UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CALEBE MICAEL DE OLIVEIRA CONCEIÇÃO

Uma Arquitetura de Co-Processador para Simulação de Algoritmos Quânticos em FPGA

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Ricardo Augusto da Luz Reis
Orientador

Porto Alegre, Outubro de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Conceição, Calebe Micael de Oliveira

Uma Arquitetura de Co-Processador para Simulação de Algoritmos Quânticos em FPGA / Calebe Micael de Oliveira Conceição. – Porto Alegre: PPGC da UFRGS, 2013.

81 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2013. Orientador: Ricardo Augusto da Luz Reis.

1. Ciência da Computação. 2. Microeletrônica. 3. Mecânica Quântica. 4. Computação Quântica. 5. Algoritmos Quânticos. 6. Simulação. 7. Ferramenta EDA. 8. Circuitos Quânticos. 9. FPGA. I. Reis, Ricardo Augusto da Luz. II. Título.

À minha amada família.

AGRADECIMENTOS

Ao bondoso Deus pelas oportunidades e pelas pessoas que encontrei ao longo dessa caminhada. Agradeço ainda mais a Ele por me dar a consciência de que ainda há muito a ser feito, e por me dar forças para persistir nesse sonho.

"Se você quer mesmo ser um bom cientista, busque estar com os melhores, no melhor lugar", certa vez me disse o meu querido orientador na graduação, o Prof. Luiz Brunelli. O sonho de ser um bom cientista me trouxe a Porto Alegre. Agradeço de coração ao Prof. Ricardo Reis, meu orientador, por me abrir as portas e me permitir trazer comigo as minhas ideias.

Agradeço à minha amada esposa Mari por aceitar caminhar comigo. Agradeço à sua família, agora também minha família, pelo carinho com que me incluíram. Amo vocês.

Agradeço aos meus colegas de laboratório, geniais, dedicados, sempre solícitos e acessíveis. É um prazer, na verdade uma honra, poder chamá-los meus amigos. Não teria conseguido sem suas inúmeras dicas, orientações, prévias e revisões ao longo de todo esse processo, sempre que eu precisei.

Agradeço aos membros da banca, tanto da banca de defesa quanto da banca de andamento do mestrado. Obrigado por terem aceitado o convite, e pelas valiosas contribuições. Estejam certos que vou utilizá-las para seguir melhorando.

Agradeço aos professores do mestrado no INF e aos meus professores da graduação na UFS. Guardei os conteúdos que aprendi associados a vocês em minha em minha memória. Esse é o laço forte que tenho com vocês, e que não se perde. Obrigado por terem compartilhado comigo parte do conhecimento e da experiência de vocês.

Agradeço também aos meus alunos do IFRS campus Restinga, onde sou professor temporário. Com vocês eu comprovei algo que eu já desconfiava: melhor que apenas aprender coisas legais, é poder compartilhar e ensinar essas coisas legais. Duvido que haja profissão mais gratificante que essa.

À minha pequenina mãe, sem a qual nada disso seria possível. Obrigado, mãe, por fazer do meu sonho o seu. A todos os meus amados familiares e amigos: obrigado pela torcida. Mais cedo ou mais tarde eu volto, podem anotar!

"Não to mandei eu? Esforça-te, e tem bom ânimo; não temas, nem te espantes; porque o Senhor teu Deus é contigo, por onde quer que andares." (Josué 1:9)

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ASIC	Application Specific Integrated Circuit
BDD	Binary Decision Diagram
EDA	Electronic Design Automation
IC	Integrated Circuit
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDL	Hardware Description Language
LQP	Logic Quantum Processor
LUT	Look Up Table
QFT	Quantum Fourier Transform
QHI	Quantum Hardware Interface
QuIDD	Quantum Information Decision Diagram
Qubit	Quantum Bit
RTL	Register Transfer Level
SIMD	Single Instruction, Multiple Data
SQRAM	Sequential Quantum Random Access Machine

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE CÓDIGOS-FONTE

RESUMO

Simuladores quânticos têm tido um importante papel no estudo e desenvolvimento da computação quântica ao longo dos anos. A simulação de algoritmos quânticos em computadores clássicos é computacionalmente difícil, principalmente devido à natureza paralela dos sistemas quânticos. Para acelerar essas simulações, alguns trabalhos propõem usar hardware paralelo programável como FPGAs, o que diminui consideravelmente o tempo de execução. Contudo, essa abordagem tem três problemas principais: pouca escalabilidade, já que apenas transfere a complexidade do domínio do tempo para o domínio do espaço; a necessidade de re-síntese a cada mudança no algoritmo; e o esforço extra ao projetar o código RTL para simulação. Para lidar com esses problemas, uma arquitetura de um co-processador SIMD é proposta, cujas operações das portas quânticas está baseada no modelo *Network of Butterflies*. Com isso, eliminamos a necessidade de re-síntese com mudanças pequenas no algoritmo quântico simulado, e eliminamos a influência de um dos fatores que levam ao crescimento exponencial do uso de recursos da FPGA. Adicionamente, desenvolvemos uma ferramenta para geração automática do código RTL sintetizável do co-processador, reduzindo assim o esforço extra de projeto.

Palavras-chave: Ciência da Computação, Microeletrônica, Mecânica Quântica, Computação Quântica, Algoritmos Quânticos, Simulação, Ferramenta EDA, Circuitos Quânticos, FPGA.

