Revisitando e adicionando portas a um processador dedicado à simulação de algoritmos quânticos em hardware

Pedro J. Silveira[1], **Breno S. Nascimento**[1], **Calebe M. Conceição**[1], **Rodolfo B. Garcia**[1]

[1]Departamento de Computação (DCOMP) – Universidade Federal de Sergipe (UFS)
Av. Marechal Rondon, – Jardim Rosa Elze – CEP 49100-000
São Cristóvão – SE – Brazil

pedro.silveira@dcomp.ufs.br, breno.nascimento@dcomp.ufs.br

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

Abstract. *Current approaches to implementing quantum-algorithm simulations in hardware suffer from performance and complexity issues. In this article, we revisit an FPGA implementation of a dedicated 3-qubit processor that supports the S gate, T gate, Hadamard gate, Pauli-X gate, CNOT gate, and Toffoli gate. The proposed architecture combines high performance with design simplicity by enabling the efficient simulation of a quantum circuit's behavior through a programmable, RISC-inspired model. This eliminates the need to re-synthesize the design, reduces potential errors, and overcomes some of the limitations of existing solutions. Such an approach could represent a viable alternative for the development and validation of new quantum-based systems.*

Resumo. *As atuais abordagens para implementação de simulação algoritmos quânticos em hardware sofrem de problemas relacionados a desempenho e complexidade. Neste artigo, revisitamos uma implementação em FPGA de um processador dedicado de 3 qubits para dar suporte às portas quânticas Sgate, Tgate, Hadamard, Paulix, CNOT, e Toffoli. A arquitetura proposta alia alto desempenho à simplicidade de projeto, já que permite simular o comportamento de um circuito quântico de forma eficiente com um modelo programável inspirado na arquitetura RISC, eliminando a necessidade de re-síntese do modelo, reduzindo possíveis erros e algumas limitações de outras soluções existentes. Tal abordagem pode representar uma alternativa viável ao desenvolvimento e validação de novas soluções baseadas em sistemas quânticos.*

1. Introdução

A computação quântica se destaca como uma das tecnologias emergentes mais promissoras da Computação, pois sua natureza exponencial confere a ela a capacidade de resolver alguns problemas intratáveis para sistemas clássicos [Feynman 1986]. No entanto, essa mesma característica impõe limitações físicas à simulação de sistemas quânticos em arquiteturas clássicas, o que dificulta sua escalabilidade [Maron et al. 2013].

Por outro lado, dispositivos reconfiguráveis oferecem a flexibilidade necessária para simular um ambiente quântico com baixa latência, pipelines configuráveis e alto grau de paralelismo. Assim, as FPGAs surgem como uma plataforma promissora para essa tarefa [Fujishima et al. 2003] [Khalid et al. 2004]. Neste trabalho, apresentamos o

projeto em SystemVerilog de seis portas elementares (S, T, Hadamard, Pauli-X, Toffoli e CNOT) em um processador dedicado, a sua síntese em FPGA e uma comparação entre os resultados de simulação para (ModelSim) e a execução na placa-alvo.

Propõe-se esse conjunto de portas como um aprimoramento de uma solução inspirada na arquitetura RISC para a simulação de um processador quântico em FPGA apresentada em [Conceição and Reis 2015]. O design utiliza um conjunto reduzido de instruções com formato uniforme, garantindo baixa latência e escalabilidade. Além disso, a estrutura parametrizada permite a integração de pipelines mais complexos, explorando o alto grau de paralelismo inerente às FPGAs.

Como metodologia de implementação e validação,são utilizadas duas abordagens: em hardware, na FPGA Cyclone IV DE2i-150, e em software, no ModelSim. Além disso, como ambiente de teste e verificação funcional, foi adotada uma abordagem de alta diversidade na geração de vetores de teste, usando números aleatórios para cada porta quântica. Mapeamos os sinais de entrada e saída e acompanhamos suas variações ao longo do tempo, confirmando que cada porta operava em um único ciclo de clock e produzia resultados conforme o esperado.

