrada de *enable*, é habilitado com um *pull up*, ou borda de subida para indicar que há dados prontos para serem lidos na entrada **din**.
- **din:** barramento de entrada do componente UART_RLAB de largura 8, é habilitado pelo **din_e** e seus dados são enviados serialmente pelo transmissor **Tx**.
- **dout:** barramento de saída do UART_RLAB de largura 8, quando é recebido o comando para transmitir os dados recebidos para o *top level*, estes são enviados para a saída **dout** e quando estiverem prontos para serem lidos é elevado o sinal da saída **dout_e**, indicando para o *top level* que há dados para serem lidos na saída do componente UART_VLAB.
- **dout_e:** saída de *enable*, é enviado um *pull up*, indicando ao *top level* que instanciou o componente UART_VLAB que há dados disponíveis para serem lidos no barramento **dout**.
- **swd:** barramento de saída de largura 16, é um barramento implementado para representar *switches* ou chaves, cada bit representando uma chave com o estado ligado ou desligado, para controlar estados no *top level*, para cada alteração feita no **swd** é enviado uma borda de saída no **swd_e**, indicando que houve uma alteração no barramento.
- **swd_e:** saída *enable*, para ser utilizada como um *trigger*, gatilho de aviso, que envia uma borda de subida indicando que houve alteração no barramento **swd**.

O componente UART_RLAB é composto por outras duas instâncias de componentes separadas que são instanciados no UART_RLAB, estes componentes são o receptor e o transmissor. A solução utilizada para o transmissor e para o receptor neste projeto é de autoria de Russell Merrick, previamente autorizado pelo autor e também disponibilizado em seu website <www.nandland.com>.

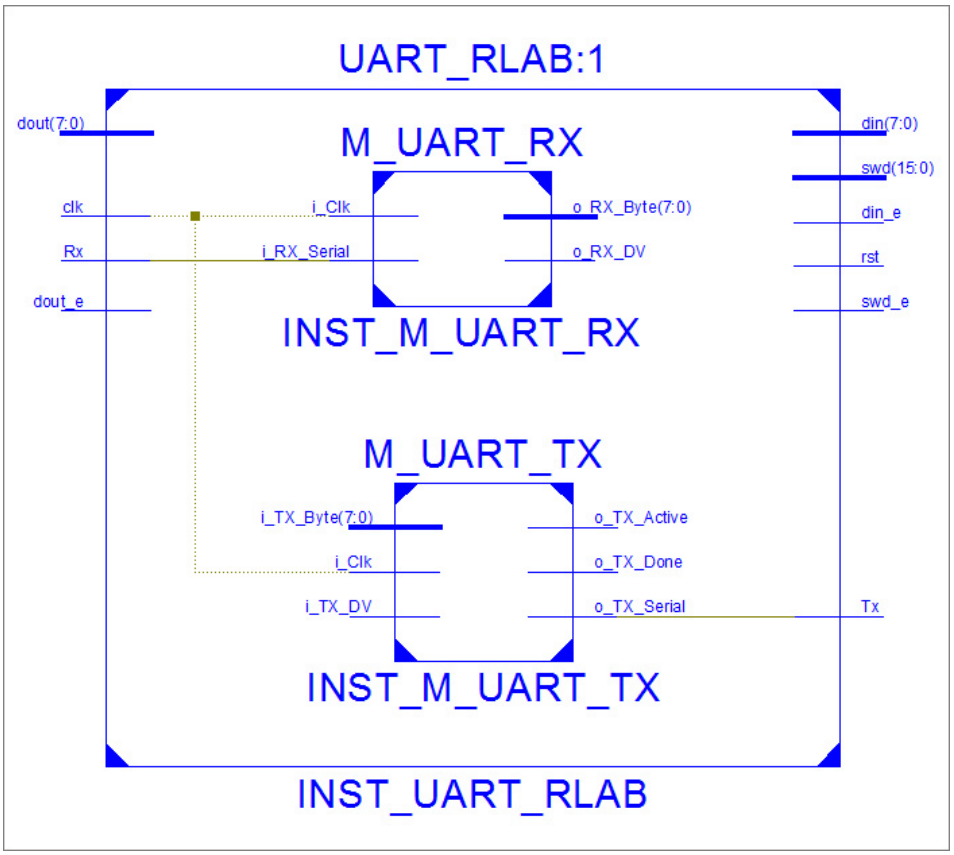
A Figura 23, ilustra o componente UART_RLAB com suas duas instâncias responsáveis pela transmissão e recepção de dados, o *top level* mostrado na figura é o encapsulamento do componente UART_RLAB mostrado na Figura 22, com os mesmos pinos descritos, internamente há os módulos M_UART_RX e M_UART_TX, ambos são ligados diretamente

no sinal de *clock* **clk** e cada um em seu pino correspondente **Rx** e **Tx** do UART_RLAB, os módulos M_UART_RX e M_UART_TX referentes a Figura 23, são detalhados a seguir:

