Universidade de Brasília

Faculdade de Tecnologia

Departamento de Engenharia Elétrica

Desenvolvimento de uma Infraestrutura em Nuvem com Aceleradores FPGA para Otimização de Aplicações em Internet das Coisas.

Matheus Barbosa de Miranda

TRABALHO DE CONCLUSÃO DE CURSO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2025

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Desenvolvimento de uma Infraestrutura em Nuvem com Aceleradores FPGA para Otimização de Aplicações em Internet das Coisas.

Matheus Barbosa de Miranda

Trabalho de Conclusão de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. Jones Yudi Mori Alves da Silva

Brasília
2025

FICHA CATALOGRÁFICA

de Miranda, Matheus Barbosa.

Desenvolvimento de uma Infraestrutura em Nuvem com Aceleradores FPGA para Otimização de Aplicações em Internet das Coisas. / Matheus Barbosa de Miranda; orientador Jones Yudi Mori Alves da Silva. –– Brasília, 2025.

42 p.

Trabalho de Conclusão de Curso (Engenharia de Controle e Automação) –– Universidade de Brasília, 2025.

1. Sistemas Digitais. 2. FPGA. 3. IoT. 4. HLS. 5. Aceleração. I. da Silva, Jones Yudi Mori Alves, orient. II. Título.

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Desenvolvimento de uma Infraestrutura em Nuvem com Aceleradores FPGA para Otimização de Aplicações em Internet das Coisas.

Matheus Barbosa de Miranda

Trabalho de Conclusão de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Trabalho aprovado. Brasília, 21 de Fevereiro de 2025:

Prof. Dr. Jones Yudi Mori Alves da Silva,
UnB/FT/ENM
Orientador

Prof. Dr. Daniel Mauricio Munoz Arboleda,
UnB/FGA/FCTE
Examinador interno

Prof. Dr. Janier Arias García,
UFMG
Examinador interno

Este trabalho é dedicado a Robson Soares de Miranda, que foi um grande marido, pai, avô, bisavô e trabalhador.

Dedico também a Renan Costa Tavares, um grande colega de curso que fez tanto pelo curso de Engenharia Mecatrônica, presidindo o Centro Acadêmico do mesmo. Uma pessoa gentil, humilde e alegre que deixou sua esposa e seus amigos tão cedo.

Agradecimentos

Agradeço ao meu orientador, Dr. Jones Yudi Mori Alves da Silva, pela paciência e humildade em me orientar em um assunto que amo tanto, mas que traz tantas dificuldades. Obrigado por me oferecer um estágio quando eu estava à beira de desistir de tudo, apesar de o senhor não saber. Obrigado por tudo, professor. Foi uma honra trabalhar com o senhor.

Agradeço à minha família, meus pais Ricardo Almeida de Miranda e minha mãe Marúcia Valença Barbosa de Miranda, pelo apoio incondicional em todas as etapas difíceis que passei, especialmente nesses últimos anos, além do amor e carinho demonstrados sempre. Amo vocês, obrigado.

Agradeço especialmente à minha noiva, Ana Luiza Espinoza Resende, que esteve comigo em todos os momentos que precisei e que me incentiva a ser um profissional e uma pessoa melhor todo dia. Obrigado pelo amor, pelo carinho, pelas risadas e pelas broncas (necessárias). Eu te amo.

Agradeço ao doutor Rodrigo Elias Neme Gebrim, que me mostrou que a vida não era o que pensava ser 10 anos atrás. O Dr. Rodrigo explicou que a vida pode ser bonita, alegre e cheia de felicidades, provando para mim que um garoto jovem, cético e triste poderia vir a vencer o que era uma doença e viver uma vida alegre, sorridente e contente. Com o senhor, o mundo deixou de ser cinza e passou a ter cores vivas. Muito obrigado, doutor, o senhor mudou minha vida para sempre.

Agradeço à Ana Paula Moraes, que me fez ver que uma conversa em um espaço seguro muda sua vida por completo e é extremamente necessário. Obrigado, Ana, espero continuar te vendo por muito tempo.

Finalmente, agradeço às amizades que fiz nessa caminhada. Amigos, sem vocês seria impossível terminar este curso, no qual tantas vezes pensei em desistir. Gabriel Bertone, João Vitor Loureiro, Guilherme Zapponi, Mateus Paula Leite Paz, Alexandre Bernardi Peres, Arthur Ribeiro, Clarice Machado Menin, Giovana Morelo Gagno, Hércules Nunes Jr. e a todos os outros que sabem que merecem estar aqui. Vocês foram e são meus pilares para continuar essa luta. Muito obrigado.

“Ando devagar
Porque já tive pressa...”
(Almir Sater e Renato Teixeira)

Resumo

Este projeto envolve o desenvolvimento de um sistema de computação em nuvem otimizado com o uso de aceleradores baseados em FPGA para aplicações em IoT . O objetivo do projeto é criar uma infraestrutura eficiente que suporte a execução de algoritmos intensivos de processamento de dados, comuns em IoT, com baixa latência e alto desempenho, aproveitando a flexibilidade e a capacidade de paralelismo das FPGAs. O sistema em nuvem desenvolvido será responsável por gerenciar, processar e analisar grandes volumes de dados gerados por dispositivos IoT, enquanto as FPGAs serão utilizadas para acelerar tarefas específicas, como criptografia, filtragem de sinais, processamento de imagens e compressão de dados. Esse projeto aborda desafios críticos, como a integração de hardware e software, a comunicação eficiente entre dispositivos IoT e a nuvem, além de explorar o uso de plataformas em FPGA para melhorar o desempenho e a escalabilidade de sistemas IoT em cenários reais, como cidades inteligentes e automação industrial.

Palavras-chave: Sistemas Digitais; FPGA; IoT; HLS; Aceleração.

Abstract

This project involves the development of a computing system in optimized cloud with the use of accelerators based on FPGA for IoT applications. The project's goal is to create an efficient infrastructure that supports the execution of intense data processing algorithms (common in IoT) with low latency and high performance, making use of the flexibility and parallelism potential of FPGAs. The cloud system developed will be responsible for management, processing e analyzing big volume of data obtained by IoT devices, while the FPGAs will be used to accelerate specific tasks, such as cryptography, signal filtering, image processing and data compressioning. This project approaches critical challenges, such as hardware-software integration, efficient communication between IoT devices and the cloud, in addition to exploring the use of FPGA platforms to get a better performance and scalability of IoT systems in real scenarios, like smart cities and industrial automation.

Keywords: Digital Systems; FPGA; IoT; HLS; Acceleration.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

FPGA	Field-Programmable Gate Array
IoT	Internet of things
LoRa	Long Range
PCI	Peripheral Component Interconnect
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XRT	Xilinx Run Time

Sumário

1 Introdução

É cada vez mais presente em projetos a existência de dispositivos com um *hardware* cada vez menor e, portanto, com maior capacidade de processamento. O termo Internet of Things foi relatado pela primeira vez em 1999 por Kevin Ashton (Gokhale; Bhat, 2018)

A ascensão da Internet das Coisas (IoT) tem impulsionado uma crescente demanda por sistemas computacionais adaptativos, capazes de lidar com a vasta quantidade de dados gerados por dispositivos inteligentes distribuídos. A IoT conecta sensores, atuadores e dispositivos de computação embarcada a redes de comunicação, permitindo a automação e otimização de processos em diversas áreas, como saúde, indústria, transporte e cidades inteligentes. No entanto, essa explosão de dados exige infraestruturas que não apenas garantam alta capacidade de processamento e armazenamento, mas que também ofereçam flexibilidade para se adaptar a diferentes cargas de trabalho. Nesse contexto, arquiteturas computacionais que integram aceleração por FPGAs e armazenamento computacional com SmartSSDs tornam-se estratégicas, pois combinam alto desempenho, baixo consumo de energia e eficiência na movimentação e no processamento de dados diretamente no local onde são gerados.

1.1 Sobre a Internet das Coisas

A Internet das Coisas pode ser considerada um pico de uma revolução sobre tecnologia ou pode ser considerada uma tecnologia disruptiva? Na verdade, vemos que a IoT não é disruptiva, e sim um paradigma da computação. Utilizando da Internet e seus diversos protocolos, como Zigbee, TCP, UDP ou LoRa para conectar diversos sensores a um servidor, seja em nuvem ou físico, para processamento dos mesmos.

Entre as vantagens da aplicação de IoT estão: eficiência energética, possibilidade de uso de sensores de baixo custo, maior possibilidade de personalização da solução para a conveniência do trabalho, monitoramento e acessibilidade remota e entre outros.

Nas últimas décadas, os sistemas computacionais evoluíram de grandes máquinas para dispositivos embarcados, impulsionados pela tecnologia VLSI e a Lei de Moore. Novos paradigmas, como Sistemas Ciber-Físicos e Internet das Coisas (IoT), ampliaram o poder computacional, levando à Computação Ubíqua e Pervasiva. Sensores Inteligentes surgiram para processar e atuar sobre o ambiente, reduzindo a dependência de servidores remotos. A fusão de dados de múltiplos sensores é essencial para compreender fenômenos ambientais, mas as arquiteturas atuais não suportam a crescente demanda por processamento avançado, exigindo novos modelos computacionais.

