Validando em FPGA um Core RISC-V dedicado à Simulação de Circuitos Quânticos

Thiago Andrade [1], **Breno Nascimento** [2], **Calebe Conceição** [3], **Rodolfo Garcia** [4]

[1]Departamento de Computação – Universidade Federal de Sergipe (UFS)
Av. Marechal Rondon, s/n – Jardim Rosa Elze – CEP 49100-000
São Cristóvão – Sergipe – Brasil

thiago.andrade@dcomp.ufs.br,breno.nascimento@dcomp.ufs.br

calebe@dcomp.ufs.br, rodolfo.botto@dcomp.ufs.br

Abstract. *The cost of developing a quantum computer is currently high. In this context, this article aims to describe the processes and results of research focused on developing a RISC-V core dedicated to quantum circuit simulation. The adopted methodology involved simplifying the Tiny RISC-V processor and conducting tests and validations on a Field-Programmable Gate Array (FPGA). The results are promising, indicating the feasibility of instantiating multiple cores to explore parallelism to perform quantum circuits simulation, as an alternative to GPU simulation.*

Resumo. *O custo para o desenvolvimento de um computador quântico é alto nos dias atuais. Nesse contexto, este artigo tem como objetivo descrever os processos e resultados de uma pesquisa voltada ao desenvolvimento de um núcleo RISC-V dedicado à simulação de circuitos quânticos. A metodologia adotada envolveu a simplificação do processador Tiny RISC-V e a realização de testes de desempenho em um Arranjo de Portas Programáveis em Campo (FPGA). Os resultados obtidos são promissores, indicando a viabilidade de instanciar múltiplos cores para explorar o paralelismo para realizar a simulação de circuitos quânticos, como uma alternativa à simulação em GPU.*

1. Intrudução

Em 2007, a empresa D-Wave Systems apresentou o primeiro protótipo funcional de computador quântico. Posteriormente, lançou em 2011 o D-Wave One, considerado o primeiro computador quântico comercial disponível no mercado [Nunes 2016]. Esse lançamento aguçou o interesse de grandes empresas de computação, como Google e IBM, que hoje perseguem o domínio da tecnologia que promete avanços na realização de cálculos complexos com significativa maior eficiência em comparação aos computadores tradicionais. No entanto, a adoção dessa tecnologia ainda enfrenta desafios importantes, especialmente os relacionados ao custo de acesso. Segundo o site Spinquanta, O computador quântico mais barato do mundo é 2-qubit Portable NMR Quantum Computer que custa de 8700 a 9000 dólares [SpinQuanta 2024] e computadores quânticos supercondutores podem chegar a 50 milhões de dólares [SpinQuanta 2025], além de demandarem grande espaço físico e apresentarem dificuldades de alocação, o que limita seu acesso a instituições de ensino e pesquisa.

Diante desse cenário, surge a necessidade de soluções mais acessíveis para explorar a computação quântica. A simulação é um caminho ainda necessário, mas enfrenta desafios de eficiência,, tendo em vista a demanda exponencial de recursos computacionais que implicam em maior tempo de execução. Soluções paralelas apresentam-se como promissoras neste sentido, como aquelas baseadas em GPUs. O objetivo deste trabalho é o desenvolvimento de um *core* RISC-V — uma unidade de processamento baseada na arquitetura de conjunto de instruções RISC-V — dedicada à simulação de circuitos quânticos, com foco na investigação sobre impactos na economia de recursos lógicos e na redução do custo de implementação.

Este artigo está organizado da seguinte forma:a Seção 2 relaciona o trabalho a outros existentes; a Seção 3 apresenta a fundamentação teórica sobre computação quântica e a arquitetura RISC-V; a Seção 4 descreve os métodos e ferramentas utilizados no desenvolvimento do projeto; a Seção 5 discute os Experimentos realizados; a Seção 6 discute os resultados obtidos; e, por fim, a Seção 7 apresenta as conclusões e propostas de trabalhos futuros.