- **M_UART_RX:** Componente que recebe os dados serialmente e os disponibiliza em seu barramento de saída **o_RX_Byte** em paralelo, possui 4 pinos, 2 entradas e 2 saídas sem uma um barramento de tamanho 8, suas entradas e saídas são representadas pela letra **i** input e **o** output no nome de seu pino.
- **i_Clk:** entrada do sinal de clock, ligado diretamente ao sinal de clock do componente UART_RLAB.
- **i_Rx_Serial:** entrada do receptor, ligado diretamente no **Rx** do UART_RLAB, descrito na descrição de seu componente.
- **o_RX_Byte:** barramento de saída do componente M_UART_RX, quando é finalizado a recepção da cadeia de bit que formam um byte, a saída **o_RX_DV** é elevada indicando que o dado recebido, está pronto para ser lido em paralelo na sua saída.
- **o_RX_DV:** saída *enable* ou *trigger*, esta envia um *pull up* ou borda de subida, indicando que o dado recebido está, pronto para ser lido na saída **o_RX_Byte**.
- **M_UART_TX:** Componente que faz a transmissão serial dos dados recebidos em paralelo no seu barramento de entrada **i_TX_Byte**, possui 6 pinos, 3 entradas e 3 saídas, sendo uma entrada um barramento de tamanho 8, suas entradas e saídas são representadas pela letra **i** input e **o** output no nome de seu pino.
- **i_Clk:** entrada do sinal de clock, ligado diretamente ao sinal de clock do componente UART_RLAB.
- **o_TX_Serial:** saída do transmissor, por onde é transmitido serialmente os dados recebidos em paralelo, é ligado diretamente ao **Tx** do componente UART_RLAB.
- **i_TX_Byte:** barramento de entrada do componente M_UART_TX, recebe a entrada a ser transmitida, quando o *enable*, **i_TX_DV** é elevado, indicado que os dados na sua entrada estão prontos para serem transmitidos.
- **i_TX_DV:** entrada *enable*, quando este é elevado *pull up*, indica que os dados no barramento de entrada estão prontos para serem transmitidos serialmente.
- **o_TX_Active:** saída, assim que inicia-se uma transmissão esta saída é elevada para indicar que o componente está ocupando em transmissão.
- **o_TX_Done:** saída que envia uma borda de subida indicando o fim da transmissão de um dado, também indicando o momento em que o componente acaba de desocupar e no

ciclo de *clock* seguinte está pronto para uma nova transmissão, esta saída é elevado apenas no termino da transmissão, por um ciclo de *clock*, serve apenas como um gatilho que indica o fim da transmissão e não servindo para identificar se o componente está ocupado ou não, para este é utilizado o **o_TX_Active**.

Figura 23 – Visão dos componentes Rx e Tx instanciados no UART_RLAB.



Fonte: (Elaborado pelo Autor, 2016).

Dos pinos descritos, três devem ser ligados diretamente a sua saída e entrada específicas, estes pinos são (clk, Rx e Tx), o clk, pino do sinal do oscilador de *clock*, deve ser ligado diretamente no oscilador de *clock* do kit de desenvolvimento, assim como a entrada e saída Rx e Tx, devem ser ligadas ao pino referente a porta USB UART do kit de desenvolvimento.

Para ligar estes pinos aos seus correspondentes do kit de desenvolvimento, utiliza se o UCF (*User Constraints File*), que é um arquivo de texto que contém as especificações e as relações dos pinos do projeto que são ligados aos pinos da placa, e.g. do pino de *clock*: (XILINX INC., Constraints Guide, 2011).

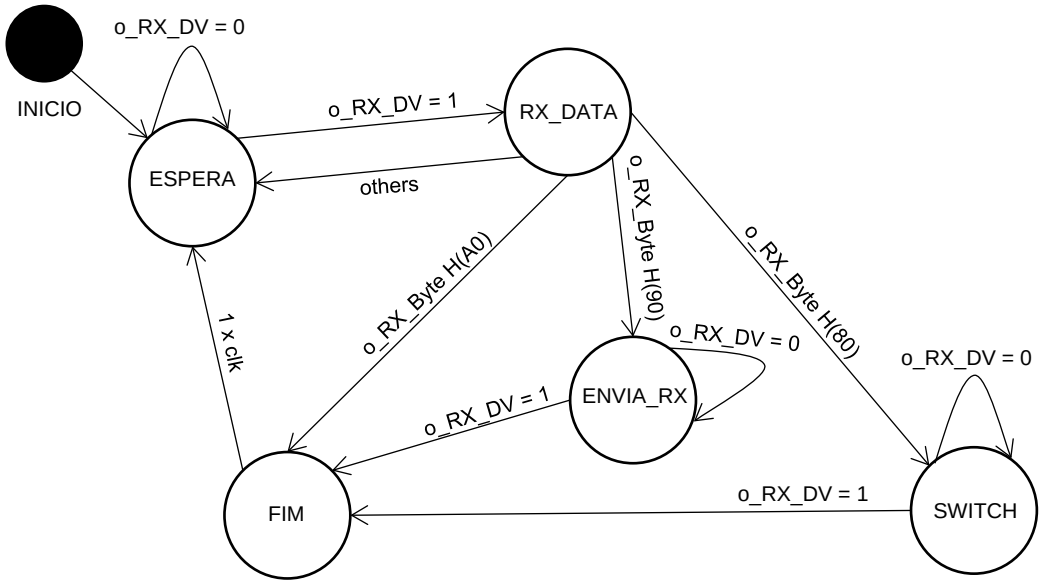
NET "clk" LOC = "V10" | IOSTANDARD = "LVCMOS33";

Esta linha indica o que pino “clk” do projeto é ligado à linha de *clock* do oscilador do kit de desenvolvimento. A referência das linhas e componentes do kit de desenvolvimento FPGA difere de para o outro, sendo necessário consultar a documentação do kit para o nome correto de cada linha e componente de cada kit, o arquivo UCF do modelo presente no Laboratório Remoto é distribuído junto do módulo UART_RLAB.

3.3.1. Protocolo de comunicação

Foi criado um protocolo para comunicação entre o servidor e o FPGA, no *top level* UART_RLAB, é implementado uma máquina de estados que reconhece o protocolo criado para esta comunicação, para reconhecimento dos comandos referentes aos *switches*, *reset* e envio dos dados recebidos para o barramento **dout**, este protocolo é reconhecido pela máquina de estados conforme ilustrado na Figura 24.

Figura 24 – Máquina de estados do UART_RLAB que trata os dados de entrada.



Fonte: (Elaborado pelo Autor, 2016).

Esta máquina de estados é ativada sempre que houver dados na saída do M_UART_RX, assim que o o_RX_DV for elevado indicando que o dado recebido está pronto para ser lido, passa-se para o próximo estado RX_DATA, que verifica se o dado recebido no barramento

o_RX_Byte, equivale a algum comando, caso não retorna para o estado de ESPERA, a Tabela 2, lista os 3 comandos básicos, segue detalhes dos 3 estados.