2. Fundamentação Teórica

A computação quântica possui como principal e mais básica unidade de informação o qubit. O qubit pode assumir os valores $|0\rangle$, $|1\rangle$ ou uma superposição de ambos, sendo $|0\rangle$ e $|1\rangle$ vetores bidimensionais de números complexos[Nielsen and Chuang 2000]. Sistemas com mais de um qubit podem ser representados por uma combinação linear dos vetores que correspondem aos possíveis estados nos quais o sistema pode colapsar:

$$|\psi\rangle \;=\; \alpha_0\,|00\cdots0\rangle \;+\; \alpha_1\,|00\cdots1\rangle \;+\; \ldots \;+\; \alpha_{2^n-1}\,|11\cdots1\rangle\,,$$

de modo que o número de componentes seja igual a 2^n, sendo n o número de qubits do sistema. Utilizando essa propriedade, podemos criar uma estrutura de dados chamada *quantum register*, que representa o sistema quântico apenas armazenando os valores dos coeficientes α_i e abstraindo a representação explícita de cada componente.

Os algoritmos quânticos são, basicamente, uma sequência de portas quânticas aplicadas ao sistema de forma sequencial, alterando seu estado [Shor 1994]. Essas portas quânticas podem ser representadas por matrizes quadradas e, ao multiplicarmos o vetor que representa o estado do sistema por uma matriz que representa uma porta, obtemos o resultado da aplicação dessa porta quântica. Assim, podemos representar os algoritmos quânticos como uma sequência de multiplicações de matrizes pelos vetores componentes do sistema.

Apesar de um sistema quântico poder assumir diversos valores ao mesmo tempo em razão do fenômeno da superposição, apenas um valor é obtido como saída durante a medição. Em resumo, os algoritmos quânticos têm como objetivo, por meio da aplicação das portas quânticas, manipular essas probabilidades de forma a garantir que, ao medir o sistema, o valor colapsado seja o desejado. Essa manipulação baseia-se na propriedade

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 \;=\; 1,$$

ou seja, a soma dos quadrados dos módulos dos coeficientes que multiplicam as componentes do vetor-estado deve ser igual a 1 [Nielsen and Chuang 2000]. Dessa forma, os algoritmos quânticos utilizam a manipulação dos coeficientes para que a probabilidade de o sistema colapsar no estado desejado durante a observação seja a maior possível.

3. Arquitetura

Como os algoritmos quânticos funcionam aplicando portas quânticas sequencialmente a um sistema , são adotadas para a arquitetura proposta um formato de instrução que armazena apenas os índices dos qubits afetados e o código da operação. Para tanto, o formato de instrução é estruturado em quatro campos: *target*, *ctrl0*, *ctrl1* e *opcode*. Os campos *target*, *ctrl0* e *ctrl1* representam, respectivamente, os índices dos qubits sobre o qual a porta será aplicada, enquanto o *opcode* identifica qual porta quântica será executada, conforme ilustrado na figura 1.

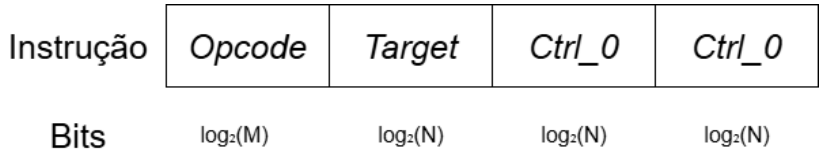

Figura 1. Formato de instrução (N número de qubits, M número de portas).

Para viabilizar a representação de um sistema quântico em hardware, são empregadas duas estruturas de dados: *quantum register* e *complex*. O *complex* é uma representação de número complexo em 32 bits (16 bits para parte real e 16 bits para parte imaginária), ambos em ponto fixo $8:8$. Já o *quantum register* é um vetor de 2^n elementos do tipo *complex*, onde n é o número de qubits; cada elemento corresponde ao coeficiente que multiplica uma das componentes do vetor-estado. Durante a execução, aplica-se uma porta de cada vez sobre o *quantum register*; ao finalizar uma instrução, a próxima é executada.