A Co-Processor Architecture for Simulation of Quantum Algorithms on FPGA

ABSTRACT

Quantum simulators have had a important role on the studying and development of quantum computing throughout the years. The simulation of quantum algorithms on classical computers is computationally hard, mainly due to the parallel nature of quantum systems. To speed up these simulations, some works have proposed to use programmable parallel hardware such as FPGAs, which considerably shorten the execution time. However this approach has three main problems: low scalability, since it only transfers the complexity from time domain to space domain; the need of re-synthesis on every change on the algorithm; and the extra effort on designing the RTL code for simulation. To handle these problems, an architecture of a SIMD co-processor is proposed, whose operations of quantum gates are based on Network of Butterflies model. Thus, we eliminate the need of re-synthesis on small changes on the simulated quantum algorithm, and we eliminated the influence of one of the factors that lead to the exponential growth on the consumption of FPGA resources. Aditionally, we developed a tool to automatically generate the synthesizable RTL code of the co-processor, thus reducing the extra design effort.

Keywords: Computer Science, Microelectronics, Quantum Mechanics, Quantum Computing, Quantum Algorithms, Simulation,EDA Tool, Quantum Circuits, FPGA.

1 INTRODUÇÃO

Em 1965, Gordon Moore – antigo CEO da Intel corp.– tentou prever o que aconteceria na industria de semicondutores nos anos seguintes, baseando sua análise em fatores tecnológicos e financeiros. Tomando por base o custo mínimo por componente, ele prospectou que a quantidade média de elementos eletrônicos em uma área fixa de um CI (Circuito Integrado) dobraria a cada ano. Essa predição, inicialmente feita para os 10 anos seguintes, não apenas se confirmou, como alcançou o status de lei. Nem mesmo Moore imaginava que sua predição poderia se manter por tanto tempo. De fato, a Lei de Moore se tornou o principal motor para a evolução da computação e da comunicação como um todo (MOORE, 2006) (INTEL, 2005).

O próprio Moore cedo preocupou-se com o quanto essa tendência duraria. Em artigo publicado em 1975, ele revisitou sua prospecção e listou (1) a tendência de crescimento da área de chip, (2) a diminuição dos componentes e (3) a melhoria dos dispositivos e dos projetos de circuitos como os três pilares fundamentais de sustentação dos avanços na complexidade dos chips. Nesse mesmo artigo ele já discute sobre os limites para esses pilares, e aponta a diminuição dos componentes como a tendência dominante dali em diante (MOORE, 1975). Os anos seguintes mostraram que ele estava certo novamente.

Acontece que do ponto de vista puramente físico, a diminuição do tamanho dos componentes dos circuitos tem nas leis da física clássica um limite absoluto. Se a lei de Moore continuasse a valer, dimensões próximas a de um único átomo seriam alcançadas em 2036 (POWELL, 2008). Mais ainda, como discutido em (MACK, 2011), a lei de Moore tem em fatores econômicos limites bem mais próximos de serem alcançados. Ela tem sido uma predição auto-realizável, considerando que a indústria de semicondutores, por interesses econômicos, tem trabalhado com afinco na sua manutenção. Seguir a lei de Moore vinha sendo um esforço compensatório, uma vez que cada nó tecnológico era gerado a um custo de investimentos praticamente constante.

A indústria de microeletrônica já está encarando limitações físicas severas que dificultam a manutenção do crescimento exponencial da complexidade dos circuitos integrados, e espera-se um decaimento dessa taxa por volta de 2020 (BAMPI; REIS, 2011). Mas isso não significa que essa indústria irá se extinguir. Ao contrário, espera-se que ela alcance maturidade e amplie seus ramos de aplicação, enquanto o mercado de produtos eletrônicos estiver aquecido. Contudo, considerando a lei de Moore e sua relevância para a história da computação, é natural associar um limite para sua continuidade com um limite para a evolução da computação como um todo. Portanto, faz-se necessário buscar novas tecnologias para continuar avançando na computação, preferencialmente que sejam compatíveis com toda a tecnologia já desenvolvida baseada no silício (BAMPI; REIS, 2011) (WARREN, 2004).

Além do cenário recente de preocupação com o futuro da computação, a busca por no-

vos modelos e tecnologias computacionais que tornem a computação mais eficiente sempre foi uma constante na pesquisa em computação. Muitas alternativas têm sido propostas para substituir ou trabalhar conjuntamente com as tecnologias e modelos tradicionais. As variadas propostas sugerem desde fazer uso de novos materiais ou fazer uso de outros aspectos físicos da matéria além da carga elétrica para representação do chaveamento lógico, até novos modelos computacionais [1], os quais sugerem paradigmas computacionais completamente diferente do modelo computacional que conhecemos (BAMPI; REIS, 2011), (Wu et al., 2012).

A computação quântica se destaca como uma alternativa provável dentre todas essas. A hipótese que a computação poderia ser mais eficiente se fizesse uso de fenômenos quânticos foi feita por Richard Feynman, em 1985, ao encontrar dificuldades para simular sistemas quânticos em computadores tradicionais, devido à natureza paralela do mundo quântico (FEYNMAN, 1985). Trabalhos subsequentes, como os de Peter Shor e Lov Grover, demonstraram essa hipótese ao apresentar algoritmos cuja complexidade em um computador quântico é melhor que os melhores algoritmos clássicos propostos para os mesmos problemas. (SHOR, 1994) (GROVER, 1996).