Os sistemas distribuídos e concorrentes, como os presentes na Internet das Coisas (IdC), apresentam diversos desafios tecnológicos devido à complexidade da execução simultânea de tarefas em múltiplos dispositivos.

Existem no mercado diversas arquiteturas de sistemas de Internet das Coisas, porém, pode-se identificar a presença de três elementos-chave, listados a seguir:

- Dispositivos de Borda (Edge Devices): são os elementos responsáveis pelo sensoriamento e atuação sobre o ambiente ou aplicação. Sua implementação é realizada com dispositivos de baixo poder computacional.
- Comunicação: equipamentos e sistemas de transmissão de dados, trazendo conectividade aos dispositivos de borda e fazendo uma ponte entre estes e os servidores na nuvem.
- Computação em Nuvem: este elemento é composto por servidores de processamento, armazenamento e visualização de dados e informações.

Neste trabalho, o foco principal está na Computação em Nuvem. Dentre as técnicas computacionais mais modernas, a virtualização permite que os recursos da nuvem sejam distribuídos de forma flexível, possibilitando diferentes modelos de serviço em que dois se destacam:

- Software-as-a-Service (SaaS): Fornece aplicativos completos como serviço, acessíveis pela internet, sem necessidade de instalação local. Exemplos incluem Google Docs, Dropbox e plataformas de IoT gerenciadas.
- Hardware-as-a-Service (HaaS): Oferece infraestrutura física (servidores, armazenamento, FPGAs, GPUs) sob demanda, permitindo que empresas utilizem hardware especializado sem precisar adquiri-lo diretamente. Essa abordagem é essencial para aplicações que exigem computação de alto desempenho (HPC).

1.2 Sobre Computação Reconfigurável

Conceituar o que é Computação Reconfigurável e como técnicas de co-projeto hardware/software (Hardware/Software Co-Design) podem trazer benefícios para prover sistemas computacionais equilibrados/otimizados/customizados para cada aplicação.

A utilização de FPGA para este projeto foi justamente para obter uma solução flexível em termos de *hardware*, como baixa latência para comunicação com o servidor. A escolha de um *hardware* programável leva a uma otimização do produto, pois uma CPU ou uma GPU já têm suas instruções e métodos de endereçamento de memória muito bem definidos.

A computação reconfigurável advém de técnicas de co-projeto hardware/software em que aceleradores de hardware são integrados dentro das chamadas de software de modo transparente ao usuário final. Utilizando módulos de hardware customizáveis em arqui-

teturas específicas de acordo com a aplicação, alcançam-se desempenhos superiores às implementações puramente em software.

Nesse contexto os FPGAs(Field-Programmable Gate Arrays), componentes eletrônicos reconfiguráveis com capacidades computacionais variadas, possibilitam a exploração do espaço de projeto de acordo com os requisitos específicos de cada aplicação. No caso deste projeto, uma arquitetura modular reconfigurável para soluções de Internet das Coisas é proposta, conforme a Figura **??**.

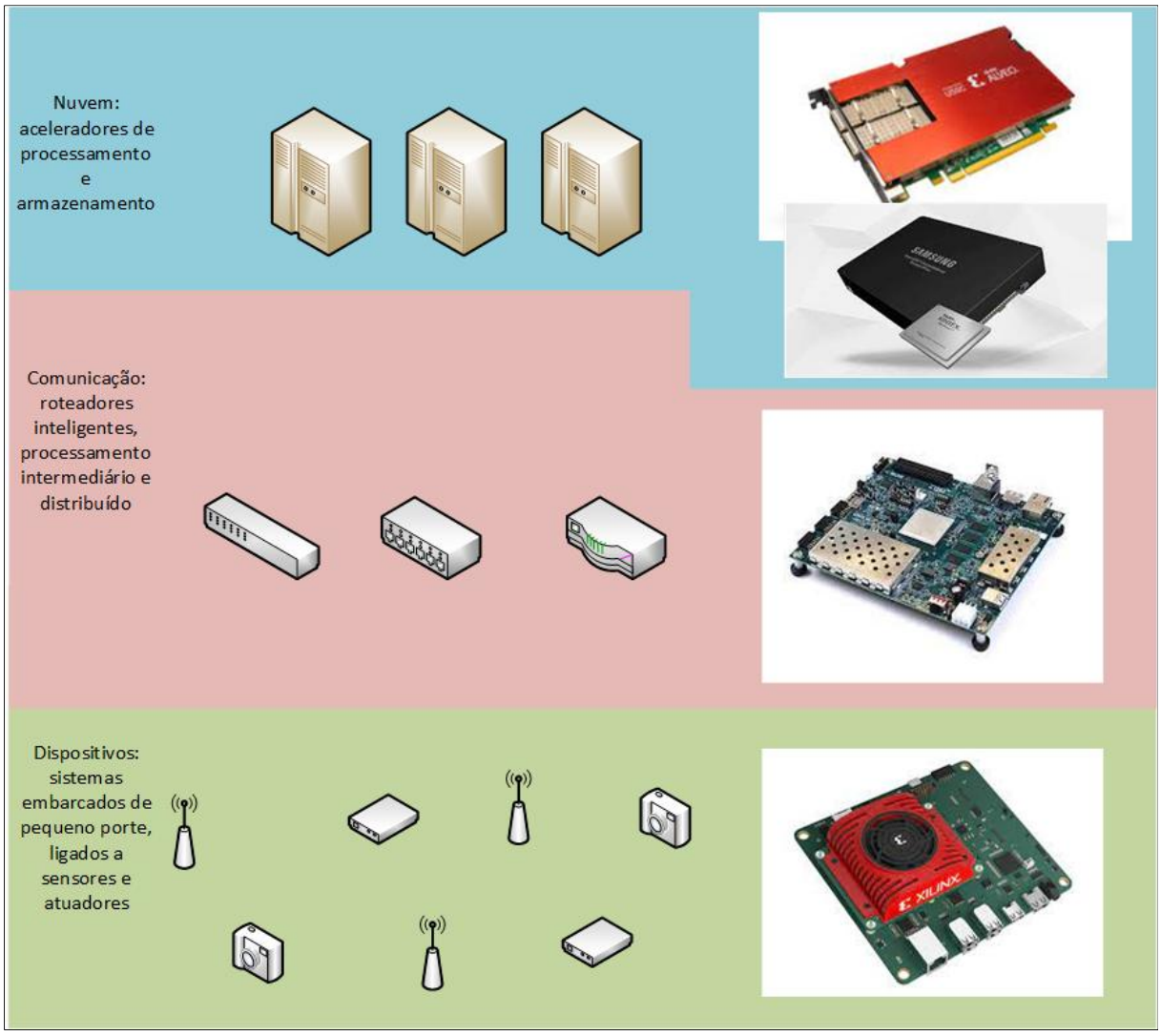


Figura 1.1 – Arquitetura reconfigurável para Internet das Coisas.

Na camada de dispositivos, estão presentes sistemas embarcados ligados a sensores e atuadores específicos de cada aplicação. Nesta camada, FPGAs de baixa densidade são utilizados para a implementação de algoritmos em geral de pouca complexidade.

Na camada de comunicação, dispositivos de médio porte ligados a antenas e rodeadores são utilizados. Os sistemas reconfiguráveis desta camada são FPGAs de densidade média.

Finalmente, na camada de nuvem temos os dispositivos de alta densidade contemplando aceleradores e armazenamento computacional (computational storage). Nesta camada, algoritmos de alta complexidade e com massivas quantidades de dados são implementados.

1.3 Sobre Nuvem Reconfigurável

A utilização de FPGAs nos sistemas de nuvem pode ajudar a superar as limitações das arquiteturas computacionais convencionais ao oferecer processamento acelerado, eficiência energética, redução da latência de movimentação de dados, entre outras vantagens.

Processamento Paralelo e Customizável

Os FPGAs permitem a exploração de paralelismo e pipelining, podendo ser reconfigurados para otimizar o processamento de cargas de trabnalho específicas, como aprendizado de máquina, análise de grandes volumes de dados e segurança cibernética. Isso se deve à possibilidade de descrição de circuitos eletrônicos extremamente refinados e dedicados aos algoritmos desejados pela aplicação.

Armazenamento Computacional

Os SmartSSDs combinam FPGAs e armazenamento em SSDs em um único dispositivo, permitindo que o processamento seja realizado diretamente onde os dados estão armazenados. Com isso, reduz-se significativamente a movimentação de dados entre CPU, RAM e armazenamento, minimizando a latência e o consumo de energia. Aplicações como Big Data e Internet das Coisas, onde há grandes volumes de dados a serem processador, podem se beneficiar sobremaneira das características dos SmartSSDs.

Eficiência Energética

A execução de algoritmos diretamente no FPGA reduz também a frequência de operação dos dispositivos, resultando em menor consumo de energia que o processamento em CPUs ou GPUs, sendo uma solução mais sustentável para modernos datacenters.

Redução da sobrecarga computacional

Em geral, a CPU é responsável por gerenciar diversas tarefas simultaneamente, o que gera gargalos de processamento quando do acesso à memória. A integração de FPGAs e SmartSSDs aos servidores resulta em liberação de recursos da CPU para tarefas essenciais de gerenciamento e orquestração, melhorando a escalabilidade do sistema como um todo e possibilitando grandes otimizações de desempenho.