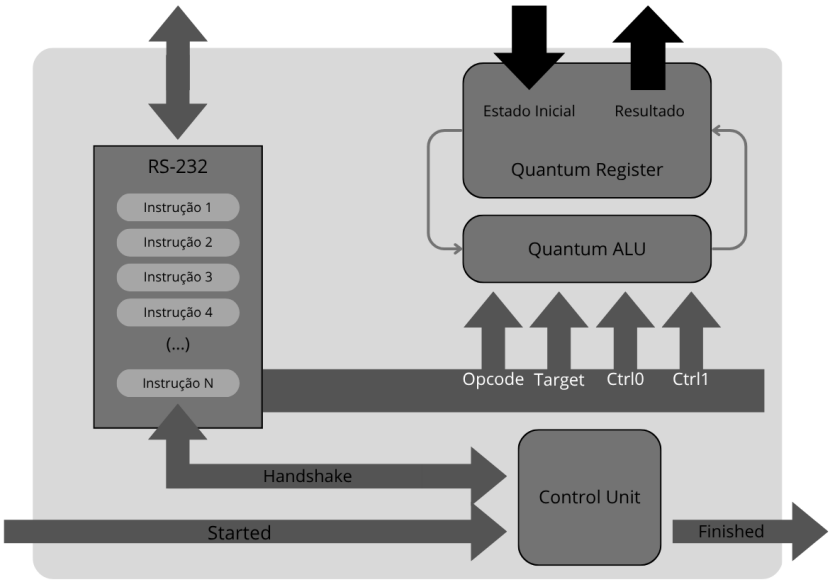

Figura 2. diagrama de blocos da arquitetura. (Fonte: [Conceição and Reis 2015])

3.1. Operação das Portas

A arquitetura é do tipo SIMD (*Single Instruction Multiple Data*) para realização das operações: cada instrução é aplicada a múltiplos dados simultaneamente, controlados pelos campos *target*, *ctrl0* e *ctrl1* por meio da ação de multiplexadores. Todas as portas recebem um *quantum register* e retornam outro *quantum register*. Na prática, o sistema consiste em uma memória única realimentada. A memória contém os dados dos coeficientes do *quantum register*, e é realimentada a cada ciclo com o resultado da parte combinacional que implementa as portas quânticas, controladas por uma unidade de controle que ativa o caminho de dados. Multiplexadores, multiplicadores e somadores de números complexos simulam o comportamento de cada porta.

As portas *Toffoli*, *Pauli-X* e *CNOT* são implementadas sobretudo com multiplexadores. Ao aplicar *Pauli-X* em um único qubit, invertemos os coeficientes $|0\rangle \leftrightarrow |1\rangle$. Em um sistema de três qubits, o comportamento é mostrado na Fig. 3. A porta *Toffoli* só troca índices quando ambos os bits de controle (*ctrl1* e *ctrl0*) são 1. A porta *CNOT* só leva em conta o controle *ctrl0*, invertendo o alvo se este for 1, como ilustrao na Figura 4 .

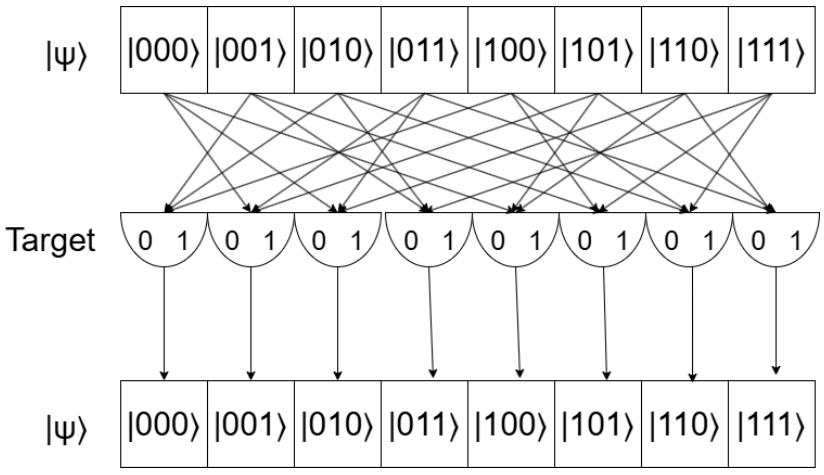

Figura 3. Comportamento da porta Pauli-X em 3 qubits.