- **SWITCH**: após o byte recebido indicando este estado, aguarda-se o próximo byte para ativar ou desativar um *switch* ou chave, quando for elevado o_RX_DV, a primeira sequência de bits recebido o_RX_Byte(7:4) e.g. "**0000**xxxx", é utilizada para indicar o estado de ligado ou desligado do *switch*, sendo "**0000**xxxx" para ligado e "**1100**xxxx" para desligado, qualquer sequência no intervalo (7:4) diferente desses dois é ignorado, nenhum switch é alterado e o estado SWITCH passa para o estado FIM, depois voltando para o estado de ESPERA, a segunda sequência de bits o_RX_Byte(3:0) e.g. "xxxx**0000**", indica o número do switch que é ativado ou desativado, dependendo do valor da primeira entrada, sendo os *switches* representados por 4 bits, então tem se 2^4 = 16, dezesseis *switches*.
- **ENVIA_RX**: este estado apenas aguarda o próximo dado no barramento de saída o_RX_Byte, assim que o o_RX_DV for elevado indicando que a saída do componente M_UART_RX está pronto para ser lido, o UART_RLAB envia o valor da saída o_RX_Byte para a sua saída **dout**, assim disponibilizando o dado de entrada para o *top level* que implementar o módulo UART_RLAB.
- **RESET**: esta não é um estado, assim que recebido o comando referente ao reset, todos os registradores do UART_RLAB são postos em *pull down* ou zerados, a saída do módulo é elevado para indicar ao *top level* que instância o UART_RLAB o comando de reset e o estado RX_DATA passa para o estado FIM aguardando 1 ciclo para retornar.

Tabela 2 – Estados do módulo UART_RLAB

o_RX_Byte	**Estados**	**o_RX_Byte(7:4)**	**o_RX_Byte(3:0)**
80_H "10000000"	SWITCH	**0**x_H "**0000**xxxx"	Ligado x**0**$_H$ "xxxx**0000**"
		Cx_H "**1100**xxxx"	Desligado x**0**$_H$ "xxxx**0000**"
90_H "10010000"	ENVIA_RX	o_RX_Byte	
$A0_H$ "10100000"	RESET		

Fonte: (Elaborado pelo Autor, 2016).

3.4. Conexão com câmera IP

A solução adotada para transmitir a visualização em vídeo do kit de desenvolvimento, foi a utilização de câmeras IP, como fora descrito no item 1.7.6, as câmeras IPs são dispositivos de *hardware* para transmissão de vídeo através de uma rede IP.

Este dispositivo é independente do servidor e dos demais componentes do projeto, sua implementação e conexão com o servidor é bastante simples, o endereço de rede, host ou IP da câmera é associado a um kit de desenvolvimento na tela de configurações, descrito no item 3.1.3, quando acessado a tela lab_area, que contém a área de exibição do vídeo, na rota da tela lab_area é importado o pacote *request*, descrito no item 3.2.1 de dependência, este pacote é responsável por fazer requisições http e implementa a interface Stream, permitindo transmitir qualquer requisição. (GITHUB, Request, 2016).

A interface Stream é um módulo que provê uma API de fluxo de dados, com quatro tipos fundamentais, *Readable* que transmite um fluxo de dados de onde podem ser lidos, *Writable* que transmite um fluxo de dados para onde pode ser escrito, *Duplex* pode transmitir um fluxo de dados onde pode ser lido e escrito e por último, *Transform* é como o *Duplex* e que pode transformar ou modificar os dados na leitura ou escrita.

Assim, o módulo *request* faz a requisição http do endereço da câmera IP, e o retransmite através do servidor, utilizado o método pipe(), e.g. **request("endereço da câmera IP").pipe(response),** o método pipe(), pertence a interface Stream e permite "ligar" um item de fluxo de leitura (*Readable*) a um (*Writable*) item de fluxo de escrita, fazendo com que ele mude automaticamente para o modo de fluxo (*streaming*) e envie todos os seus dados para o método *response*. O fluxo de dados é gerenciado automaticamente para que o fluxo de envio para o destino não seja superado pelo fluxo de leitura (NODE.JS, Stream, 2016).

Esta solução permite a retransmissão do vídeo da câmera através do servidor, assim em caso de a câmera IP pertencer apenas a rede local do servidor, está possa ser transmitida para o usuário lado cliente através da rota http do servidor do laboratório.

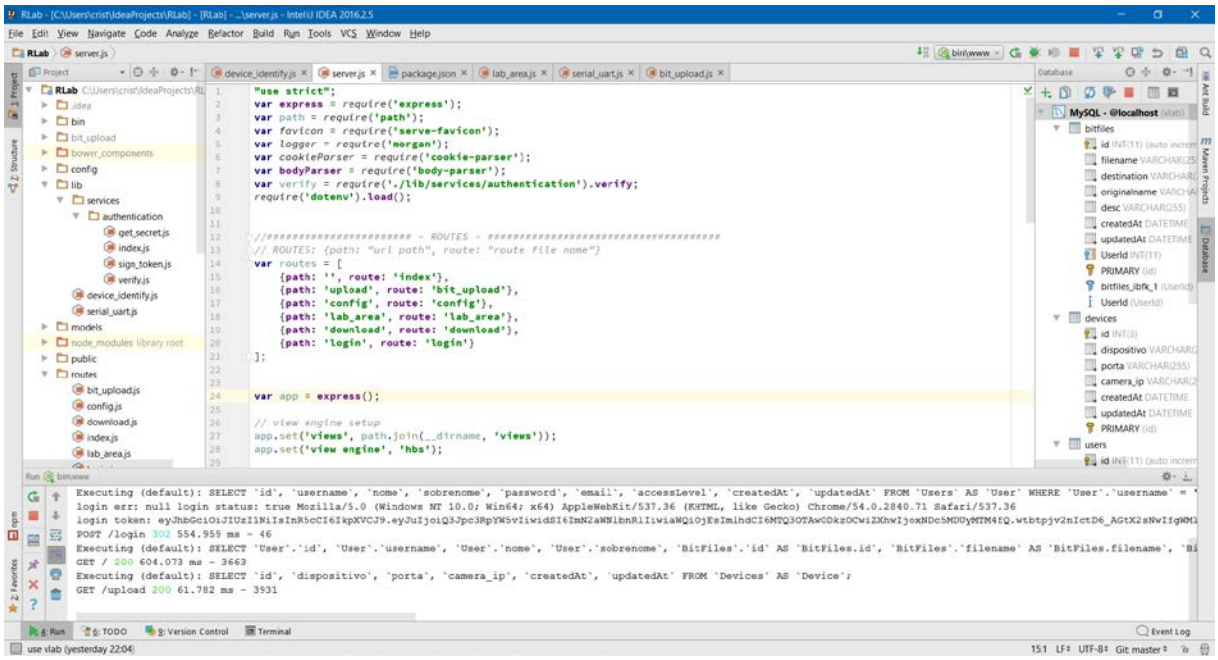
4. Implementação e implantação

Este capítulo aborda o processo de implementação e implantação do Laboratório Remoto, as tecnologias e ferramentas utilizados na sua implementação e como é feito a implantação do laboratório.

