UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Marcos Azevedo dos Santos

Atualização remota em sistemas embarcados baseados em FPGAs

Blumenau
2024

Marcos Azevedo dos Santos

Atualização remota em sistemas embarcados baseados em FPGAs

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.
Orientador: Prof. Marcos Vinicius Matsuo, Dr.

Blumenau
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Santos, Marcos Azevedo dos
Atualização remota em sistemas embarcados baseados em FPGAs / Marcos Azevedo dos Santos ; orientador, Marcos Vinicius Matsuo, 2024.
55 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2. FPGA. 3. Sistemas Embarcados. 4. Atualização Remota. 5. Bootloader. I. Matsuo, Marcos Vinicius. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Marcos Azevedo dos Santos

Atualização remota em sistemas embarcados baseados em FPGAs

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Engenheiro de Controle e Automação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 08 de Julho de 2024.

Banca Examinadora:

Documento assinado digitalmente
MARCOS VINICIUS MATSUO
Data: 11/07/2024 16:33:38-0300
CPF: ***.580.029-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Marcos Vinicius Matsuo, Dr.
Instituição Universidade Federal de Santa Catarina

Documento assinado digitalmente
Carlos Roberto Moratelli
Data: 11/07/2024 16:39:37-0300
CPF: ***.999.369-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Carlos Roberto Moratelli, Dr.
Instituição Universidade Federal de Santa Catarina

Documento assinado digitalmente
Guilherme Brasil Pintarelli
Data: 11/07/2024 16:41:21-0300
CPF: ***.523.129-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Guilherme Brasil Pintarelli, Dr.
Instituição Universidade Federal de Santa Catarina

Aos meus pais, as minhas filhas, à minha companheira e
todos àqueles que contribuíram para a minha formação.

AGRADECIMENTOS

Gostaria de agradecer a todos que contribuíram para a minha formação como engenheiro: Em primeiro lugar, a Deus que me colocou neste caminho, que me abençoou e me possibilitou estar aqui, aos meus amados pais que me deram força e, com muito esforço, me deram o suporte necessário para persistir nesta caminhada; às minhas filhas, fontes de luz, que me iluminam e me motivam, cada dia mais a perseverar por um futuro melhor; à minha companheira Alice, que, com paciência, incentivo e compreensão, me acompanha em todos os desafios na trajetória da vida; à todos os meus professores que contribuíram para o engrandecimento e esclarecimento das habilidades e conhecimentos necessários para o exercício da profissão, e a todos os colegas com os quais compartilhei as dores da caminhada e as glórias do sucesso. Gostaria de agradecer ainda à empresa que abriu as portas e disponibilizou os recursos necessários para a implementação do projeto. E, por fim, agradeço ao professor Marcos Matsuo, honorável mestre, pela orientação e paciência durante o processo.

“A imaginação é mais importante que o conhecimento. O conhecimento é limitado, enquanto a imaginação abraça o mundo inteiro, estimulando o progresso, e dando origem à evolução.” (Einstein, 1931)

RESUMO

No contexto da indústria 4.0, no qual se tem a aplicação de tecnologias avançadas envolvendo automação, inteligência artificial e análise de dados, cada vez mais, as indústrias têm demandado por soluções que aumentem o desempenho de seus processos produtivos, a fim de se destacar em um mercado mais competitivo. Visando ampliar a lucratividade e melhorar a eficiência dos processos, as empresas têm adotado tecnologias avançadas como FPGAs em seus produtos. Essas tecnologias oferecem maior flexibilidade de projeto, alto poder computacional e reduzem o tempo de lançamento dos produtos no mercado. No entanto, é imprescindível uma infraestrutura integrada, que possibilite a atualização remota para corrigir falhas ou otimizar os sistemas computacionais destes produtos, visto que fornecer suporte de atualização *in loco* se torna impraticável para produtos vendidos em larga escala. Assim, o desenvolvimento de uma arquitetura que viabilize a atualização de sistemas baseados em FPGAs é importante tanto para manter os produtos sempre atualizados com hardwares/softwares mais eficientes e robustos, quanto para reduzir custos associados ao envio de técnicos apenas para este fim. Considerando que os sistemas com FPGAs são normalmente programados utilizando um gravador com acesso físico à interface de comunicação JTAG, o presente trabalho de conclusão de curso destina-se ao projeto, desenvolvimento e implementação de uma solução capaz de oferecer uma infraestrutura que viabilize a atualização remota de sistemas eletrônicos que utilizam FPGAs através de uma interface de comunicação personalizada, com suporte ao protocolo TCP/IP. Para o desenvolvimento deste projeto, foram escolhidos FPGA e os softwares de desenvolvimento da empresa Altera. Além do projeto da arquitetura, também foi construído um protótipo da aplicação, permitindo a demonstração do funcionamento do sistema e sua conexão com um servidor remoto, para demonstrar o funcionamento da atualização remota dos módulos eletrônicos contendo FPGAs. Durante os testes de bancada realizados, foi obtido um tempo de gravação similar aos métodos convencionais, que não oferecem o benefício de uma gravação simutânea e remota.

Palavras-chave: PLD; Sistemas Computacionais; CI/CD (Integração Contínua/Entegra Contínua); Entraga Contínua; Atualização Remota; Bootloader; Integração de Sistemas.

ABSTRACT

In the context of Industry 4.0, where advanced technologies involving automation, artificial intelligence, and data analysis are applied, industries are increasingly demanding solutions that enhance the performance of their production processes in order to stand out in a more competitive market. Aiming to increase profitability and improve process efficiency, companies have been adopting advanced technologies such as FPGAs in their products. These technologies offer greater design flexibility, high computational power, and reduce product's time-to-market. However, an integrated infrastructure is essential to enable remote updates to fix bugs or optimize the computational systems of these products, as providing on-site update support becomes impractical for products sold on a large scale. Thus, developing an architecture that enables the remote updating of FPGA-based systems is important both for keeping products up-to-date with more efficient and robust hardware/software, and to reducing costs associated with sending technicians solely for this purpose. Considering that FPGA systems are typically programmed using a programmer with physical access to the JTAG communication interface, this thesis focuses on the design, development, and implementation of a solution capable of providing an infrastructure that enables the remote updating of electronic systems using FPGAs through a customized communication interface with TCP/IP protocol support. For the development of this project, Altera's FPGA and development software were chosen. In addition to the architecture design, a prototype of the application was also built, allowing the demonstration of the system's functionality and its connection to a remote server to showcase the remote updating of electronic modules containing FPGAs. During bench tests, a programming time similar to conventional methods was achieved, which do not offer the benefit of simultaneous and remote programming.

Keywords: PLD; Computational Systems; CI/CD; Continuous Integration; Continuous Delivery; Remote Update; Bootloader; Systems Integration.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ALM	*Adaptive Logic Module*
AS	*Active Serial*
ASIC	*Application Specific Integrated Circuits*
ASMI	*Active Serial Memory Interface*
BSP	*Board Support Package*
CLB	*Configuration Logic Block*
CMOS	*Complementary Metal Oxide Semiconductor*
CPLD	*Complex Programmable Logic Device*
CRC	*Cyclic Redundancy Check*
DUT	*Device Under Test*
EDA	*Electronic Design Automation*
EPCS	*Enhanced Programmable Configuration device*
FPGA	*Field Programmable Gate Array*
HAL	*Hardware Abstraction Layer*
HDL	*Hardware Description Language*
JTAG	*Joint Test Action Group*
LAB	*Logic Array Block*
LUT	*Look-Up Table*
LVDS	*Low Voltage Differential Signaling*
M10K	*Memory 10 Kbytes*
PAL	*Programable Array Logic*
PLA	*Programable Logic Array*
PLD	*Programmable Logic Device*
PS	*Passive Serial*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computing*
RTL	*Register Transfer Level*
SoPC	*System on a Programmable Chip*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UART	*Universal Asynchronous Receiver/Transmitter*
VLSI	*Very Large Scale of Integrations*
VPS	*Virtual Private Server*

SUMÁRIO

1 INTRODUÇÃO

A revolução tecnológica da *Internet of Things* (IoT), impulsionada pelo avanço dos sistemas digitais programáveis e reconfiguráveis, tem fomentado a transformação tanto da sociedade de forma geral como dos meios de produção da indústria (OZTEMEL, 2018). Da indústria tradicional para a indústria 4.0, a IoT vem ganhando cada vez mais espaço, tornando máquinas e processos fabris mais automatizados, integrados e inteligentes, a fim de reduzir custos e torná-los mais eficientes para garantir a competitividade no mercado. Além disso, existe uma demanda crescente no mercado por soluções especializadas que tem incentivado a indústria a desenvolver novos produtos para explorar e atender as novas demandas. Neste contexto, diversas empresas têm optado por utilizar em seus produtos, tecnologias que oferecem rápida prototipagem, programabilidade e reuso de recursos, aprimorando o desempenho dos sistemas e minimizando principalmente o tempo de desenvolvimento, o que reduz significativamente o *time-to-market* do produto (GROUT, 2008).

Uma das tecnologias que vem sendo amplamente utilizada em aplicações que envolvem sistemas críticos e de tempo real é o FPGA, que possibilita não só a programação de software, mas a reconfiguração interna do hardware, proporcionando flexibilidade, alta velocidade e baixa latência em comparação com outras tecnologias programáveis existentes, como os microcontroladores. É esperado que o mercado de FPGAs cresça a uma média anual de 8,32% até 2028 (MORDORINTELLIGENCE, 2023), impulsionado principalmente pelas aplicações em telecomunicações, automotivos, militar e aeroespacial.

No entanto, devido à urgência com a qual os projetos precisam ser entregues e levando em consideração que os produtos estão em constante aprimoramento, se faz necessário muitas vezes atualizar o *firmware* destes dispositivos a fim corrigir *bugs*, implementar novas funcionalidades e até mesmo corrigir vulnerabilidades que podem comprometer a segurança da aplicação como um todo (TECHBEACON, 2017). Usualmente, a atualização de sistemas computacionais é focada no *software*. No entanto, em sistemas contendo FPGAs, também é possível reconfigurar os sistemas digitais internos dos FPGAs, o que confere uma maior flexibilidade no que diz respeito a atualização do sistema computacional.

Não obstante a atualização de sistemas embarcados baseados em FPGAs no âmbito industrial muitas vezes é realizada no formato tradicional, isto é, técnicos da empresa fabricante do sistema são enviados a campo para realizar a atualização manual do FPGA utilizando algum hardware de gravação (por exemplo, USB-Blaster). A entrega da atualização manual pode levar até meses, o que torna o processo de entrega lento, o que muitas vezes gera prejuízos, insatisfação ao cliente e implica em ciclos de *feedback* longos que podem fazer com que um bug leve meses para ser identificado e corrigido (LESZKO, 2022). Além disso, realizar essa atualização manual pode ser impraticável do ponto de vista logístico e financeiro, pois seria necessário enviar técnicos para atualizar todas as

máquinas espalhadas pelo mundo, a fim de manter o controle de versões. Isso abre margem para que se tenha produtos com *firmwares* desatualizados, podendo resultar no comprometimento das funcionalidades do sistema. Com o intuito de reduzir custos e viabilizar um procedimento operacional escalonável surgiu o conceito de entrega contínua que é amplamente aplicado em empresas de tecnologia da informação, mas ainda embrionário dentro da indústria.

Este trabalho foi desenvolvido tendo como base um sistema eletrônico de uma das máquinas de uma indústria do setor alimentício. O sistema eletrônico possui diversos módulos eletrônicos contendo FPGAs, os quais poderiam ser atualizados simultaneamente caso fosse utilizado o sistema de comunicação da máquina. No contexto de integração e entrega contínua, o presente trabalho consiste em projetar, desenvolver e implementar uma infraestrutura, que viabilize a atualização remota de sistemas embarcados contendo FPGAs. A infraestrutura envolve desde o servidor, que disponibiliza a imagem de atualização das FPGAs, até os sistemas embarcados com o FPGA, chamados neste texto de módulos de aplicação. Especificamente, o foco deste trabalho está na implementação de um *bootloader* (sintetizado no FPGA) que possibilite a gravação da imagem de atualização em uma memória *flash*, presente no módulo de aplicação. A memória *flash* é utilizada para carregar os dados binários, que contém a configuração de *hardware/software* do FPGA, toda vez que o módulo de aplicação é inicializado. Para dar suporte e facilitar a demonstração do *bootloader*, foi desenvolvido um sistema completo envolvendo o servidor remoto contendo a imagem de atualização e uma interface gráfica, a partir da qual um usuário pode realizar a atualização dos módulos de aplicação, configurados com a imagem de *bootloader*. Por fim, o objetivo final do projeto é realizar a transferência do binário de atualização desde o servidor que realiza o controle de versões até a memória *flash* de cada módulo eletrônico que contém uma FPGA.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Projetar, desenvolver e implementar um sistema de atualização remota que viabilize a entrega contínua de atualizações e realize a gravação simultânea em módulos contendo FPGA.

1.1.2 Objetivos Específicos

1. Projetar a arquitetura do sistema consistindo em servidor, interface gráfica e módulos de aplicação com FPGA;
2. Desenvolver o *hardware* e *software* embarcado nos módulos de aplicação;

3. Desenvolver a aplicação cliente e servidor, necessária para implementação da atualização remota de FPGAs;
4. Desenvolver um protótipo de bancada;
5. Avaliar o sistema desenvolvido, comparando-o com os métodos tradicionais.

1.1.3 Organização do trabalho

O presente trabalho está organizado como segue. O Capítulo 2 apresenta uma revisão dos principais conceitos teóricos e tecnologias relacionadas a este trabalho, como por exemplo, sistemas computacionais, PLDs e FPGAs. Em seguida, no Capítulo 3, são discutidos a metodologia e os recursos utilizados no desenvolvimento deste trabalho, bem como é apresentado um passo a passo do desenvolvimento do sistema. No Capítulo 4, são apresentados os resultados obtidos na simulação, compilação e implementação da aplicação, discutindo os principais pontos do projeto. Nesse capítulo, também, são apresentadas análises quantitativas e qualitativas, visando avaliar o desempenho do sistema desenvolvido. Por fim, no Capítulo 5, são sumarizados os principais resultados alcançados com o desenvolvimento deste trabalho, bem como apontados alguns pontos de melhorias futuras.

2 REVISÃO TEÓRICA

O surgimento do transistor, em 1947, foi um marco histórico no desenvolvimento das tecnologias e dispositivos eletrônicos que temos hoje. A invenção desse componente disruptivo proporcionou aos dispositivos eletrônicos uma maior durabilidade, menor volume, maior confiabilidade e custo reduzido em comparação com seus antecessores valvulados, o que foi fundamental para o desenvolvimento da eletrônica digital moderna e da computação como conhecemos.

Neste contexto, o surgimento da tecnologia CMOS que tornou possível a fabricação de portas lógicas por meio do processamento planar de silício, a constante miniaturização dos transistores prevista pela Lei de Moore, aliada à crescente demanda por dispositivos eletrônicos mais complexos, desempenharam um papel crucial na invenção dos circuitos integrados, que inicialmente agrupavam apenas algumas dezenas de transistores. Hoje estes circuitos, classificados como VLSI (*Very Large Scale of Integrations*), agrupam centenas de milhões de transistores em escalas nanométricas fabricados sob uma única pastilha de silício (RAJ, 2018).

Com o desenvolvimento da microeletrônica e sua aplicação nos mais diversos setores, a demanda por soluções especializadas cresceu na mesma proporção que estes chips se tornaram cada vez menores e mais complexos, iniciando a trajetória de desenvolvimento dos ASICs (*Application Specific Integrated Circuits*), chips projetados especificamente para uma determinada aplicação ao mesmo passo que surgiam também os primeiros microprocessadores, circuitos integrados programáveis capazes de executar instruções (CHEN, 2003). No entanto, conforme crescia o aumento da complexidade e tamanho dos circuitos *on-chip* e *off-chip*, crescia na mesma proporção a complexidade e as chances de falhas de desenvolvimento dos circuitos integrados, que até meados da década de 70 eram projetados ao nível de portas lógicas utilizando diagramas esquemáticos como metodologia de projeto para circuitos eletrônicos (WOODS *et al.*, 2017). Visando contornar estes desafios de *design*, emergiram nesta época, ferramentas e técnicas que melhoraram significativamente o fluxo de trabalho dos projetistas, mudando completamente o cenário de desenvolvimento de circuitos digitais e que são amplamente utilizados atualmente.

Os *softwares* de automação de projetos eletrônicos, ou EDAs (*Electronic Design Automation*), são ferramentas que auxiliam no projeto de circuitos eletrônicos, facilitando o *design* de chips. Essas ferramentas fornecem aos projetistas um arsenal poderoso para realizar análise, verificação e simulação de um sistema antes do chip ser fabricado, averiguando se o comportamento do *hardware* projetado funciona de acordo com as especificações desejadas (NELSON, 2014). Além disso, a utilização de HDLs (*Hardware Description Languages*), que são linguagens que descrevem o *hardware* de maneira textual, facilitaram o desenvolvimento e a compreensão de arquiteturas digitais complexas, tornando possível a implementação de uma metodologia de projeto que abstrai os detalhes da camada lógica

do circuito, proporcionando aos projetistas os meios para desenvolver arquiteturas ao nível de sistema. Essa metodologia de projeto ao nível de sistema, também conhecida como ao nível de transferência entre registradores ou RTL (*Register Transfer Level*), possibilita ao projetista focar o desenvolvimento no comportamento dos componentes, deixando que as ferramentas computacionais derivem os detalhes da implementação física do projeto através do processo de síntese.