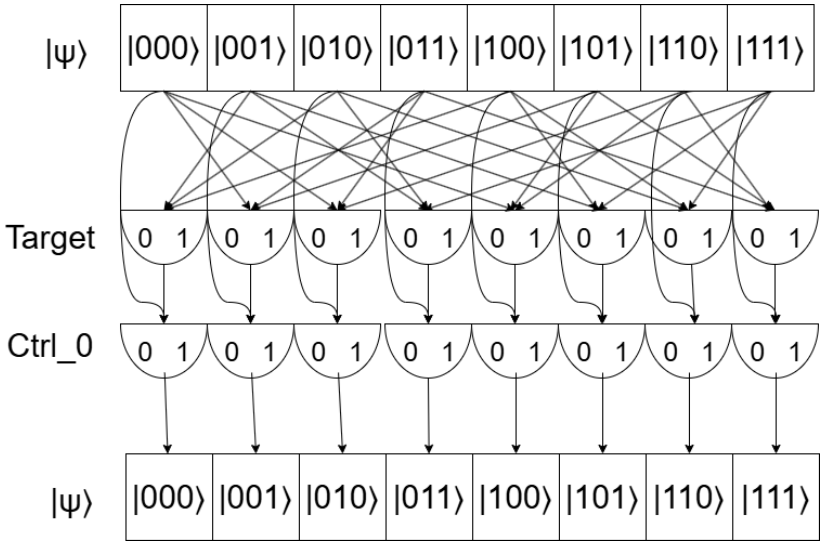

Figura 4. Comportamento da porta CNOT.

Já para as portas de fase (*Hadamard*, *S-gate* e *T-gate*), somadores e multiplicadores de números complexos são combinados aos multiplexadores. A porta *Hadamard*, quando aplicada a um qubit em seu estado base, cria um estado de superposição entre os componentes do vetor de estado, fazendo com que a probabilidade de colapso em qualquer um dos resultados seja igual, conforme exemplificado na Figura 5. Já a *S-gate* multiplica por $i = e^{i\pi/2}$ os componentes $|1\rangle$ do qubit alvo, como mostrado na figura 6, gerando uma rotação de fase de $\pi/2$ (90°). De forma análoga, a *T-gate* multiplica por $e^{i\pi/4}$ os componentes $|1\rangle$ do qubit alvo, correspondendo a uma rotação de fase de $\pi/4$ (45°), como mostrado na figura 7 e 8.

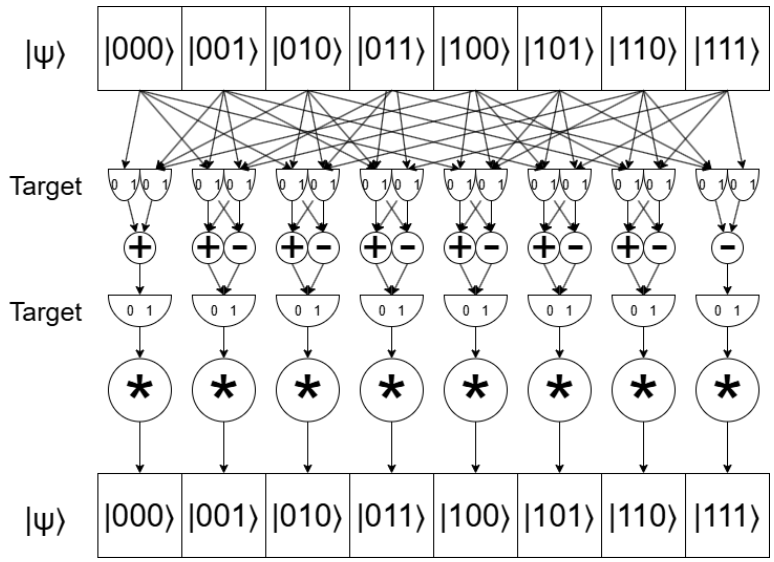

Figura 5. Comportamento da porta Hadamard.

