CENTRO UNIVERSITÁRIO DO INSTITUTO MAUÁ DE TECNOLOGIA

ESCOLA DE ENGENHARIA MAUÁ

ENGENHARIA DE COMPUTAÇÃO

ANDRÉ SOLANO FERREIRA RODRIGUES MAIOLINI
DURVAL CONSORTI SORANZ DE BARROS SANTOS
LEONARDO ROBERTO AMADIO
ESTEVAN DELAZARI FEHER
LUCAS CASTANHO PAGANOTTO

PROJETO E ANÁLISE DE UM SOC RISC-V COM ACELERAÇÃO NEURAL EM FPGA

SÃO PAULO
2026

André Solano Ferreira Rodrigues Maiolini
Durval Consorti Soranz de Barros Santos
Leonardo Roberto Amadio
Estevan Delazari Feher
Lucas Castanho Paganotto

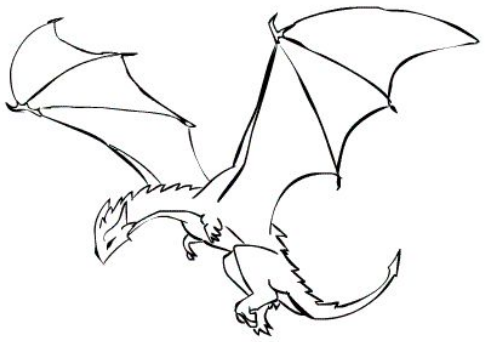
PROJETO E ANÁLISE DE UM SOC RISC-V COM ACELERAÇÃO NEURAL EM FPGA

Trabalho de conclusão de curso apresentado como requisito parcial para obtenção do título de Engenheiro de Computação, abordando o projeto, implementação e análise de desempenho de um System-on-Chip (SoC) baseado na arquitetura RISC-V com suporte à computação heterogênea.

Orientador: Prof. Me. Nuncio Perrella

SÃO PAULO
2026

Este trabalho é dedicado àqueles cuja curiosidade os empurra sempre um pouco além do necessário.



"Fairy tales are more than true: not because they tell us that dragons exist, but because they tell us that dragons can be beaten."

(Neil Gaiman)

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
abnTeX	ABsurdas Normas para TeX

LISTA DE SÍMBOLOS

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence

SUMÁRIO

1 INTRODUÇÃO

A computação moderna é construída sobre múltiplas camadas de abstração. Do clique de um mouse à execução de uma inteligência artificial, existem bilhões de transistores operando em uma sinfonia lógica coordenada. No entanto, para a maioria dos profissionais da área, essa base opera como uma caixa-preta. O hardware tornou-se algo que se usa, mas não é necessariamente compreendido.

Este trabalho propõe romper com a visão estritamente utilitária. O projeto aqui documentado não se limita à construção de um microprocessador funcional, mas constitui uma investigação prática sobre os fundamentos arquiteturais que sustentam toda a era digital. Por meio do projeto, implementação e validação de um sistema-em-chip (SoC) completo baseado na arquitetura RISC-V, busca-se rersgatar o domínio sobre a camada de hardware subjacente às abstrações modernas.

A arquitetura RISC-V foi adotada como eixo central deste estudo por sua natureza aberta, modular e pedagogicamente adequada. Sua simplicidade estrutural permite observar, de forma explícita, a relação entre o conjunto de instruções, o datapath, os sinais de controle e o comportamento temporal do sistema. Dessa forma, o processador desenvolvido não é tratado apenas como um meio para execução de programas, mas como um objeto de análise arquitetural, no qual decisões de projeto impactam diretamente desempenho, previsibilidade e uso de recursos.

À medida que aplicações contemporâneas passam a demandar volumes crescentes de processamento especializado - especialmente em domínios como aprendizado de máquina e inferência em dispositivos embarcados - torna-se evidente a limitação de arquiteturas puramente generalistas. Nesse contexto, aceleradores dedicados emergem como uma extensão natural do sistema computacional. Assim, este trabalho evolui a partir do núcleo RISC-V para a integração de uma Unidade de Processamento Neural (*Neural Processing Unit* - NPU), utilizada como estudo de caso para explorar os efeitos da especialização arquitetural dentro de um SoC.