2. Trabalhos Relacionados

O RISC-V é uma instruction set architecture (ISA) open-source e livre de royalties, caracterizada por sua modularidade e extensibilidade, o que viabiliza adaptações para aplicações específicas. A literatura sobre o tema é extensa, como exemplificado por [Cui et al. 2023], que apresenta um mapeamento das principais pesquisas relacionadas ao uso de extensões na arquitetura. O estudo evidencia sua ampla adoção em microprocessadores embarcados, processadores de domínio específico e sistemas de alto desempenho, abordando extensões privilegiadas e não privilegiadas.

Entre os trabalhos aplicados, [Zhao et al. 2023] utiliza a extensão vetorial em algoritmos de criptografia pós-quântica, enquanto [Zhao et al. 2022] propõe uma extensão customizada para operações matriciais voltadas a aplicações criptográficas. De forma complementar, [Wang et al. 2024] demonstra os ganhos de desempenho proporcionados por extensões customizadas. Em contraste, neste trabalho optamos por não empregar extensões adicionais, com o objetivo de reduzir o número de operadores lógicos e otimizar o uso dos recursos de hardware.

Um trabalho relacionado é apresentado por [Gebauer et al. 2021], que descreve um firmware modular em FPGA para controle de qubits supercondutores. Cada célula digital integra um núcleo RISC-V com um sequenciador, dois geradores de sinal e um registrador, comunicando-se por um barramento Wishbone modificado. A ISA utilizada inclui o conjunto base, extensões de multiplicação e uma extensão customizada para sequenciamento. Em nosso projeto, por outro lado, eliminamos inclusive as instruções básicas da ISA RISC-V, buscando minimizar a complexidade lógica do processador.

3. Fundamentação Teórica

Este trabalho concentra esforços em desenvolver um core RISC-V dedicado à simulação de circuitos quânticos. Sendo assim, é necessário apresentar a fundamentação sobre simulação de sistemas quânticos que se apresentam como requisitos para o core proposto, bem como as características da arquitetura RISC-V e o modelo de referência a partir do qual a customização proposta foi elaborada.

3.1. Simulação de circuitos quânticos

Diferentemente da computação clássica, na qual cada bit pode assumir apenas os estados discretos 0 ou 1, na computação quântica um bit quântico (ou qubit) é descrito como uma combinação linear desses dois estados base de um espaço vetorial complexo. Desse modo, um qubit possui coeficientes complexos associados a cada um dos estados base $|0\rangle$ e $|1\rangle$.

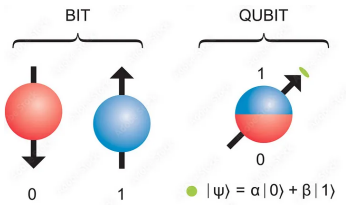

Figura 1. Comparação entre bits clássicos e qubits. Fonte: [NLP 2023]

A notação bra-ket, introduzida por Paul Dirac, permite representar de forma clara as amplitudes e funções de onda dos estados quânticos. Essa notação é vantajosa por possibilitar a descrição de estados quânticos como vetores coluna em um espaço de Hilbert, simplificando o arcabouço matemático que descreve o sistema computacional quântico. [Nielsen and Chuang 2010]. Uma operação em um sistema quântico isolado pode ser descrita como uma multiplicação de matriz unitária U pelo vetor de estado $|\psi\rangle$ que representa o sistema, representada na notação de Dirac como $U|\psi\rangle$.

Um algoritmo quântico nada mais é do que a aplicação dessas transformações unitárias sobre o vetor de estado de forma sequencial e controlada para obter um resultado previsível. Uma forma de representar um algoritmo quântico é por meio de um circuito quântico, como o exemplificado na Figura 2. As linhas correspondem aos qubits, as operações são representadas pelas caixas, cuja disposição da esquerda para a direita representa a ordem em que são realizadas. Algumas operações são aplicadas sobre mais de um qubit.

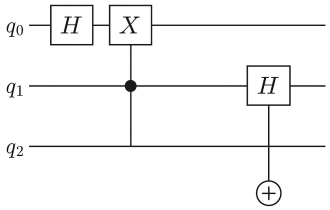

Figura 2. Exemplo de Circuito Quântico com Operações Básicas de Superposição e Controle.