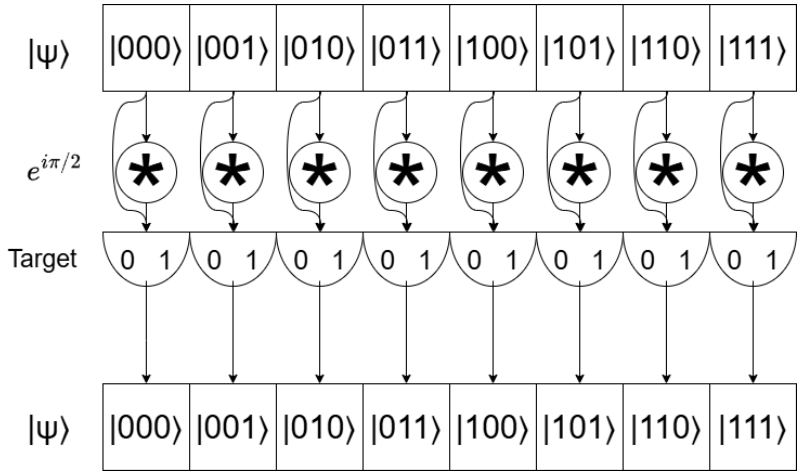

Figura 6. Comportamento da S-gate ($\pi/2$ de fase).

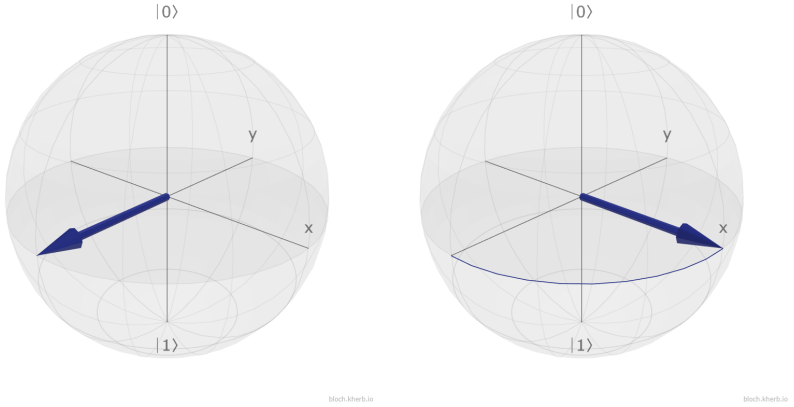

Figura 7. Exemplo da S-gate na esfera de bloch ($\pi/2$ de fase).

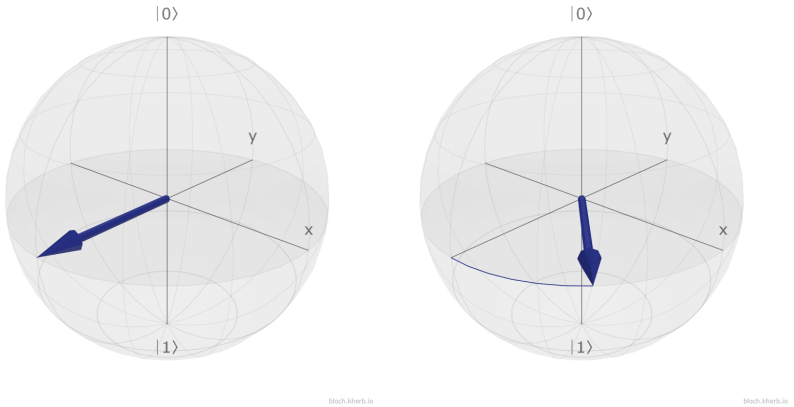

Figura 8. Exemplo da T-gate na esfera de bloch ($\pi/4$ de fase).

.