A NPU não é apresentada como um fim em si mesma, mas como uma consequência direta da análise das camadas de abstração do sistema. Sua iniclusão permite investigar, de forma controlada, como a divisão de reponsabilidades entre CPU e aceleradores impacta métricas como latência, throughput, movimentação de dados e limites teóricos de aceleração - destravando a era da computação heterogênea. Dessa forma, o projeto estabele uma ponte concreta entre fundamentos clássicos de arquitetura de computadores e desagios atuais da computação embarcada e de edge-AI.

1.1 Motivação

Por que, na era da computação em nuvem e linguagens de altíssimo nível, um engenheiro deve despender tempo entendendo barramentos, datapaths e sinais de controle? A despeito do avanço das abstrações oferecidas por linguagens modernas, frameworks e plataformas em

nuvem, os sistemas computacionais continuam fundamentados em princípios microarquiteturais que determinam seu desempenho, seu comportamento e suas limitações. Esses conhecimentos permanecem essenciais porque tais componentes constituem o nível de base sobre o qual todas as abstrações são construídas. Sem essa compreensão, o engenheiro passa a operar máquinas sem entender suas características - tais como: latência, gargalos, paralelismo, hierarquia de memória e restrições — que afetam diretamente a eficiência e a confiabilidade de qualquer solução de software ou hardware desenvolvida.

Do ponto de vista formativo, a familiaridade com os elementos internos do processador também capacita o engenheiro a compreender o funcionamento da cadeia completa de execução — da especificação de uma instrução ao seu efeito no hardware físico. Essa visão abrangente é indispensável em áreas como sistemas embarcados, computação de alto desempenho, aceleradores especializados, dispositivos IoT, aplicações críticas e de tempo real, nas quais a interação direta com o hardware é frequente e decisiva.

A motivação deste projeto apoia-se em três pilares fundamentais:

1. **Domínio da Arquitetura Heterogênea:** o desempenho computacional moderno é limitado majoritariamente pelo "Gargalo de Von Neumann- o custo de mover dados entre memória e processador. Compreender como mitigar esse gargalo através de técnicas como DMA (Acesso Direto à Memória) e hardware de aceleradores especializados é uma competência esesencial para o projeto de sistemas eficientes;

2. **Abordagem Pedagógica e Visual:** a abstração excessiva pode ser prejudicial ao aprendizado profundo. Ao desenvolver um sistema que não inclui apenas o hardware, mas ferramentas de visualização, o projeto torna tangível conceitos abstratos.

Em suma, mesmo em um cenário em que as abstrações se tornam cada vez mais sofisticadas, a compreensão do nível fundamental da computação permite que o engenheiro avalie limitações, explore otimizações, antecipe comportamentos do sistema e desenvolva soluções mais robustas e eficientes. Assim, o estudo desses conceitos continua sendo um elemento central na formação e na prática da engenharia moderna.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho propõe o projeto, a implementação e a análise de um sistema-em-chip (SoC) baseado em um processador RISC-V em FPGA, investigando como decisões microarquiteturais em diferentes níveis de abstração impactam o desempenho, o comportamento temporal e os limites de aceleração do sistema. A integração de uma unidade de processamento neural (NPU) dedicada ao SoC permite estudar o desempenho de sistemas heterogêneos e avaliar os ganhos potenciais de aceleração computacional.

Além disso, o projeto visa o desenvolvimento de um ecossistema educacional de ferramentas voltadas para o ensino de Arquitetura e Organização de Computadores. Diferentemente de abordagens que utilizam blocos de propriedade intelectual (IPs) proprietários fornecidos por fabricantes de FPGA, este trabalho adota uma filosofia de implementação integral da microarquitetura, permitindo inspeção detalhada de todos os componentes e favorecendo uma abordagem de pesquisa e ensino baseada em *white-box*.