A combinação de FPGAs e SmartSSDs representa um salto na eficiência computacional, resolvendo os desafios de desempenho, consumo energético e movimentação de dados que limitam as arquiteturas tradicionais. Essa abordagem possibilita sistemas mais ágeis, escaláveis e otimizados para cargas de trabalho modernas, tornando-se uma solução essencial

para a próxima geração de infraestrutura de computação em nuvem e IoT.

1.4 Objetivos

1.4.1 Objetivo Principal

Este trabalho tem como objetivo a implementação de um acelerador de processamento em hardware reconfigurável em um contexto de *Hardware-as-a-Service*.

1.4.2 Objetivos Específicos

- Instalar e configurar a placa aceleradora no servidor.
- Selecionar aplicações no contexto de Internet das Coisas.
- Implementar em Software e Hardware uma ou mais aplicações.
- Analisar criticamente o desempenho alcançado.

1.5 Organização do Trabalho

O trabalho está dividido em:

- Capítulo 2: Uma revisão bibliográfica sobre temas semelhantes ao presente trabalho. Temas como Computação em nuvem, HaaS e Nuvem reconfigurável são mostrados.
- Capítulo 3: É mostrado como a implementação do Datacenter foi feita. São descritos todos os componentes, a configuração das máquinas no servidor remoto, bem como todo o procedimento de instalação dos firmwares, softwares e pacotes necessários. Também é mostrado como validar as instalações no servidor.
- Capítulo 4: Descreve o experimento como um todo, a utilização do software utilizado de forma remota e como montar o experimento. Descreve e compara os diferentes resultados obtidos.
- Conclusão e trabalhos futuros.

2 Revisão Bibliográfica

2.1 Computação em Nuvem

O artigo intitulado "Models of Computing as a Service and IoT: an analysis of the current scenario with applications using LPWAN"(Bezerra Fernando Luiz Koch, 2020) explora a integração entre modelos de computação em nuvem e a Internet das Coisas (IoT), especialmente no contexto de redes de longa distância e baixa potência (LPWAN).

O estudo destaca que o paradigma da computação em nuvem transformou a implementação de soluções industriais por meio da comoditização de infraestruturas de TI compartilhadas. Com o advento da IoT em larga escala, surgem novos desafios que exigem configurações flexíveis, onde os recursos são distribuídos mais próximos das fontes de dados.

2.2 Hardware-as-a-Service

Um artigo de 10 anos atrás que tem uma abordagem muito parecida com o presente trabalho é o "RC3E: Provision and Management of Reconfigurable Hardware Accelerators in a Cloud Environment"(Knodel, 2015).

O RC3E (Reconfigurable Common Cloud Computing Environment) é uma arquitetura desenvolvida para integrar aceleradores de hardware reconfiguráveis, especificamente FPGAs, em ambientes de computação em nuvem. Esta abordagem visa fornecer recursos de hardware reconfigurável como serviço, permitindo que múltiplos usuários executem seus projetos virtuais em um único dispositivo físico.

Já em outro trabalho, "Kernel-as-a-Service: A Serverless Interface to GPUs"(Pemberton Anton Zabreyko; Gonzalez, 2022) há a apresentação do KaaS, uma interface que trata GPUs como recursos de primeira classe em plataformas serverless. O KaaS permite que kernels de GPU sejam invocados como funções serverless, gerenciando a memória da GPU e escalonando tarefas em um pool de GPUs disponíveis. Essa abordagem visa compartilhar eficientemente recursos de GPU em ambientes multitenant na nuvem, melhorando a utilização e o desempenho. Os resultados mostram que o KaaS pode aumentar em até 50 vezes a taxa de transferência e reduzir em até 16 vezes a latência quando os recursos de GPU estão concorridos.

2.3 Nuvem Reconfigurável

O artigo "A Survey on Reconfigurable Accelerators for Cloud Computing"(Kachris, 2020) apresenta uma análise abrangente sobre o uso de aceleradores reconfiguráveis, especialmente FPGAs, em ambientes de computação em nuvem.

O artigo examina as cargas de trabalho típicas em data centers modernos, destacando a necessidade de soluções que ofereçam alto desempenho e eficiência energética para lidar com o crescente volume de dados e processamento. Os autores fornecem uma comparação detalhada dos diferentes esquemas de aceleração, considerando métricas como ganho de desempenho, eficiência energética e consumo de energia. Essa análise auxilia na compreensão das vantagens e desafios associados à adoção de FPGAs em ambientes de nuvem.

3 Implementação do Hardware (Datacenter)

3.1 Descrição do data center

O servidor foi planejado como na figura 3.1, utilizando duas máquinas iguais, cada uma com uma placa FPGA da Alveo e um SmartSSD cada. Porém, devido a incompatibilidades encontradas (kernel, drivers de dispositivo e entre outros), a escolha feita foi pela imagem 3.2, na qual em uma máquina há duas placas da Alveo e, na outra, quatro SmartSSDs.

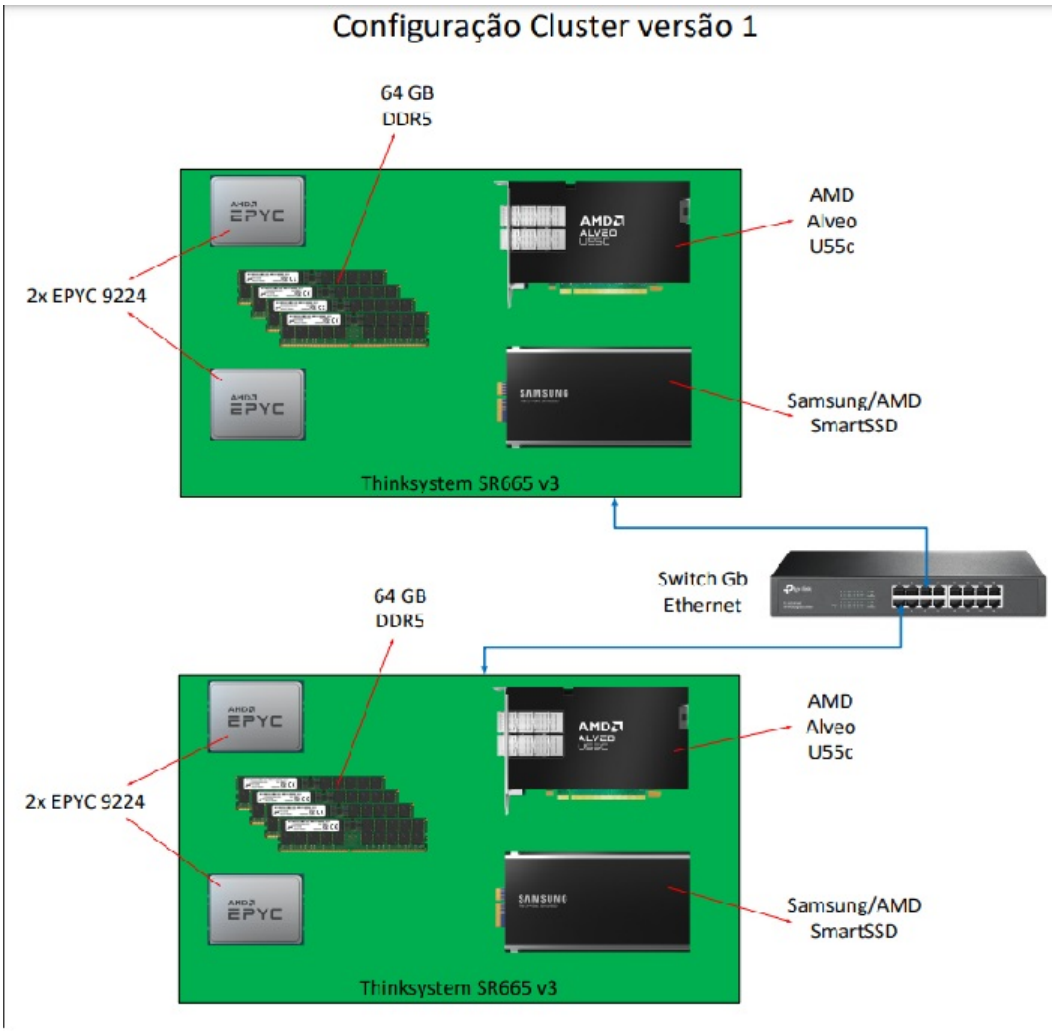


Figura 3.1 – Primeira opção desejada das duas máquinas do servidor.

É importante ressaltar que, devido a dificuldades de acessar e programar os SmartSSDs via servidor, o presente trabalho utiliza apenas a máquina com as placas FPGAS da Alveo.