Apesar das HDLs, ferramentas computacionais e metodologia de projeto RTL terem aumentado significativamente as chances de sucesso do *design* de sistemas digitais, ainda se constituia um desafio realizar atualizações em um sistema já implementado, tanto para corrigir falhas de projeto quanto visando o atendimento de novos requisitos de funcionamento (CHEN, 2003). Neste sentido, os microprocessadores davam ao projetista maior flexibilidade para realizar modificações visando corrigir o sistema já desenvolvido (principalmente no contexto de *software*), mesmo que as alterações em circuitos montados sobre uma PCI (Placa de Circuito Impresso) estivessem restritas as conexões de entradas e saídas do microprocessador. Para contornar as dificuldades no projeto de sistemas digitais (principalmente no que diz respeito ao *hardware*), foram desenvolvidos os dispositivos lógicos programáveis ou PLDs (*Programmable Logic Devices*). Estes dispositivos foram desenvolvidos inicialmente com o intuito de construir protótipos para testar o *design* do chip antes da sua fabricação em massa. Os primeiros PLDs lançados no mercado foram o PLA (*Programmable Logic Array*) e o PAL (*Programmable Array Logic*).

Para (ANDINA; ARNANZ; PEÑA, 2017) há uma distinção entre dispositivos programáveis e configuráveis. Apesar de ambos os conceitos se basearem na primitiva de possibilitar ao projetista programar ou reprogramar uma funcionalidade do sistema, o termo programável está mais relacionado ao *software* ou conjunto de instruções executadas sequencialmente por um microprocessador. Já o termo reconfigurável abrange a dispositivos de *hardware* (usualmente PLDs) que possuem flexibilidade na configuração dos seus recursos e interconexões. A reconfiguração permite adaptar funções de acordo com a operação, permitindo adaptar o *hardware* do dispositivo a diferentes condições de operação para responder a modificações nos requisitos de funcionalidade, alterações no ambiente, ou falhas que possam ocorrer, possibilitando estender a sua usabilidade.

Em diversas aplicações, selecionar a tecnologia mais adequada para atingir os requisitos de operação pode não ser uma tarefa trivial, tendo em vista a necessidade de se considerar diversos fatores interdependentes. Dentre esses fatores destacam-se, custo, desempenho, consumo de energia, disponibilidade de recursos (como memória de armazenamento, quantidades de entradas e saídas), confiabilidade e tratamento de falhas. Neste contexto, é usual a utilização de sistemas que tenham flexibilidade no que se refere a programação e/ou configuração dos sistemas de controle e processamento, visando a correção de *bugs* e/ou inserção de novas funcionalidades.

Dependendo da aplicação, há algumas vantagens em se utilizar dispositivos reconfi-

guráveis ao invés de dispositivos programáveis. Diferentemente dos microcontroladores que possuem o *hardware* fixo, os PLDs permitem maior flexibilidade, possibilitando ajustes dos recursos de *hardware* o que torna possível otimizações e/ou alterações de funcionalidades de acordo com os requisitos do sistema. Além disso, o processador insere uma limitação ao sistema no que tange o processamento digital de sinais, limitado a execução de apenas uma instrução por vez (considerando processadores de apenas 1 núcleo) (WOODS *et al.*, 2017), o que não é o caso em tecnologias cujo paralelismo é uma característica intrísceca, como os PLDs.

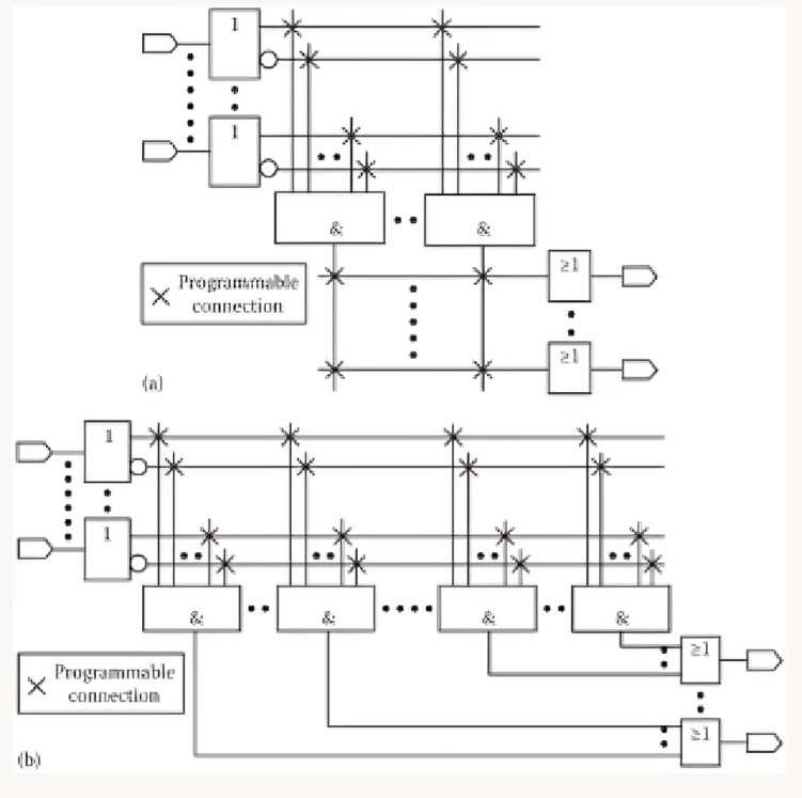
2.1 DESENVOLVIMENTO DOS PLDS

Os PLDs são circuitos integrados cujo circuito interno pode ser configurado após a sua fabricação, possuindo a capacidade de reconfigurar as conexões internas do chip. Os primeiros PLDs lançados no mercado foram o PLA e o PAL. Dispondo de uma arquitetura organizada de modo a implementar circuitos combinacionais a partir da soma de produtos de uma função booleana (SASS; SCHMIDT, 2010), estes dispositivos são compostos por matrizes de conexões configuráveis de portas AND e OR organizadas conforme a Figura 1, onde as entradas do dispositivo são conectadas às entradas das portas AND, e as saídas são conectadas a portas OR.

No passado a programação destes PLDs era realizada a partir da destruição das conexões indesejadas através da queima de fusíveis, o que limitava a programação desses PLDs a uma única vez. No entanto, essa abordagem logo foi substituída pela configuração das conexões através de memórias reprogramáveis de tecnologia CMOS. A diferença entre entre um PLA e um PAL é que o primeiro possui uma arquitetura do tipo *programmable-AND/fixed-OR*, enquanto o último possui ambas as matrizes de entrada e saída programáveis, constituindo uma estrutura do tipo *programmable-AND/programmable-OR*. Com o avanço da tecnologia, as fabricantes destes dispositivos acrescentaram à arquitetura *flip-flops* do tipo D para armazenar a saída do circuito combinacional configurado, o que possibilitou a criação de circuitos sequenciais e a implementação de máquinas de estados.

Para acompanhar a demanda por arquiteturas mais robustas, novas infraestruturas foram desenvolvidas na corrida por dispositivos mais rápidos, flexíveis, e com mais funcionalidades levando ao surgimento dos CPLDs e dos FPGAs. Os CPLDs consistem em aglomerados de mais de 2000 macrocélulas compostas por blocos de arquitetura PAL com conexões programáveis entre eles (PAUL HOROWITZ, 2015). No entanto, esta tecnologia se mostrou limitada ao longo do tempo principalmente no que se refere a flexibilidade de configuração dos recursos e rapidamente se tornou evidente que a arquitetura do CPLD não era adequada para implementação de sistemas digitais cada vez mais complexos (ANDINA; ARNANZ; PEÑA, 2017). Neste sentido, os FPGAs ganharam cada vez mais espaço devido à flexibilidade que esta plataforma oferece enquanto dispositivo lógico programável. Ao contrário dos CPLDs que possuem sua organização baseada em macrocélulas com

Figura 1 – Arquitetura dos PLA (a) e PAL (b).



Fonte: ANDINA, Juan (2017, p. 12)

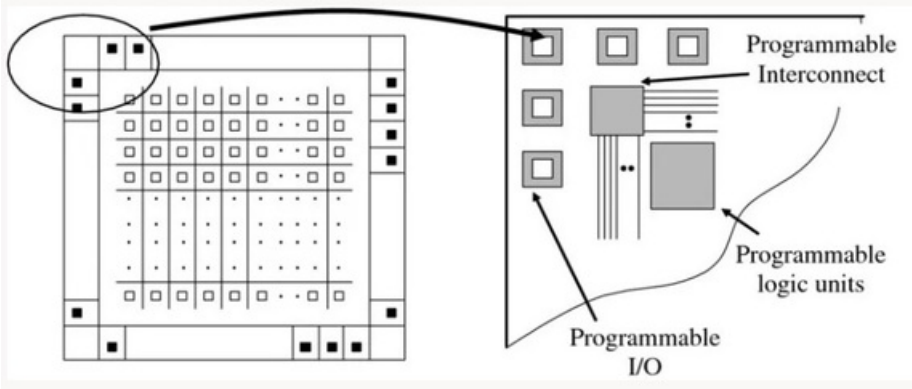
interconexões reprogramáveis, os FPGAs adotam uma abordagem baseada no roteamento das matrizes de interconexões entre células lógicas, unidades menores e mais versáteis que as macrocélulas, que fornecem elementos de armazenamento e geradores de função para a implementação da lógica do sistema. Atualmente estes chips são capazes de abrigar sistemas inteiros, deixando de ser utilizados apenas como uma simples "cola lógica" para se tornar uma tecnologia poderosa na implementação de arquiteturas customizadas com capacidade de realizar grande carga de processamento de sinais de maneira concorrente, sendo ideais para aplicações que exigem alto desempenho, como sistemas de tempo real, sistemas de visão computacional, equipamentos médicos e até aplicações no mercado financeiro.

2.2 ARQUITETURA DO FPGA

A infraestrutura básica de um FPGA, exibida na Figura 2, é composta por 3 blocos básicos, a saber:

- Blocos de I/O (IOB): compostos de pinos de entrada e saída localizados ao longo do perímetro do chip, realizam a interface do sistema *on-chip* com o sistema *off-chip*. Constituídos por *buffers* bidirecionais com saída em alta impedância,

Figura 2 – Composição básica de um FPGA.



Fonte: WOODS, Roger (2015, p. 97)

geralmente operam em níveis padronizados de 1,2V e 3,3V, podendo atuar como pinos isolados ou pares diferenciais;

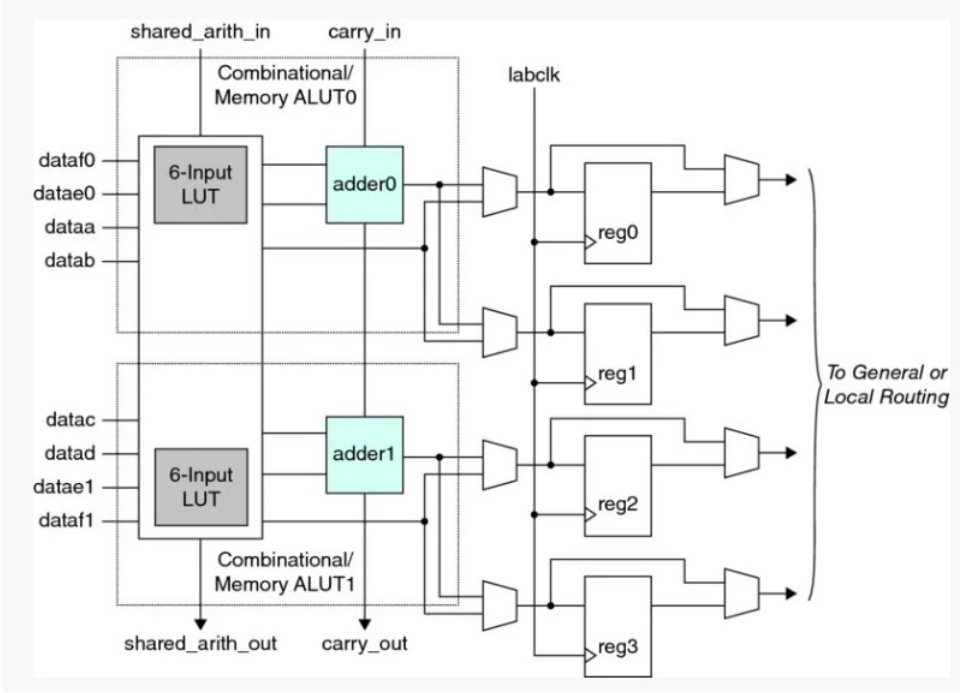
- Blocos de Lógica Configurável (CLB): compostos por agrupamentos de células lógicas programáveis, estas estruturas são organizadas em um arranjo bi-dimensional posicionados por todo dispositivo, e são utilizadas para implementar a lógica do sistema. Presentes ao longo de todo o chip, são posicionados junto a outros blocos para implementar funcionalidades dedicadas. Nos FPGAs da Altera esta estrutura recebe o nome de LAB (*Logic Array Block*);
- Blocos de interconexão configurável: compostos por conexões estruturadas em formato de grade com chaveamento programável que realiza o roteamento entre os elementos do sistema.

A estrutura mais fundamental de um FPGA (exibida na Figura 3) denominada de célula ou elemento lógico (localizada dentro do CLB), contém os seguintes componentes básicos necessários para a construção de circuitos combinacionais e sequenciais:

1. LUT (*Look-Up Table*);
2. Flip-flops, tipo D;
3. Registradores de deslocamento;
4. Somadores.

A quantidade de elementos lógicos utilizados em um *design* é frequentemente utilizada para avaliar a dimensão do circuito em projetos com FPGA. As células lógicas próximas são agrupadas em blocos lógicos para reduzir o tempo de propagação dos sinais (útil na síntese de somadores/subtratores, onde um sinal de *carry* precisa ser transmitido entre os circuitos básicos de soma/subtração). Os blocos lógicos, por sua vez, são conectados por uma rede de rotas, conhecidas como caixas de chaveamento ou *switching boxes*, utilizadas para rotear as entradas e saídas de um bloco lógico para a rede principal do chip.

Figura 3 – Composição de um Elemento Lógico.



Fonte: WOODS, Roger (2015, p. 99)

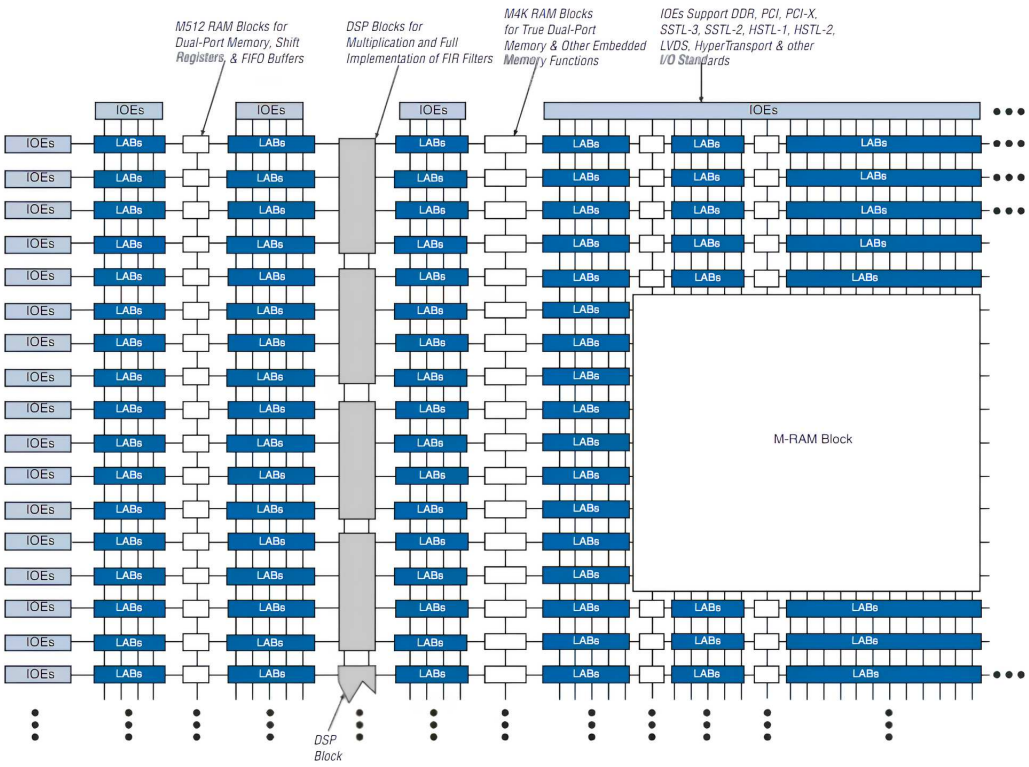
Adicionalmente, vale citar que os FPGAs modernos, conforme demonstra a arquitetura do Stratix II exibido na Figura 4, possuem alguns componentes ASICs embarcados que compõe blocos com funcionalidades especiais colocados por todo o FPGA junto aos CLBs, utilizados para fornecer ao projetista recursos especializados, removendo a necessidade de sintetizar tais blocos no FPGA o que poderia ocupar mais recursos e área, otimizando o chip (SASS; SCHMIDT, 2010). Dentre estes recursos estão memórias, FIFOs, blocos de DSP e processadores.