1.2.2 Objetivos Específicos

1. **Core RISC-V:** projetar um processador RV32I multiciclo, detalhando a relação entre conjunto de instruções, *datapath* e unidade de controle;

2. **NPU Sistólica:** projetar um acelerador de redes neurais baseado em *Systolic Array* com micro-sequenciador próprio, explorando a execução autônoma de cargas de trabalho de IA;

3. **Arquitetura de Comunicação:** implementar a infraestrutura do SoC, incluindo barramento compartilhado, arbitragem centralizada e controlador de DMA (*Direct Memory Access*) para transferências de dados;

4. **Ecossistema Educacional:** desenvolver bibliotecas de abstração de hardware (HAL) e aplicações gráficas em Python que permitam a inspeção visual da memória e interação em tempo real com o sistema embarcado;

5. **Validação Experimental:** validar o sistema completo em FPGA, executando inferências reais para analisar os ganhos de desempenho da aceleração por hardware frente à execução puramente via software.

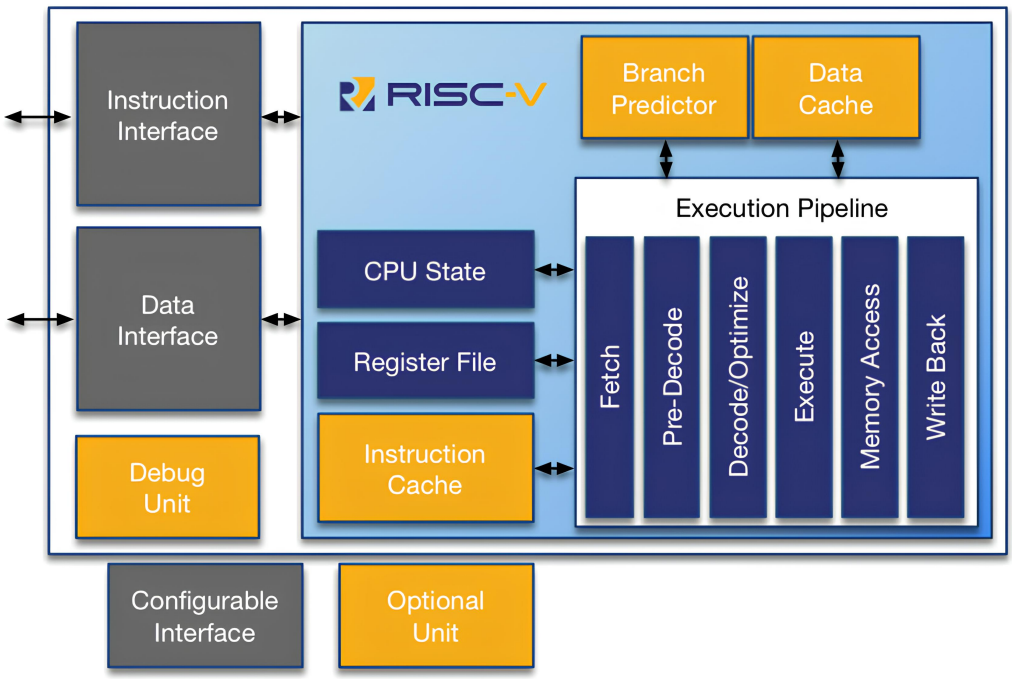
2 REVISÃO BIBLIOGRÁFICA

2.1 Arquitetura RISC-V

Historicamente, o projeto de processadores seguiu duas filosofias principais de arquiteturas de conjunto de instruções (*Instruction Set Architectures - ISAs*). A arquitetura CISC (*Complex Instruction Set Computing*), exemplificada pela família x86 da Intel, privilegia a densidade de instruções por meio de um conjunto amplo de instruções complexas e especializadas. Em contraponto, a abordagem RISC (*Reduced Instruction Set Computing*) busca um hardware mais simples e previsível, apoiando-se em um conjunto de instruções reduzido e regular (PATTERSON; HENNESSY, 2021).

O RISC-V é uma arquitetura de conjunto de instruções (*ISA*) baseada no conceito RISC (*Reduced Instruction Set Computer*). Em termos práticos, a especificação RISC-V define o contrato entre hardware e software em seu nível mais fundamental. Diferentemente de arquiteturas proprietárias, como x86 e ARM, ela é aberta e livre de royalties, permitindo que universidades, empresas e desenvolvedores a utilizem, implementem e modifiquem livremente.