4.1. Lado cliente e servidor

Para a implementação dos lados cliente e servidor foi utilizado a IDE IntelliJ IDEA 2016.2.5 (Figura 25), que é um ambiente de desenvolvimento integrado, especialmente para o desenvolvimento em Java, porém possui suporte a tecnologias web como HTML5/CSS e JavaScript, suporte para desenvolvimento com Node.js e manuseamento do banco de dados (este apenas para fins de teste de desenvolvimento, não é uma ferramenta para administração de banco de dados).

Figura 25 – IDE de desenvolvimento IntelliJ IDEA 2016.2.5.



Fonte: (Elaborado pelo Autor, 2016).

Ao iniciar um novo projeto no IDE utilizado, a IDE conta com a categoria *Node.js and NPM* que possui o modelo *Node.js Express App*, neste modelo inicial de projeto é executado o

express-generator, descrito no item 3.2.2, gerando a estrutura base para iniciar a implementação, vale notar que é necessário ter instalado o Node.js antes de criar um projeto, se não o mesmo não é possível pois o *express-generator* é um módulo do Node.js que é executado pelo processo do Node.js.

O controle de dependências do projeto é feito pelos gerenciadores de dependências, NPM e o Bower, que foram descritos no item 3.2.1, o NPM faz parte da instalação do Node.js, já o Bower deve ser instalado o Bower é distribuído pelo próprio NPM, para sua instalação basta executar o comando ***npm install -g bower***, após sua instalação basta utilizar o comando ***bower init*** no diretório raiz do projeto, após um questionário básico como, nome do projeto, descrição e autor, este gera o *bower.json*, arquivo que contém as descrições do projeto e suas dependências, também descrito no item 3.2.1 e visto na Figura 16.

E por fim, este projeto utiliza o Git, um VCS, que é um sistema de controle de versão, utilizado no desenvolvimento de software para controlar versões do código fonte, histórico e documentação, quase todas as suas operações são locais, mas podendo ser remota, em um repositório de controle de versões em um servidor na rede local ou na nuvem. (GIT, 2016).

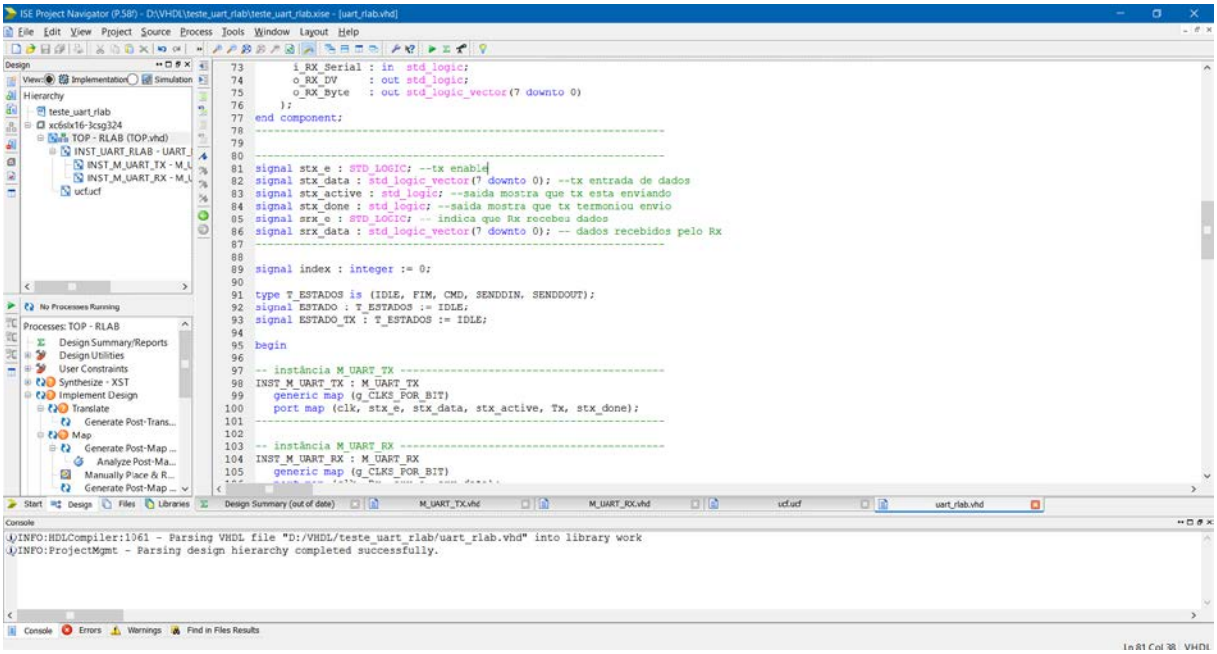
Este projeto utiliza publicação em nuvem no repositório online Github o Laboratório Remoto é *open-source* ou de código fonte aberto, seu código fonte pode ser encontrado publicado no repositório online do auto disponível no Github <https://github.com/CristSky/RLab>.

4.2. Módulo UART VHDL

O módulo UART_RLAB foi implementado utilizando a IDE ISE Design Suite 14.5 da Xilinx, Figura 26, esta IDE é responsável por analisar, sintetizar, mapear, gera as rotas e o arquivo de programação .bit, dos projetos para os FPGAs da Xilinx, seu fluxo de projeto está descrito no item 1.3.2.