A placa FPGA escolhida para o trabalho é a ALVEO® U55C (figura 3.3), da AMD®

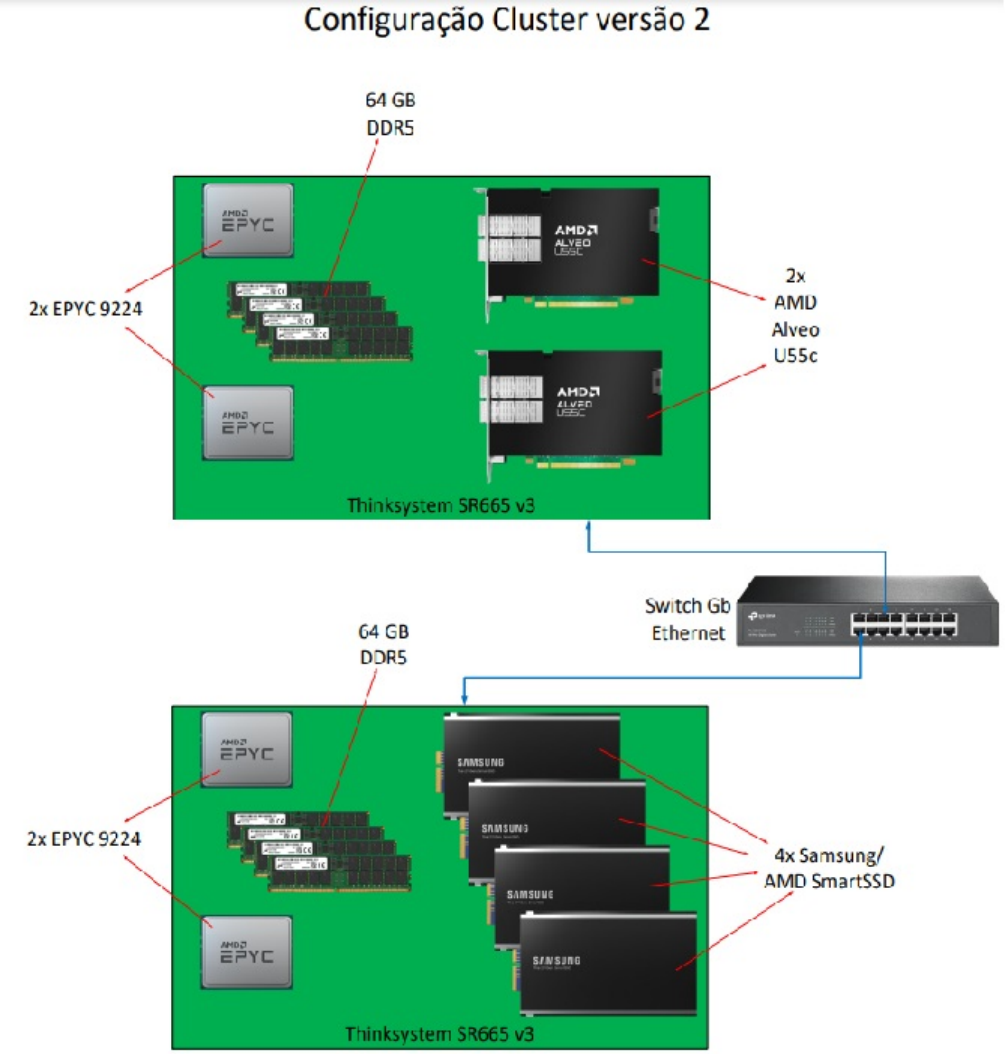


Figura 3.2 – Esquemático atual das duas máquinas do servidor.

A própria AMD dispõe de um tutorial de como instalar a placa, porém, claro, há muitos desafios para instalá-la da maneira que o experimento exige.

O *data center accelerator* da XILINX® ALVEO® U55C é um PCI EXPRESS® (PCI-e) que converse com a versão 3.0 (mínimo) com um conector x16. A especificação mínima de memória RAM para apenas instalação é de 16GB, porém o instalado foi de 64GB, que é o mínimo especificado para desenvolvimento. As demais especificações para o data center podem ser encontradas no próprio guia da Xilinx (a)[1].

[1] Disponível em: https://docs.amd.com/v/u/en-US/ug1468-alveo-u55c

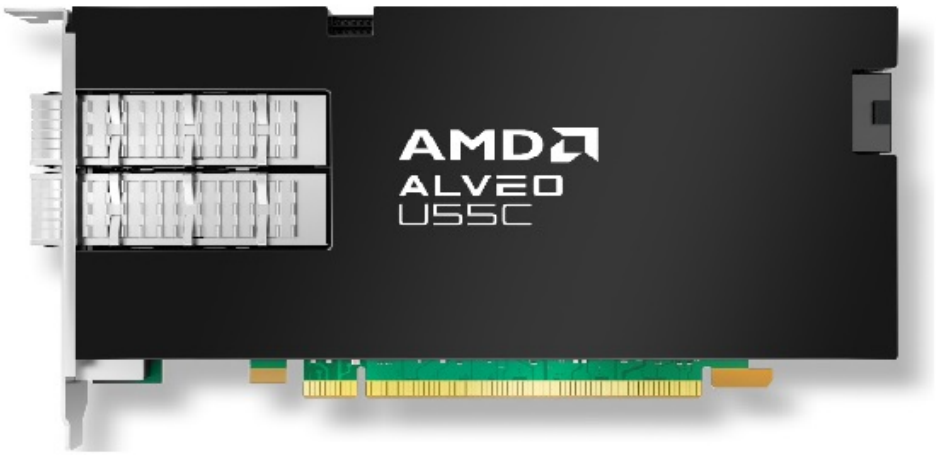
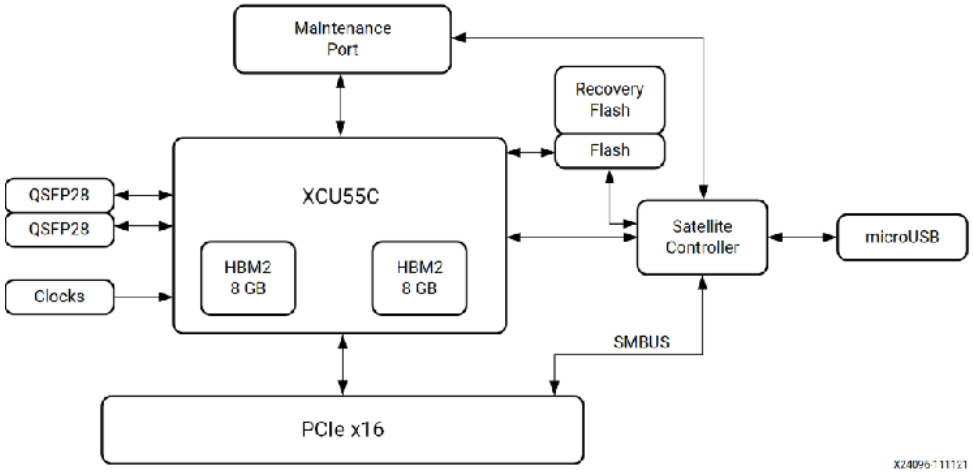


Figura 3.3 – Placa ALVEO u55c.



Fonte: Xilinx (b)

Figura 3.4 – Diagrama de blocos com uma ALVEO u55c aceleradora.

A tabela 3.1 lista os equipamentos utilizados no servidor, localizado no laboratório do GRACO - na Faculdade de Tecnologia -, para o trabalho em questão.

3.2 Preparação do ambiente remoto

3.2.1 Preparação do Hardware

De acordo com o Xilinx (a), deve-se desligar o *host* e tirar da tomada. Em seguida, abrir o computador e conectar o Alveo na placa-mãe no conector PCIe x16. Então, conecte o cabo de potência com o Alveo, certificando-se que está mecanicamente fixado com um clique.

Reconecte o *casing* do computador e conecte o cabo de potência, ligando o computador.

Tabela 3.1 – Elementos do Servidor

Recurso	Qtd
ThinkSystem V3 2U 24x2.5″Chassis	1
Data Center Environment 25 Degree Celsius	1
ThinkSystem AMD EPYC 9224 24C 200W 2.5GHz Processor	2
ThinkSystem SR665 V3 2U High Performance Heatsink	2
ThinkSystem 16GB TruDDR5 4800MHz (1Rx8) RDIMM-A	4
ThinkSystem 4-Port PCIe Gen4 NVMe Retimer Adapter	1
ThinkSystem 2U/4U 8x2.5″NVMe Backplane	3
ThinkSystem M.2 NVMe 2-Bay RAID Enablement Kit	1
M.2 NVMe	1
ThinkSystem M.2 7450 PRO 960GB Read Intensive NVMe PCIe 4.0 x4 NHS SSD	2
ThinkSystem Intel X710-T2L 10GBASE-T 2-port OCP Ethernet Adapter	1
ThinkSystem V3 2U x16/x8/x8 PCIe Gen4 Riser1 or 2	2
ThinkSystem 2600W 230V Titanium Hot-Swap Gen2 Power Supply	2
ThinkSystem 2U V3 Performance Fan Module	6
ThinkSystem Toolless Slide Rail Kit v2	1
ThinkSystem 2U V3 EIA Latch Standard	1
TPM 2.0	1

Fonte: Elaborada pelo autor

3.2.2 Preparação de *software*

Como o experimento conta com um acesso remoto ao servidor que fica no GRACO, entrar em um *software* de VPN para conectar-se ao servidor. Assim, deve-se executar os comandos para verificar a instalação da placa.

Execute o código 3.1 e verifique o retorno como no código 3.2.