Figura 1 – Arquitetura geral de um núcleo RISC-V, destacando o *pipeline* de execução, o banco de registradores e as interfaces de memória.



Fonte: (Roa Logic B.V., 2018)

As principais características da arquitetura RISC-V incluem:

1. **Modularidade:** núcleo mínimo (base ISA) e extensões opcionais.

2. **Simplicidade**: instruções de tamanho fixo (geralmente 32 bits).

3. **Portabilidade**: projetada para uso desde microcontroladores até supercomputadores.

4. **Ecossistema aberto**: suporte crescente em compiladores, simuladores e hardware.

Apesar do domínio da ARM (*Advanced RISC Machine*) no mercado de embarcados e móveis, a flexibilidade, o custo zero de licenciamento e a capacidade de customização do RISC-V o posicionam como um competidor formidável e uma alternativa estratégica para o futuro da computação.

2.1.1 Conjunto de Instruções RV32I

O conjunto de instruções RV32I constitui a especificação base da arquitetura RISC-V para processadores de 32 bits (RISC-V International, 2019). Trata-se de uma arquitetura do tipo *load-store*, na qual operações aritméticas e lógicas são realizadas exclusivamente sobre operandos residentes em registradores. O acesso à memória é efetuado apenas por meio de instruções explícitas de *load* e *store*, responsáveis, respectivamente, por transferir dados da memória para os registradores e por escrever os resultados de volta na memória.

A nomenclatura RV32I indica uma largura de palavra de 32 bits e o suporte a operações com números inteiros, que formam a base obrigatória sobre a qual todas as demais extensões da ISA RISC-V são construídas. A Tabela 1 apresenta o conjunto de instruções fundamentais suportadas pelo núcleo RV32I.

Tabela 1 – Instruções do Conjunto RV32I

Categoria	Mnemônico	Descrição
R-Type	ADD	Soma aritmética entre registadores.
	SUB	Subtração aritmética entre registadores.
	SLL	Deslocamento lógico à esquerda (*Shift Left Logical*).
	SLT	Definir se menor que (*Set Less Than*, com sinal).
	SLTU	Definir se menor que (*Set Less Than Unsigned*, sem sinal).
	XOR	Ou exclusivo bit a bit.
	SRL	Deslocamento lógico à direita (*Shift Right Logical*).
	SRA	Deslocamento aritmético à direita (*Shift Right Arithmetic*).
	OR	Ou lógico bit a bit.
	AND	E lógico bit a bit.
I-Type	ADDI	Soma com valor imediato.
	SLTI	Definir se menor que imediato (com sinal).
	SLTIU	Definir se menor que imediato (sem sinal).
	XORI	Ou exclusivo imediato.

Categoria	Mnemônico	Descrição
	ORI	Ou lógico imediato.
	ANDI	E lógico imediato.
	SLLI	Deslocamento lógico à esquerda imediato.
	SRLI	Deslocamento lógico à direita imediato.
	SRAI	Deslocamento aritmético à direita imediato.
U-Type	LUI	Carregar Imediato Superior (*Load Upper Immediate*). Preenche os 20 bits mais significativos.
	AUIPC	Adicionar Imediato Superior ao PC (*Add Upper Immediate to PC*).
Jumps	JAL	Salto e Link (*Jump and Link*). Salto incondicional relativo ao PC.
	JALR	Salto e Link Registrador (*Jump and Link Register*). Salto incondicional via registrador.
Branches	BEQ	Desvio se Igual (*Branch if Equal*).
	BNE	Desvio se Diferente (*Branch if Not Equal*).
	BLT	Desvio se Menor (*Branch if Less Than*, com sinal).
	BGE	Desvio se Maior ou Igual (*Branch if Greater/Equal*, com sinal).
	BLTU	Desvio se Menor (*Branch if Less Than*, sem sinal).
	BGEU	Desvio se Maior ou Igual (*Branch if Greater/Equal*, sem sinal).
Loads	LB	Carregar Byte (estende sinal).
	LH	Carregar Meia-Palavra (*half-word*, estende sinal).
	LW	Carregar Palavra (*word*, 32 bits).
	LBU	Carregar Byte (sem sinal, preenche com zeros).
	LHU	Carregar Meia-Palavra (sem sinal, preenche com zeros).
Stores	SB	Armazenar Byte.
	SH	Armazenar Meia-Palavra.
	SW	Armazenar Palavra.
System	FENCE	Sincronização de memória e I/O.
	ECALL	Chamada de sistema (*Environment Call*).
	EBREAK	Paragem para depuração (*Environment Break*).