4. Validação da arquitetura

Para assegurar a validade da implementação proposta, foram realizadas rotinas de verificação funcional por simulação, e prototipação em hardware na placa FPGA DE2i-

150. Ambas as etapas são descritas em detalhes a seguir.

4.1. Verificação Funcional

A verificação funcional do processador foi realizada com o software ModelSim. Adotou-se uma abordagem em duas etapas, testando separadamente o Módulo Serial e o Módulo de Processamento.

No teste do Módulo Serial, o objetivo principal foi verificar se a máquina de estados, a decodificação das instruções e os sinais de controle funcionavam conforme o esperado. Para isso, entradas aleatórias foram aplicadas ao módulo a uma frequência igual à utilizada na integração com a placa de desenvolvimento. Em seguida, os sinais de saída e os pulsos de controle gerados foram comparados com os resultados previstos.

Já na verificação do Módulo de Processamento, avaliou-se o comportamento das máquinas de estados e a correção das operações realizadas pela ULA. Durante a simulação, foram usados vetores de entrada selecionados para exercitar as portas quânticas em condições normais e extremas, garantindo a estabilidade e a robustez do circuito.

4.2. Validação em FPGA

Na validação em FPGA, a principal preocupação foi a propagação correta das instruções dentro do processador e a comunicação com a estação de trabalho. Para isso, aplicaram-se sequências curtas de instruções em diferentes *baud rates*, com o objetivo de assegurar que os dados transitavam corretamente por todo o sistema.

Com a utilização de um clock de 50 MHz e um *baud rate* de 115 200 bps, os resultados foram conforme o esperado, tanto os resultados finais quanto os resultados intermediários.

4.2.1. Comunicação com a estação de trabalho

Para a comunicação entre a FPGA e a fonte de instruções, foi escolhida a especificação serial RS-232. Essa especificação baseia-se em comunicação assíncrona, transmitindo 8 bits de informação, 1 bit de início e 1 bit de parada. A transmissão é feita por duas linhas principais: TXD, responsável pela transmissão do sinal, e RXD, responsável pelo recebimento do sinal. A velocidade de comunicação é definida pelo *baud rate*, que na FPGA de teste foi configurado em 115200 bps.

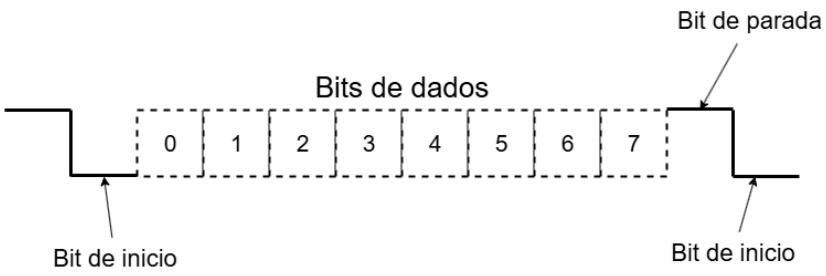

Figura 9. Formato de envio e recebimento no RS-232.

Como o RS-232 suporta apenas 8 bits por quadro e a instrução da arquitetura exige 9 bits (3 do opcode, 2 do target, 2 do ctrl0 e 2 do ctrl1), foi necessário implementar uma máquina de estados no módulo serial para controlar o envio e o recebimento sem perdas. Essa máquina divide cada instrução em duas partes, enviando uma metade primeiro e a outra em seguida, garantindo que nenhum bit seja perdido.

Para o carregamento das instruções no processador, não utilizamos memória de instruções na FPGA. Diferente do artigo de referência, aplicamos uma máquina de estados e sinais de controle que comandam o processo de recepção e execução das instruções, criando um *handshake* entre a FPGA e a fonte de instruções. Após o envio do estado inicial do sistema ao processador, a fonte aguarda um sinal de confirmação da FPGA. Esse sinal faz com que o processador envie a instrução, a qual é então processada. Quando a execução é concluída, o processador envia o resultado provisório de volta à fonte de instruções, que imediatamente passa à transmissão da próxima instrução, isso se repete até o fim das instruções e o envio do estado final para o computador.