Mais do que uma alternativa tecnológica, a computação quântica é um novo paradigma de processamento da informação, que pode ter diferentes implementações físicas, e implica em uma nova forma de construir algoritmos (NIELSEN; CHUANG, 2000). O paralelismo inerente aos sistemas quânticos, a escalabilidade exponencial desses sistemas aliados à correta manipulação de fenômenos tipicamente quânticos, a exemplo do emaranhamento, podem conferir à computação quântica potencial computacional sem precedentes. Mas ao mesmo tempo, ela é compatível com a computação clássica, o que a torna uma forte candidata para emergir como um passo além da computação tradicional (RIEFFEL; POLAK, 2000).

Apesar da aceitação do seu potencial computacional, nas duas últimas décadas, devido à notória dificuldade de manipular sistemas físicos dessas dimensões, houve grande ceticismo quanto à possibilidade de construção do hardware de computadores quânticos reais escaláveis e confiáveis, mesmo com alguns protótipos de pequena escala já tendo sido demonstrados (CHUANG; GERSHENFELD; KUBINEC, 1998) (STEFFEN et al., 2001). Em 2010, contudo, uma start-up canadense chamada D-Wave apresentou o primeiro computador quântico comercial, o D-Wave One, que usa um modelo computacional diferenciado das iniciativas adotadas até então (D-WAVE, 2013). Experimentos recentes demonstram que esse computador quântico de fato apresenta o desempenho computacional esperado, e já atraiu investimentos de cerca de 30 milhões de dólares no ano de 2012 (NEVEN et al., 2009) (MCGEOCH; WANG, 2013) (HSU, 2013) (MCBRIDE, 2012).

1.1 Escopo deste trabalho

Muito antes do desenvolvimento dos primeiros protótipos de computadores quânticos, algoritmos quânticos já vinham sendo desenvolvidos e testados por simulação em computadores tradicionais. Contudo, essa atividade recai no problema que motivou Feynman a propor a computação quântica: a simulação de sistemas quânticos em computadores clássicos é computacionalmente difícil, e apresenta complexidade exponencial em relação ao tamanho do sistema simulado. Tais simulações tendem a ser lentas.

O comportamento de um sistema quântico pode ser inteiramente modelado usando o arcabouço matemático da mecânica quântica. Há simuladores de algoritmos quânti-

[1] A comparação entre essas tecnologias e modelos emergentes está fora do escopo dessa dissertação

cos que fazem uso de matrizes para representar os dados do sistema e operar sobre eles (KARAFYLLIDIS, 2005). Outros simuladores usam estruturas de dados mais elaboradas como variantes de BDDs (Binary Decision Diagrams) para alcançar melhores resultados em ultilização de memória (VIAMONTES; MARKOV; HAYES, 2004). Mas qualquer que seja a estrutura de dados adotada, em todos os simuladores disponíveis a complexidade temporal é exponencial com o tamanho do sistema simulado. Simuladores em software são lentos principalmente porque cada operação que seria executada de forma atômica em um computador quântico precisa ser quebrada em operações aritméticas e lógicas simples e escalonadas para execução sequencial em um processador clássico.

Como alternativa, alguns trabalhos usam o paralelismo intrínseco de dispositivos lógicos programáveis como FPGAs (*Field Programmable Gate Array*) para realizar tais simulações (MOHAMED; BADAWY; JULLIEN, 2009)(AMINIAN et al., 2008)(KHALID; ZILIC; RADECKA, 2004), alcançando tempos de execução cerca de três ordens de magnitude menores que as simulações em computadores de propósito geral. Isso acontece porque o hardware necessário para realizar as operações mais simples como somas e multiplicações são replicadas para executar em paralelo, transportando portanto a complexidade do problema do domínio do tempo para o domínio do espaço.

Por outro lado, os recursos disponíveis na FPGA são limitados, e os limites são facilmente alcançados com o aumento do sistema quântico simulado. Além disso, cada modificação feita no algoritmo quântico simulado demanda uma nova síntese, cujo tempo necessário também aumenta exponencialmente quando o sistema é aumentado. Ademais, o uso de FPGA implica em um esforço extra de projeto para descrição do sistema em HDL (*Hardware Description Language*).

Neste sentido, nessa dissertação é proposta uma arquitetura de co-processador voltado para a simulação de algoritmos quânticos. O co-processador proposto é uma máquina SIMD (*Single Instruction, Multiple Data*) e a implementação das operações quânticas está baseada numa adaptação do modelo *Network of Butterflies*, proposto por (NEGOVETIC et al., 2002), para realização em hardware. Além disso, desenvolvemos uma ferramenta para geração automática da descrição HDL (*Hardware Description Language*) sintetizável do co-processador, eliminando assim a necessidade do projetista do algoritmo quântico conhecer o fluxo de projetos pra FPGA. Até onde sabemos, essa é a primeira iniciativa desse tipo.

1.2 Organização do Documento

Essa dissertação está organizada da seguinte forma. No Capítulo 2 são apresentadas os principais conceitos da computação quântica necessários ao entendimento deste trabalho. No Capítulo 3 é apresentada uma revisão bibliográfica dos trabalhos relacionados. No Capítulo 4 apresentamos a arquitetura do co-processoador proposto, e sua implementação alternativa. O Capítulo 5 contém os resultado alcançados com essa proposta, comparados com os demais trabalhos relacionados, como também apresenta a ferramenta que desenvolvemos. As conclusões são tecidas no Capítulo 6.