2.3 ECADS

Os ECADs (*Electronic Computer-Aided Design*) são *softwares* usados para o desenvolvimento e projeto de sistemas eletrônicos que oferece conjuntos de ferramentas que possibilitam ao projetista criar, modificar e analisar sistemas eletrônicos. Estas plataformas são essenciais em todo o fluxo de design com FPGA e geralmente são fornecidos pelo próprio fabricante (ANDINA; ARNANZ; PEÑA, 2017). O Quartus é um ECAD proprietário da Altera, que fornece uma gama de ferramentas para auxiliar no processo de desenvolvimento, compilação, e verificação de *designs* para FPGAs. A sua principal função é compilar o HDL para gerar o arquivo que contém as informações da configuração a partir do processo de síntese. O processo de compilação consiste em 3 etapas:

- Análise e Síntese;

Figura 4 – Arquitetura interna do FPGA Stratix II.



Fonte: Altera Stratix II Device Handbook (2007)

- Fitter (*place & route*);
- Análise de temporização.

O primeiro passo de compilação é verificar se a sintaxe do código escrito está coerente com a linguagem escolhida. Em seguida tem-se o processo de síntese lógica no qual o comportamento desejado descrito em uma HDL com código sintetizável é transformado em uma implementação do projeto ao nível hierárquico. Especificamente, o sintetizador RTL é responsável por realizar a conversão do código HDL, produzindo na saída deste processo uma representação sintetizada do circuito, no qual a estrutura básica pode ser identificada, mas não possui nenhuma tecnologia atrelada, ou seja, o resultado desta etapa da compilação não representa o circuito a ser implementado utilizando os recursos internos do chip. Em particular, esta tarefa é realizada na etapa de *Fitter*, onde a representação sintetizada na etapa anterior é implementada utilizando os recursos disponíveis no FPGA selecionado, mapeando os componentes na localização apropriada do chip, criando as interconexões necessárias utilizando os recursos de roteamento. A este processo também dá-se o nome de *place & route* (ANDINA; ARNANZ; PEÑA, 2017). Em seguida, é verificado se os requisitos de temporização (*timing*) são respeitados para a correta execução do circuito sintetizado.

Se o processo for bem-sucedido é gerado um arquivo chamado de *bitstream* que

pode ser utilizado para programar a FPGA diretamente, ou armazenado em uma memória não volátil para ser carregado na FPGA durante a sua inicialização. Especificamente, o *bitstream* é o arquivo binário que contém as informações da imagem que são escritos na memória de configuração do FPGA para que os elementos do dispositivo sejam arranjados adequadamente para operar conforme especificado. O *bitstream* pode ser comprimido visando minimizar a utilização de memória no dispositivo de armazenamento, tanto quanto reduzir o tempo de reconfiguração (ANDINA; ARNANZ; PEÑA, 2017).

Uma ferramenta importante do Quartus que auxilia no processo de desenvolvimento de sistemas com FPGA é o Platform Designer. Esta ferramenta permite ao projetista interconectar os componentes do sistema, através de uma representação por blocos em sua interface gráfica. Estes blocos, chamados de IPs (*Intelectual Property*), consistem em módulos com interfaces padronizadas que agilizam a integração e facilitam a reutilização de componentes, tornando o desenvolvimento mais eficiente. É possível utilizar tanto IPs fornecidos pela fabricante, quanto gerar o IP a partir de um HDL.

O ModelSim é outra ferramenta do Quartus bastante utilizada para realizar a verificação de designs HDL. Esta ferramenta fornece um ambiente de simulação onde é possível verificar o funcionamento do DUT (*device under test*). Durante a simulação é possível testar as funcionalidades do sistema desenvolvido verificando se estão de acordo com as especificações do projeto.

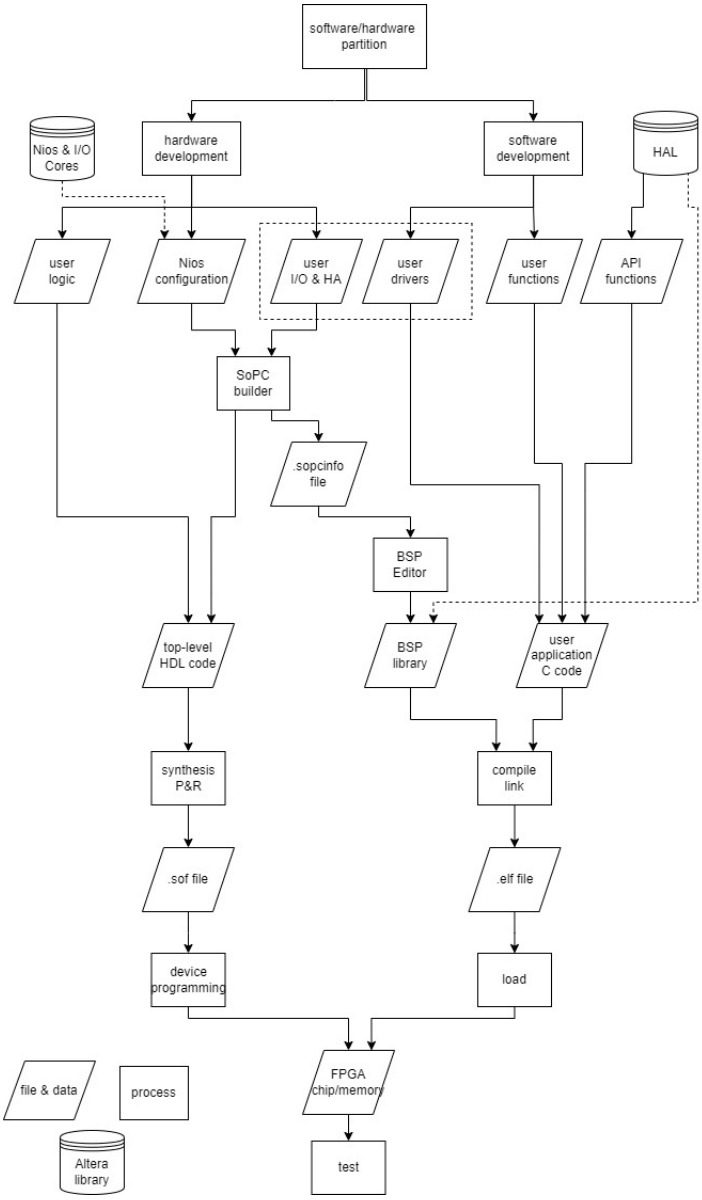
Para ampliar as possibilidades de um sistema utilizando FPGA, é muito comum utilizar uma arquitetura do tipo SoPC (*System on a programmble chip*), que consiste na utilização de um processador (*hard* ou *soft*) embarcado no FPGA. Especificamente, conforme demonstrado na Figura 5, a implementação de sistemas SoPC envolve tarefas relacionadas tanto ao desenvolvimento de *software* quanto de *hardware*. Os SoPCs podem ser criados na plataforma de desenvolvimento da Altera, a partir da inserção do IP de um *softcore* no Platform Designer, onde é possível configurar diversas características do *softcore*, como o tamanho da memória.

No projeto de SoPCs é necessário o suporte de ferramentas tanto para o projeto de *hardware*, para definir inteconexões mestre/escravo e os barramentos mapeados em memória, quanto para o projeto de *software*, que fornecem as bibliotecas e os *drivers* necessários para acesso aos hardwares utilizados.

O Fluxo de desenvolvimento básico para sistemas embarcados SoPC segundo (CHU, 2011) consiste em:

1. Divisão de tarefas entre *software* e *hardware*;
2. Desenvolvimento do *hardware*, incluindo aceleradores de *hardware*, periféricos de I/O e integração com o processador.
3. Desenvolvimento do *software*.
4. Testes da implementação de *hardware* e *software*.

Figura 5 – Fluxo de desenvolvimento de um sistema SoPC.



Fonte: CHU, Pong (2011, p. 6), adaptado pelo autor, 2024

O Nios2 é um *softcore* de arquitetura RISC de propósito geral desenvolvido pela Altera onde é possível embarcar *softwares* desenvolvidos em linguagem C. Este IP possui uma arquitetura 32-bits e utiliza o barramento *Avalon Memory-Mapped* para se comunicar com outros componentes. Para estabelecer a conexão com o hardware, a Altera possui uma ferramenta para gerar os arquivos BSP (*Board support package*), chamada de SoPC Builder. O BSP consiste em um conjunto de *drivers*, bibliotecas e informações que auxiliam no processo de implementação do software em uma determinada tecnologia ou dispositivo-alvo.

A HAL (*hardware abstraction layer*) faz parte dos arquivos de biblioteca gerados

no BSP, sendo responsável por abstrair a interface do *hardware*, representada por mapa de registradores e endereçamento de memória do dispositivo, que possibilita ao *software* acessar os recursos do componente.

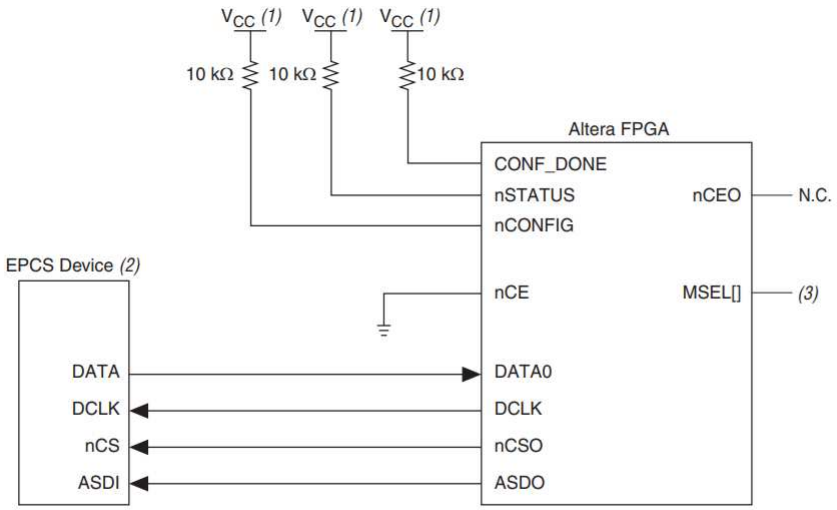
2.4 CIRCUITO DE CONFIGURAÇÃO DA FPGA

Nos FPGAs da Altera a configuração do dispositivo pode ser realizada em 3 modos (ALTERA, 2014):

1. Modo AS (*Active Serial*): a configuração é realizada através de dispositivos de configuração serial, que configura a FPGA em toda inicialização do sistema.
2. Modo PS (*Passive Serial*): a configuração realizada por um controlador externo, como um processador ou outro PLD.
3. Modo JTAG: configuração realizada utilizando a interface de testes JTAG (*Joint Test Action Group*).

O esquemático exibido na Figura 6 demonstra a ligação entre o dispositivo de configuração serial e a FPGA para o modo AS. Os pinos MSEL são responsáveis por selecionar o modo de configuração da FPGA.

Figura 6 – Esquema de configuração Active Serial.



Fonte: Altera Serial Configuration (EPCS) Devices (2014, p. 5)

Durante a inicialização, a FPGA aterra os pinos de *nStatus* para sinalizar que está ocupada e *CONF_DONE* para indicar que ainda não está configurada. A inicialização da FPGA só é concluída após o sinal *CONF_DONE* subir para *High*, o que só ocorre após a transferência de todo o *bitstream*. Se acontecer algum erro na configuração o sinal de

CONF_DONE se mantém em *Low*. Todos os pinos da FPGA são colocados em *tri-state* durante o processo de configuração. A gravação do dispositivo de configuração serial pode ser realizada através do acesso a interface *ASMI* (*Active Serial Memory Interface*), que pode ser acessada diretamente, ou através da FPGA.

2.5 RTL

A metodologia de *design* de arquitetura de *hardware* RTL é uma abordagem que, segundo (VAHID, 2011), consiste em capturar o comportamento desejado e então converter o comportamento em um circuito. Geralmente o comportamento desejado é traduzido para um controlador e um *datapath*, que modelado usando uma máquina de estados.

Para (VAHID, 2011), o projeto RTL é uma metodologia onde o circuito é descrito como uma série de registradores e funções de transferências que definem o fluxo de dados entre registradores. Os registradores são elementos de memória síncronos, e as funções de transferência são modelos resultantes de blocos com lógica combinacional.

A metodologia RTL segundo (JASINSKY, 2016) resume-se basicamente em 3 passos:

- Especificar os registradores do sistema: definidos para cada uma das variáveis de entrada e saída do sistema incluindo todas as variáveis intermediárias dos subprocessos do sistema.
- Definir as funções de transferência entre registradores: definição do comportamento desejado para todos os subsistemas, traduzidos posteriormente para os circuitos combinacionais.
- Criar a lógica de controle: modelado geralmente como uma máquina de estados que realiza a gerência do processamento interno das variáveis de saída do componente, de acordo com as variáveis controladas de entrada.

A abordagem RTL permite ao projetista especificar a arquitetura de *hardware* sem se preocupar com cada porta lógica, deixando o trabalho para ferramentas de síntese e otimização. O processo de converter automaticamente uma descrição RTL para uma implementação ao nível lógico em hardware é chamado de síntese. Desenvolver em nível RTL resulta em um *design* facilmente compreendido por ferramentas de síntese e minimiza os riscos potenciais de um circuito criado seguindo uma abordagem mal estruturada. Em resumo, a metodologia de projeto RTL fornece uma abordagem sistemática para transformar um algoritmo em um circuito digital. Quanto mais complexo os componentes maior deve ser o nível de abstração para que o projetista consiga lidar. Quanto maior o nível de abstração, menor é o tempo e o esforço de desenvolvimento de circuitos complexos.

3 PROPOSTA E IMPLEMENTAÇÃO DO SISTEMA DE ATUALIZAÇÃO REMOTA PARA FPGAS

3.1 DESCRIÇÃO DO CENÁRIO

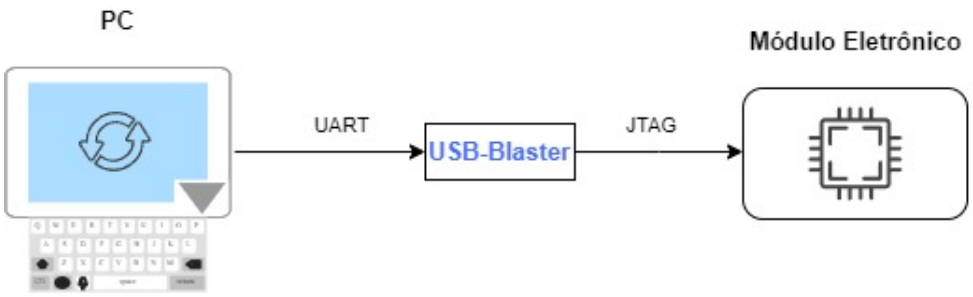
Levando em consideração as demandas da Indústria 4.0, o presente trabalho tem como objetivo desenvolver um modelo de infraestrutura que possibilite a atualização remota de FPGAs presentes em módulos eletrônicos de máquinas industriais.

O processo de atualização de sistemas que contenham FPGAs (Figura 7) consiste em transferir o *bitstream* de configuração para o módulo eletrônico por meio de um USB-Blaster. O USB-Blaster é um dispositivo de *hardware* que converte a interface UART (*Universal Asynchronous Receiver/Transmitter*) da conexão USB para a interface JTAG, que comunica com o FPGA. O processo de atualização exige que o USB-Blaster esteja acoplado fisicamente ao módulo eletrônico e a um computador que contenha o programa da fabricante que transfere o *bitstream*.

Na Figura 8 tem-se uma ilustração da parte interna do módulo eletrônico aonde temos o FPGA e a memória *flash* (dispositivo que armazena o *bitstream*). O *bitstream* é transferido pelo USB-Blaster para a FPGA através da interface JTAG, e a FPGA transfere para a *flash* através da interface ASMI (*Active Serial Memory Interface*). O *bitstream* é então gravado no dispositivo de armazenamento, carregado para a FPGA toda vez que o módulo eletrônico é inicializado.

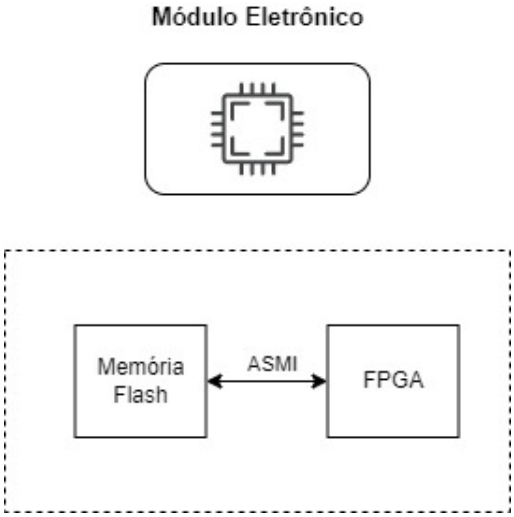
O sistema projetado neste trabalho teve como base uma máquina industrial que possui sistemas embarcados com FPGA. Informações mais detalhadas dos módulos eletrônicos e da máquina não puderam ser apresentados neste trabalho para fins de preservação de segredos industriais. Em todo caso, uma versão simplificada da arquitetura do sistema eletrônico é demonstrada na Figura 9. Especificamente, o sistema é composto por uma interface gráfica, permitindo ao usuário ter acesso aos recursos e funcionalidades da máquina. Esta unidade de comando se comunica com os demais módulos eletrônicos através do módulo de roteamento, responsável por gerenciar os pacotes de dados e realizar a re-

Figura 7 – Processo de atualização de FPGAs.



Fonte: O Autor

Figura 8 – Comunicação entre FPGA e a memória *flash*.