A ferramenta iMPACT descrito em vários itens deste documento, em principal nos itens 3.2.4 e 3.2.5, é distribuída no pacote da IDE ISE Design Suite, é necessário ter esta ferramenta no servidor para Laboratório Remoto consiga programar o FPGA o iMPACT também está presente na instalação do Lab Tools, que é um pacote com apenas as ferramentas menores sem a IDE, assim não sendo necessário fazer uma instalação completa caso precise apenas do iMPACT.

Figura 26 – IDE desenvolvimento ISE Design Suite 14.5



Fonte: (Elaborado pelo Autor, 2016).

4.3. Implantação do Laboratório Remoto

Este item descreve o processo de implantação ou *deploy*, do Laboratório Remoto e faz um levantamento de compatibilidade dos *softwares* utilizados com os sistemas operacionais, baseado em suas documentações.

4.3.1. Compatibilidade.

Para a implantação do Laboratório Remoto, foi levantado a compatibilidade das ferramentas com os sistemas operacionais de acordo com suas documentações referentes as tecnologias utilizadas para o seu correto funcionamento.

Para as ferramentas da Xilinx pertencentes ao pacote ISE Design Suite, está descrito em documentos de seu suporte online, que sua última versão 14.7, suporta os seguintes sistemas operacionais, “Windows XP Pro/Windows 7 Pro 32/64-bit/Windows Server 2008; RHEL WS 5 32/64-bit; RHEL WS 6 32/64-bit; SUSE 32/64-bit”, (XILINX INC., Operating System (OS)

Support on Xilinx ISE Design Tools, 2016). Vale notar que esta ferramenta pode ser compatível com outros sistemas operacionais não listados em sua documentação oficial, mediante testes de compatibilidade, pois o Laboratório Remoto foi desenvolvido no sistema operacional Windows 10, onde a ferramenta iMPACT apresentou total compatibilidade, este item não visa testar a compatibilidade das ferramentas utilizadas.

Para o Node.js, este projeto foi implementado utilizando a versão 6.9.1 LTS do Node.js, de acordo com a documentação desta versão, esta oferece suporte a diversos sistemas operacionais, porém como a ferramenta ISE Design Suite oferece suporte apenas a sistemas Windows e Linux, é considerado a compatibilidade apenas desses sistemas, os sistemas suportados são, “Windows 7 32/64 bits, ou superior; RHEL 5 32/64 bits, ou superior; CentOS 5 32/64 bits, ou superior; SUSE 11.4 32/64 bits, ou superior”, (Node.js, Installing Node.js e Github, Node.js v6 ChangeLog, 20160).

Dos módulos adicionais do Node.js, estes não relatam em suas documentações compatibilidade com sistemas operacionais, apenas com as versões do Node.js e a Câmera IP é um dispositivo de rede independente dos sistemas operacionais listados.

Após esta análise, tem se a lista de plataforma qual o Laboratório Remote pode ser instalado;

- Windows 7 32/64 bits: listado na documentação de ambos.
- Windows 10 64bits: Foi testado e desenvolvido nesta plataforma e apresentou total compatibilidade.
- RHEL 5 e 6 32/64 bits: listado na documentação de ambos.
- SUSE 11.4+ 32/64 bits: listado na documentação de ambos, porém não discrimina versão na documentação do ISE Design Suite.

Esta lista de sistemas compatíveis é formada de acordo com as informações de compatibilidade descritas nas documentações dos softwares necessários para o correto funcionamento do Laboratório Remoto, não incluindo o banco de dados. O Laboratório Remoto pode ser compatível com outras plataformas mediante a realização de teste. Este item não visa testar a compatibilidade das ferramentas utilizadas com outros sistemas operacionais.

4.3.2. Processo de implantação

Este item não visa descrever o processo de instalação de cada software necessário, apenas da implantação do laboratório.

Os *softwares* necessários que devem estar instalados são;

- Xilinx ISE Design Suite ou Lab Tools: (ambos contem a ferramenta iMPACT) é necessário adicionar o caminho da ferramenta iMPACT ao PATH do sistema operacional para possibilitar o laboratório executar a ferramenta.
- Node.js v6 LTS com NPM.
- Bower: após instalado o Node,js com NPM basta executar o comando "*npm install -g bower*" para instalar o Bower.
- Git: (mesmo que opte por copiar os arquivos manualmente, este item é necessário, pois os gerenciadores de pacotes NPM e Bower o utilizam, para baixar dependências de repositórios Git.)
- Banco de dados: ver item 3.2.3, os bancos de dados compatíveis com o ORM.

Tendo todos os *softwares* necessários instalados, escolhe-se um diretório para implantação do Laboratório Remoto, pode-se fazer o download do código fonte do laboratório no repositório online <https://github.com/CristSky/RLab> e extrair no diretório escolhido ou utilizar o Git para clonar o código para o diretório com o seguinte comando no prompt do Windows ou terminal do Linux no diretório designado.

git clone https://github.com/CristSky/RLab

Após a extração ou clonagem do código fonte do laboratório, é necessário instalar as dependências do projeto, como citado no item 3.2.1, as dependências são gerenciadas por 2 gerenciadores de pacotes, o NPM e o Bower, para a instalação das dependências basta executar dois comandos no diretório raiz de implantação do Laboratório Remoto no prompt ou terminal, em qualquer ordem, porém o segundo após o termino do primeiro.

npm install

bower install

Após a execução dos dois comandos o projeto deve ter todas as dependências listados nos arquivos Figura 16, instaladas.