2 COMPUTAÇÃO QUÂNTICA

O objetivo deste capítulo é apresentar uma visão geral sobre a computação quântica, a qual encontra-se na intersecção entre a Física e a Ciência da Computação. Neste capítulo, são apresentados os temas da física essenciais ao entendimento desse trabalho, e um foco maior é dado para as visões estruturais e comportamentais dos computadores quânticos, conforme divisão proposta por Udrescu et al e sintetizada na Fig. 2.1 (UDRESCU; PRODAN; VLADUTIU, 2004).

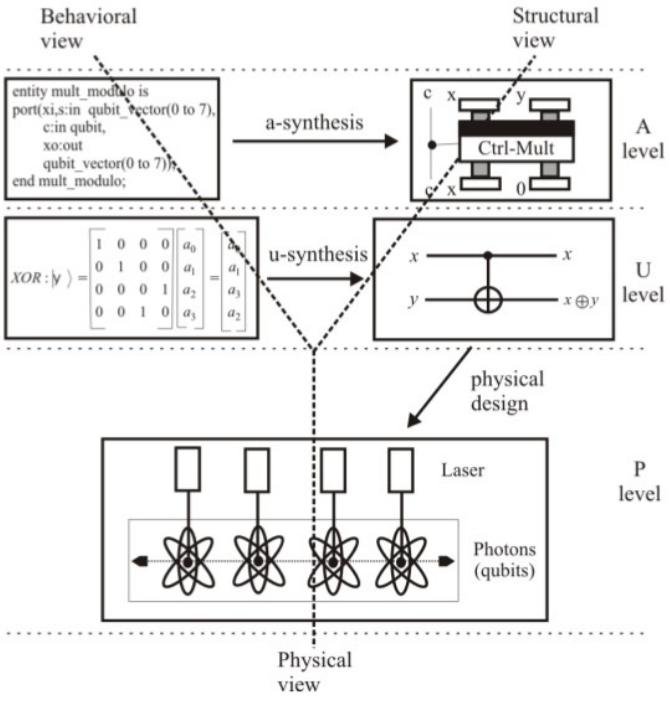


Figura 2.1: Carta Y para a computação quântica

Essa divisão é uma analogia à famosa carta Y, proposta por Gajski para descrever as diferentes perspectivas para o projeto de hardware clássico (GEREZ, 1999). Nesse trabalho, nos concentramos nos níveis arquitetural (*Level A* na figura), que descreve o fluxo de dados do algoritmo codificado nos estados quânticos globais; e unitário (*Level U*, na figura), que descreve as redes de portas quânticas e as transformações unitárias básicas a que estão associadas.

Sempre que possível são feitas analogias com a computação clássica. Para um entendimento mais aprofundado, recomendamos consultar as referências (RIEFFEL; POLAK, 2000) e (NIELSEN; CHUANG, 2000).

2.1 Postulados da Mecânica Quântica

Assim como na computação clássica, na qual não é necessário conhecer o exato comportamento de um transistor para criar um programa, na computação quântica é suficiente concentrar-se nos postulados da mecânica quântica e abstrair a implementação física do sistema quântico para entender seus algoritmos. A mecânica quântica é um arcabouço matemático que é sumarizado em quatro postulados sobre os quais todo o comportamento do mundo quântico pode ser modelado (NIELSEN; CHUANG, 2000). Conhecimentos básicos em álgebra linear podem ser importantes para entender a ideia geral desses postulados.

O sistema quântico mais importante e de maior interesse para a computação quântica é o qubit (**qu**antum **bit**). Tal qual o bit para a computação clássica, o qual é uma abstração de um sistema físico a exemplo da existência de carga no circuito, o qubit é a unidade básica de representação de dados e uma abstração lógica que pode ser implementado fisicamente de diferentes formas. Por exemplo, pode ser implementado como a polarização de um fóton, ou como níveis de energia em um átomo de hidrogênio, dentre outras formas (NIELSEN; CHUANG, 2000). A implementação física dos sistemas quânticos não estão no escopo dessa dissertação.

Tal qual um bit clássico, um único qubit pode assumir os estados $|0)$ ou $|1)$, mas também pode assumir ambos os estados simultaneamente, em um fenômeno chamado superposição, como será discutido no primeiro postulado. Sistemas com mais de dois valores lógicos – chamados sistemas multi-valorados – também são possíveis e largamente explorados nas pesquisas em computação quântica, mas não estão no escopo desta dissertação.

2.1.1 Postulado do Espaço de Estados

> A qualquer sistema físico isolado existe associado um espaço vetorial complexo com produto interno (ou seja, um espaço de Hilbert), conhecido como *espaço de estados* do sistema. O sistema é completamente descrito pelo seu *vetor de estado*, um vetor unitário no espaço de estados.