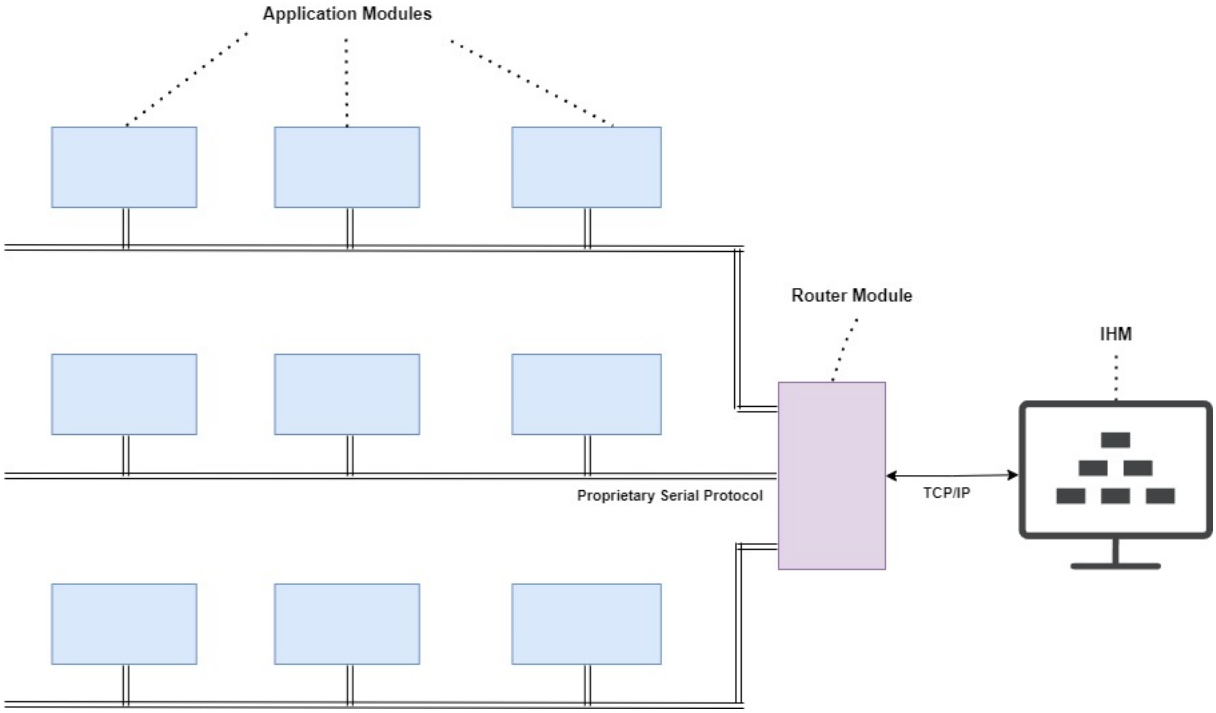
Fonte: O Autor

transmissão dos pacotes de dados para os módulos de aplicação responsáveis pela execução das funcionalidades. No contexto deste trabalho, o módulo de roteamento (em roxo, na Figura 9) pode ser entendido como uma caixa preta que recebe os pacotes através do protocolo TCP/IP e retransmite para os módulos de aplicação por meio de uma interface serial proprietária. Cada módulo de aplicação (em azul, na Figura 9) possui em síntese uma FPGA para embarcar a lógica de *hardware* e software de aplicação, conectores para interface de comunicação serial, uma memória *flash*, LEDs, além de outros periféricos. O circuito de configuração que seleciona o modo de programação dos módulos eletrônicos com FPGA, estão configurados internamente para funcionar no modo Ativo Serial (Figura 6), possibilitando a programação automática assim que o sistema é inicializado.

Do ponto de vista de comunicação, os sistemas são conectados por meio de uma rede com topologia mista, onde os módulos de aplicação são conectados em barramento, com os barramentos conectados em estrela ao módulo de roteamento. Cada barramento possui uma quantidade N de módulos de aplicação. A camada física de comunicação é implementada por meio de pares LVDS (*Low-Voltage Differential Signaling*), que conferem maior robustez a interferência de ruídos brancos. Toda lógica de *hardware* e *software* relacionada com o sistema de comunicação é compilada para um *bitstream* ou imagem de configuração embarcada na *flash* e carregada para a FPGA durante a inicialização.

O foco do projeto está relacionado ao desenvolvimento de um sistema que atue como um *bootloader* nos módulos de aplicação, tendo como sua principal função a capacidade de atualizar o módulo eletrônico a partir da transferência de um novo *bitstream* de aplicação para a memória *flash*. O *bootloader* trata-se de um sistema digital que é sintetizado no FPGA a partir do seu *bitstream*, sendo tal *bitstream* armazenado em uma seção protegida

Figura 9 – Arquitetura do sistema eletrônico da máquina.



Fonte: O Autor

da memória *flash*.

Em aplicações comuns envolvendo FPGA, é gravado apenas um *bitstream* na memória *flash*, mas no sistema desenvolvido, em particular, a memória *flash* foi dividida em duas metades, a primeira contendo o *bitstream* do *bootloader*, responsável por inicializar o sistema do módulo de aplicação e carregar outro *bitstream*, chamado de imagem de aplicação (localizada na segunda metade da memória *flash*), responsável por executar as funções de operação normal da máquina.

O *bootloader* inicializa os componentes de comunicação para comunicar com uma interface gráfica, que contém entre suas funcionalidades a capacidade de apagar os setores que armazenam o *bitstream* de aplicação, alternar entre os *bitstreams* de *bootloader* ou de aplicação, entrar em modo de transferência e formatar a *flash* completamente para apagar o próprio *bootloader* (este último necessário apenas caso o mesmo precise ser atualizado). A interface gráfica desenvolvida possui três funções principais: fornecer uma interface homem-máquina para controlar o módulo eletrônico, se comunicar com um servidor em busca de uma atualização e baixar e transferir o binário de configuração para o módulo eletrônico. O servidor remoto é responsável por armazenar o *bitstream* de atualização e fornecer, por meio de uma API, informações requisitadas pela interface gráfica a fim de verificar se há uma versão mais atualizada da aplicação instalada no módulo eletrônico.

Deste modo, este trabalho propõe uma arquitetura para um sistema de atualização

remota de FPGA, bem como a implementação de um protótipo de demonstração. Neste capítulo, o sistema desenvolvido é descrito sendo também abordados os recursos utilizados para a sua implementação.

3.2 RECURSOS UTILIZADOS

Para o desenvolvimento do projeto, foi utilizada uma das FPGAs da Altera, bem como o Quartus 18.1 (ECAD da Altera utilizado para desenvolver e compilar os projetos em FPGA) e o Platform Designer, uma ferramenta do Quartus que possibilita o acoplamento dos módulos IPs por meio de uma interface gráfica para a conexão dos componentes presentes no projeto. Adicionalmente, foi utilizado o *software* VScode para desenvolvimento dos códigos em linguagem C para o *softcore* Nios2, implementado nos FPGAs utilizados no projeto. Foi utilizado para realizar *logs* de informações do *firmware* embarcado no FPGA, o Nios2-terminal, uma ferramenta que permite ao FPGA ter acesso a um console no computador a partir da interface UART utilizada pelo USB-Blaster (gravador de FPGAs que converte a interface UART em interface serial JTAG). Além disso, os componentes de comunicação foram desenvolvidos através da metodologia RTL utilizando a linguagem de descrição de *hardware* VHDL. Por fim, o *software* de simulação Modelsim foi utilizado para verificar o funcionamento do sistema desenvolvido conforme os requisitos de projeto, os quais não são compartilhados neste trabalho a pedido da empresa onde o projeto foi desenvolvido.

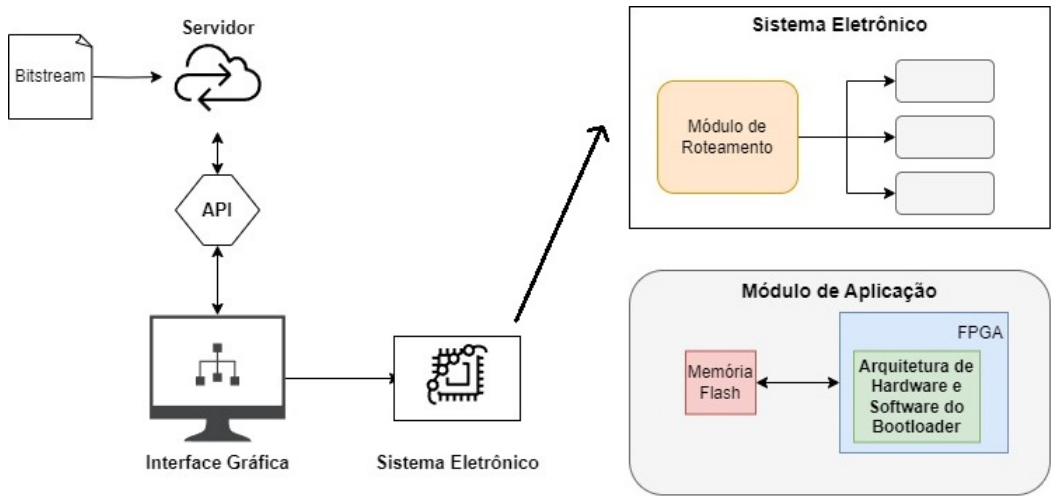
3.3 SISTEMA DESENVOLVIDO

O sistema desenvolvido neste trabalho (cujo diagrama é ilustrado na Figura 10) tem por objetivo ser capaz de atualizar os módulos de aplicação, por meio de um *pipeline* de entrega constituído pelo sistema eletrônico da máquina, uma interface gráfica, e o servidor responsável por disponibilizar o *bitstream* para a atualização.

A interface gráfica deve ficar responsável por realizar as checagens periodicamente para verificar a disponibilidade de uma atualização. Ao constatar uma atualização, o *download* é iniciado e uma mensagem de atualização disponível é emitida na interface gráfica, sinalizando para o operador que ele deve alternar o sistema eletrônico da máquina para o modo de *bootloader* para efetivar a atualização do sistema. O pacote de comando é enviado para o módulo de roteamento, sendo retransmitido para todos os módulos de aplicação simultaneamente. Os módulos de aplicação então realizam a troca do *bitstream* de aplicação para o *bitstream* de *bootloader*.

O *bootloader*, cujo *hardware* e *software* são capazes de realizar o acesso direto a memória *flash*, pode executar operações que permitem apagar e escrever dados no periférico, possibilitando assim realizar a atualização da imagem de aplicação do sistema.

Figura 10 – Arquitetura geral do projeto.



Fonte: O Autor

3.3.1 Interface Gráfica

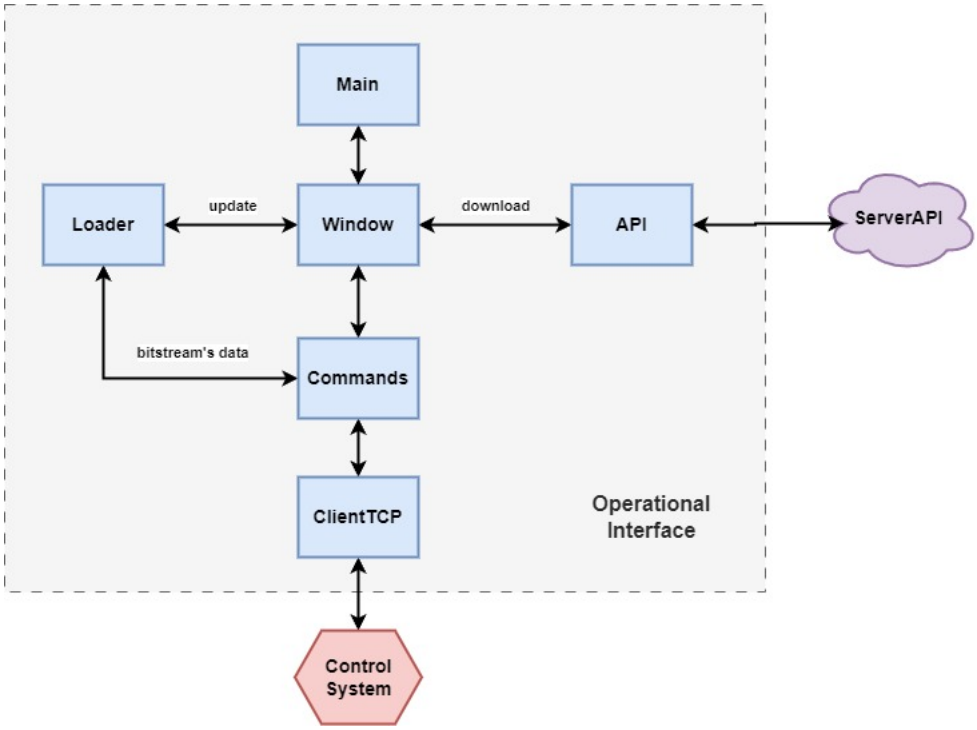
A interface gráfica desenvolvida é responsável por realizar a interface homem-máquina, possibilitando ao operador ter acesso às funcionalidades dos módulos eletrônicos com FPGA. No programa, são exibidos o modo de operação dos sistemas com FPGA (*bootloader* ou aplicação), a versão atual dos firmwares instalados em cada módulo da máquina e um indicador da versão mais atual.

O programa realiza periodicamente uma requisição ao servidor remoto para verificar se existe uma versão mais atual do *bitstream* do que a versão vigente no módulo. Caso seja confirmado a existência de uma atualização, um botão de *update* é liberado, para que a atualização possa ser baixada. Além disso, o operador deve alternar para o modo de bootloader para realizar a atualização do bitstream de aplicação das FPGAs.

Na Figura 11, é ilustrada a estrutura da lógica do programa da interface gráfica. O sistema foi separado em 6 arquivos fonte para facilitar a organização das funcionalidades sendo:

- *Main*: Função principal que chama o módulo da interface gráfica chamado de Window.
- *Window*: Módulo que cria a janela de aplicação da interface gráfica para interação com usuário.
- *API*: Módulo contendo as funções para comunicar com a API do servidor remoto.
- *Loader*: Módulo que realiza fragmentação e os processos necessários para o empacotamento e envio do *bitstream*.

Figura 11 – Esquemático lógico da programação da interface gráfica.



Fonte: O Autor

- *Commands*: Módulo que executa a conexão e envio dos comandos com o sistema eletrônico de controle da máquina/roteamento da máquina.
- *clientTCP*: Módulo que recebe e envia as requisições para o módulo de roteamento através do protocolo TCP/IP.

Para a transferência do binário, a aplicação fica encarregada de fragmentar o *bitstream* da aplicação, montar os *frames* no formato do protocolo serial proprietário, e transferir estes pacotes para o módulo de roteamento, que retransmite para o restante do sistema. A interface gráfica foi desenvolvida em Python utilizando a biblioteca PySimpleGUI, que provê os *widgets* como botões, frames e demais componentes. A escolha da biblioteca foi devido a praticidade e agilidade necessária para o desenvolvimento do projeto.

3.3.2 Servidor

O servidor consiste em uma aplicação hospedada remotamente, onde o *bitstream* de aplicação é armazenado e disponível para ser baixado, caso seja solicitado. A interface do sistema com o servidor é realizada por meio de um restAPI que fornece os metadados da versão do *bitstream* presente no servidor, necessários para verificar uma atualização. O servidor é hospedado em uma máquina com o sistema operacional Ubuntu 20.04 LTS,

na plataforma de hospedagem Locaweb. A aplicação servidor foi criada em Python e foi utilizado o Apache com o módulo *mod_wsgi* para executar a aplicação Python no servidor.

3.3.3 Sistema eletrônico

Conforme apresentado na Figura 9, o sistema eletrônico da máquina é composto por 2 subsistemas: os módulos de aplicação e o módulo de roteamento. Os módulos de aplicação, como o nome sugere, são responsáveis por executar e controlar as funcionalidades da aplicação, sendo essenciais para a finalidade do sistema. Estes módulos estão conectados por um barramento ao módulo de roteamento de pacotes. O módulo de roteamento é o sistema intermediário que gerencia e transmite os pacotes de comando para os barramentos do sistema e recebe os pacotes de retorno, encaminhando para a interface gráfica. No sentido contrário, os pacotes vindos da interface são recebidos via TCP/IP, e são processados pelo sistema, que encaminha para os módulos eletrônicos utilizando um protocolo de comunicação proprietário.

Especificamente, para realizar a atualização do *bitstream* presente nos módulos de aplicação, são armazenados duas imagens de configuração em uma mesma memória *flash*, conforme demonstra a Figura 12. Além da imagem de aplicação, o módulo possui uma imagem de *bootloader* responsável por iniciar as funcionalidades básicas do sistema. A principal função do *bootloader* é garantir a execução do módulo eletrônico em um modo seguro que possibilite e viabilize a atualização, evitando a corrupção completa do sistema caso aconteça algum erro durante uma atualização, evitando, portanto o comprometimento do funcionamento da máquina. Com o sistema possuindo o *bootloader* como uma imagem de configuração segura, será sempre possível realizar uma nova gravação da imagem de aplicação.