A cada passo do algoritmo quântico é necessário expandir a operação matricial que é realizada sobre todos os qubits como uma combinação por produto Kronecker da matriz da porta quântica aplicada naquele instante com a operação identidade que é aplicada sobre os qubits que não sofrem a operação. Em um sistema com N qubits, a matriz de operação resultante é uma matriz quadrada complexa de dimensão 2^N. Uma forma de realizar a simulação de um dado algoritmo é computando a sequência encadeada das

multiplicações matriciais correspondentes a cada passo. Nesse contexto, o desafio computacional reside no incremento exponencial que a adição de um único qubit no sistema impacta na dimensão dessas matrizes.

3.2. O processador Tiny RISC-V

Para compreender o contexto do trabalho proposto, é fundamental compreender a arquitetura RISC-V, em especial a versão do processador *Tiny RISC-V*. Trata-se de um processador de pequeno porte e baixa complexidade, projetado para fins educacionais e de pesquisa, servindo como uma plataforma acessível ao estudo de arquiteturas de processadores. O *Tiny RISC-V* adota a arquitetura RISC-V e oferece suporte ao conjunto de instruções RV32I, correspondente ao núcleo de instruções inteiras de 32 bits. Ele utiliza um barramento de memória do tipo Harvard, no qual as memórias de instrução e de dados são separadas, permitindo acessos simultâneos e independentes e não há suporte a operações de ponto flutuante.

Sua arquitetura pode ser implementada em diferentes configurações — ciclo único, multiciclo ou com suporte a pipeline — de acordo com a complexidade desejada no projeto. Em termos de componentes, o processador é composto por uma unidade de controle, um banco de registradores, uma unidade lógico-aritmética (ULA), um contador de programa (PC), memória de instruções e memória de dados. A interação entre esses elementos se dá por meio de um *datapath*, conforme ilustrado na Figura 3.

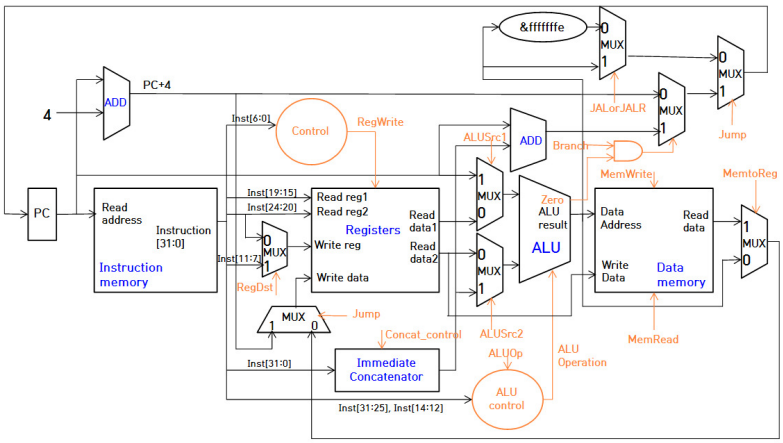

Figura 3. Diagrama estrutural do processador Tiny RISC-V. Fonte: [Hushon 2020].

O processador opera com instruções de 32 bits, divididas em campos específicos: *Opcode* (identifica a operação), campos de registradores (especificam os registradores-fonte e destino), campos de função (refinam a operação) e campos imediatos (valores constantes utilizados em operações aritméticas ou de endereçamento) [Patterson and Hennessy 2017]. Esses campos são decodificados pela unidade de controle, que gera sinais para seleção da operação na ULA (que realiza operações aritméticas como soma, subtração, e lógica como AND, OR, XOR, NOT), ativação da escrita no banco de registradores e controle de acesso à memória (*load*/*store*).