O conjunto de instruções base da RISC-V constitui a especificação mais simples e fundamental da arquitetura, servindo como ponto de partida obrigatório para qualquer implementação. Isoladamente, essa base é suficiente para a construção de um processador de propósito geral e para a execução de código completo escrito em C, desde que utilizado o compilador e a *toolchain* apropriados da RISC-V.

Neste projeto, adota-se o subconjunto **RV32I**, que define, entre outros aspectos funda-

mentais da arquitetura:

1. Banco de 32 registradores de propósito geral (`x0` a `x31`), sendo o registrador `x0` rigidamente fixo em zero;

2. Um contador de programa (`PC` - *Program Counter*) de 32 bits.

A presença de um registrador constantemente nulo (`x0`) simplifica o conjunto de instruções e o hardware do processador, permitindo a implementação eficiente de operações como movimentação de dados, inicialização de registradores e comparação com zero, sem a necessidade de instruções dedicadas ou lógica adicional. Dentre os usos recorrentes estão:

- **Inicialização e limpeza de registradores**: instruções como "`ADD xN, x0, x0`"ou "`ADDI xN, x0, 0`" permitem zerar registradores sem a necessidade de uma instrução dedicada para esse fim;

- **Movimentação de dados**: a pseudo-instrução `MV` é implementada como "`ADD xD, xS, x0`", explorando o fato de `x0` sempre fornecer o valor zero;

- **Comparações com zero**: instruções de desvio condicional, como `BEQ` e `BNE`, utilizam `x0` para realizar testes contra zero de forma direta e eficiente;

- **Descartar resultados**: operações cujo resultado não precisa ser preservado podem ter seu destino apontado para `x0`, eliminando a necessidade de lógica adicional para supressão de escrita.

Um processador que se declara compatível com **RV32IMC**, por exemplo, implementa a base de inteiros (I), a extensão de multiplicação e divisão (M) e o conjunto de instruções comprimidas de 16 bits (C), mantendo plena compatibilidade com o ecossistema de software RISC-V.

2.1.2 Extensões relevantes

A arquitetura RISC-V distingue-se pela sua modularidade. O conjunto de instruções base (como o RV32I implementado neste trabalho) é a única parte obrigatória; sobre ela, projetistas podem acoplar módulos opcionais para otimizar o desempenho em domínios específicos.

As extensões padronizadas são identificadas por letras sufixas à nomenclatura do processador. As principais são:

- **M (*Integer Multiplication and Division*)**: adiciona hardware dedicado para multiplicação e divisão de inteiros. Sem esta extensão, estas operações devem ser realizadas via software (rotinas de emulação), o que é significativamente mais lento.

- **A (*Atomic*)**: fornece instruções de leitura-modificação-escrita atômicas. Estas são essenciais para a sincronização de memória e gerenciamento de exclusão mútua em sistemas operacionais e arquiteturas *multicore*.

- **F/D (*Single/Double Precision Floating-Point*)**: adiciona um banco de registradores separado e unidades de cálculo para operações com ponto flutuante, seguindo o padrão IEEE 754 (32 bits para F e 64 bits para D).

- **C (*Compressed*)**: introduz um conjunto de instruções de 16 bits que mapeiam para as operações mais comuns de 32 bits. Isso aumenta a densidade de código, reduzindo o tamanho dos binários e o uso de largura de banda de memória.

- **V (*Vector*)**: habilita operações vetoriais (SIMD - *Single Instruction, Multiple Data*), permitindo o processamento paralelo de grandes volumes de dados, comum em aplicações de IA e processamento de sinal.