4.2.2. Módulos de integração e validação

No controle interno da FPGA, foi necessário implementar máquinas de estado que coordenam as trocas de informação entre o módulo serial e o módulo de processamento, evitando perdas ou atrasos.

Após o início da transmissão entre o computador e a FPGA, uma máquina de estados controla o sincronismo dos módulos. Quando uma instrução é recebida, o módulo serial emite um pulso de aviso indicando dados novos e aguarda um pulso do módulo de processamento indicando término da última instrução. Concluída a execução, esse pulso libera o módulo serial para enviar a próxima instrução e atualizar seu registro com o dado mais recente recebido pela comunicação serial que, por sua vez, espera o término da execução. Isso se repete até o fim das instruções e o envio do estado final do sistema de volta para o computador, como demonstrado na Figura .

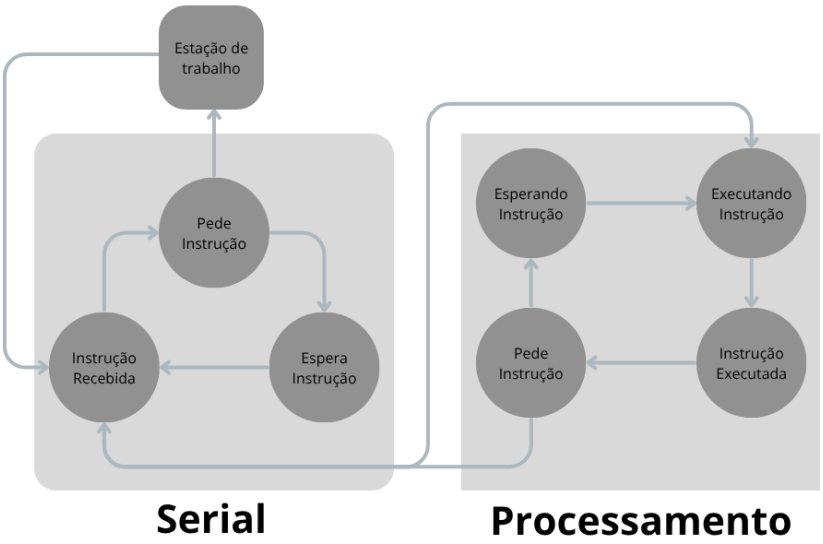

Figura 10. Diagrama de blocos que representa as maquinas de estado.

5. Resultados

A partir da implementação em FPGA, foram coletados dados de ocupação do dispositivo, visando subsidiar trabalhos futuros rumo à escalabilidade da solução para suporte a mais qubits. A análise concentra-se em três grandezas principais: ocupação de elementos lógicos, tempo de síntese e tempo de execução por instrução.

Área de ocupação lógica (LUTs) Foram utilizadas 9.404 LUTs no total, sendo 9.074 no módulo de processamento e 318 no módulo de comunicação serial. Aproximadamente 8.000 LUTs foram dedicadas à implementação das portas quânticas, o que corresponde a cerca de 6% da capacidade total da FPGA. Considerando a arquitetura RISC com um conjunto reduzido de portas e instruções, o número de LUTs deverá crescer principalmente em função do aumento do número de qubits, que irá demandar um incremento no número de bits dos campos da instrução.

Tempo médio de síntese A síntese completa do projeto no Quartus II VERSÃO 13.0.sp1 levou cerca de 3 minutos em um computador Linux/Windows com processador athlon 3000g e 16 GB de memória ram DDR IV. Embora a síntese para um maior número de qubits tenda a crescer de forma quase exponencial devido à complexidade combinacional, o mesmo não vale para o tempo de desenvolvimento, visto que o código foi parametrizado de modo que a inclusão de novos qubits dependa apenas da alteração de constantes e parâmetros, permitindo reaproveitamento quase total do design.