Conforme descrito em [John 2017], as instruções são organizadas em categorias de acordo com os sinais de controle exigidos e os componentes lógicos envolvidos. A

arquitetura RISC-V define seis categorias principais de instruções em sua implementação básica (RV32I), agrupadas em dois blocos funcionais:

Instruções de Núcleo

- **Tipo R (Register-Register):** operações aritméticas e lógicas entre registradores, como adição e operações booleanas.
- **Tipo I (Immediate):** operações com registrador e valor imediato, incluindo acessos à memória (*loads*) e saltos relativos.
- **Tipo S (Store):** operações de armazenamento em memória, calculando o endereço a partir da soma entre um registrador base e um deslocamento imediato.
- **Tipo U (Upper Immediate):** manipulação de constantes e endereços, inserindo valores de 20 bits nos bits mais significativos do registrador.

Instruções de Controle de Fluxo

- **Tipo B (Branch):** desvios condicionais baseados em comparações entre registradores, com deslocamento relativo.
- **Tipo J (Jump):** saltos incondicionais de longo alcance, úteis para chamadas de função e retornos.

As informações contidas na instrução de 32 bits são interpretadas de forma combinacional, de acordo com o *opcode*, conforme ilustrado na Tabela 1.

Tabela 1. Formatos das instruções base do RISC-V conforme apresentado em [Harris and Harris 2021].

Formato	31–25	24–20	19–15	14–12	11–7	6–0
R-type	func7	rs2	rs1	func3	rd	opcode
I-type	imm[11:0]		rs1	func3	rd	opcode
S-type	imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode
B-type	imm[12—10:5]	rs2	rs1	func3	imm[4:1—11]	opcode
U-type	imm[31:12]				rd	opcode
J-type	imm[20—10:1—11—19:12]				rd	opcode

4. Metodologia

Nesta pesquisa, investiga-se a viabilidade de implementar uma customização do *Tiny RISC-V* que mantém apenas as estruturas que dão suporte às operações de multiplicação de matrizes necessárias à simulação de algoritmos quânticos descritos no modelo de circuitos.

A primeira etapa da pesquisa consiste em selecionar quais operações são necessárias à multiplicação de matrizes. Para isso, foi implementado em C um código para multiplicação de matriz, esse que pode ser observado abaixo, e analisado seu código assembly do RISC-V gerado por meio da compilação usando o compilador GCC (*GNU Compiler Collection*) com o comando `gcc -S mulmatrizes.c`. O código gerado foi então refinado com o auxílio do simulador RARS – *RISC-V Assembler and Runtime Simulator* – , com o objetivo de reduzir ao máximo o conjunto de instruções necessárias, mantendo a funcionalidade. Neste estudo, verificou-se que era necessário manter apenas o suporte a instruções dos tipos **R**, **I** e **S** na arquitetura.

A segunda etapa consistiu na customização da arquitetura do *Tiny RISC-V*. Para realizar simulação funcional, foi usado o suporte do simulador *Icarus Verilog* e do visualizador de forma de onda *GTKwave*, visando à eliminação dos componentes lógicos dedicados ao processamento das instruções dos tipos **U**, **B** e **J**. O objetivo é avaliar o impacto no desempenho e na complexidade do circuito, e quantificar os ganhos em termos de elementos lógicos necessários à implementação da arquitetura. Com isso, almeja-se realizar uma avaliação inicial do custo-benefício para o desenvolvimento de um processador dedicado, em aprofundamentos futuros.

```
vector<vector<float>> multiplicarMatrizes(const vector<vector<
   float>>& A, const vector<vector<float>>& B) {
    int linhasA = A.size(), colunasA = A[0].size(), colunasB = B
       [0].size();
    vector<vector<float>> resultado(linhasA, vector<float>(
       colunasB, 0.0f));
    for (int i = 0; i < linhasA; i++)
        for (int j = 0; j < colunasB; j++)
            for (int k = 0; k < colunasA; k++)
                resultado[i][j] += A[i][k] * B[k][j];
    return resultado;
}
...
```

5. Experimentos realizados

O Tiny RISC-V customizado no contexto deste trabalho apresenta as seguintes características fundamentais:

- Banco de 32 registradores de propósito geral
- Arquitetura Harvard modificada com:
  - Memória de instruções (ROM)
  - Memória de dados (RAM)
- Módulo de memória unificado com portas diferenciadas por função
- Suporte completo ao conjunto de instruções RV32I

5.1. Implementação Inicial

Desenvolvemos um algoritmo em assembly RISC-V utilizando exclusivamente instruções dos tipos:

- **R**: Operações registrador-registrador
- **I**: Operações com imediato
- **S**: Armazenamento em memória

Este código implementa multiplicação matricial com entrada direta por meio de um computador hospedeiro, conforme ilustrado na Figura 5.