O próximo passo é a configuração do banco de dados, descrito no item 3.2.3 basta criar um *schema* ou *database*, com o nome que preferir e configurar o seu acesso no arquivo "*config.json*" encontrado no diretório "*config*" localizado no diretório raiz do laboratório, com as seguintes configurações;

- *username*: Nome de usuário do banco de dados.
- *password:* Senha do usuário do bando de dados.
- *database*: Nome do *schema* ou *database* criado para o Laboratório Remoto.
- *host*: Endereço do banco de dados, se caso for na mesma máquina e.g. "*localhost*" ou "127.0.0.1"
- *dialect*: Nome do banco de dados utilizado e.g. "*mysql, mariadb, postgres, sqlite, ...*"

A este ponto tendo tudo instalado, conecte o kit de desenvolvimento FPGA na máquina em que foi implantado o laboratório, bastando agora executar o servidor, do diretório raiz do Laboratório Remoto, execute o comando, no prompt ou terminal.

node /bin/www

O arquivo www é o script de inicialização do servidor, descrito no item 3.2.2, na sua primeira inicialização o ORM cria as tabelas do banco de dados e o Laboratório Remoto está disponível para acesso. No primeiro acesso é necessário configurar o dispositivo como descrito no item 3.1.3, associando o kit de desenvolvimento com uma porta COM e uma câmera IP.

5. Conclusão

Este capítulo relata os resultados atingidos, as dificuldades no desenvolvimento do Laboratório Remoto, suas limitações e oportunidades para trabalhos futuros.

5.1. Dos Resultados

O Laboratório Remoto desenvolvido, permite a um usuário, o envio de seu projeto sintetizado para ser programado remotamente em um FPGA de um kit de desenvolvimento real, permitindo ao usuário o controle, a visualização por vídeo, o envio e recebimento de dados para o FPGA programado no kit de desenvolvimento, através de uma plataforma web acessível de um navegador ou *browser* de Internet.

O projeto desenvolvido possui uma estrutura simples e de baixo custo sendo acessível e de fácil implantação, pois para a implantação mínima do laboratório é necessário apenas:

- *Hardwares*: um computador PC normal, um kit de desenvolvimento com 2 cabos USB para liga-lo ao PC e uma câmera IP, nenhum outro *hardware* adicional é necessário.
- *Softwares*: Caso utilize Windows, apenas o custo da licença do sistema operacional, pois do *software* da Xilinx, de acordo com a página de downloads do fabricante Xilinx, a versão *Lab Tools* que conta apenas com ferramentas, incluindo o iMPACT, não requer certificado ou chave de ativação (XILINX INC., Downloads, 2016), não tendo custo com licença deste *software* e os demais *softwares* utilizados são de licença gratuita.

A simplicidade de sua estrutura, o não requerimento de nenhum *hardware* específico para o seu funcionamento e sem custos adicionais com licenças de *softwares*, salvo a do sistema operacional caso seja Windows, o faz de baixo custo, acessível e simples de manter.

Portanto, os objetivos de desenvolver um laboratório de acesso remoto para auxiliar na programação de circuitos digitais, utilizando kits de desenvolvimento FPGAs através de uma plataforma web, visando um projeto de baixo custo e fácil manutenção, acessível e de fácil implantação, foi alcançado ao final deste trabalho.

5.2. Das dificuldades no desenvolvimento

Durante o processo de desenvolvimento das dificuldades encontradas, duas tiveram grande impacto para desenvolvimento, que são descritas a seguir.

A primeira dificuldade no desenvolvimento deste projeto, deu se no desenvolvimento do módulo UART para a comunicação entre o FPGA e o servidor, em sua fase inicial de desenvolvimento foi realizado testes te envio e recebimento de dados, os dados eram enviados a partir do servidor e retransmitidos de volta do FPGA, de acordo com a documentação do módulo *serialport* para o Node.js, os dados a serem enviados podem ser do tipo *string* ou *Buffer*, foi então definido que tudo seria convertido para *string* e assim enviado para o FPGA.

O protótipo inicial do módulo UART, recebia alguns dos dados enviados e outros não, a princípio foi realizado uma pesquisa em busca de reparar o erro da inconsistência no funcionamento do módulo UART, porém neste processo foi notado que todos os dados com caracteres comuns e sem acentos eram enviados e recebidos corretamente, apontando para um possível erro de codificação no lado servidor, onde foi realizado um estudo no código fonte do módulo *serialport*, neste foi constatado que todos os dados recebidos pelo módulo que faz o envio dos dados, se os dados não forem do tipo Buffer, estes então são convertidos para *Buffer* para serem transmitidos de forma binaria na porta serial.

A partir deste ponto foi realizado um estudo na classe *Buffer* do Node.js, onde foi constatado que todos os dados convertidos pela classe, por padrão são do tipo UTF-8, que tem tamanho de 1 byte para caracteres latinos sem acentos compatíveis com a tabela ASCII no intervalo de 00_H a $7F_H$, já os caracteres com acentos ou símbolos especiais são representados por 2 bytes no UTF-8, informação descrita na documentação RFC 3629 (IETF, Internet Standard, 2003).

Para a solução desta divergência de codificações, os dados que eram convertidos para *string* antes do envio passaram a ser convertidos diretamente para *Buffer*, passando para o método da classe o parâmetro *"binary"*, descrito no item 3.2.4, que é uma forma de fazer classe *Buffer* codificar os caracteres de uma *string* em um byte apenas.

A segunda dificuldade no desenvolvimento foi da implementação da visualização por vídeo, inicialmente foi proposto a utilização de uma Webcam, nas pesquisas iniciais, não foi encontrado nenhuma forma eficiente de acesso ao vídeo de uma Webcam utilizando o Node.js, apenas alguns projetos que não apresentaram resultados.

A solução encontrada para este problema foi a utilização de câmeras IPs, descrito no item 3.4, que além de solucionar este problema trouxe dois pontos positivo no quesito de fácil manutenção e simplicidade, não sendo necessário, ter preocupações com *drivers* do dispositivo, diferenças entre fabricantes ou compatibilidade com o sistema operacional, a câmera IP simplificou isso, sendo necessário apenas relacionar o endereço *host* ou IP em que a câmera transmite o vídeo nas configurações do laboratório item 3.1.3.

5.3. Das limitações do projeto desenvolvido

O módulo UART desenvolvido possui as seguintes limitações:

- Tamanho do dado de apenas 8 bits.
- Número máximo de chaves é 16.
- É limitado a apenas 3 instruções, devido ao atraso em seu desenvolvimento.
- Não possui uma memória de Buffer.