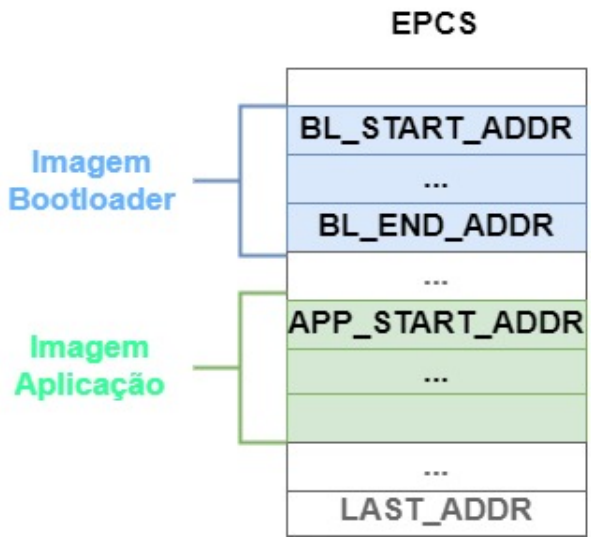
O *bootloader* conta com as seguintes funcionalidades:

- Comunicação com a interface gráfica;
- Sistema de atualização de hardware;
- Sistema de status;
- Acesso a R/W da memória *flash*.

No que diz respeito a arquitetura de *hardware* (Figura 13), o projeto do *bootloader* conta com os seguintes submódulos:

- **Serializer/Desserializer**: Módulos de comunicação que fazem a conversão dos dados de paralelo-serial e serial-paralelo, respectivamente.
- **FIFO**: IP da Altera utilizado para bufferizar o stream de dados recebidos do Desserializer.
- **Nios2-processor**: Softcore que possui a unidade de processamento para execução do software embarcado.

Figura 12 – Localização das imagens de bootloader e aplicação.



Fonte: O Autor

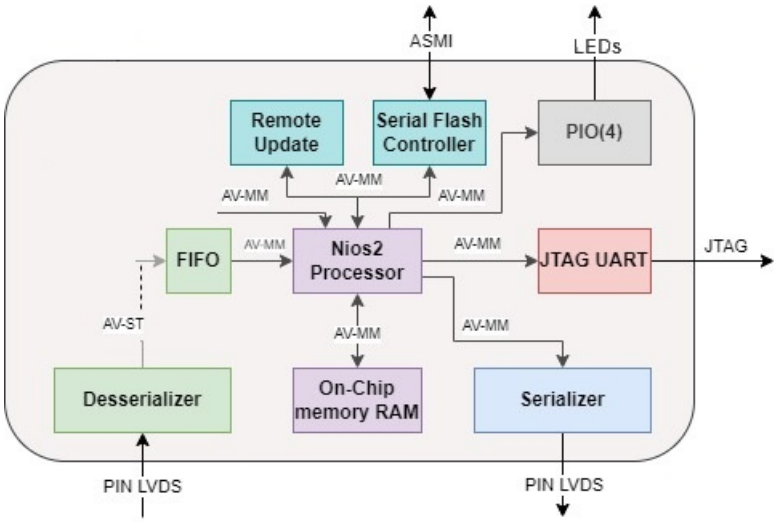
- **On-Chip-Memory**: IP da Altera que fornece uma estruturada de memória RAM para armazenar e auxiliar na execução do software embarcado.
- **Parallel I/O (PIO)**: IP da Altera que permite converter a interface de um sinal externo para a interface *Avalon Memory-Mapped*, possibilitando ao *software* acessar o sinal através dos *drivers*.
- **Serial Flash Controller**: IP da Altera que permite acessar a interface ASMI da FPGA.
- **Remote Update**: IP da Altera que permite acessar o circuito de reconfiguração
- **JTAG UART**: IP da Altera que permite utilizar um terminal em um computador a partir do barramento JTAG.

3.3.3.1 Comunicação do sistema

O sistema de comunicação entre os módulos eletrônicos é *full duplex*, ou seja, as informações trafegam por barramentos separados, podendo ser transmitidas e recebidas ao mesmo tempo. Essas informações trafegam em pacotes tanto no sentido da interface gráfica para a cadeia de módulos de aplicação, definido como sentido de comando, quanto no sentido inverso, definido como sentido de retorno conforme ilustra a Figura 14. O módulo de roteamento realiza a ponte de comunicação entre os módulos de aplicação e a interface gráfica e serve principalmente para rotear dados entre os barramentos.

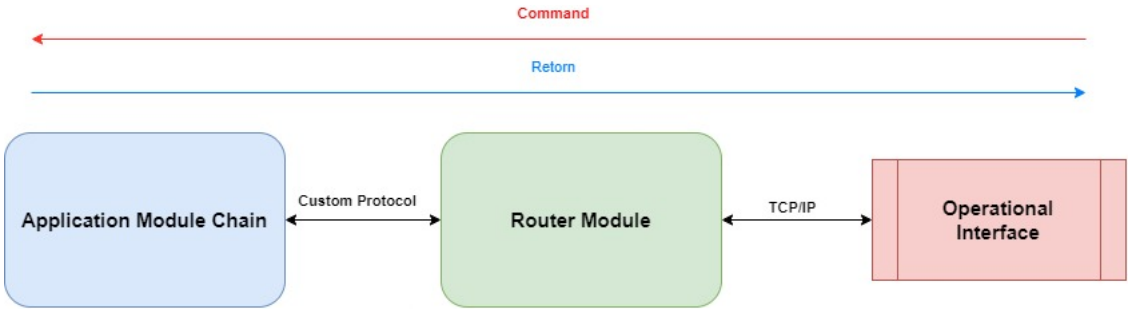
O componente serializador presente nas FPGAs, como o nome sugere, é responsável por serializar os bytes de dados para uma transferência bit a bit por meio de um barramento LVDS. Os dados são inseridos neste componente a partir da escrita de dados na interface

Figura 13 – Arquitetura de *hardware* do *bootloader* embarcado no FPGA



Fonte: O Autor

Figura 14 – Sentidos dos pacotes de comunicação.

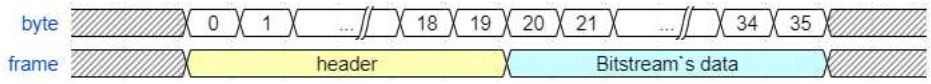


Fonte: O Autor

Avalon-MM (*Avalon Memory-Mapped*) acessível ao processador embarcado, de onde vêm as informações escritas no componente, e são disponibilizados na saída serializada deste módulo no formato da interface Avalon-ST (*Avalon Stream*). Este componente adiciona informação de *header* aos conjunto de dados, compondo o pacote. Este componente também possui a integração com outro componente que gera os bytes de CRC (*Cyclic Redundancy Check*) que são inclusos ao final do pacote e são utilizados na verificação de integridade de pacote no módulo de desserialização.

O componente desserializador é responsável pela reconstrução dos bits transferidos pelo barramento de comunicação para bytes (dados paralelos). Os dados que trafegam no sentido de comando são compartilhados por todos os módulos, mas apenas aqueles ao qual o pacote é endereçado processam os dados. Para os demais módulos, aos quais não são endereçados, os pacotes são descartados. Este componente inicia a sua máquina de

Figura 15 – Estrutura do pacote de comunicação.



Fonte: O Autor

estados a partir da identificação de uma assinatura de transmissão proprietária gerada pelo módulo serializador. Ao identificar uma transmissão o módulo converte os bits em bytes, que são verificados a partir da checagem do CRC transferido pelo módulo serializador, a fim de verificar a integridade dos dados transmitidos. Após a verificação, os pacotes são encaminhados para um *buffer*, onde os dados recebidos pelo barramento serão lidos a partir da interface Avalon-MM pelo *firmware* através dos *drivers*.

A estrutura do pacote de comunicação encontra-se descrito na Figura 15 e desempenha um papel importante na identificação do tipo de pacote, da entidade geradora e a entidade receptora. O pacote de comunicação possui 36 bytes e o seu frame é organizado em dois tipos de informações diferentes: o *header* e os dados. Dentro do *header* são encaminhados metadados que carregam informações de utilização dos dados que estão sendo transmitidos, tais como o identificador de endereçamento e o comando a ser executado. Para os pacotes de dados relacionados a transferência do *bitstream*, a informação do endereço da memória *flash* aonde estes dados devem começar a ser escritos são encaminhados no *header*. Na estrutura utilizada, o header ocupa 20 bytes, restando 16 bytes de dados.

3.3.3.2 A memória Flash

Para realizar o acesso ao *flash* conectada a FPGA, foi utilizado o IP da Altera *Serial Flash Controller*, que possibilita o acesso ao dispositivo utilizando o processador Nios2 a partir dos *drivers* implementados pela HAL durante o processo de geração do BSP. As informações disponíveis do IP na documentação da Altera, definem que se pode acessar as funções da *flash* a partir dos registradores do componente, definidos no mapa de registradores exibido na Figura 16.

Para realizar a transferência dos dados do *bitstream* de configuração para a *flash* foi necessário compreender a estrutura de endereços dos registradores do dispositivo, bem como os comandos relacionados. Dentre os comandos fornecidos pela *flash* o mais importante é o que realiza a proteção de setores, pois setores protegidos não podem realizar operações de apagamento nem formatação da *flash*, de modo que os dados armazenados se mantenham preservados.

Nas FPGAs da Altera é comum utilizar os dispositivos de configuração serial ou EPCS para o armazenamento do *bitstream*. A memória *flash* no projeto dos módulos eletrônicos foi a EPCS64, com capacidade de até 64Mbits, ou 8 MBytes. Este dispositivo

Figura 16 – Mapa de registradores do IP Serial Flash Controller.

Register	Offset	Width	Access	Description
FLASH_RD_STATUS	0x0	8	R	Perform read operation on flash device status register and store the read back data.
FLASH_RD_SID	0x1	8	R	Perform read operation to extract flash device silicon ID and store the read back data. Only support in EPCS16 and EPCS64 flash devices.
FLASH_RD_RDID	0x2	8	R	Perform read operation to extract flash device memory capacity and store the read back data.
FLASH_MEM_OP	0x3	24	W	To protect and erase memory
FLASH_ISR	0x4	2	RW	Interrupt status register
FLASH_IMR	0x5	2	RW	To mask of interrupt status register
FLASH_CHIP_SELECT	0x6	3	W	Chip select values: • B'000/b'001 -chip 1 • B'010 - chip 2 • B'100 - chip 3

Fonte: Altera Embedded Peripherals IP User Guide (2024, p. 232)

Figura 17 – Mapa de endereços da EPCS64.

Sector	Address Range (hex)	
	Start	End
127	0x7F0000	0x7FFFFF
126	0x7E0000	0x7EFFFF
125	0x7D0000	0x7DFFFF
...	...	...
...	...	...
2	0x020000	0x02FFFF
1	0x010000	0x01FFFF
0	0x000000	0x00FFFF

Fonte: O Autor

utiliza a interface ASMI, e é conectado ao FPGA conforme a Figura 6. Neste projeto todos os módulos eletrônicos com FPGA operam utilizando a configuração serial ativa, o que implica que as FPGAs carregarão o *bitstream* desta memória *flash* toda vez que a placa for inicializada. A memória *flash* utilizada possui 128 setores conforme a Figura 17, onde cada setor possui 65536 endereços de 8 bits.

Os setores de memória foram divididos em duas regiões, uma para cada *bitstream* de configuração. O *bitstream* do *bootloader* é armazenado no primeiro setor da *flash*,

pois segundo a documentação da Altera, é onde a FPGA utilizada, que está configurada para o modo Ativo Serial, busca o *bitstream* de configuração. O *bitstream* de aplicação é armazenado em outra região, que deve ser indicada para o componente de reconfiguração, o IP Remote Update. Um espaço vazio de endereços é reservado após cada *bitstream* como uma margem de segurança caso o bitstream cresça e ocupe mais endereços, evitando a sobrescrita entre os setores de uma imagem com a outra.

3.3.3.3 A arquitetura de transferência.

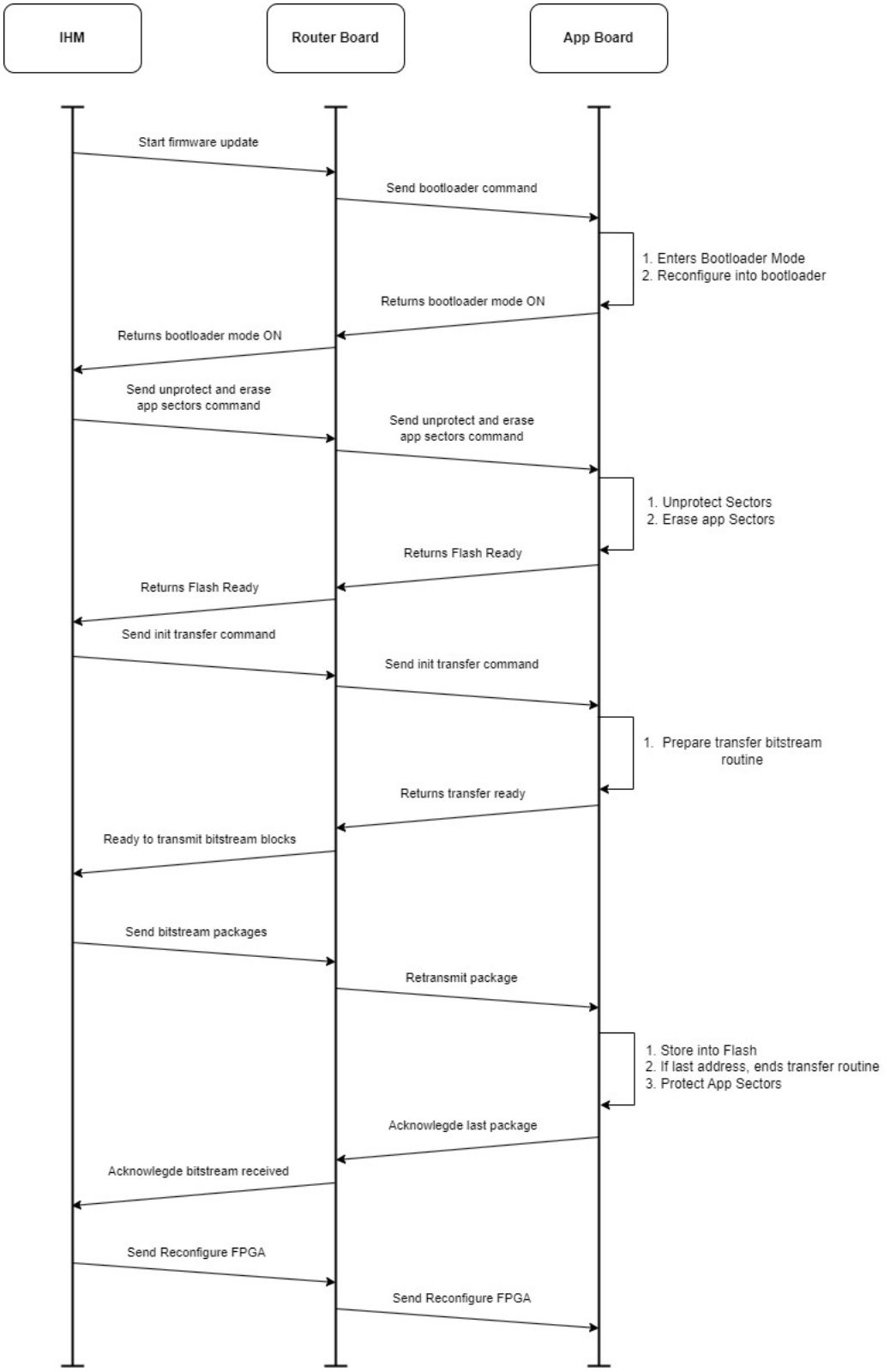
Para realizar a transmissão do *bitstream* de aplicação de forma apropriada foi definido uma estrutura de transferência exibida no diagrama de comunicação na Figura 18. Esse diagrama contempla os 3 nós da rede de comunicação, a saber: a interface gráfica (IHM), o módulo de roteamento (*router board*), e o módulo de aplicação (*application board*). A IHM é responsável por fragmentar o arquivo binário, separando em pacotes os dados do bitstream, encaminhando-os para a placa de roteamento a partir do protocolo TCP/IP. A placa de roteamento é responsável por receber estes pacotes, processá-lo e reencaminhá-lo para a uma ou mais placas de aplicação as quais os pacotes são endereçados. Para iniciar o processo de atualização do sistema, a IHM deve solicitar que o módulo de aplicação reconfigure primeiramente a FPGA para o *bitstream* de *bootloader*. Uma vez que o módulo encontra-se em modo *bootloader*, a IHM pode solicitar os comandos necessários para preparar a placa para a transferência do *bitstream*.

Para a correta transferência do *bitstream* de aplicação, os setores de aplicação da *flash* devem ser desbloqueados e posteriormente apagados, mantendo os setores aonde se encontra o *bitstream* do *bootloader* protegidos para evitar o apagamento acidental. Uma vez que os módulos eletrônicos estiverem prontos para receber o *bitstream*, a IHM envia o comando para entrar em modo de transferência. O *bitstream* de aplicação é transferido em seguida, em um processo de envio dos pacotes em modo de enxurrada. Ao receber o último pacote, a placa de aplicação sinaliza que o bitstream foi transferido. Um comando de reconfiguração é enviado para que o módulo entre em modo de operação, agora com a imagem atualizada.

O fluxograma lógico da rotina do *firmware* do *bootloader* é exibido nas Figuras 19 e 20. Os pacotes de dados armazenados na FIFO, são lidos no *software* a partir dos *drivers* do componente. A rotina do *software* é dividida em dois modos de operação: o modo *standby*, onde o módulo recebe, lê e processa os comandos do sistema, e o modo de transferência onde a FIFO fica inteiramente dedicada a receber os pacotes do *bitstream*. Na Figura 18 são mostrados todos os comandos reconhecidos pelos módulos eletrônicos durante o modo *bootloader*, a saber:

- Configuração de comunicação: Configura os parâmetros do módulo de comunicação para o retorno de dados do módulo de aplicação para a IHM.
- Reconfigurar FPGA: Comando para executar a reconfiguração da FPGA.