5.2. Otimização da Arquitetura

Foram realizadas simplificações no processador base Tiny RISC-V por meio da remoção dos circuitos dependentes das instruções dos tipos U, B e J. Essas modificações foram

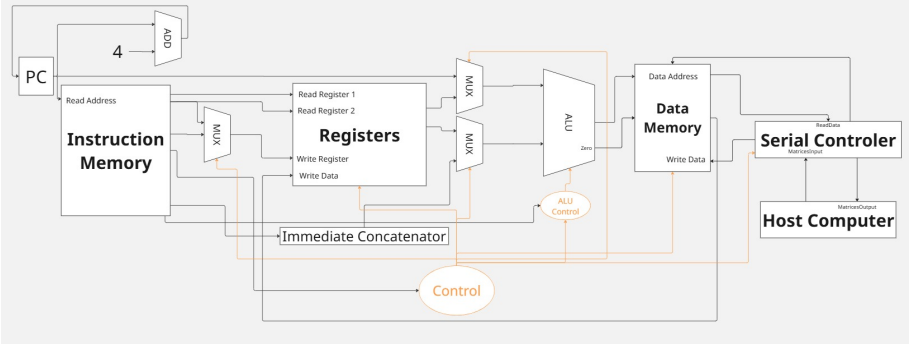

Figura 4. Fornecimento de dados do computador hospedeiro para o processador.

implementadas por meio de alterações no código-fonte no Visual Studio Code, especialmente nas lógicas de multiplexação presentes na Unidade Lógica Aritmética (ULA), na Unidade de Controle Principal, no Contador de Programas (PC) e no Caminho de Dados (Datapath), que originalmente dependiam dessas instruções. Por exemplo, no Tiny RISC-V original.

Por exemplo:

```
assign NXTPC = (~RSTn)? 0 : (Jump)? ( (JALorJALR)? JALRAddress :
    JALAddress ) : ((BranchTaken)? BranchTarget : PC+4 );
```

Essa era a linha responsável por controlar qual seria o próximo valor atribuído ao Contador de Programas. Como pode ser observado, havia diversos sinais dependentes das operações do tipo B e J. Na nossa versão final, removemos a dependência do processador em relação a esses sinais que eram atribuídos na Unidade de controle.

```
assign NXTPC = (~RSTn)? 0 : (PC+4 );
```

5.3. Validação em FPGA

Para a validação em FPGA, foi implementado um sistema completo baseado em um processador RISC-V personalizado com interface serial RS-232, conforme detalhado na Figura 5. Esta arquitetura integra três componentes principais: o núcleo de processamento RISC-V, o controlador serial dedicado e a interface de comunicação com o computador hospedeiro.

Essa implementação utiliza comunicação RS-232, um padrão de comunicação serial assíncrona para permitir a troca de dados entre computadores e dispositivos periféricos, configurada a 115.200 bps, velocidade máxima estável. O protocolo emprega os sinais convencionais TXD (transmissão) e RXD (recepção).

O fluxo de operação é governado por uma máquina de estados finitos com três estados principais. No estado inicial (*IDLE*), o sistema aguarda a recepção dos valores das matrizes de entrada, que são enviados em uma sequência específica a_1, b_1, c_1, d_1 para a primeira matriz, seguidos por a_2, b_2, c_2, d_2 para a segunda matriz. Todos os valores são armazenados na memória de dados, utilizando representação em ponto fixo Q16.8, onde 16 bits representam a parte inteira e outros 8 bits a parte fracionária. Devido à

limitação decorrente da ausência de instrução de multiplicação no Tiny RISC-V, os 8 bits mais significativos ficaram inutilizados para uso na multiplicação, sendo usados para o processamento interno do produto.

$$\begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix} \times \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix}$$

Quando a última entrada é recebida, o sistema automaticamente transiciona para o estado *RUN*, onde o processador RISC-V inicia a execução do programa de multiplicação matricial previamente carregado na memória de instruções enquanto nos estados anteriores. Este programa realiza as operações necessárias para calcular o produto das matrizes e armazena o resultado na memória de dados.