- **B (*Bit Manipulation*)**: oferece instruções eficientes para manipulação de bits individuais (rotações, contagens, extrações), úteis em criptografia e drivers de dispositivos.

É comum na literatura encontrar a designação RV32G ou RV64G, onde 'G' (*General*) representa um agrupamento das extensões mais utilizadas em sistemas de propósito geral.

Cabe destacar que, embora o conjunto *G* agregue as extensões mais comuns, extensões como Zicsr são frequentemente tratadas como obrigatórias na prática, pois viabilizam o uso dos registradores de controle necessários ao tratamento de exceções e interrupções.

Além do padrão, a ISA reserva espaço de opcodes para **extensões customizadas (X)**. Esta característica permite a criação de aceleradores de hardware proprietários para tarefas especializadas (domain-specific architectures) sem quebrar a compatibilidade com o ecossistema de software base.

2.2 Microarquitetura

A microarquitetura define a organização lógica e física dos componentes internos do processador para implementar o conjunto de instruções (ISA) alvo. Geralmente, a microarquitetura é utiliza-se da separação entre o Caminho de Dados (*Datapath*) - responsável pelo armazenamento e manipulação das informações - e a Unidade de Controle (*Control*) — responsável por orquestrar o fluxo de dados através de sinais de controle.

O caminho de dados do processador é composto por elementos de estado (memórias e registradores) e elementos combinacionais (ALU, somadores, multiplexadores). No qual o fluxo principal de execução segue a seguinte sequência lógica dentro do hardware:

1. **Busca (Fetch):** O Contador de Programa (PC) endereça a memória de instruções.

2. **Decodificação (Decode):** A instrução é quebrada em campos; os registradores fonte são lidos e valores imediatos são estendidos.

3. **Execução (Execute):** A ALU realiza operações aritméticas/lógicas ou calcula endereços de memória.

4. **Memória (Memory):** Dados são lidos ou escritos na Memória de Dados (se aplicável).

5. **Escrita (Write-Back):** O resultado é escrito de volta no banco de registradores.

Para uma mesma ISA, existem infinitas microarquiteturas possíveis, que variam em desempenho, área e consumo de energia. As organizações clássicas são:

1. **Monociclo** (*Single-cycle*): executa cada instrução em um único ciclo de clock, no qual todas as etapas - busca, decodificação, execução, acesso à memória e escrita de resultado - ocorrem sequencialmente. Embora conceitualmente simples, seu desempenho é limitado pelo caminho crítico mais longo do datapath (tipicamente associado a instruções de *load*), o que impõe um período de clock elevado e, consequentemente, uma baixa frequência de operação;

2. **Multiciclo** (*Multi-cycle*): fragmenta a execução das instruções em uma sequência de etapas menores, reutilizando os mesmos recursos de hardware ao longo de múltiplos ciclos. Cada instrução pode demandar um número variável de ciclos para ser concluída, o que resulta em um índice de ciclos por instrução $CPI > 1$, porém permite uma redução significativa do caminho crítico e do período de clock;

3. **Pipeline**: organiza a execução das instruções em estágios sobrepostos, de modo que múltiplas instruções possam estar em processamento simultaneamente. Essa abordagem maximiza o paralelismo temporal e constitui o padrão adotado em processadores de alto desempenho. Entretanto, introduz desafios adicionais, como conflitos estruturais, de controle e de dados (*hazards*), além da necessidade de técnicas para mitigação de penalidades associadas a desvios de controle, como a previsão de desvios (*branch prediction*). Idealmente, um processador pipeline apresenta *CPI* próximo de 1, embora esse valor raramente seja atingido na prática;

4. **Superescalar**: estende o modelo pipeline ao permitir a emissão e execução de múltiplas instruções por ciclo de clock, explorando paralelismo em nível de instruções (*Instruction-Level Parallelism - ILP*). Essa abordagem requer lógica adicional para despacho, escalonamento e detecção de dependências, aumentando significativamente a complexidade do hardware;

5. **Execução fora de ordem** (*Out-of-Order Execution*): permite que instruções sejam executadas fora da ordem original do programa, desde que as dependências de dados e de

controle sejam respeitadas. Essa técnica melhora a utilização dos recursos de execução e reduz bolhas no pipeline, à custa de estruturas complexas como filas de reordenação (*Reorder Buffer - ROB*) e mecanismos de renomeação de registradores;