Para explicar esse postulado, considere um sistema quântico de um único qubit cujo espaço de estados é bidimensional. Como se trata de um espaço vetorial complexo, todo vetor $|\psi)$ pode ser descrito como uma combinação linear de suas bases, por exemplo a base composta pelos vetores ortonormais $|0)$ e $|1)$. Por definição de combinação linear, temos:

$$|\psi) = \alpha|0) + \beta|1) \tag{2.1}$$

onde α e β são coeficientes complexos e, de acordo com a notação de Dirac, $|0)$ e $|1)$ são expressos como matrizes:

$$|0) = \begin{matrix} 1 \\ 0 \end{matrix}, |1) = \begin{matrix} 0 \\ 1 \end{matrix} \tag{2.2}$$

de modo que:

$$|\psi) = \alpha|0) + \beta|1) = \alpha \begin{matrix} 1 \\ 0 \end{matrix} + \beta \begin{matrix} 0 \\ 1 \end{matrix} = \begin{matrix} \alpha \\ 0 \end{matrix} + \begin{matrix} 0 \\ \beta \end{matrix} = \begin{matrix} \alpha \\ \beta \end{matrix} \tag{2.3}$$

O postulado também afirma que o vetor de estado é unitário, o que implica na relação

de completude:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.4}$$

É também importante destacar que a base canônica $\{|0), |1)\}$ é apenas uma das bases possíveis sobre as quais o vetor de estado pode ser escrito. De acordo com a definição de base de um sistema linear, qualquer outro par de vetores complexos ortonormais dessa dimensão pode ser usado como base do espaço de estados.

Este primeiro postulado mostra que o estado completo de um sistema físico isolado pode ser abstraído como um vetor em um espaço vetorial complexo. Uma vez que o qubit é um vetor expresso por um componente α em $|0)$ e outro componente β em $|1)$, isso permite modelar o comportamento de superposição, em que o qubit é $|0)$ e $|1)$ ao mesmo tempo.

2.1.2 Postulado da Evolução

A evolução de um sistema quântico fechado é descrita por uma transformação unitária. Ou seja, o estado $|\psi)$ do sistema no tempo t_1 está relacionado ao estado $|\psi')$ do sistema em t_2 por um operador unitário U que depende somente de t_1 e t_2.

$$|\psi') = U|\psi) \tag{2.5}$$

Esse postulado descreve quais tipos de operações podem ser feitas sobre um sistema quântico fechado, e abre espaço para um grande número de operadores que podem ser usados. Ele expõe que a relação entre os vetores de estado do sistema quântico em dois momentos distintos é descrita por um operador unitário. Como consequência, a relação de completude se mantém durante cada operação. Além disso, uma vez que o operador U é unitário, é válido que:

$$U^\dagger U = \boldsymbol{I} \tag{2.6}$$

onde $U^\dagger$ é a transposta conjugada de U, e $\boldsymbol{I}$ é a matriz identidade. Isso significa que toda operação U realizada em um sistema quântico fechado é reversível, sendo necessário simplesmente aplicar o operador $U^\dagger$ para voltar ao estado anterior.

2.1.3 Postulado da Medição

As medidas quânticas são descritas por determinados operadores de medida M_m. Esses operadores atuam sobre o espaço de estados do sistema. O índice m se refere aos possíveis resultados da medida. Se o estado de um sistema quântico for $|\psi)$, imediatamente antes da medida, a probabilidade do resultado m ocorrer é dada por:

$$p(m) = (\psi|M_m^\dagger M_m|\psi) \tag{2.7}$$

e o estado do sistema após a medida será

$$\frac{M_m|\psi)}{\sqrt{(\psi|M_m^\dagger M_m|\psi)}} \tag{2.8}$$

Os operadores de medida satisfazem a relação de completude:

$$\sum_m M_m^\dagger M_m = \boldsymbol{I} \tag{2.9}$$

A relação de completude expressa o fato de que a soma das probabilidades deve ser igual a 1:

$$1 = \sum_m p(m) = \sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle \qquad (2.10)$$

Este postulado apresenta uma das mais curiosas características dos sistemas quânticos: a observalidade parcial do estado de um sistema quântico. Quando uma medição é realizada sobre o estado de um sistema quântico, isso perturba a premissa de isolamento e faz o sistema colapsar para uma das base subjacentes do aparato de medição com probabilidade igual ao quadrado da amplitude associada àquele vetor base.

Assim, de acordo com a equação 2.7, um sistema quântico de um único qubit tem a probabilidade α^2 de ser colapsado para $|0\rangle$ e β^2 de ser colapsado para $|1\rangle$ quando medido. Portanto, o estado $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ desse sistema após a medição será $|\psi\rangle = 1|0\rangle + 0|1\rangle$ com probabilidade α^2 ou $|\psi\rangle = 0|0\rangle + 1|1\rangle$ com probabilidade β^2, de acordo com a equação 2.8. Note que após a medição o qubit medido não pode estar mais em uma superposição daquela base.

2.1.4 Postulado do Sistema Composto

O espaço de estado de um sistema físico composto é o produto tensorial dos espaços de estados dos sistemas físicos individuais. Se os sistemas forem numerados de 1 até n, e o sistema i for preparado no estado $|\psi_i\rangle$, decorre que o estado do sistema composto será $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \ldots \otimes |\psi_n\rangle$.

De fato muito pouca computação pode ser feita com apenas um qubit. As mais interessantes aplicações para sistemas de computação quântica sugeridas na literatura relacionada são baseadas em sistemas quânticos com mais de um qubit. Esse postulado descreve como esses sistemas trabalham.

Nele é afirmado que o espaço de estados de um sistema quântico composto é formado pela combinação dos seus componentes por meio do produto tensorial (também conhecido como produto Kronecker). Uma característica dessa operação é que a dimensão do espaço vetorial resultante é dado pela multiplicação das dimensões dos espaços vetoriais individuais.