Figura 18 – Diagrama de comunicação de Transferência de um *bitstream*.



Fonte: O Autor

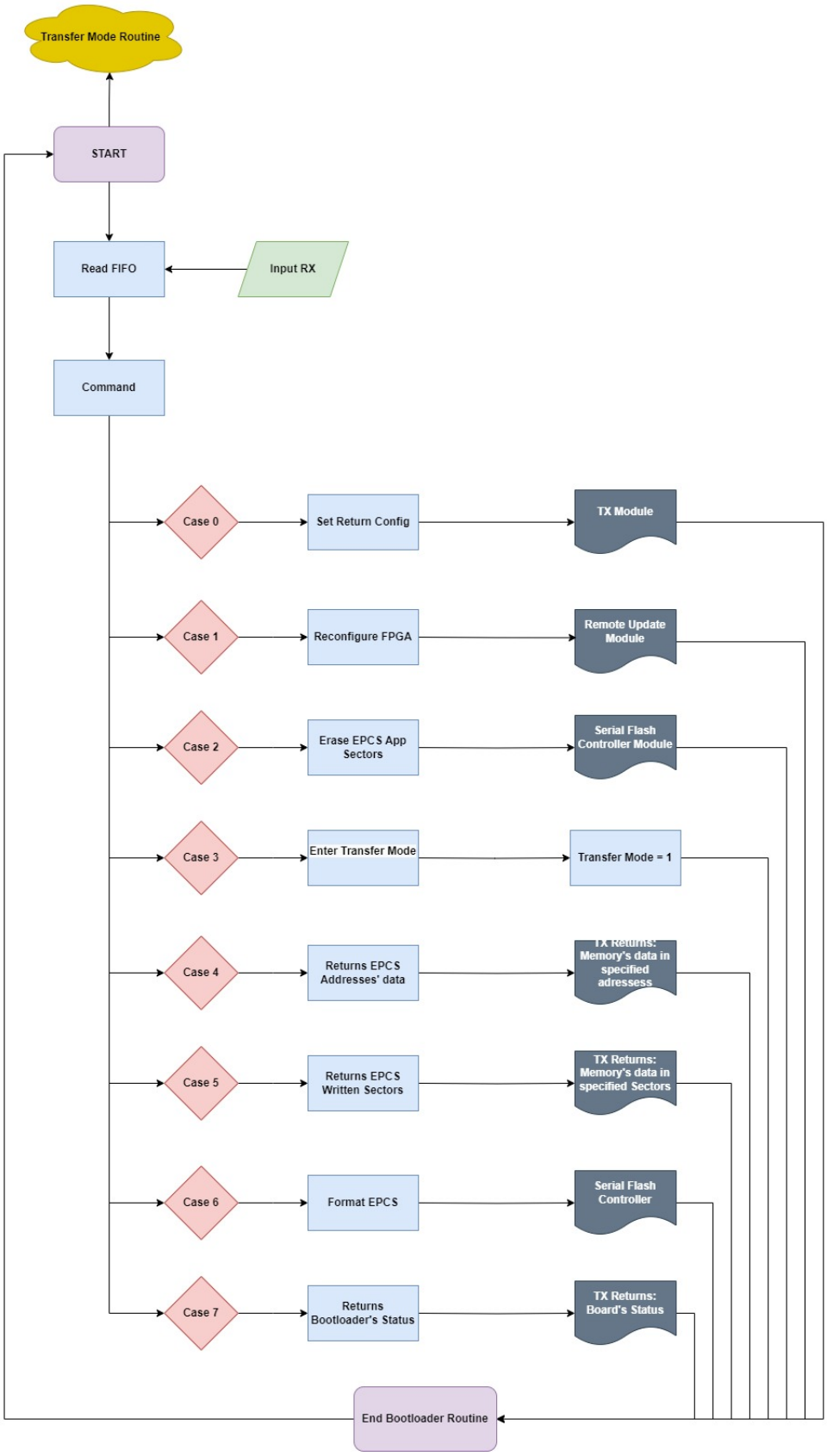
- Apagar setores aplicação: Desprotege e apaga os setores da memória *flash* onde a imagem de aplicação está armazenada. Limpar os setores de aplicação é um dos processos necessários para realizar a atualização.
- Entrar em modo de transferência: Troca para a rotina de transferência.
- Retorno de Status da placa: Retorna para a IHM o status da placa, como status de transferência, erros de transmissão, status da *flash*.
- Formatar EPCS: Este comando desprotege e formata todos os setores da EPCS, apagando todas as imagens. Esse comando deve ser utilizado apenas caso seja necessário atualizar o próprio *bootloader*.

O módulo de aplicação realiza o processamento de todos os pacotes que chegam ao barramento, mas apenas capturam aqueles que são endereçados a ele. Como *bitstream* é transmitido por todo o barramento serial conforme é ilustrado na Figura 9, os pacotes transmitidos pelo barramento serial são acessíveis a todos os módulos da cadeia, proporcionando ao sistema a capacidade de transferência do *bitstream* em *broadcast*, possibilitando a atualização simultânea de todos os módulos do barramento, acelerando o processo de atualização do sistema como um todo.

3.3.3.4 Remote Update

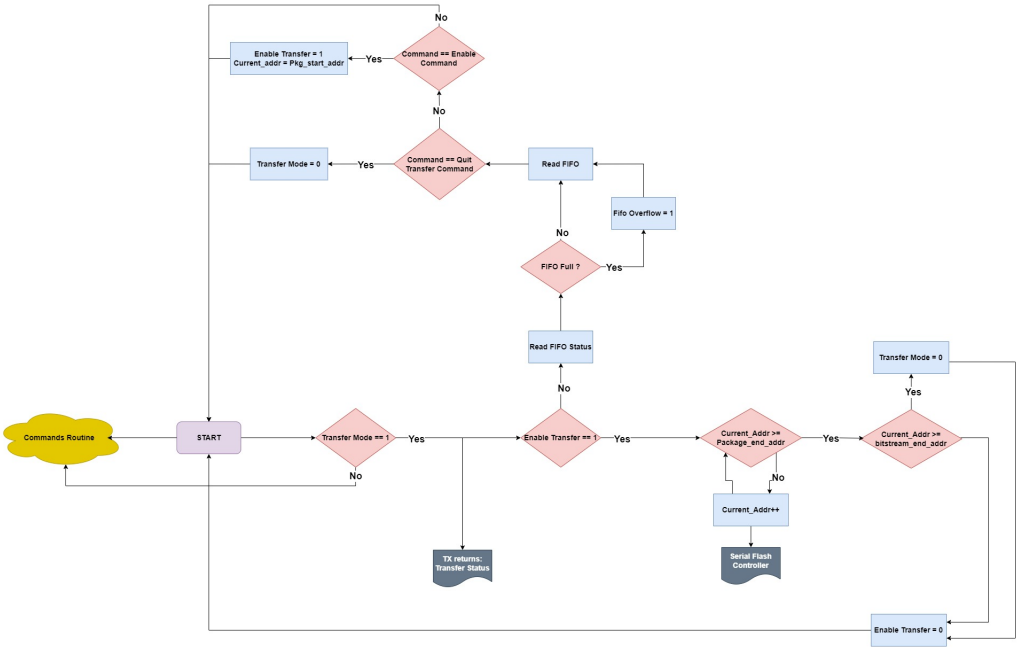
O componente Remote Update é o IP responsável por realizar a reconfiguração da FPGA. Segundo a documentação da Altera, este componente possui controle sobre o circuito de reconfiguração interno do chip, acessível através da interface Avalon-MM. Os *drivers* fornecem acesso aos registradores que realizam as configurações de reconfiguração, onde é possível configurar um *watch dog timer* do circuito, o endereço da *flash* aonde o circuito irá buscar o *bitstream* e o disparo de reconfiguração. Na Figura 21, é apresentado um diagrama que descreve o funcionamento do circuito de reconfiguração. Durante a inicialização no modo ativo serial, a placa é iniciada na imagem localizada no endereço de *boot* padrão da FPGA utilizada. Ao tentar fazer uma reconfiguração o Remote Update irá tentar buscar no endereço indicado o *bitstream* de configuração secundário que será carregado logo em seguida. O componente possui um *watch dog timer*, que emite uma interrupção para o componente caso o circuito não consiga carregar o *bitstream*, seja porque o mesmo está corrompido, ou seja porque não foi encontrado na localização indicada. Caso o circuito falhe em reconfigurar a FPGA, o Remote Update irá carregar o *bitstream* localizada no endereço padrão de *boot*. Segundo (ELI, 2017) é importante notar que o Remote Update IP não tem nada a ver com a programação da *flash*: é apenas uma função que permite ter acesso ao recurso de reconfiguração do FPGA.

Figura 19 – Fluxograma lógico do *software bootloader*.



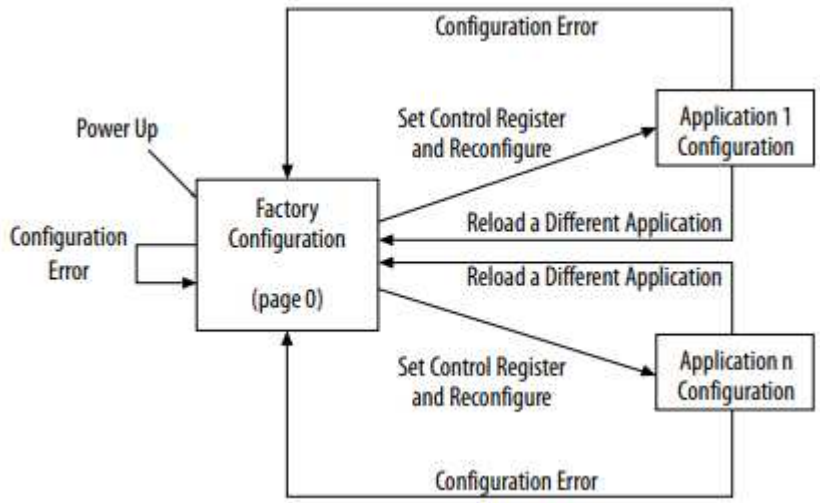
Fonte: O Autor

Figura 20 – Fluxograma lógico da rotina de transferência.



Fonte: O Autor

Figura 21 – Funcionamento de reconfiguração do IP Remote Update.



Fonte: Altera Remote Update IP User Guide (2022, p. 16)

4 RESULTADOS

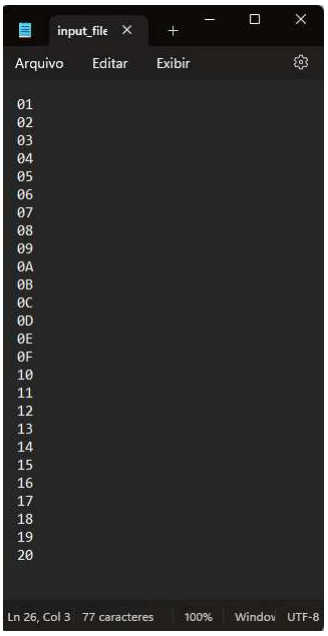
4.1 SIMULAÇÃO

O *Black Box Testing* (ou Teste de Caixa Preta) é uma metodologia de teste de software e hardware que visa avaliar a funcionalidade do mesmo sem conhecer a estrutura interna, arquitetura ou o código-fonte. O foco está em realizar uma análise funcional, verificando se as entradas fornecidas ao sistema produzem as saídas esperadas, conforme as especificações e requisitos do sistema.

Para verificar o funcionamento e integração entre os componentes de comunicação foi realizada uma transmissão de pacotes dentro do ambiente de simulação do Modelsim, onde foram transferidos dados entre os componentes Serializador e Desserializador. Os dados transferidos entre os componentes (na simulação) foram inseridos a partir de um arquivo de texto, cujo conteúdo é apresentado na Figura 22. Nessa figura, tem-se 2 grupos de sinais relacionados ao Serializador, presente em uma placa eletrônica fictícia 1, e um Desserializador (representado em uma placa fictícia 2). No componente "SER" da Placa 1, pode-se observar os dados representados pelo sinal *Data* (circulado em laranja) exibidos na Figura 23 sendo serializados e enviados no sinal *Sent Package*.

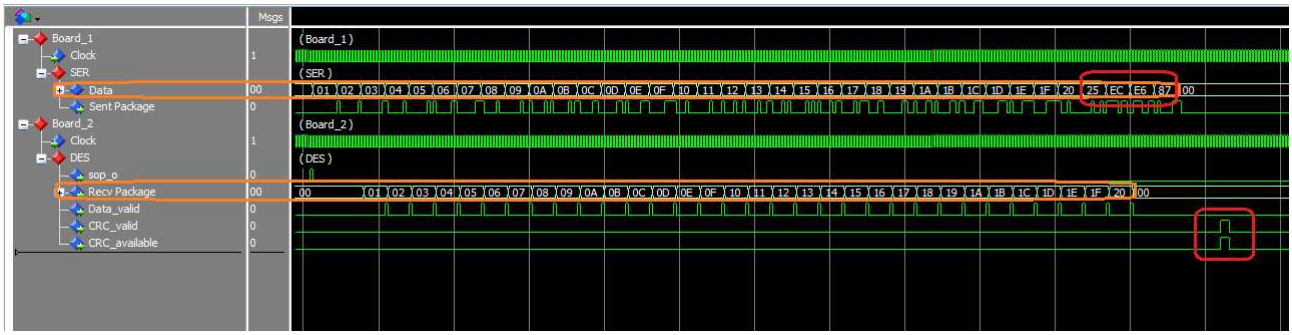
No componente "DES" na Placa 2, pode-se observar em laranja o pacote recebido pelo componente, sendo os dados recebidos congruentes com os enviados, o que caracteriza uma transferência bem-sucedida. Para garantir que os dados são íntegros podemos verificar que o sinal do CRC é alterado para o estado alto no final da transmissão, garantindo que

Figura 22 – Dados de entrada.



Fonte: O Autor

Figura 23 – Resultado da Simulação da comunicação.



Fonte: O Autor

o CRC enviado foi verificado, o que sinaliza a integridade dos dados recebidos na Placa 2.

4.2 COMPILAÇÃO

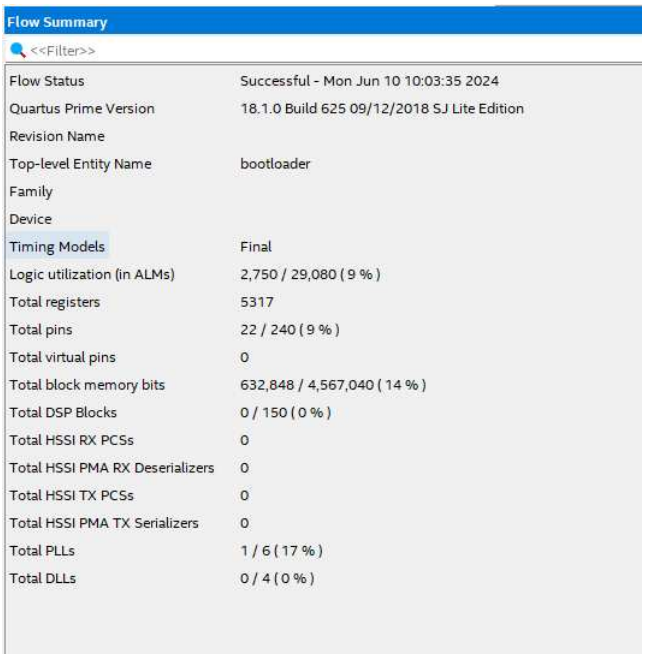
Par avaliar a compilação observou-se a ocupação percentual dos recursos lógicos utilizados em relação aos recursos lógicos disponíveis no FPGA da Altera utilizado. No relatório de compilação do Quartus exibidos nas Figuras 24 e 25, podemos observar que o resultado da compilação do projeto como todo teve uma ocupação de 2750 ou 9% dos Recursos lógicos (*ALM utilization*) disponíveis no chip, utilizando cerca de 633 Kbits ou 14% dos bits dos recursos memória RAM. Além disso, foi verificado na Figura 25 que o projeto utilizou apenas 16% das LABs disponíveis e em torno de 18% dos blocos de memória M10K do chip.

Os resultados da compilação do projeto se mostram satisfatórios, visto que uma das maiores preocupações em projetos com FPGA é a capacidade do chip de abrigar toda a lógica do sistema. Por este sistema possuir uma ocupação reduzida em termos de elementos lógicos, ele possui também a vantagem de ser facilmente portável para chips com uma capacidade menor, o que viabiliza a sua utilização em uma ampla gama de FPGAs comerciais com capacidade limitada.

4.3 INTERFACE GRÁFICA

A Interface gráfica desenvolvida, exibida na Figura 26, ilustra as funcionalidades do sistema projetado. A interface gráfica possui duas *threads* de execução: uma que fica periodicamente se comunicando com o módulo eletrônico, requisitando o modo de operação e a versão vigente, e outra que se comunica com o servidor para checar a versão mais atual do sistema, disponível através de uma restAPI. O status exibido na Figura 26 é acessado utilizando esta mesma restAPI e é monitorado a cada segundo.

Figura 24 – Relatório de compilação do projeto.