Após a conclusão do processamento, o sistema entra no estado *DONE*, onde os resultados são lidos da memória e enviados de volta ao computador hospedeiro através da mesma interface serial. Nesta fase, ocorre o processo inverso ao de recepção, com cada valor de 32 bits sendo dividido em quatro transmissões seriais de 8 bits. Todo esse fluxo é coordenado por sinais de controle que garantem a sincronização entre os diferentes componentes do sistema.

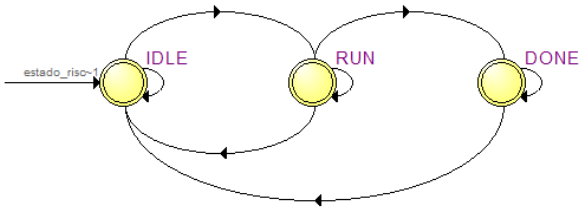

Figura 5. Diagrama de transição de estados.

Para validar o correto funcionamento da implementação, foram realizados testes extensivos tanto em simulação quanto na plataforma física. Na fase de simulação, foi possível verificar o comportamento da máquina de estados, o armazenamento correto dos dados na memória e a precisão dos cálculos realizados. Já nos testes práticos, matrizes com valores conhecidos foram enviadas ao sistema, com os resultados sendo comparados com cálculos de referência. Por exemplo, ao processar as matrizes $\left(\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right)$ e $\left(\begin{smallmatrix} 5 & 6 \\ 7 & 8 \end{smallmatrix}\right)$, o sistema retornou corretamente $\left(\begin{smallmatrix} 19 & 22 \\ 43 & 50 \end{smallmatrix}\right)$. Adicionalmente, como as memórias usadas foram geradas usando o recursos Mega Wizard Plugin, pertmiu o uso do visualizador de memória do Quartus II para inspecionar internamente os valores armazenados, oferecendo uma segunda camada de verificação.

6. Resultados

6.1. Avaliação de Desempenho

A análise comparativa foi realizada utilizando o *Quartus Prime*, com foco em:

- Redução no número de elementos lógicos (GE - *Gate Equivalents*)
- Impacto no consumo de memória

Tabela 2. Resultados comparados após alterações

Versão do Design	Elementos Lógicos	Memória (bits)
Processador Personalizado	4383	65536
Tiny RISC-V	4604	32768

Nos testes realizados foi percebido que houve, de fato, uma redução na quantidade de elementos lógicos no processador em cerca de 5%, de 4.604 para 4.383, como consequência da remoção das instruções **U**, **B** e **J**. O maior impacto ocorreu na Unidade Lógico-Aritmética e no Concatenador de Imediato, que passaram de 789 e 60 elementos para 542 e 7, respectivamente.

Contudo, houve um impacto negativo na memória, cujo consumo aumentou de 32.768 para 65.536 bits. Esse aumento se justifica pelo crescimento no número de linhas de instrução do código de multiplicação de matrizes, que passou de 46 para 2.003 linhas, a fim de contornar a ausência dos tipos de instrução mencionados anteriormente. Esse problema pode ser facilmente contornado ao fazer uso de uma das memórias físicas externas ao chip FPGA, disponíveis na placa. A Tabela 2 sumariza os resultados experimentais alcançados.

7. Conclusões e trabalhos futuros

Com o estudo desenvolvido, pode-se concluir que realmente é possível economizar elementos lógicos ao customizar o processador de propósito geral para um propósito específico. Foi possível ainda verificar que a eliminação de instruções pode implicar na perda de otimizações no código assembly necessário para implementar uma solução. Ainda que isso tenha gerado um impacto na memória, que demandou um incremento para abarcar o código assembly mais extenso da aplicação, a proposta não fica inviabilizada, considerando que ainda é possível utilizar da memória física disponível na placa FPGA ou ainda de esquemas de hierarquia de memória.