Portanto, se a dimensão do espaço vetorial de um sistema de um único qubit é 2, a dimensão do espaço de estados de um sistema composto com 2 qubits é 2x2 = 4. Isso significa que o conjunto de vetores base do espaço vetorial resultante tem 4 vetores, os quais, por sua vez, são também formados pelo produto tensorial de cada par de vetores base dos sistemas individuais.

Como exemplo, considere o sistema composto de dois qubits e também considere a base canônica $\{|0\rangle, |1\rangle\}$ do espaço vetorial dos sistemas de um único qubit. O espaço de estados do sistema composto tem dimensão 4 – como um resultado da aplicação do produto tensorial ($\otimes$) – e seu conjunto de vetores base é composto pelo produto tensorial de cada par de vetores base de cada qubit, como segue:

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{matrix}0\\1\end{matrix} \otimes |0\rangle = \begin{matrix}0|0\rangle\\1|0\rangle\end{matrix} = \begin{matrix}0\\0\\0\\1\end{matrix} \otimes \begin{matrix}0\\1\end{matrix} = \begin{matrix}0\\0\\0\\0\end{matrix}$$

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{matrix}0\\1\end{matrix} \otimes |1\rangle = \begin{matrix}0|1\rangle\\1|1\rangle\end{matrix} = \begin{matrix}0\\0\\1\\1\end{matrix} = \begin{matrix}0\\0\\1\\0\end{matrix}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{matrix}1\\0\end{matrix} \otimes |0\rangle = \begin{matrix}1|0\rangle\\0|0\rangle\end{matrix} = \begin{matrix}1\\0\\0\\0\end{matrix} = \begin{matrix}0\\0\\1\\0\end{matrix}$$

$$|11\rangle = |1\rangle \otimes |1\rangle = \begin{matrix}1\\0\end{matrix} \otimes |1\rangle = \begin{matrix}1|1\rangle\\0|1\rangle\end{matrix} = \begin{matrix}1\\0\\0\\1\end{matrix} = \begin{matrix}0\\0\\0\\1\end{matrix}$$

Portanto, qualquer estado $|\psi\rangle$ desse sistema de dois qubits é descrito nessa base como:

$$|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle \tag{2.11}$$

onde α_0, α_1, α_2, α_3 são números complexos.

De forma mais genérica, o vetor de estados $|\psi\rangle$ de um sistema computacional quântico composto de n qubits é descrito como uma combinação linear de 2^n vetores bases como:

$$|\psi\rangle = \sum_{i}^{2^n} \alpha_i|\psi_i\rangle, \psi_i \in \{0,1\}^n, \alpha_i \in \mathbb{C} \tag{2.12}$$

Uma forma de representar o vetor de estados de um sistema composto de n qubits é como uma lista de 2^n valores complexos $(\alpha_0 \ldots \alpha_{2^n-1})$, usualmente chamada de registrador quântico.

As relações estabelecidas nos postulados anteriores também são aplicadas a sistemas quânticos compostos, com as devidas adaptações, a exemplo da relação de completude:

$$\sum_{i}^{2^n} \alpha_i^2 = 1 \tag{2.13}$$

Uma implicação direta desse postulado é que, assim como na computação clássica, em um sistema computacional quântico com n qubits, 2^n palavras de tamanho n podem ser representadas. Contudo, na computação quântica essas palavras podem ser representadas ao mesmo tempo, devido ao fenômeno de superposição. Em outras palavras, o vetor de estado de um sistema composto pode estar em uma combinação linear de até 2^n vetores bases com coeficientes não nulos. O paralelismo está na representação dos dados, de forma que um paralelismo real pode ser alcançado na computação quântica sem necessidade de replicação de hardware, como é feito na computação clássica.

Em um sistema composto, uma medição parcial de um único qubit pode ser feita. Após isso, o espaço de estados desse qubit irá colapsar para o estado base resultante da medição, e os coeficientes do vetor de estado do sistema composto serão ajustados para manter a relação de completude.

Como exemplo, considere um sistema quântico quadridimencional, composto por dois

qubits, em que seu vetor de estado $|\psi)$ é preparado em uma igual superposição dos estados da base, então:

$$|\psi) = 0.5|00) + 0.5|01) + 0.5|10) + 0.5|11) \quad (2.14)$$

A seguir, suponha que a medição é feita sobre apenas o primeiro qubit e resulta no estado $|0)$. Portanto, os coeficientes de $|\psi)$ serão ajustados para a probabilidade nula de um segunda medição sobre o mesmo qubit resultar em $|1)$, enquanto no segundo qubit é mantido o estado de superposição. O estado final $|\psi)$ será:

$$|\psi) = \frac{1}{\sqrt{2}}|00) + \frac{1}{\sqrt{2}}|01) + 0|10) + 0|11) \tag{2.15}$$

2.1.4.1 Emaranhamento

De acordo com o postulado dos sistemas compostos, dados dois qubits cujos respectivos estados são $\alpha_0|0) + \alpha_1|1)$ e $\beta_0|0) + \beta_1|1)$ é possível construir o estado composto da forma mostrada em (2.16):

$$(\alpha_0|0) + \alpha_1|1))(\beta_0|0) + \beta_1|1)) = \alpha_0\beta_0|00) + \alpha_0\beta_1|01) + \alpha_1\beta_0|10) + \alpha_1\beta_1|11) \tag{2.16}$$

Diz-se que um estado de um sistema quântico composto está emaranhado quando não é possível decompor o estado do sistema como o produto tensorial dos estados individuais. (RIEFFEL; POLAK, 2000). Um exemplo é estado apresentado em (2.17).