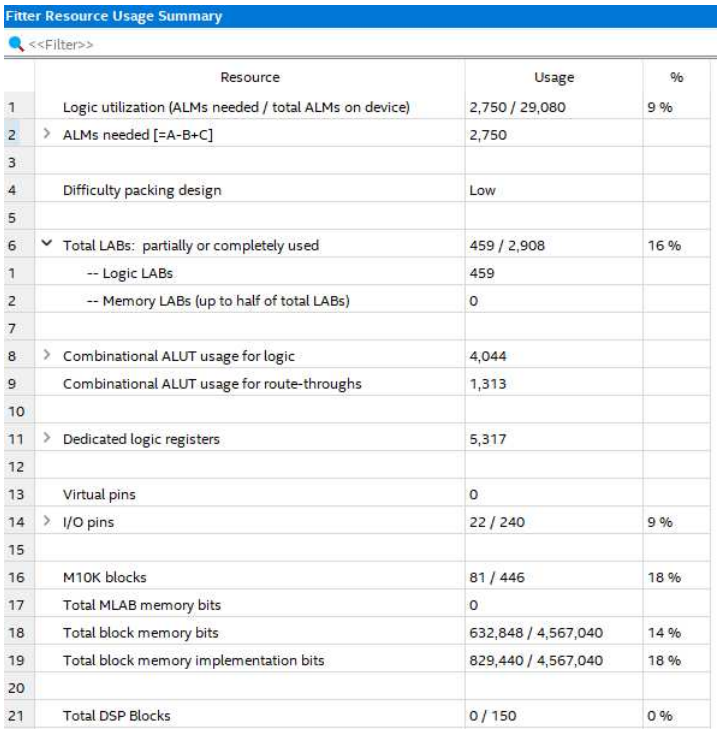
Flow Summary

<<Filter>>

Flow Status	Successful - Mon Jun 10 10:03:35 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	
Top-level Entity Name	bootloader
Family	
Device	
Timing Models	Final
Logic utilization (in ALMs)	2,750 / 29,080 (9 %)
Total registers	5317
Total pins	22 / 240 (9 %)
Total virtual pins	0
Total block memory bits	632,848 / 4,567,040 (14 %)
Total DSP Blocks	0 / 150 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1 / 6 (17 %)
Total DLLs	0 / 4 (0 %)

Fonte: O Autor

Figura 25 – Relatório de recursos utilizados da FPGA.



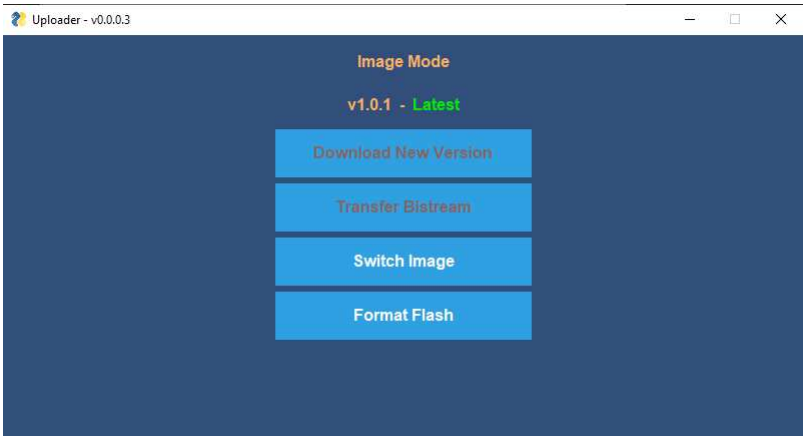
Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	2,750 / 29,080	9 %
2	ALMs needed [=A-B+C]	2,750	
3			
4	Difficulty packing design	Low	
5			
6	Total LABs: partially or completely used	459 / 2,908	16 %
1	-- Logic LABs	459	
2	-- Memory LABs (up to half of total LABs)	0	
7			
8	Combinational ALUT usage for logic	4,044	
9	Combinational ALUT usage for route-throughs	1,313	
10			
11	Dedicated logic registers	5,317	
12			
13	Virtual pins	0	
14	I/O pins	22 / 240	9 %
15			
16	M10K blocks	81 / 446	18 %
17	Total MLAB memory bits	0	
18	Total block memory bits	632,848 / 4,567,040	14 %
19	Total block memory implementation bits	829,440 / 4,567,040	18 %
20			
21	Total DSP Blocks	0 / 150	0 %

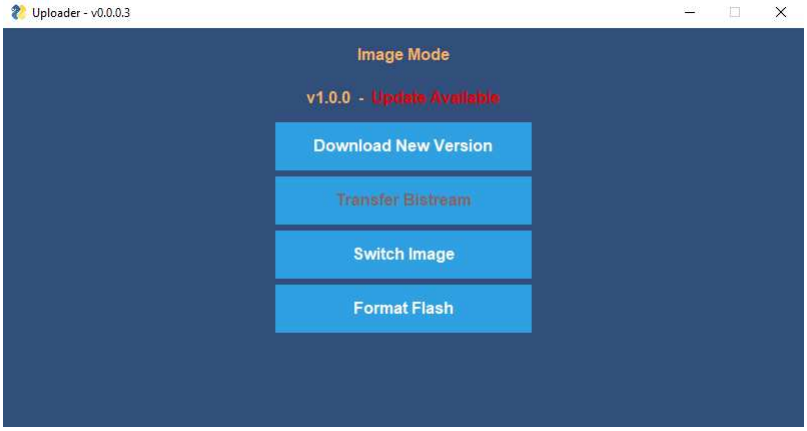
Fonte: O Autor

Figura 26 – Interface gráfica indicando sistema atualizado.



Fonte: O Autor

Figura 27 – Interface gráfica indicando que há uma atualização disponível.



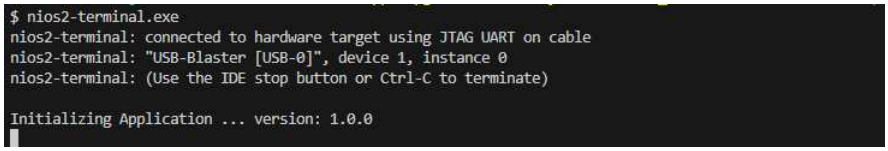
Fonte: O Autor

O sistema realiza então a comparação entre as versões vigente e a remota a fim de garantir a concordância das versões. Caso haja uma nova versão disponível no servidor, a mensagem de atualização disponível é exibida conforme a Figura 27, podendo ser baixada ao clicar no botão *Download New Version*. Ao baixar a versão o sistema poderá ser atualizado, bastando apenas trocar a imagem clicando em *Switch Image* para entrar no modo bootloader e, por fim, clicando no botão *Transfer Bitstream* para atualizar a imagem das FPGAs presentes nos módulos eletrônicos.

4.4 OPERAÇÃO

Para demonstrar a operação do sistema projetado, foram desenvolvidas, duas imagens (*bitstreams* de configuração de FPGA) distintas, contendo internamente os módulos

Figura 28 – *log* do *bitstream* de aplicação vigente.



Fonte: O Autor

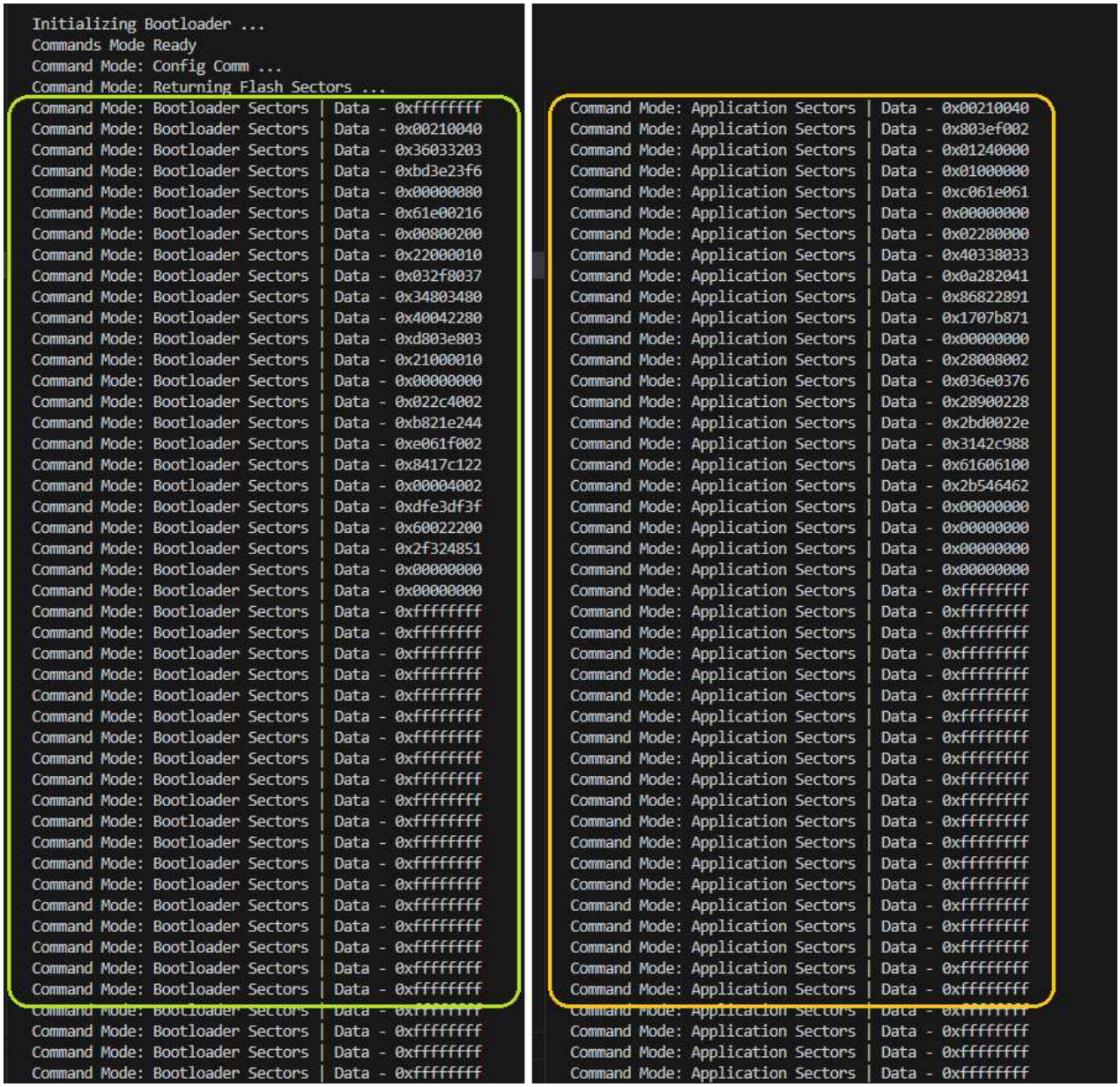
de comunicação, e um processador embarcado sobre o qual, executa o *firmware*. Para demonstração de atualização, cada *bitstream* contém uma versão diferente de *firmware* com um *bitstream* estando armazenado na FPGA do módulo eletrônico e o outro armazenado em um servidor remoto. Para realizar um cenário de atualização remota, foi contratado um serviço de hospedagem de uma VPS Linux, com o *bitstream* de atualização sendo armazenado em uma máquina virtual com sistema operacional Ubuntu 22.04 operando como servidor remoto. O *bitstream* é fornecido através de uma restAPI.

Para monitorar as operações executadas durante e após os testes realizados, foram inseridos *logs* na interface gráfica e nos módulos de aplicação visando verificar se os subsistemas estavam operando de forma correta. Para acessar o *log* do módulo de aplicação foi utilizado o conjunto módulo de aplicação conectado a um PC através de um USB-Blaster, que possibilita a utilização do Nios2-terminal. Para medir o desempenho do sistema desenvolvido e poder fazer uma análise quantitativa do sistema, foram definidos dois critérios principais para avaliação dos testes realizados, a saber: o primeiro dado pelo tempo de gravação do sistema implementado, e o segundo sendo a taxa de sucesso das gravações. Vale ressaltar que todos os testes foram realizados utilizando o mesmo setup, e os mesmos parâmetros de transmissão. Para realizar a transferência foi definido um tempo de envio entre pacotes de 0,85 ms, sendo ao todo cerca de 196500 pacotes transferidos durante uma atualização.

Em modo *bootloader*, na interface gráfica são disponibilizados os comandos exibidos na Figura 19. Antes de realizar uma gravação primeiro é executado o comando de apagamento para limpar os setores da memória flash onde está o *bitstream* de aplicação. Nas Figuras 29 e 30 temos o antes e o depois, respectivamente, da operação de apagamento dos setores relacionados a aplicação (localizados na região de setores destacado em amarelo). Os setores de *bootloader* (destacados em verde) correspondem aos dados referentes ao sistema desenvolvido, e que durante uma operação de apagamento devem estar protegidos, evitando uma formatação acidental do mesmo. Pelas imagens é possível notar que a operação apaga apenas os dados da imagem de aplicação.

Durante uma gravação, o sistema desenvolvido entra em modo de transferência (rotina do *firmware* descrita na Figura 18) para preparar-se para receber os pacotes da transmissão. Os logs de pacotes são exibidos tanto na saída da interface gráfica como na

Figura 29 – Setores antes da limpeza dos setores.

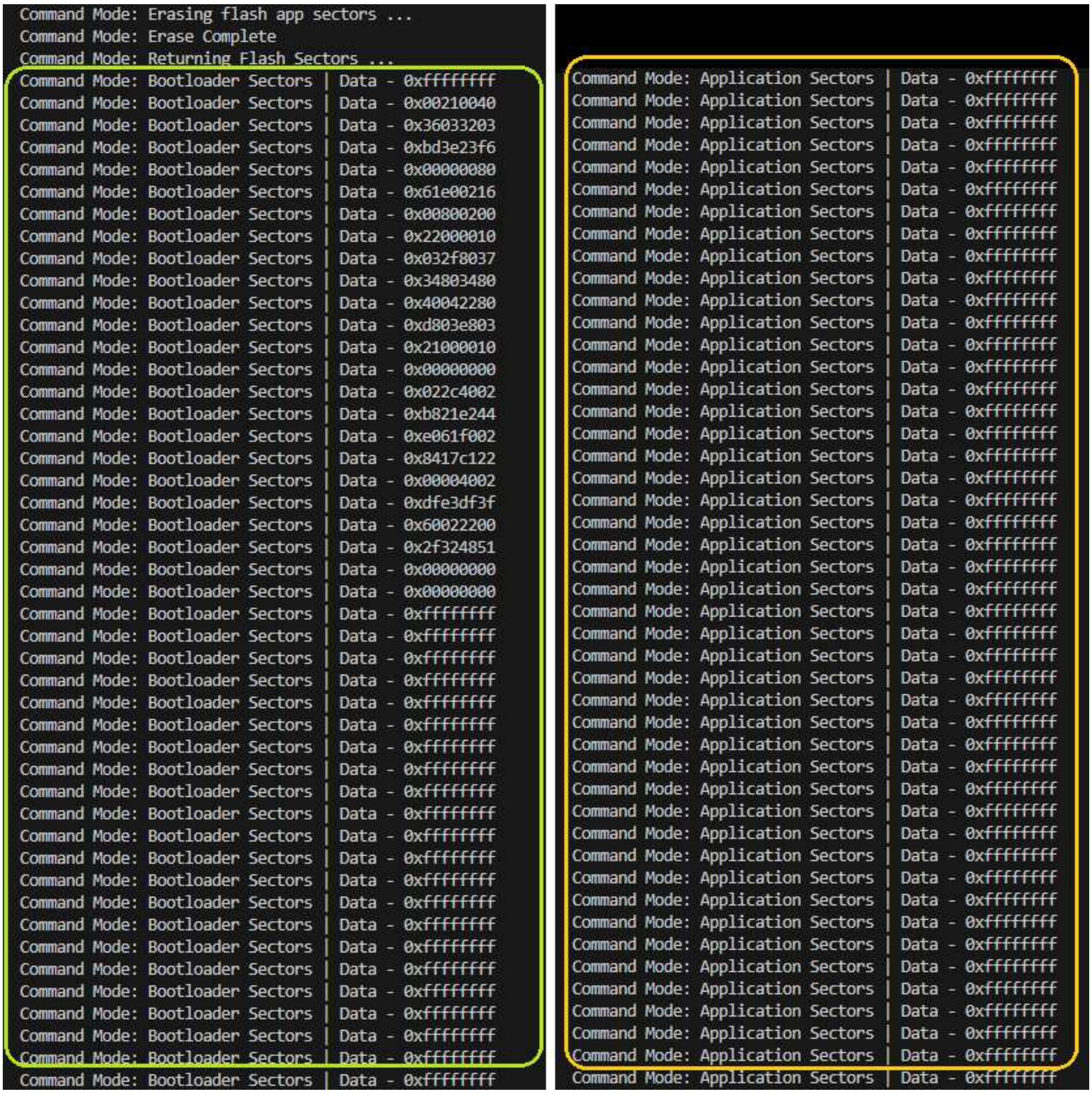


```
Initializing Bootloader ...
Commands Mode Ready
Command Mode: Config Comm ...
Command Mode: Returning Flash Sectors ...
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0x00210040
Command Mode: Bootloader Sectors | Data - 0x36033203
Command Mode: Bootloader Sectors | Data - 0xbd3e23f6
Command Mode: Bootloader Sectors | Data - 0x00000080
Command Mode: Bootloader Sectors | Data - 0x61e00216
Command Mode: Bootloader Sectors | Data - 0x00800200
Command Mode: Bootloader Sectors | Data - 0x22000010
Command Mode: Bootloader Sectors | Data - 0x032f8037
Command Mode: Bootloader Sectors | Data - 0x34803480
Command Mode: Bootloader Sectors | Data - 0x40042280
Command Mode: Bootloader Sectors | Data - 0xd803e803
Command Mode: Bootloader Sectors | Data - 0x21000010
Command Mode: Bootloader Sectors | Data - 0x00000000
Command Mode: Bootloader Sectors | Data - 0x022c4002
Command Mode: Bootloader Sectors | Data - 0xb821e244
Command Mode: Bootloader Sectors | Data - 0xe061f002
Command Mode: Bootloader Sectors | Data - 0x8417c122
Command Mode: Bootloader Sectors | Data - 0x00004002
Command Mode: Bootloader Sectors | Data - 0xdfe3df3f
Command Mode: Bootloader Sectors | Data - 0x60022200
Command Mode: Bootloader Sectors | Data - 0x2f324851
Command Mode: Bootloader Sectors | Data - 0x00000000
Command Mode: Bootloader Sectors | Data - 0x00000000
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
Command Mode: Bootloader Sectors | Data - 0xffffffff
```

```
Command Mode: Application Sectors | Data - 0x00210040
Command Mode: Application Sectors | Data - 0x803ef002
Command Mode: Application Sectors | Data - 0x01240000
Command Mode: Application Sectors | Data - 0x01000000
Command Mode: Application Sectors | Data - 0xc061e061
Command Mode: Application Sectors | Data - 0x00000000
Command Mode: Application Sectors | Data - 0x02280000
Command Mode: Application Sectors | Data - 0x40338033
Command Mode: Application Sectors | Data - 0x0a282041
Command Mode: Application Sectors | Data - 0x86822891
Command Mode: Application Sectors | Data - 0x1707b871
Command Mode: Application Sectors | Data - 0x00000000
Command Mode: Application Sectors | Data - 0x28008002
Command Mode: Application Sectors | Data - 0x036e0376
Command Mode: Application Sectors | Data - 0x28900228
Command Mode: Application Sectors | Data - 0x2bd0022e
Command Mode: Application Sectors | Data - 0x3142c988
Command Mode: Application Sectors | Data - 0x61606100
Command Mode: Application Sectors | Data - 0x2b546462
Command Mode: Application Sectors | Data - 0x00000000
Command Mode: Application Sectors | Data - 0x00000000
Command Mode: Application Sectors | Data - 0x00000000
Command Mode: Application Sectors | Data - 0x00000000
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
Command Mode: Application Sectors | Data - 0xffffffff
```