6. **Arquiteturas multithreaded**: suportam múltiplos contextos de execução concorrentes em um mesmo núcleo, alternando ou sobrepondo a execução de diferentes *threads* para mascarar latências, especialmente de acesso à memória. Exemplos incluem *fine-grained multithreading* e *simultaneous multithreading* (SMT);

7. **Arquiteturas multicore**: integram múltiplos núcleos de processamento em um único sistema, explorando paralelismo em nível de tarefas (*Thread-Level Parallelism - TLP*). Essa abordagem impõe desafios adicionais relacionados à coerência de cache sincronização e comunicação entre núcleos.

Este trabalho adota uma organização **multiciclo** para o núcleo de processamento. Essa escolha permite a operação em frequências de clock superiores às de uma implementação monociclo, uma vez que o período de clock passa a ser limitado pelo estágio mais lento do datapath, e não pela soma das latências de todas as etapas de uma instrução.

Além disso, a arquitetura multiciclo evita a complexidade adicional de hardware inerente a processadores *pipelined*, como a necessidade de mecanismos para tratamento de *hazards* e controle refinado de desvios, o que simplifica tanto o processo de depuração quanto a análise didática da microarquitetura.

Outro benefício dessa abordagem é o reuso eficiente de componentes microarquiteturais ao longo dos diferentes estágios de execução, reduzindo a área de hardware ao evitar a duplicação de unidades funcionais que permaneceriam ociosas em determinados ciclos. Adicionalmente, o modelo multiciclo oferece maior flexibilidade no tratamento da latência de acesso à memória, uma vez que operações de leitura e escrita podem ser naturalmente distribuídas ao longo de múltiplos ciclos.

Considerações relacionadas à temporização do sistema, aos atrasos de propagação dos componentes e ao acesso às memórias serão discutidas em maior detalhe em seções posteriores.

2.3 Sistemas-em-Chip (SoC) e Interconexões

Um sistema-em-chip (SoC - *System-on-Chip*) integra, em um único circuito, todos os componentes necessários para um sistema computacional funcional: processador, memória, interfaces de entrada/saída (E/S) e aceleradores. O desafio central no projeto de um SoC não é apenas o design dos componentes individuais, mas a eficiência da comunicação entre eles.

2.3.1 Barramento de Interconexão

A comunicação em um SoC é tipicamente feita através de um barramento (*bus*) - onde um protocolo define o *handshaking* para leitura e escrita. Protocoolos industriais comuns incluem o AMBA (AHB/APB) da ARM e o TileLink. Neste projeto, implementa-se um protocolo de barramento customizado simplificado.

2.3.2 Acesso Direto à Memória (DMA)

Em sistemas onde a CPU é o único mestre do barramento, a movimentação de grandes blocos de dados (e.g. copiar uma imagem da memória para um acelerador) monoopoliza o processador, impedindo-o de realizar trabalho útil. O controlador de DMA (*Direct Memory Access*) resolve esse problema. Ele é um mestre secundário no barramento capaz de realizar transferências de dados de forma autônoma. A CPU configura o DMA - informando o endereço de origem, endereço de destino e o tamanho do bloco de dados - e é liberada para outras tarefas, recebendo uma interrupção apenas quando a transferência termina.

2.3.3 CSRs de Interface

2.3.4 Arbitragem

2.4 Aceleração de Hardware e Redes Neurais

A popularização da utilização de redes neurais e aprendizagem profunda expôs as limitações das CPUs de propósito geral. O treinamento e a inferência de redes neurais são dominados por operações de álgebra linear, especificamente a multiplicação de matrizes (GEMM - *General Matrix Multiply*).

2.4.1 Gargalo de Von Neumann

Processadores tradicionais buscam dados e instruções da memória sequencialmente. [...]

2.4.2 Arquitetura de Arranjo Sistólico

Para mitigar esse gargalo, aceleradores como o Google TPU (*Tensor Processing Unit*) utilizam a arquitetura de Arranjo Sistólico (*Systolic Array*).