Vale destacar que o processador customizado implementado neste trabalho foi concebido para atuar como um co-processador dedicado à multiplicação de matrizes, visando descarregar essa operação computacionalmente intensiva de uma CPU principal. A escolha pela implementação em FPGA permite otimizações específicas para operações matriciais, como paralelismo de cálculos e acesso eficiente à memória, características que seriam menos eficientes em processadores de propósito geral.

Planejamos analisar os *trade-offs* entre completude funcional e otimização de hardware, especialmente relevante para aplicações específicas para computação quântica, que demandam operações matriciais intensivas. Também cabe destacar que para esta aplicação, o espaço de operações é o espaço de Hilbert, que trabalha com coeficientes complexos. Adicionalmente, é usual que os valores desses coeficientes estejam limitados ao intervalo [-1, 1]. Por conta disso, pretende-se como trabalhos futuros implementar duas melhorias fundamentais no núcleo do processador: o suporte nativo para números complexos, permitindo que o co-processador execute multiplicações matriciais com elementos complexos de forma direta e eficiente; e adicionar uma unidade de ponto fixo otimizada para operações no intervalo [-1, 1]. Ao limitar a faixa de valores conhecida, é

possível dedicar mais bits para a parte fracionária, aumentando a precisão sem sacrificar desempenho.

Referências

Cui, E., Li, T., and Wei, Q. (2023). Risc-v instruction set architecture extensions: A survey. *IEEE Access*, 11:24696–24711.

Gebauer, R., Karcher, N., and Sander, O. (2021). A modular rfsoc-based approach to interface superconducting quantum bits. In *2021 International Conference on Field-Programmable Technology (ICFPT)*, pages 1–9.

Harris, D. M. and Harris, S. L. (2021). *Digital Design and Computer Architecture: RISC-V Edition*. Morgan Kaufmann.

Hushon (2020). Tiny-riscv-cpu: A simple risc-v cpu written in verilog. `https://github.com/hushon/Tiny-RISCV-CPU`. Acesso em: jun. 2025.

John, L. (2017). *Computer Organization and Design Risc-v Edition-the Hardware Software Int*. Elsevier Science & Technology.

Nielsen, M. A. and Chuang, I. L. (2010). *Quantum Computation and Quantum Information*. Cambridge University Press, 10th anniversary edition edition.

NLP, R. (2023). Bases matemáticas da computação quântica: Álgebra linear, superposição e entrelaçamento. `https://medium.com/@recogna.nlp/bases-matem%C3%A1ticas-da-computa%C3%A7%C3%A3o-qu%C3%A2ntica-%C3%A1lgebra-linear-superposi%C3%A7%C3%A3o-e-entrela%C3%A7amento-7d8b10dda10f`. Acessado em: 16 jun. 2025.

Nunes, J. (2016). Computadores quânticos, informação e computação quânticas. *Correio dos Açores*, pages 17–17.

Patterson, D. A. and Hennessy, J. L. (2017). *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Morgan Kaufmann. Acesso em: 21 jun. 2025.

SpinQuanta (2024). World's cheapest quantum computer: Affordable for quantum education. Accessed: 2025-06-20.

SpinQuanta (2025). Superconducting quantum computer price range: Full overview. Accessed: 2025-06-20.

Wang, S., Wang, X., Xu, Z., Chen, B., Feng, C., Wang, Q., and Ye, T. T. (2024). Optimizing cnn computation using risc-v custom instruction sets for edge platforms. *IEEE Transactions on Computers*, 73(5):1371–1384.

Zhao, Y., Kuang, H., Sun, Y., Yang, Z., Chen, C., Meng, J., and Han, J. (2023). Enhancing risc-v vector extension for efficient application of post-quantum cryptography. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 10–17.

Zhao, Y., Xie, R., Xin, G., and Han, J. (2022). A high-performance domain-specific processor with matrix extension of risc-v for module-lwe applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(7):2871–2884.