$$|\psi) = \frac{1}{\sqrt{2}}|00) + \frac{1}{\sqrt{2}}|11) = (\alpha_0|0) + \alpha_1|1))(\beta_0|0) + \beta_1|1)) \tag{2.17}$$

Não existem valores para $\alpha_0,\alpha_1,\beta_0,\beta_1$ que satisfaçam a equação (2.17). Sabe-se, pela equação (2.16), que $\alpha_0\beta_0 = \frac{1}{\sqrt{2}}$ e $\alpha_1\beta_1 = \frac{1}{\sqrt{2}}$. Portanto $\alpha_0,\alpha_1,\beta_0,\beta_1$ são todos não nulos. Acontece que os coeficientes associados aos estados base $|01)$ e $|10)$ em $|\psi)$ são nulos. Ou seja, de acordo com (2.16), $\alpha_0\beta_1 = 0$ e $\alpha_1\beta_0 = 0$, o que leva a uma contradição.

2.2 Modelo de Circuitos

Ao manipular um sistema quântico para realizar cômputos, é necessário controlar a sequência de operações que ele realiza. Esse controle é realizado ao selecionar a sequência correta de operações unitárias, de acordo com o segundo postulado da mecânica quântica. Existem diversos modelos que representam como as operações são feitas nos sistemas quânticos, a exemplo do modelo de Máquina de Turing Quântica, o modelo *Quantum Cellular Automata*, o modelo de circuitos quânticos, dentre outros (MISZCZAK, 2011).

O modelo de circuitos para representação de algoritmos quânticos é usado nessa dissertação. Neste modelo, os algoritmos são descritos como uma rede de portas quânticas. Cada porta quântica é associada a uma transformação unitária, que pode ser aplicada sobre um ou mais qubits, os quais por sua vez são representados por fios conectados às entradas e às saídas das portas. Este modelo é largamente utilizado na literatura relacionada. Na Fig. 2.2 é apresentado um exemplo de circuito quântico.

Em um circuito quântico não há ciclos nem sinais de realimentação. A disposição das portas significa a ordem em que as operações que elas representam são aplicadas sobre o sistema. Começando por um vetor de estado inicial, a sequência de execução segue da esquerda para a direita enquanto o estado do sistema vai sendo modificado à medida que as portas quânticas são alcançadas. As operações são realizadas pela multiplicação de cada operador correspondente pelo vetor de estado corrente do sistema.

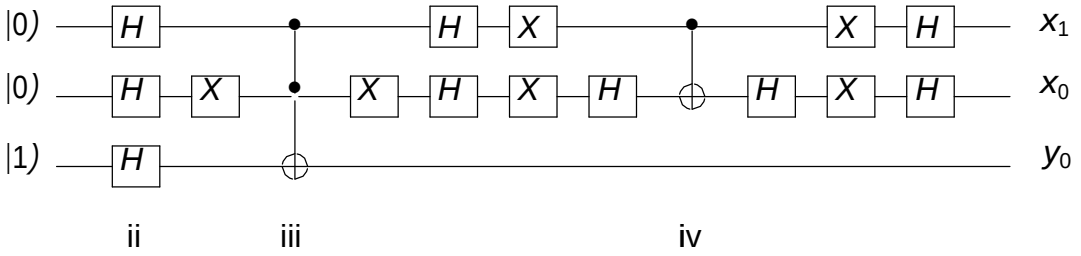


Figura 2.2: Exemplo de um circuito quântico

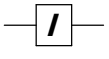
2.2.1 Porta Identidade

Esta é a porta quântica mais simples. Essa porta de um único qubit, também conhecida como Pauli I, mantém o estado do qubit inalterado. Apesar de ser usualmente suprimida nas representações de circuitos quânticos, ela é combinada por meio do produto tensorial com outras portas quânticas que são aplicadas sobre os outros qubits em um mesmo instante.

Os resultados de sua aplicação são:

$$I|0\rangle \rightarrow |0\rangle$$
$$I|1\rangle \rightarrow |1\rangle$$
$$I(\alpha|0\rangle + \beta|1\rangle) \rightarrow \alpha|0\rangle + \beta|1\rangle$$

Seu simbolo e a matriz do operador correspondente são apresentados na Fig. 2.3.



$$\sigma_0 = I = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

Figura 2.3: Representações da porta Pauli I.

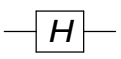
2.2.2 Porta Hadamard

A Hadamard é uma porta de um único qubit que quando aplicada a um qubit em seu estado fundamental – isto é, colapsado em $|0\rangle$ ou em $|1\rangle$ – o coloca em uma igual superposição de seus vetores base. Isso significa que uma medição após a sua aplicação resultaria em $|0\rangle$ ou $|1\rangle$ com iguais probabilidades. Os resultados da aplicação da Hadamard são:

$$H|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$H|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$H(\alpha|0\rangle + \beta|1\rangle) \rightarrow \frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle$$

Suas representações são apresentadas na Fig. 2.4.



$$H = \frac{1}{\sqrt{2}}\begin{matrix} 1 & 1 \\ 1 & -1 \end{matrix}$$

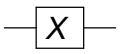
Figura 2.4: Representações gráfica e matricial da porta Hadamard.