Código 3.1 – Comando para verificar instalação

```
1 $ sudo lspci -vd 10ee:
```

O terminal deve retornar o código 3.2.

Caso haja algum erro, refira-se a Xilinx, AMD (2021).

Código 3.2 – Resposta do terminal

```
86:00.0 Processing accelerators: Xilinx Corporation Device 505c
Subsystem: Xilinx Corporation Device 000e
Physical Slot: 5
Flags: bus master, fast devsel, latency 0, NUMA node 1
Memory at d7ff2000000 (64-bit, prefetchable) [size=32M]
Memory at d7ff4040000 (64-bit, prefetchable) [size=256K]
Capabilities: [40] Power Management version 3
Capabilities: [60] MSI-X: Enable+ Count=32 MaskedCapabilities: [70]
   Express Endpoint, MSI 00
Capabilities: [100] Advanced Error Reporting
Capabilities: [1c0] #19
Capabilities: [e00] Access Control Services
Capabilities: [e10] #15
Capabilities: [e80] Vendor Specific Information: ID=0020 Rev=0 Len=010
Kernel driver in use: xclmgmt
Kernel modules: xclmgmt
```

3.2.3 *Deployment Installing - Ubuntu*

A plataforma de *deployment* entrega o *firmware* básico para rodar programas pré compiladas, não sendo usadas para criar novas aplicações (que é parte do desenvolvimento de *software*, que não é necessário para rodar aplicações, mas nesse presente projeto irá ser usado para criar novos programas.

Para instalar o XRT, rode o código 3.3 para instalar os cabeçalhos e os pacotes de desenvolvimento do *kernel*

Código 3.3 – Instalação XRT

```
$ sudo apt install ./xrt_<version>.deb
```

Em seguida, descompacte o tar.gz em um diretório à sua escolha, notando que o mesmo não deve ter nenhum outro arquivo. Execute então o comando 3.4

Código 3.4 – Instalação pacotes XRT

```
sudo apt install ./*.deb
```

Faça em seguida um *flash* na plataforma de *firmware* da placa. Após a instalação, o código 3.5 deve aparecer no terminal.

Para fazer o *flash*, execute o comando 3.6.

Código 3.5 – Instalação pacotes XRT

```
Partition package installed successfully.
Please flash card manually by running below command:
sudo /opt/xilinx/xrt/bin/xbmgmt program --base --device <bdf> --image
xilinx_u55c_gen3x16_xdma_base_2
```

Código 3.6 – Instalação pacotes XRT

```
sudo /opt/xilinx/xrt/bin/xbmgmt program --base --device <management bdf>
--image xilinx_u55c_gen3x16_xdma_base_2
```

Esse procedimento pode levar vários minutos. Em obtendo sucesso, a mensagem 3.7 deve aparecer em seu terminal.

Código 3.7 – Instalação pacotes XRT

```
Device flashed successfully.
**************************************************
Cold reboot machine to load the new image on device(s).
**************************************************
```

Em seguida, atualize o *Storage Center firmware* da placa com o código 3.8. Se o procedimento for executado com sucesso, aparecerá uma mensagem como no código 3.9.

Código 3.8 – SC firmware update

```
$ sudo /opt/xilinx/xrt/bin/xbmgmt program --base --device <management
BDF> --image xilinx_u55c_gen3x16_xdma_base_2
```

Código 3.9 – SC firmware update successfull

```
Report
 [0000:af:00.0] : Successfully flashed
1 device(s) flashed successfully.
****************************************************************
Warm reboot is required to recognize new SC image on the device.
****************************************************************
```

3.2.4 Validação da Placa

No Bash, utilize o comando apresentado no código 3.10. O resultado esperado no terminal, diferentemente de quando executado o código 3.1, deve ser como no código 3.11.

Código 3.10 – Comandos para Teste

```
$ source /opt/xilinx/xrt/setup.sh
$ sudo lspci -vd 10ee:
```

Código 3.11 – Resultado dos testes

```
5e:00.0 Processing accelerators: Xilinx Corporation Device 505c
 Subsystem: Xilinx Corporation Device 000e
 Flags: bus master, fast devsel, latency 0, NUMA node 0
 Memory at 383ff2000000 (64-bit, prefetchable) [size=32M]
 Memory at 383ff4040000 (64-bit, prefetchable) [size=256K]
 Capabilities: [40] Power Management version 3
 Capabilities: [60] MSI-X: Enable+ Count=32 Masked-
 Capabilities: [70] Express Endpoint, MSI 00
 Capabilities: [100] Advanced Error Reporting
 Capabilities: [1c0] Secondary PCI Express
 Capabilities: [e00] Access Control Services
 Capabilities: [e10] Resizable BAR <?>
 Capabilities: [e80] Vendor Specific Information: ID=0020 Rev=0
Len=010 <?>
 Kernel driver in use: xclmgmt
 Kernel modules: xclmgmt
5e:00.1 Processing accelerators: Xilinx Corporation Device 505d
 Subsystem: Xilinx Corporation Device 000e
 Flags: bus master, fast devsel, latency 0, IRQ 303, NUMA node 0
 Memory at 383ff0000000 (64-bit, prefetchable) [size=32M]
 Memory at 383ff4000000 (64-bit, prefetchable) [size=256K]
 Memory at 383fe0000000 (64-bit, prefetchable) [size=256M]
 Capabilities: [40] Power Management version 3
 Capabilities: [60] MSI-X: Enable+ Count=32 Masked-
 Capabilities: [70] Express Endpoint, MSI 00
 Capabilities: [100] Advanced Error Reporting
 Capabilities: [e00] Access Control Services
 Capabilities: [e10] Resizable BAR <?>
 Capabilities: [e80] Vendor Specific Information: ID=0020 Rev=0
Len=010 <?>
 Kernel driver in use: xocl
 Kernel modules: xocl
```

Para confirmar a instalação do *firmware*, as entradas da Plataforma e SC devem ser as mesmas. Caso encontre problemas, deve-se referir a Card BDF Values () para, além de resolver problemas de versionamento da Plataforma e do SC, obter informações sobre o valor BDF da placa. O comando do código 3.12 deve gerar algo parecido com o código 3.13.

Código 3.12 – Examinando as Versões

```
$ sudo /opt/xilinx/xrt/bin/xbmgmt examine --device <management BDF>
```

Código 3.13 – Resposta

```
----------------------------------------------------
1/1 [0000:5e:00.0] : xilinx_u55c_gen3x16_xdma_base_2
----------------------------------------------------
Flash properties
 Type : spi
 Serial Number : XFL1GKKP3E2D
Device properties
 Type : u55c
 Name : ALVEO U55C
 Config Mode : 7
 Max Power : 225W
Flashable partitions running on FPGA
 Platform : xilinx_u55c_gen3x16_xdma_base_2
 SC Version : 7.1.14
 Platform UUID : FCC68C40-94CC-3A1E-9A6C-A76BCA494718
 Interface UUID : 738F93F1-1FCF-2AAB-F1AF-DFEB9C8992C7
Flashable partitions installed in system
 Platform : xilinx_u55c_gen3x16_xdma_base_2
 SC Version : 7.1.14
 Platform UUID : FCC68C40-94CC-3A1E-9A6C-A76BCA494718
```

Note que, no exemplo, a Plataforma, descrita por xilinx_u55c_gen3x16_xdma_base_2 possui a mesma versão SC que a mesma nas partições *flash* instaladas no sistema.

Para validação da placa, utiliza-se o código 3.14 -usando o BDF desejado-, que deve responder no terminal na forma similar ao código 3.15

Código 3.14 – Validando a Placa

```
$ /opt/xilinx/xrt/bin/xbutil validate --device <user BDF>
```

Código 3.15 – Resposta da Validação

```
Starting validation for 1 devices
Validate Device : [0000:5e:00.1]
 Platform : xilinx_u55c_gen3x16_xdma_base_2
 SC Version : 7.1.14
 Platform ID : FCC68C40-94CC-3A1E-9A6C-A76BCA494718
-------------------------------------------------------------------------------
---
Test 1 [0000:5e:00.1] : PCIE link
 Test Status : [PASSED]
-------------------------------------------------------------------------------
---
Test 2 [0000:5e:00.1] : SC version
 Test Status : [PASSED]
-------------------------------------------------------------------------------
---
Test 3 [0000:5e:00.1] : Verify kernel
 Test Status : [PASSED]
-------------------------------------------------------------------------------
```

```
---
Test 4 [0000:5e:00.1] : DMA
 Details : Host -> PCIe -> FPGA write bandwidth = 11937.2
MB/s
 Host <- PCIe <- FPGA read bandwidth = 12076.0
MB/s
 Test Status : [PASSED]
--------------------------------------------------------------------------
---
Test 5 [0000:5e:00.1] : iops
 Details : IOPS: 462327 (verify)
 Test Status : [PASSED]
--------------------------------------------------------------------------
---
Test 6 [0000:5e:00.1] : Bandwidth kernel
 Details : Throughput (Type: HBM) (Bank count: 1) :
12098.7MB/s
 Test Status : [PASSED]
--------------------------------------------------------------------------
---
Test 7 [0000:5e:00.1] : vcu
Validation completed. Please run the command '--verbose' option for more
details
```

Pode-se ver, pelo código 3.15, que todos os testes passaram.