Tempo de execução por instrução Na FPGA, cada instrução foi executada em um único ciclo de clock (50 MHz). Esse desempenho foi alcançado sem o uso de técnicas de pipeline profundo ou paralelismo adicional. Dessa forma, mesmo com mais qubits, espera-se que a latência por instrução se mantenha próxima, já que no modelo proposto pelo processador dedicado adotado como referência neste trabalho, cada operação quântica é realizada em paralelo de forma combinacional, aproveitando eficientemente o tempo de processamento. Apesar do tempo de processamento de cada operação na FPGA demandar apenas o período de *clock*, o gargalo fica a cargo da interface de comunicação, que representa uma parcela significativa no tempo total de processamento na solução em FPGA.

6. Conclusões

Neste artigo revisitamos uma arquitetura de processador baseada em RISC com capacidade de simular o comportamento de algoritmos quânticos em FPGA, e ampliamos o conjunto de portas quânticas suportadas para um conjunto de portas quânticas universais aproximadas. Com o conhecimento absorvido, podemos direcionar esforços para as próximas etapas da pesquisa, que são ampliar o número de qubits suportados e adicionar suporte a portas de coeficientes arbitrários para alcançar um conjunto universal exato.

Os desafios enfrentados em grande parte das soluções existentes para simulação de algoritmos quânticos em hardware estão atrelados principalmente a três fatores: o crescimento exponencial no número de LUTs utilizadas conforme aumenta o número de qubits no sistema; o crescimento exponencial no tempo médio de síntese pelo mesmo motivo; e a necessidade de grandes modificações no código HDL para alterar o número de qubits suportados e o tamanho da mantissa [Aminian et al. 2008], [Negovetic et al. 2002], [Khalid et al. 2004].

A arquitetura adotada demonstra a obtenção de ganhos consideráveis tanto no uso de LUTs quanto no tempo de síntese, pelo fato de ser programável. Como a aplicação de cada porta é análoga à execução de uma instrução, o tempo necessário para a execução de um algoritmo quântico é linear e proporcional ao número de portas que ele utiliza. Com essa solução, uma nova síntese só é demandada quando há alteração do número de qubits ou do tamanho da mantissa empregados pelo sistema. Além disso, o código implementado encontra-se parametrizado, de maneira que a maior parte do código HDL pode ser reaproveitado em expansões futuras de escala ou precisão, exigindo apenas pequenas modificações em constantes e parâmetros predefinidos.

Para trabalhos futuros, consideramos expandir a implementação para suportar um número maior de qubits e realizar testes com alguns algoritmos quânticos conhecidos. Além disso, planejamos substituir a comunicação via RS-232 por outras de maior velocidade, como a USB ou PCI-e, e assim aumentar a taxa de transferência de dados. O objetivo é avaliar a solução completa como estrutura de co-processamento, usada como alternativa à simulação sequencial em software, preferencialmente integrada a alguma plataforma popular de simulação de algoritmos quânticos.

Referências

Aminian, M., Saeedi, M., Zamani, M. S., and Sedighi, M. (2008). Fpga-based circuit model emulation of quantum algorithms. In *2008 IEEE Computer Society Annual Symposium on VLSI*, pages 399–404.

Conceição, C. and Reis, R. (2015). Efficient emulation of quantum circuits on classical hardware. In *2015 IEEE 6th Latin American Symposium on Circuits & Systems (LASCAS)*, pages 1–4.

Feynman, R. P. (1986). Quantum mechanical computers. *Found. Phys.*, 16(6):507–532.

Fujishima, M., Saito, K., and Hoh, K. (2003). 16-qubit quantum-computing emulation based on high-speed hardware architecture. *Japanese Journal of Applied Physics*, 42(4S):2182.

Khalid, A. U., Zilic, Z., and Radecka, K. (2004). Fpga emulation of quantum circuits. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, pages 310–315. IEEE.

Maron, A., Reiser, R., and Pilla, M. (2013). High-performance quantum computing simulation for the quantum geometric machine model. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 474–481. IEEE.

Negovetic, G., Perkowski, M., Lukac, M., and Buller, A. (2002). Evolving quantum circuits and an fpga-based quantum computing emulator.

Nielsen, M. A. and Chuang, I. L. (2000). Quantum information and quantum computation. *Cambridge: Cambridge University Press*, 2(8):23.

Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.