O pacote de *softwares* da Xilinx, ISE Design Suite, que contém o iMPACT, que é utilizado para o reconhecimento e programação dos FPGAs é limitado até a 6ª geração de FPGAs, a atual geração é a 7ª, que é suportado pelo pacote de *softwares* Vivado Design Suite, que também possui a ferramenta iMPACT, porém este não foi testado com o projeto atual (Xilinx Inc., 2016). Assim este projeto é limitado a 6ª, podendo ser compatível com a 7ª geração, mediante teste de validação.

Do laboratório, não há informações da capacidade máxima de kits FPGAs que podem ser utilizados, a ferramenta iMPACT, responsável pelo reconhecimento do kit conectado ao servidor, pode reconhecer diversos dispositivos conectados e atribuir a cada um deles, um id (Xilinx Inc., 2016), porém sua documentação não específica a quantidade máxima de dispositivos que podem ser reconhecidos. O laboratório desenvolvido foi testado com apenas um dispositivo conectado ao mesmo tempo, carecendo a informação de capacidade máxima de kits que podem ser usados.

5.4. Para trabalhos futuros

O projeto desenvolvido alcançou seu objetivo como um laboratório remoto para programação de circuitos digitais, da plataforma resultante, pode-se por exemplo, melhorar sua compatibilidade com a nova geração de FPGAs ou até mesmo a compatibilidade com dispositivos de outros fabricantes, adicionar novas funcionalidades à plataforma web ou ao módulo de comunicação, aumentando as suas instruções e até desenvolver um laboratório de educação a distância para desenvolvimento de circuitos digitais.

Referências

AXELSON, J. **Serial Port Complete:** COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems. Second Edition. ed. Madison: Lakeview Research LLC, v. I, 2007, p. 4-5.

BOOTSRAP, **JavaScript Overview**. Disponível em: <http://getbootstrap.com/javascript/>. Acesso em: 03 de novembro de 2016.

DIGILENT INC., **About Us**. Disponível em: <http://store.digilentinc.com/pages.php?pageid=26>. Acesso em: 13 de março de 2016.

EXPRESS, **Debugging Express.** Disponível em: <http://expressjs.com/en/guide/debugging.html>. Acesso em: 08 de novembro de 2016.

EXPRESS, **Express application generator.** Disponível em: < https://expressjs.com/en/starter/generator.html>. Acesso em: 06 de novembro de 2016.

FUNDING UNIVERSE, **Xilinx, Inc. History**, Vol. 16. St. James Press, (1997). Disponível em: <http://www.fundinguniverse.com/company-histories/xilinx-inc-history/>. Acesso em: 12 de março de 2016.

GIT. **Noções Básicas de Git**. Disponível em: <https://git-scm.com/book/pt-br/v1/Primeiros-passos-No%C3%A7%C3%B5es-B%C3%A1sicas-de-Git>. Acesso em: 13 de novembro de 2016.

GITHUB, NODE.JS., **Node.js v6 ChangeLog.** Disponível em: <https://github.com/nodejs/node/pull/5167>. Acesso em: 14 de novembro de 2016.

GITHUB, Request - Simplified HTTP client. Disponível em: <https://github.com/request/request#streaming>. Acesso em: 09 de novembro de 2016.

GOLDBERG, K., GENTNER, S., WIEGLEY, J. (1999) **The Mercury Project: A Feasibility Study for Internet Robots**, IEEE Robotics and Automation Magazine. Special Issue on Internet Robotics, dezembro de 1999.

GOOGLE DEVELOPERS, **Chrome V8: Google's high performance, open source, JavaScript engine.**, Disponível em: < https://developers.google.com/v8/ >. Acesso em: 02 de novembro de 2016.

HASHEMIAN, R. E RIDDLEY, J., **FPGA e-Lab, a Technique to Remote Access a Laboratory to Design and Test**, 2007 IEEE International Conference on Microelectronic Systems Education (MSE'07), San Diego, CA, p. 139-140, 2007.

HENRY, J. (2003), **7 Years Experience with Web Laboratories.** Int Conf Engineering Education, Valencia, Spain.

ICINSIGHTS, **Seven Top-20 1Q16 Semiconductor Suppliers** Show Double-Digit Declines. Disponível em: <http://www.icinsights.com/news/bulletins/Seven-Top20-1Q16-Semiconductor-Suppliers-Show-DoubleDigit-Declines-/>. Acesso em: 20 de novembro de 2016.

IEEE STANDARD, **VHDL Language Reference Manual**, in IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002), p.c1-626, 26 de janeiro de 2009.

IETF, INTERNET STANDARD. **UTF-8**, a transformation format of ISO 10646, 2003, Disponícel em: <https://tools.ietf.org/html/rfc3629>. Acesso em: 15 de novembro de 2016.

MAXFIELD, CLIVE (2004). **The Design Warrior's Guide to FPGAs**: Devices, Tools and Flows. Elsevier. p. 4. ISBN 978-0-7506-7604-5, 2004.

MEDEIROS, D. S., Predefinição:DiegoMedeiros-SST20707: **Tecnologia de FPGA**. Disponível em: <http://wiki.sj.ifsc.edu.br/wiki/index.php/Predefini%C3%A7%C3%A3o:DiegoMedeiros-SST20707#Tecnologia_de_FPGA>. Acesso em: 06 de novembro de 2016.

MICHAELIS, **Dicionário Brasileiro da Língua Portuguesa**, Disponível em: <http://michaelis.uol.com.br/busca?r=0&f=0&t=0&palavra=laboratorio> Acesso em: 01 de novembro de 2016.

MIDORIKAWA, EDSON T., **Uma Introdução às Linguagens de Descrição de Hardware**, EPUSP - PCS 2304 - Projeto Lógico Digital, (2001). Disponível em: <http://www.pcs.usp.br/~edson/intro-hdl.pdf>. Acesso em: 15 de junho de 2016.

MORGAN, F. E CAWLEY, S., **Enhancing learning of digital systems using a remote FPGA lab**, Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on, Montpellier, p. 1-8, 2011.