4 Análise de Resultados

Dentro os programas disponíveis para desenvolvimento e síntese, como o Vivado, o INTEL® HLS Compiler e outros *open source*, como o *LegUp HLS*, o escolhido para O desenvolvimento do código para colocar na placa foi feito utilizando o *software* VITIS® 2024.1, ferramenta que engloba o próprio Vivado. O Vitis é da Xilinx.

4.1 Utilização do Vitis de Forma Remota

Para utilizar o servidor descrito na tabela 3.1, foi utilizado o protocolo SSH para que o envio de mensagens seja feito via TCP. Um usuário foi criado para o aluno, que utilizando o *software* MOBAXTERM®, um IP foi criado em uma porta protegido por senha. O sistema operacional é Ubuntu 20 e, a princípio, não tem interface gráfica; porém há instalado o Vitis, google-chrome e firefox, o que abre na tela do *host*.

Para abrir o Vitis, há a necessidade de rodar o código 4.1 no terminal

Código 4.1 – Preparação para Vitis

```
source <Vitis_install_path>/Vitis/2024.2/settings64.sh
source <xrt_install_path>/setup.sh
export PLATFORM_REPO_PATHS=/opt/xilinx/platforms
Vitis&
```

4.2 Criando *Workspace*

Na pasta contendo os arquivos de código-fonte, crie um diretório e, logo depois, um *workspace* na pasta.

Código 4.2 – Workspace

```
mkdir <path_to_reference_files/workspace
vitis -w workspace
```

O workspace é onde serão armazenados códigos-fonte e arquivos para compilar o projeto.

4.3 DCT em HLS sem Otimizações.

No presente experimento, foi utilizado um algoritmo de Transformada Discreta do Cosseno (DCT). Em aplicações de imagem, o DCT é capaz de transformar uma figura do

domínio do espaço para o domínio da frequência. Aqui, foi utilizada apenas a transformação de uma matriz 8x8 utilizando uma matriz ortonormal, que tem sua inversa igual à sua transposta. Em implementações computacionais da DCT, pode-se utilizar a abordagem de transformada por matriz transposta para otimizar a execução em hardware. Então, uma matriz A pode ser computada como $B = T * A * T$ e a DCT de B é $A = T * B * T$

A AMD possui um algoritmo simples em C++ para ser usado no Vitis, utilizando diretivas importantes como #pragma. O algoritmo está disponível no anexo A. Há também uma necessidade de utilizar testes unitários para o Vitis rodar e sintetizar o controle.

Com o Vitis aberto no workspace, cria-se um componente HLS vazio. Na figura 4.1 é colocado o arquivo .cpp principal e na parte de *testbench* os arquivos necessários para testes.

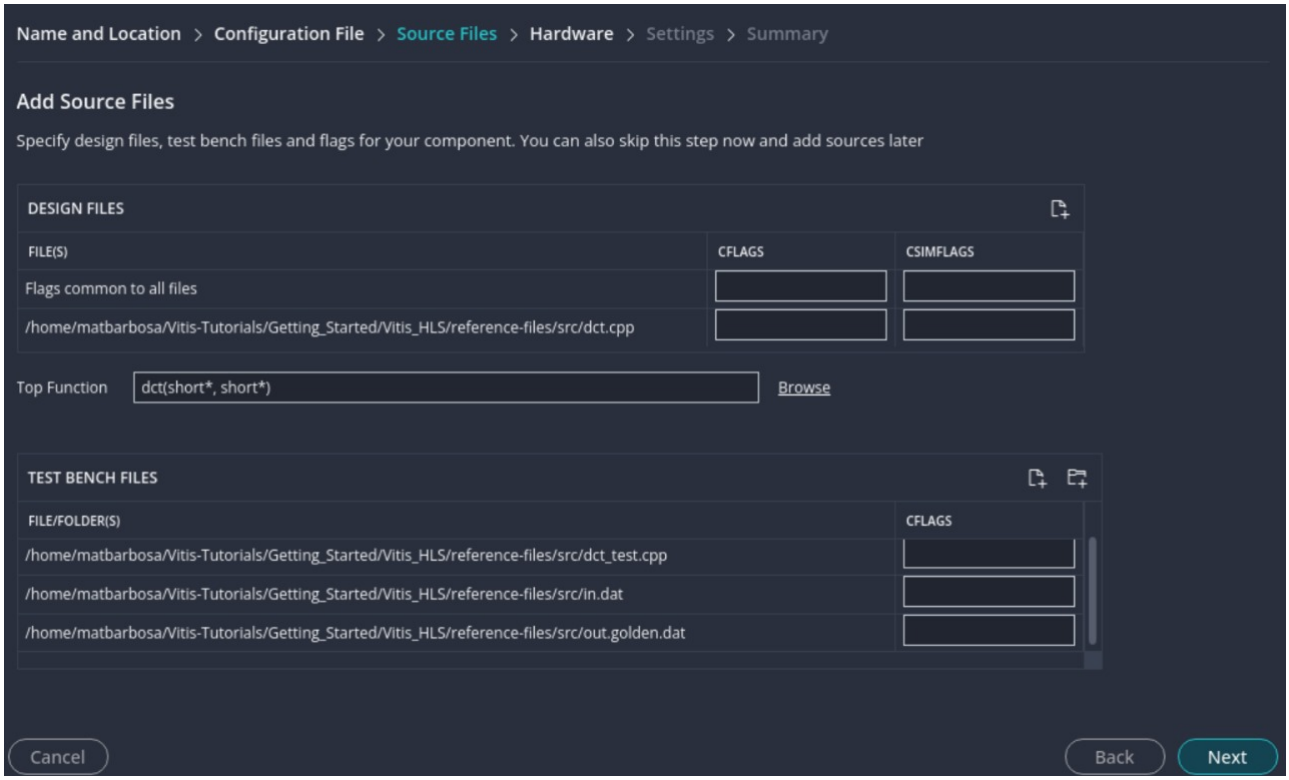


Figura 4.1 – Arquivos de design e testes, além de colocar a função principal.

O próximo procedimento é escolher a placa, como na figura 4.2. A placa no caso é a Alveo u55c.

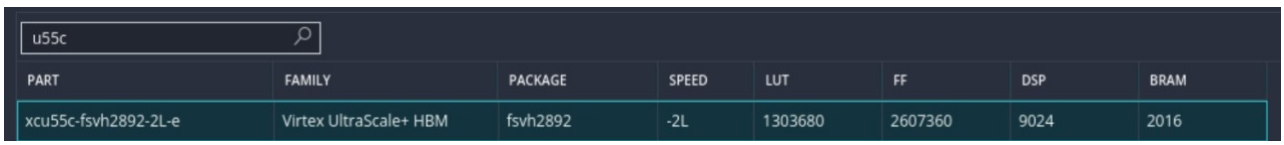


Figura 4.2 – Escolha da placa.

O próximo passo da simulação é definir o clock, sua precisão, o alvo e o estilo do output.

Foram utilizadas as sugeridas pela AMD (figura 4.3).