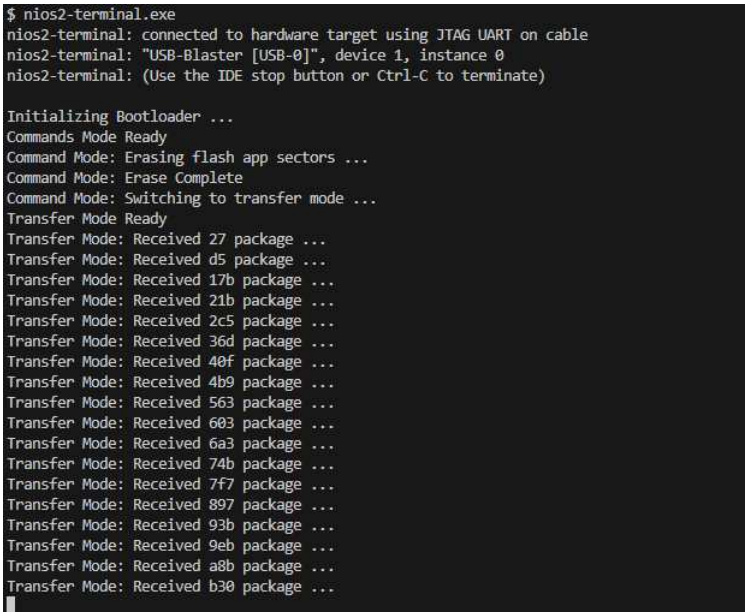
Fonte: O Autor

Figura 30 – Setores após da limpeza dos setores.



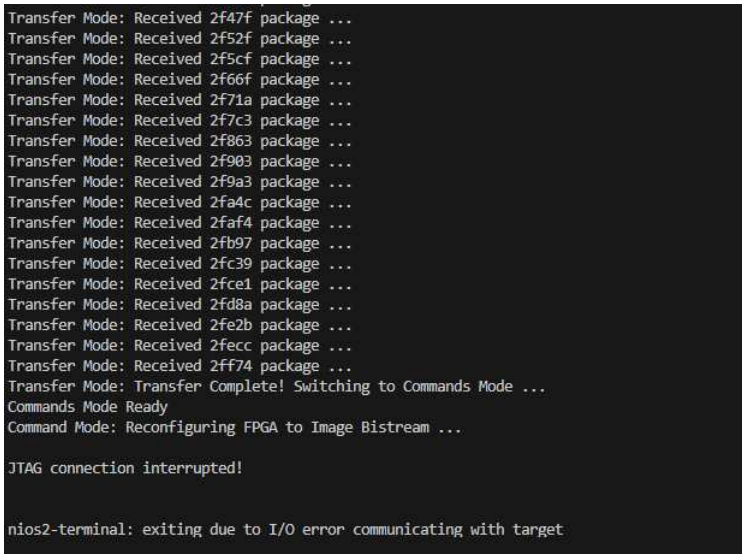
Fonte: O Autor

Figura 31 – Tela de log de transferência do bitstream.



Fonte: O Autor

Figura 32 – Tela de log de reconfiguração da FPGA.



Fonte: O Autor

chegada no *bootloader* através do Nios2-terminal (Figura 31).

Ao finalizar a transferência o sistema retorna para o modo de comando onde é possível realizar as demais operações. Na Figura 32 é ilustrado um exemplo da execução do comando de reconfiguração do módulo após uma transferência. Quando o comando de reconfiguração é executado, devido à troca de imagem, o terminal que se conectava a

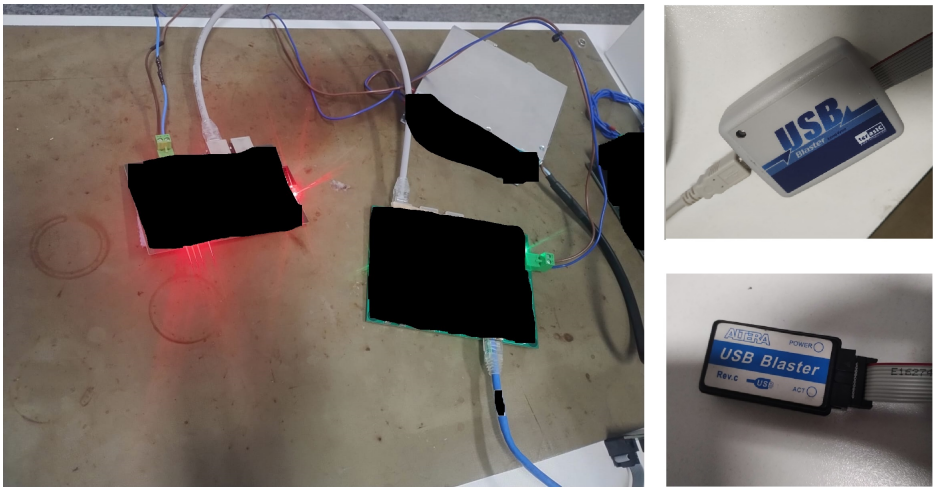
Figura 33 – Tela de log do bitstream de aplicação atualizado.

```
$ nios2-terminal.exe
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Initializing Application ... version: 2.0.0
```

Fonte: O Autor

Figura 34 – teste em bancada do protótipo.



Fonte: O Autor

imagem vigente até então é encerrado. O comando de reconfiguração busca o *bitstream* de aplicação recém transferido no setor correspondente ao início do *bitstream* na *flash*. Na Figura 33 é apresentada uma tela com o *log* do *bitstream* de aplicação após a reconfiguração realizada anteriormente. Pode-se observar na comparação entre as Figuras 28 e 33 a alteração da versão nos firmwares logados.

4.5 TESTE DE BANCADA

Com o intuito de avaliar tanto a velocidade de transmissão quanto a integridade da solução implementada, foi montada uma bancada de teste (Figura 34) contendo os módulos de aplicação e roteamento (utilizados nas máquinas industriais) e um computador executando a interface gráfica desenvolvida. O teste de velocidade consistiu em utilizar 3 métodos de gravação diferentes a fim de comparar a velocidade de comunicação entre os métodos. A gravação foi realizada utilizando um USB-Blaster (hardware de gravação de FPGAs) da Terasic, um USB-Blaster genérico e o sistema desenvolvido. O segundo teste foi realizado para avaliar a taxa de gravações bem sucedidas utilizando o sistema desenvolvido.

Figura 35 – Resultados do teste de bancada.

Método	Interface Gravação	Tempo gravação(min)	Broadcast	Tempo Total de Gravação para 9 placas	Quantidade Gravações Integras
USB-Blaster Terasic	JTAG	01:18	Não	11:42	100%
USB-Blaster Genérico	JTAG	03:38	Não	32:42	100%
Sistema desenvolvido	ASMI	04:06	Sim	04:06	90%

Fonte: O Autor

Os resultados dos testes são exibidos na Figura 35. Ao todo foram realizados 3 gravações (uma para cada método), sendo que nos testes realizados foi verificado que o USB-Blaster da Terasic foi o mais veloz com um tempo de gravação de aproximadamente 1:18, seguido do USB-Blaster genérico que levou cerca de 3:38 minutos e ficando o sistema desenvolvido (que realiza a gravação remota) por último com um tempo aproximado de 4:06 minutos. Apesar do tempo de gravação ser 315% superior ao USB-Blaster da Terasic, o sistema desenvolvido tem a capacidade de transmitir o *bitstream* em *broadcast*, ou seja, é capaz de gravar mais de um módulo de uma vez, o que não é possível utilizando os demais métodos. Além disso, o sistema desenvolvido é capaz de transferir o arquivo de configuração acessando diretamente a interface ASMI através de um barramento serial personalizado, por outro lado, o USB-Blaster precisa acessar a interface JTAG. Isto é uma vantagem no caso de ocorrer a inutilização da interface JTAG por conta de curto-circuitos. Houve situações durante testes nas quais o USB-Blaster, devido a um problema de aterramento, produziu um curto-circuito que inutilizou a interface JTAG, impossibilitando uma gravação posterior.

O segundo teste foi realizado para avaliar a taxa de gravações bem sucedidas utilizando o sistema desenvolvido. Para este teste foram realizados 20 gravações utilizando o *bootloader* (sistema desenvolvido), todas com a mesma velocidade de transmissão de 0,85 ms por pacote. Segundo os resultados do teste, a percentagem de gravações bem sucedidas ficou em torno de 90%. Este resultado é satisfatório, visto que há o benefício da gravação remota e transferência simultânea, portanto mesmo quando há a necessidade de regravar um ou outro módulo, o sistema desenvolvido é vantajoso em comparação ao método tradicional (USB Blaster), onde cada módulo é gravado um de cada vez. Além disso, durante os testes realizados na empresa para averiguar onde se encontrava o gargalo da comunicação, notou-se que o módulo de roteamento não estava conseguindo transferir os pacotes na velocidade esperada, perdendo pacotes, o que corrompia o *bitstream* transferido, o que evidencia um possível gargalo de comunicação que fica além do escopo deste trabalho.

4.6 CONSIDERAÇÕES FINAIS

Apesar da velocidade de gravação utilizando o sistema desenvolvido não ter superado o tempo de gravação utilizando um USB-Blaster, o *bootloader* desenvolvido viabiliza a transferência para vários módulos simultaneamente, ou seja, o processo de gravação utilizando o *bootloader* se torna mais eficaz. Além disso, nota-se que no sistema desenvolvido não existe esforço manual na manipulação de gravadores (como o USB-Blaster) nos módulos eletrônicos energizados, eliminando também a possibilidade de incidentes durante a conexão do USB-Blaster com os módulos, onde há riscos de curto-circuitos que podem danificar os módulos. Além disso, ressalta-se que o custo do gravador (como USB-Blaster) costuma ser elevado, e os gravadores genéricos não possuem um grau de confiabilidade adequado para atender a demandas em campo. Ao se utilizar um USB-Blaster, a atualização de uma máquina industrial composta por diversos módulos eletrônicos com FPGAs pode durar cerca de 1 hora, o que pode se resumir a 5 minutos ao utilizar-se o *bootloader* desenvolvido, o qual permite a gravação de diversas FPGAs ao mesmo tempo devido a transmissão do *bitstream* em *broadcast*, ou seja, a utilização deste tipo de gravação possibilita a atualização simultânea de sistemas com FPGA que estejam conectadas a um barramento comum, acelerando o processo de gravação.

5 CONCLUSÃO

O presente trabalho tratou sobre a implementação de um *bootloader* em sistemas com FPGA, bem como discutiu sobre a infraestrutura que permeia a atualização remota de modo geral. O sistema conseguiu alcançar os objetivos e requisitos da aplicação superando os desafios propostos referente a gravação de um *bitstream* de configuração através de um método personalizado sem a necessidade de utilização de um USB-Blaster. O método implementado além de possuir uma velocidade de gravação comparável aos métodos de gravação tradicionais, possibilita a gravação do *bitstream* em vários módulos simultaneamente, reduzindo drasticamente o tempo de gravação total de todos os módulos eletrônicos do sistema ao qual foi aplicado.

Para trabalhos futuros, acredita-se que seja possível implementar um método de verificação de integridade no sistema para garantir que a imagem esteja integra antes de tentar carregá-la no FPGA. Além disso, uma melhoria para trabalhos futuros seria adicionar uma mensagem de confirmação de recebimento de pacotes dos módulos contendo FPGA para a interface gráfica. Assim, tornaria possível, caso algum pacote não seja recebido, o reenvio do pacote perdido, aumentando a confiabilidade do sistema. No que se refere a infraestrutura de atualização, acredita-se que é possível melhorar o sistema de integração entre o servidor e a aplicação visando implementar conceitos de integração e entrega contínua (CI/CD), possibilitando desenvolver inclusive métodos de autenticação e verificação do sistema da máquina, para a implementação de mecanismos de segurança e assinatura de arquivos de configuração. Além disso, como expansão de um projeto de CI/CD completo para trabalhos futuros, seria possível utilizar esta mesma infraestrutura para implementar um sistema que realize diagnósticos remotos com sistemas que utilizam FPGAs, possibilitando até mesmo realizar debugs nos módulos eletrônicos remotamente.

REFERÊNCIAS

ALTERA, Intel. **Serial Configuration (EPCS) Devices**. [*S.l.*], 2014. Disponível em: https://semiconductors.es/semiconductors.php?id=571383. Acesso em: 27 jun. 2024.

ANDINA, J.; ARNANZ, E.; PEÑA, M. **FPGAs**: Fundamentals, Advanced Features, and Applications in Industrial Electronics. 1. ed. New York, USA: Taylor Francis, CRC Press, 2017.

CHEN, Wai-Kai. **Memory, microprocessor, and ASIC**. 1. ed. Florida, USA: CRC Press, 2003.

CHU, Pong P. **Embedded SoPC Design with Nios II Processor and Vhdl Examples**. 2. ed. New Jersey, USA: Wiley, 2011.

ELI. **Remote Update from ECPQ flash on Altera Cyclone IV**. 2017. Disponível em: https://billauer.co.il/blog/2017/06/remote-update-intel-fpga-cyclone-iv/.

GROUT, I. **Digital Systems Design with FPGAs and CPLDs**. 1. ed. Burlington: Newnes, 2008.

PAUL HOROWITZ, Winfield Hill. **The Art of Electronics**. 3. ed. New York, USA: Cambridge University Press, 2015.

JASINSKY, R. **Effective Coding with VHDL**: Principles and best practice. 1. ed. Cambridge: The MIT Press, 2016.

LESZKO, Rafal. **Continuous Delivery with Docker and Jenkins**: Create secure applications by building complete CI/CD pipelines. 3. ed. Birmingham: Packt Publishing, 2022.

MORDORINTELLIGENCE. **Tamanho do mercado de matriz de portão programável em campo e análise de ações – Tendências e previsões de crescimento (2024 – 2029)**. 2023. Disponível em: https://www.mordorintelligence.com/pt/industry-reports/field-programmable-gate-array-fpga-market.

NELSON, Yadiel. **Electronic Design Automation**. 1. ed. Delhi, Índia: The English Press, 2014.

OZTEMEL. Literature review of Industry 4.0 and related technologies. **Springer**, 2018.

RAJ, A. Arockia Bazil. **FPGA-Based Embedded System Developer's Guide**. 1. ed. New York, USA: Taylor Francis, CRC Press, 2018.

SASS, Ron; SCHMIDT, Andrew. **Emdedded Systems Design with Platform FPGAs**: Principles and Practices. 1. ed. Massachusetts, USA: Morgan Kaufmann Publishers, 2010.

TECHBEACON. The Importance Of Regular Software Updates And Patches. **Findmyservicecenter**, 2017.

VAHID, F. **Digital Design with RTL Design, VHDL, and Verilog**. 2. ed. Danvers: John Wiley Sons, inc., 2011.

WOODS, Roger; MCALLISTER, John; LIGHTBODY, Gaye; YI, Ying. **FPGA-based Implementation of Signal Processing Systems**. 2. ed. New Jersey, USA: John Willey Sons, 2017.