MORGAN, F. et al., **Remote FPGA Lab with Interactive Control and Visualisation Interface**, 2011 21st International Conference on Field Programmable Logic and Applications, Chania, p. 496-499, 2011.

MORGAN, F., CAWLEY, S., KANE, M., COFFEY, A. E CALLALY, F., **Remote FPGA Lab applications, interactive timing diagrams and assessment**, Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET, Limerick, p. 221-226, 2014.

MORRIS, KEVIN, **Xilinx vs Intel**: The New Showdown in Programmable Logic, (2016). Disponível em: <http://www.eejournal.com/archives/articles/20160920-xilinxvsintel/>. Acesso em: 20 de novembro de 2016.

MORRIS, KEVIN, **Xilinx vs. Altera**: Calling the Action in the Greatest Semiconductor Rivalry, (2014). Disponível em: <http://www.eejournal.com/archives/articles/20140225-rivalry/>. Acesso em: 10 de março de 2016.

MOZILLA DEVELOPER NETWORK, **Sobre JavaScript**, **O que é JavaScript?**, Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript>. Acesso em: 03 de novembro de 2016.

MYSQL, **Download MySQL Community Server.** Disponível em: <https://dev.mysql.com/downloads/mysql/>. Acesso em: 05 de novembro de 2016.

NANDLAND, MERRICK, R., **UART**, Serial Port, RS-232 Interface. Disponível em: <https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html>. Acesso em: 12 de novembro de 2016.

NODE.JS FOUNDATION, **About Node.js®.** Disponível em: <https://nodejs.org/en/about/>. Acesso em: 02 de novembro de 2016.

NODE.JS, Documentation: **Buffer**. Disponível em: <https://nodejs.org/api/all.html#buffer_buffer>. Acesso em: 08 de novembro de 2016.

NODE.JS, Documentation: **Child Process.** Disponível em: <https://nodejs.org/api/child_process.html>. Acesso em: 07 de novembro de 2016.

NODE.JS, Documentation: **Events.** Disponível em: <https://nodejs.org/api/events.html>. Acesso em: 08 de novembro de 2016.

NODE.JS, Documentation: **Stream**. Disponível em: <https://nodejs.org/api/stream.html#stream_readable_pipe_destination_options>. Acesso em: 10 de novembro de 2016.

NODE.JS., **Installing Node.js.** Disponivel em: <https://nodejs.org/en/download/package-manager/#debian-and-ubuntu-based-linux-distributions>. Acesso em: 14 de novembro de 2016.

NODEJITSU, **What is the file `package.json`?** Disponível em: <https://docs.nodejitsu.com/articles/getting-started/npm/what-is-the-file-package-json/>. Acesso em: 06 de novembro de 2016.

NPM, **What is npm?** Disponível em: <https://docs.npmjs.com/getting-started/what-is-npm#what-is-npm>. Acesso em: 06 de novembro de 2016.

PASTOR, J. S., GONZALEZ, I., LOPEZ, J., GOMEZ-ARRIBAS, F. E MARTINEZ, J., **A remote laboratory for debugging FPGA-based microprocessor prototypes** Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on, p. 86-90, 2004.

PEREIRA, FÁBIO DACÊNCIO, et al. **Projeto, Desempenho e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs)**, Brasil: BLESS Gráfica e Editora Ltda., 2003. 241 p.

SEQUELIZE, **Sequelize | The Node.js / io.js ORM for PostgreSQL, MySQL, SQLite and MSSQL.** Disponível em: <http://docs.sequelizejs.com/en/latest/>. Acesso em: 06 de novembro de 2016.

SITEHOSTING, **Câmera IP – Tudo Sobre Câmera IP**, Disponível em: <http://www.sitehosting.com.br/camera-ip/>. Acesso em: 03 de novembro de 2016.

STRONGLOOP, INC E IBM., **Framework web rápido, flexível e minimalista para Node.js**, Disponível em: <http://expressjs.com/pt-br/>. Acesso em: 02 de novembro de 2016.

TAYLOR, K. e TREVELYAN, J. P. (1995) **Australia's telerobot on the web.** 26th Symposium on Industrial Robotics, Singapura, p 39-44, outubro de1995.

TREVELYAN, J., **Lessons Learned from 10 Years Experience with Remote Laboratories**, International Conference on Engineering Education and Research ‘‘Progress Through Partnership”, p 687-697, 2004.

VICILOGIC, **The Vicilogic Development Team**. Disponível em: <http://www.vicilogic.com/about-us/>. Acesso em: 20 de novembro de 2016.

XILINX INC, **iMPACT: Alphabetical Listing of Batch Mode Commands.** Disponível em: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/pim_n_alphabetical_batchmode_commands.htm>. Acesso em: 07 de novembro de 2016.

XILINX INC., **All Programmable FPGAs and 3D ICs**. Disponível em: <http://www.xilinx.com/products/silicon-devices/fpga.html>. Acesso em: 08 de abril de 2016.

XILINX INC., **Constraints Guide.** UG625 (v. 13.3), 19 de outubro de 2011, 34 p.

XILINX INC., **Downloads**. Disponível em: <https://www.xilinx.com/support/download.html>. Acesso em: 15 de novembro de 2016.

XILINX INC., **FPGA Design Flow Overview.** Disponível em: <http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm>. Acesso em: 21 de abril de 2016.

XILINX INC., **iMPACT Overview.** Disponível em: <http://www.xilinx.com/itp/xilinx10/isehelp/pim_c_overview.htm>. Acesso em: 21 de abril de 2016.

XILINX INC., **iMPACT User Guide.** Disponível em: <http://file.wiener-d.com/documentation/NIMbox-NEMbox-FPGA/Programming_Guides/impact_user_guide.pdf>. Acesso em: 21 de abril de 2016.

XILINX INC., **Implementation Overview for FPGAs.** Disponível em: <http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_implement_fpga_design.htm>. Acesso em: 21 de abril de 2016.

XILINX INC., **Operating System (OS) Support on Xilinx ISE Design Tools**. Disponível em: <https://www.xilinx.com/support/answers/18419.html>. Acesso em: 14 de novembro de 2016.