2.5 VHDL e Projeto de Hardware

A descrição de hardware deste projeto foi realizada utilizando a linguagem **VHDL** (*VHSIC Hardware Description Language*). Diferente de linguagens de programação de software,

que descrevem sequências de instruções, o VHDL é uma linguagem de descrição de hardware utilizada para modelar o comportamento concorrente e a estrutura de circuitos digitais.

Para este trabalho, adotou-se especificamente o padrão **VHDL-2008** (IEEE 1076-2008). Esta revisão da linguagem introduz construções modernas que aumentam a legibilidade e a capacidade de síntese do código, permitindo uma descrição mais concisa de lógicas complexas sem sacrificar o rigor tipológico característico da linguagem.

2.6 Tecnologia de FPGA

Field-Programmable Gate Arrays (FPGAs) são diispositivos semicondutores baseados em uma matriz de blocos lógicos configuráveis (CLBs) conectados por interconexões programáveis. Diferente de processadores que executam instruções de esoftware seqeuencialmente, o FPGA permite a construção de circuitos digitais verdadeiramente paralelos em nível de hardware.

A natureza reconfigurável do FPGA o torna a plataforma ideal para prototipagem de SoCs e desenvolvimento de arquiteturas experimentais (como a proposta neste trabalho), permitindo validação física do cicruito sem os custos proibitivos de fabricação de um ASIC (*Application-Specific Iintegrated Circuit*).

2.7 Ferramentas de Ensino em Engenharia

3 DESENVOLVIMENTO

3.1 Metodologias de Projeto

A complexidade inerente ao desenvolvimento de um microprocessador exige uma metodologia de projeto rigorosa e estruturada. Este trabalho seguiu uma abordagem de desenvolvimento **Bottom-Up** (de baixo para cima) combinada com **Prototipagem Evolutiva**.

3.1.1 Desenvolvimento Modular e Hierárquico

O sistema foi decomposto em módulos fundamentais de menor complexidade. O desenvolvimento iniciou-se pelos componentes base do *datapath* (Multiplexadores, ALU, Banco de Registradores), progredindo para a Unidade de Controle e, finalmente, para a integração no nível de topo (*Top Level*). Cada módulo foi projetado como uma entidade independente, com interfaces (*ports*) claramente definidas, facilitando o isolamento de falhas.

3.1.2 Estratégia de Verificação e Simulação

Dada a dificuldade de depurar erros diretamente no hardware físico, adotou-se a abordagem de testes através de simulações (*textbenches*). Nenhum componente foi integrado ao sistema principal sem antes ser validado individualmente.

O ambiente de simulação utilizado foi o **GHDL**, um simulador *open-source* de alto desempenho, integrado a um fluxo de automação via **GNU Make**. Isso garante que todo o processo de compilação, simulação e visualização de ondas (via GTKWave) seja reprodutível e automatizado.

3.2 Arquitetura de Hardware

3.3 Ecossistema de Software (BARE-METAL)

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a,

ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

3.3.1 Toolchain RISC-V e Compilação

3.3.2 Automação com Makefile

3.3.3 Integração com simulação e verificações

3.4 Implementação em FPGA

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

3.4.1 Síntese

3.4.2 Pinagem e restrições

3.4.3 Testes em hardware

3.5 Ecossistema Educacional e Ferramentas

4 RESULTADOS E DISCUSSÃO

4.1 Validação Funcional e Simulação

4.2 Resultados de Síntese em FPGA

4.3 Análise de Desempenho (Benchmarks)

4.4 Avaliação do Uso Educacional

5 CONCLUSÃO

5.1 Considerações Finais

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

5.2 Trabalhos Futuros

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras

ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

REFERÊNCIAS

PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization and Design RISC-V Edition: The Hardware/Software Interface**. 2. ed. [S.l.]: Morgan Kaufmann, 2021.

RISC-V International. **The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA**. [S.l.], 2019. Version 20191213.

Roa Logic B.V. **RV12 RISC-V 32/64-bit CPU Core Datasheet**. [S.l.], 2018. Disponível online: <https://www.roalogic.com/>.