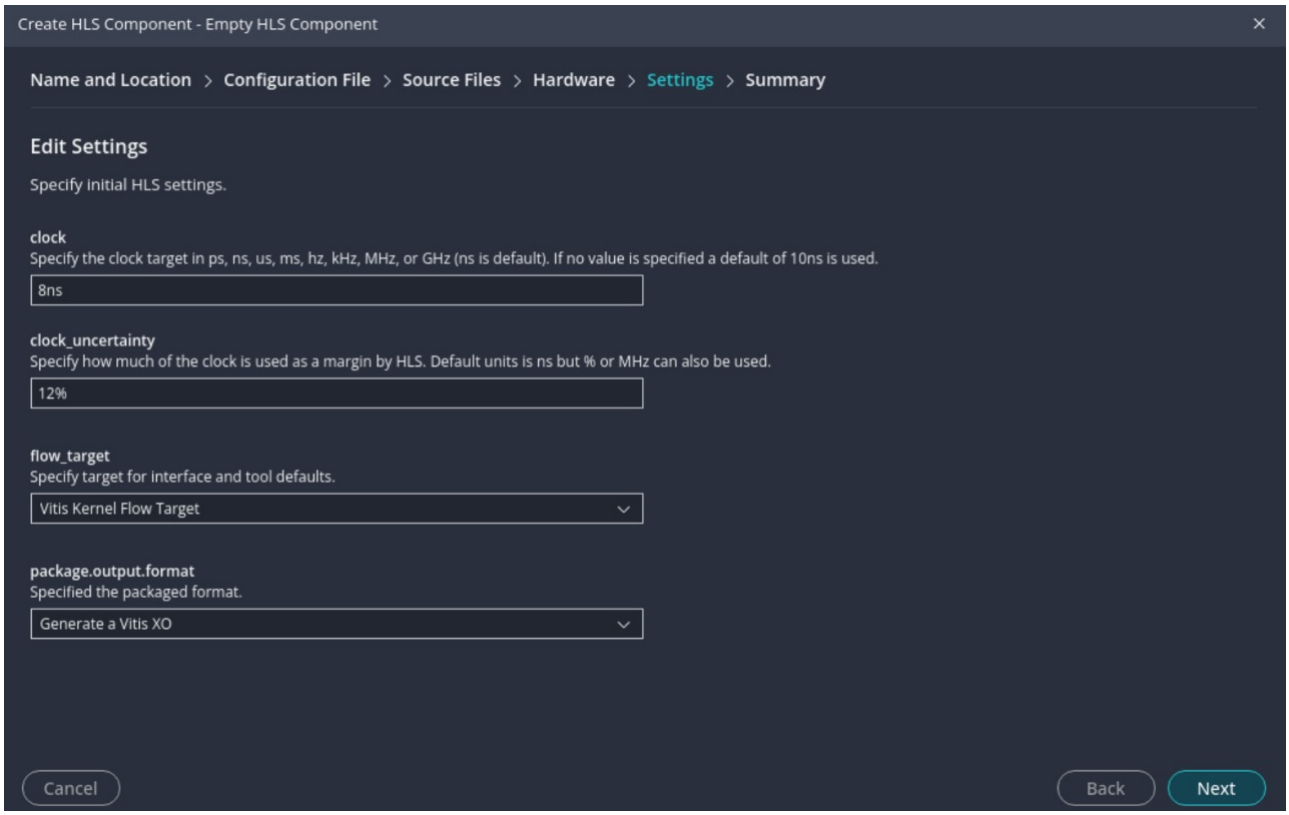


Figura 4.3 – Escolha dos parâmetros de simulação.

O próximo passo é realizar a compilação e a síntese do código. Funcionando, teremos os outputs desejados. Note que na figura 4.4 temos as chamadas das funções. Na figura 4.5, o tempo de latência em cada loop, pois alguns dependiam dos resultados de outras funções do código. Também podemos ver na figura 4.6 (que é continuação da figura 4.4) as rotinas que conseguiram ser feitas em pipeline.

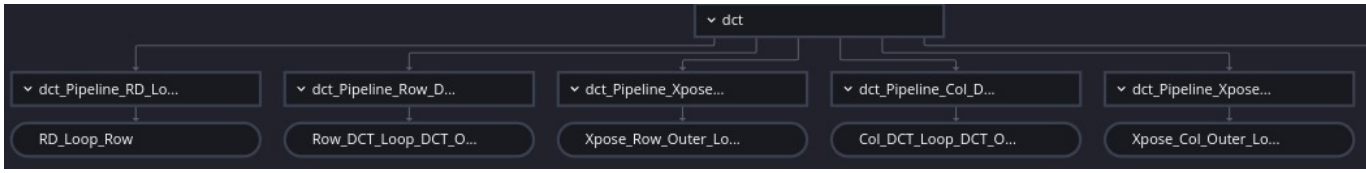


Figura 4.4 – Chamadas de funções.

Por fim, temos o resultado da frequência esperada em relação à frequência simulada (figura 4.7).

Summary Synthesis Report - dct

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL
dct (6)	446	3.568E3	-	447
dct_Pipeline_RD_Loop_Row (1)	11	88.000	-	9
RD_Loop_Row	9	72.000	3	1
dct_Pipeline_Row_DCT_Loop_DCT_Outer_Loop (1)	70	560.000	-	65
Row_DCT_Loop_DCT_Outer_Loop	68	544.000	6	1
dct_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop (1)	66	528.000	-	65
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	64	512.000	2	1
dct_Pipeline_Col_DCT_Loop_DCT_Outer_Loop (1)	70	560.000	-	65
Col_DCT_Loop_DCT_Outer_Loop	68	544.000	6	1
dct_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop (1)	66	528.000	-	65
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	64	512.000	2	1
dct_Pipeline_WR_Loop_Row (1)	11	88.000	-	9
WR_Loop_Row	9	72.000	3	1

Figura 4.5 – Síntese.

TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
-	no	30	16	6512	7224	0
-	loop auto-rewind (delay=0 clock	0	0	917	109	0
8	yes	-	-	-	-	-
-	loop auto-rewind (delay=0 clock	0	8	256	333	0
64	yes	-	-	-	-	-
-	loop auto-rewind (delay=0 clock	0	0	25	160	0
64	yes	-	-	-	-	-
-	loop auto-rewind (delay=0 clock	0	8	256	333	0
64	yes	-	-	-	-	-
-	loop auto-rewind (delay=0 clock	0	0	25	160	0
64	yes	-	-	-	-	-
-	loop auto-rewind (delay=0 clock	0	0	523	424	0
8	yes	-	-	-	-	-

Figura 4.6 – Continuação da Síntese.

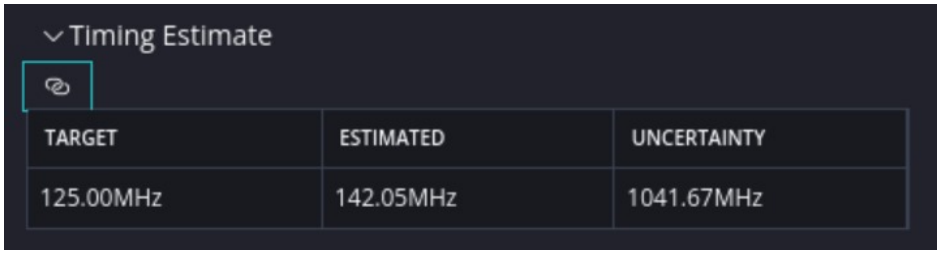


Figura 4.7 – Frequências.

4.4 DCT em HLS com otimizações

O experimento foi corretamente executado, porém sem padrões de aceleração, tanto do Vitis quanto do código .cpp.

O padrão para que o Vitis utilize o *unroll* em loops de pipeline é de 64 repetições. Foi mudado para 5 no arquivo hls_config.cfg, o que implica que cada loop de 6 ou mais repetições será feito com *unrolling*. Um loop *unrolling* faz com que o compilador HLS crie N cópias do hardware utilizado no loop e as rodarão em paralelo. Isso possibilita a finalização de várias iterações em apenas um ciclo de clock.

Outra otimização é utilizar o #pragma HLS PIPELINE. Na função no anexo 1 (dct.cpp), foi adicionado o comando #pragma HLS PIPELINE II=4, indicando que o programa tentará usar uma pipeline de 4 ciclos de clock.

O C Synthesis foi executado novamente. Na figura 4.8, podemos ver uma diferença nas calls das funções em relação ao algoritmo não acelerado.

Note que o intervalo, mostrado na figura 4.5, foi de 447 ciclos, enquanto no exemplo acelerado foi de 189 ciclos (figura 4.10). Lembrando que ambos utilizaram a mesma frequência de clock (tempo de 9ns).

Figura 4.8 – Call Graph com II-4.

Foi feito um teste com o um algoritmo C++ de DCT compilado pelo g++ e executado pelo servidor. Utilizando funções para marcação de tempo de duração do algoritmo, obteve-se um resultado muito melhor que a FPGA (7ms) em relação ao tempo de 4.032 microssegundos do HLS não acelerado e 1.072 microssegundos do HLS acelerado.

Porém tal comportamento é esperado: uma matriz tão pequena vai ser naturalmente processada de forma mais rápida no processador do servidor. Com uma matriz 100x100, a CPU i7 levou 7.7 microssegundos, perdendo para a alveo, que foi 1.9 vezes mais rápida com HLS não acelerado e 7.18 vezes mais rápida com o HLS acelerado.

NAME	II	LATENCY	BRAM	URAM	DSP
dct	189	188	30	0	0
dct_Pipeline_RD_Loop_Row	9	11	0	0	0
RD_Loop_Row	1	9			
dct_2d	4	20	0	0	0
dct_1d	1	4	0	0	28
dct_1d	1	4	0	0	28
dct_1d	1	4	0	0	28
dct_1d	1	4	0	0	28
dct_Pipeline_WR_Loop_Row	9	11	0	0	0
WR_Loop_Row	1	9			

Figura 4.9 – Síntese com a aceleração.

INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
189	-	no	30	112	11263	11852	0
9	-	loop auto-rew	0	0	917	109	0
1	8	yes	-	-	-	-	-
4	-	yes	0	112	5590	6222	0
1	-	yes	0	28	851	1032	0
1	-	yes	0	28	851	1032	0
1	-	yes	0	28	851	1032	0
1	-	yes	0	28	851	1032	0
9	-	loop auto-rew	0	0	523	424	0
1	8	yes	-	-	-	-	-

Figura 4.10 – Síntese com a aceleração - continuação.

5 Conclusões

Os experimentos realizados demonstraram duas abordagens: uma utilizando um algoritmo compilado com HLS sem aceleração de hardware e outra com aceleração habilitada. Além disso, foi feita a comparação com a execução do mesmo algoritmo na CPU do servidor. Os resultados indicaram que, para matrizes menores, a CPU apresentou melhor desempenho em relação às implementações na Alveo. No entanto, conforme o tamanho da matriz aumentou, a Alveo superou a CPU, tanto na versão acelerada quanto na não acelerada.

Isso sugere que a síntese de código em alto nível (C++) utilizando aceleradores por meio de diretivas como pragmas pode resultar em um desempenho superior ao da CPU do servidor, especialmente para cargas computacionais mais intensivas. Com aprimoramentos na implementação em C++ e otimizações mais eficientes, espera-se que a Alveo obtenha vantagem mesmo para matrizes menores.

Diante desses achados, recomenda-se a continuidade da utilização do servidor disponível para explorar novas aplicações e cenários computacionais, aproveitando ao máximo os recursos da plataforma de aceleração. Investigar técnicas mais avançadas de otimização e adaptação do código poderá ampliar ainda mais o potencial de desempenho da Alveo, tornando-a uma alternativa viável para um espectro mais amplo de problemas computacionais.

5.1 Trabalhos Futuros

Para pesquisas futuras, recomenda-se a continuidade dos experimentos utilizando o servidor disponível, explorando não apenas a FPGA Alveo, mas também os SmartSSDs, de modo a avaliar seu impacto no desempenho de diferentes aplicações. Além disso, sugere-se a implementação e análise de um conjunto mais amplo de exemplos fornecidos pela AMD/Xilinx, o que pode contribuir para um melhor entendimento das capacidades da plataforma.

A longo prazo, esse esforço poderá viabilizar a criação de um ambiente computacional na UnB com aplicações próprias otimizadas para aceleração em FPGA, proporcionando um servidor eficiente e de fácil utilização para a comunidade acadêmica. Dessa forma, espera-se que esse trabalho sirva como base para novas pesquisas e desenvolvimentos em computação de alto desempenho na instituição.

Referências

BEZERRA FERNANDO LUIZ KOCH, C. B. W. Wesley dos R. Models of computing as a service and iot: an analysis of the current scenario with applications using lpwan. **Revista de Sistemas de Informação da FSMA**, n. 25, p. 56–65, 2020. Citado na p. 18.

CARD BDF VALUES. **Card BDF Values**. [*S.l.*], [*s.d.*]. Disponível em: https://xilinx.github.io/Alveo-Cards/master/debugging/build/html/docs/common-steps.html#displaying-card-bdf-valuesc. Citado na p. 26.

GOKHALE, O. B. P.; BHAT, S. Introduction to iot. **International Advanced Research Journal in Science, Engineering and Technology**, v. 5, n. 1, p. 1–2, 2018. Citado na p. 13.

KACHRIS, D. S. C. A survey on reconfigurable accelerators for cloud computing. **Institute of Communication and Computer Systems (ICCS/NTUA)**, 2020. Citado na p. 19.

KNODEL, R. G. S. O. Rc3e: Provision and management of reconfigurable hardware accelerators in a cloud environment. **Technische Universität Dresden Dresden, Germany**, 2015. Citado na p. 18.

PEMBERTON ANTON ZABREYKO, Z. D. R. K. N.; GONZALEZ, J. Kernel-as-a-service: A serverless interface to gpus. **UC Berkeley**, 2022. Citado na p. 18.

XILINX. **Alveo U55C Data Center Accelerator Card Installation Guide (UG1468)**. [*S.l.*], [*s.d.*]. Disponível em: https://docs.amd.com/v/u/en-US/ug1468-alveo-u55c. Citado nas pp. 21 e 22.

XILINX. **XCU55C Block Diagram**. [*S.l.*], [*s.d.*]. Disponível em: https://docs.amd.com/v/u/en-US/ug1468-alveo-u55c. Citado na p. 22.

XILINX, AMD. **Troubleshooting for installation**. [*S.l.*], 2021. Disponível em: https://docs.amd.com/api/khub/documents/2ZAlG7XB3f4~rPJDbRfTxQ/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=5.0.54#page=24&zoom=100,0,108. Citado na p. 23.

Anexos

Anexo A – Código dct.cpp e dct.h

Código A.1 – Arquivo C++ para A transformada discreta do cosseno.

```
#include "dct.h"

void dct_1d(dct_data_t src[DCT_SIZE], dct_data_t dst[DCT_SIZE])
{
   unsigned int k, n;
   int tmp;
   const dct_data_t dct_coeff_table[DCT_SIZE][DCT_SIZE] = {
#include "dct_coeff_table.txt"
   };

DCT_Outer_Loop:
   for (k = 0; k < DCT_SIZE; k++) {
DCT_Inner_Loop:
      for(n = 0, tmp = 0; n < DCT_SIZE; n++) {
         int coeff = (int)dct_coeff_table[k][n];
         tmp += src[n] * coeff;
      }
      dst[k] = DESCALE(tmp, CONST_BITS);
   }
}

void dct_2d(dct_data_t in_block[DCT_SIZE][DCT_SIZE],
      dct_data_t out_block[DCT_SIZE][DCT_SIZE])
{
   dct_data_t row_outbuf[DCT_SIZE][DCT_SIZE];
   dct_data_t col_outbuf[DCT_SIZE][DCT_SIZE],
       col_inbuf[DCT_SIZE][DCT_SIZE];
   unsigned i, j;

   // DCT rows
Row_DCT_Loop:
   for(i = 0; i < DCT_SIZE; i++) {
      dct_1d(in_block[i], row_outbuf[i]);
   }
   // Transpose data in order to re-use 1D DCT code
Xpose_Row_Outer_Loop:
   for (j = 0; j < DCT_SIZE; j++)
Xpose_Row_Inner_Loop:
      for(i = 0; i < DCT_SIZE; i++)
         col_inbuf[j][i] = row_outbuf[i][j];
   // DCT columns
Col_DCT_Loop:
   for (i = 0; i < DCT_SIZE; i++) {
      dct_1d(col_inbuf[i], col_outbuf[i]);
   }
   // Transpose data back into natural order
```

```
Xpose_Col_Outer_Loop:
   for (j = 0; j < DCT_SIZE; j++)
Xpose_Col_Inner_Loop:
      for(i = 0; i < DCT_SIZE; i++)
         out_block[j][i] = col_outbuf[i][j];
}

void read_data(short input[N], short buf[DCT_SIZE][DCT_SIZE])
{
   int r, c;

RD_Loop_Row:
   for (r = 0; r < DCT_SIZE; r++) {
RD_Loop_Col:
      for (c = 0; c < DCT_SIZE; c++)

buf[r][c] = input[r * DCT_SIZE + c];
   }
}

void write_data(short buf[DCT_SIZE][DCT_SIZE], short output[N])
{
   int r, c;

WR_Loop_Row:
   for (r = 0; r < DCT_SIZE; r++) {
WR_Loop_Col:
      for (c = 0; c < DCT_SIZE; c++)
         output[r * DCT_SIZE + c] = buf[r][c];
   }
}

void dct(short input[N], short output[N]) {


   short buf_2d_in[DCT_SIZE][DCT_SIZE];
   short buf_2d_out[DCT_SIZE][DCT_SIZE];

   read_data(input, buf_2d_in);
   dct_2d(buf_2d_in, buf_2d_out);
   write_data(buf_2d_out, output);
}
```

Código A.2 – dct.h, declarações.

```
#ifndef __DCT_H__
#define __DCT_H__

#include <fstream>
#include <iostream>
#include <iomanip>
#include <cstdlib>

```

```
9  #define DW 16
10 #define N 1024/DW
11 #define NUM_TRANS 16
12
13 typedef short dct_data_t;
14
15 #define DCT_SIZE 8
16 #define CONST_BITS  13
17 #define DESCALE(x,n)  (((x) + (1 << ((n)-1))) >> n)
18
19 extern "C" {
20   void dct(short input[N], short output[N]);
21 }
22 #endif //
```

Anexo B – Código dct_test.cpp

Código B.1 – Arquivo C++ para testes unitários de dct.cpp

```
#include "dct.h"

int main() {
   short a[N], b[N], b_prime[N], x[10*N];
   int retval = 0, i, j, z;
   FILE *fp;

   fp=fopen("in.dat","r");
   for (i=0; i<(10*N); i++){
      int tmp;
      fscanf(fp, "%d", &tmp);
      x[i] = tmp;
   }
   fclose(fp);


   //Call the DCT function 10 times for Dataflow
   for (i=0; i<10; i++){
     printf("Checkpoint: i = %d\n", i);
     // Iterate through the large x[] array to populate the smaller input
        array a[]
     for (j=0; j<N; j++){
        a[j] = x[j+(N*i)];
       printf("Checkpoint: j = %d; ", j);
     }//End j_loop
     printf("\n");

     //Call DCT for each iteration of a[]
     dct(a, b);

     //For testing purposes, save the first iteration of b[] to b_prime[]
     //To compare to out.golden.dat
     if (i==0){
       printf("Copying array b to b_prime\n");
       for (z=0; z<N; z++) {
        b_prime[z] = b[z];
       }//End For
     }//End IF
   }//End i_loop


  fp=fopen("out.dat","w");
  //printf(" Din Dout \n");
  for (i=0;i<N;i++) {
    fprintf(fp, "%d \n", b_prime[i]);
    //printf("  %d   %d\n", a[i], b[i]);
```

```
46    }
47    fclose(fp);
48
49    // Compare the results file with the golden results
50    retval = system("diff --brief -w out.dat out.golden.dat");
51    if (retval != 0) {
52      printf("Test failed  !!!\n");
53      retval=1;
54    } else {
55      printf("Test passed !\n");
56    }
57     return retval;
58  }
```


UnB