2.2.3 Porta Pauli X

Ela também é conhecida como a porta NOT quântica, por que seu comportamento é similiar à homônima clássica. Essa porta de um único qubit troca as amplitudes associadas aos estados da base, fazendo com que o qubit mude para o estado oposto – por exemplo, se está em $|0\rangle$ muda para $|1\rangle$. Os resultados da aplicação da porta Pauli X são:

$$X|0\rangle \rightarrow |1\rangle$$
$$X|1\rangle \rightarrow |0\rangle$$
$$X(\alpha|0\rangle + \beta|1\rangle) \rightarrow \beta|0\rangle + \alpha|1\rangle$$

Suas representações são mostradas na Fig. 2.5:



$$\sigma_1 = \sigma_X = X = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$$

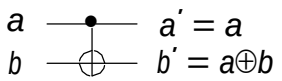
Figura 2.5: Representações gráfica e matricial da porta Pauli X.

2.2.4 Porta CNOT

Também conhecida como porta Feynman, a CNOT (*Controlled Not*) é uma porta de dois qubits e realiza a inversão controlada de um dos qubits de acordo com o valor do segundo. O valor do qubit de destino é invertido se, e somente se, o qubit de controle é $|1\rangle$, do contrário o qubit destino é inalterado. O círculo preenchido na sua representação gráfica corresponde ao qubit de controle. Os resultados da aplicação de uma porta CNOT são:

$$CNOT|00\rangle \rightarrow |00\rangle$$
$$CNOT|01\rangle \rightarrow |01\rangle$$
$$CNOT|10\rangle \rightarrow |11\rangle$$
$$CNOT|11\rangle \rightarrow |10\rangle$$
$$CNOT(\alpha|0\rangle + \beta|1\rangle)|1\rangle \rightarrow \alpha|01\rangle + \beta|10\rangle$$
$$CNOT|0\rangle(\alpha|0\rangle + \beta|1\rangle) \rightarrow \alpha|00\rangle + \beta|01\rangle$$
$$CNOT|1\rangle(\alpha|0\rangle + \beta|1\rangle) \rightarrow \beta|10\rangle + \alpha|11\rangle$$

As representações da porta CNOT são apresentadas na Fig. 2.6.



$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figura 2.6: Representação gráfica e matricial da porta CNOT.

2.2.5 Porta Toffoli

Também conhecida como C^2-NOT, esta é uma porta que é aplicada sobre três qubits. Os dois qubits de controle condicionam a inversão do qubit de destino, de forma similar à porta CNOT. Suas representações são mostradas na Fig. 2.7. Os resultados da aplicação da porta Toffoli são:

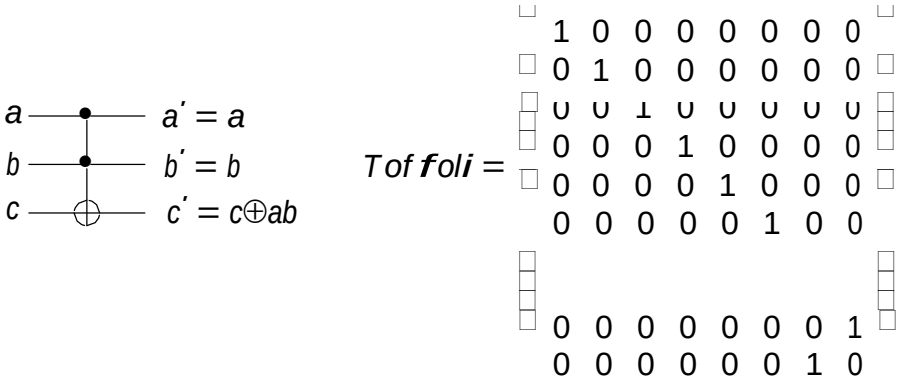


Figura 2.7: Representação gráfica e matricial da porta Toffoli

$$
\begin{aligned}
Toffoli|000\rangle &\rightarrow |000\rangle \\
Toffoli|001\rangle &\rightarrow |001\rangle \\
Toffoli|010\rangle &\rightarrow |010\rangle \\
Toffoli|011\rangle &\rightarrow |011\rangle \\
Toffoli|100\rangle &\rightarrow |100\rangle \\
Toffoli|101\rangle &\rightarrow |101\rangle \\
Toffoli|110\rangle &\rightarrow |111\rangle \\
Toffoli|111\rangle &\rightarrow |110\rangle
\end{aligned}
$$

Uma característica interessante da computação quântica é a possibilidade de com ela realizar todas as operações básicas da computação clássica. O principal argumento para essa discussão está em demostrar que a porta Toffoli pode realizar a operação de uma porta NAND clássica. Para isso, conforme é ilustrado em 2.8, é suficiente inicalizar o primeiro qubit de controle em $|1\rangle$ e associar as entradas a e b da porta NAND aos demais qubits. Deste modo, após a operação da Toffoli, o qubit destino conterá o resultado da expressão $|1\rangle \otimes ab = \overline{ab}$.

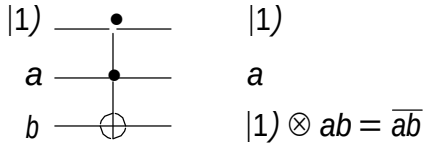


Figura 2.8: Porta Toffoli realizando a porta NAND clássica

Assim, é possível implementar circuitos clássicos usando portas quânticas. Como um exemplo demonstrativo, na Fig. 2.9 é apresentado um circuito somador completo implementado com as portas quânticas Toffoli e CNOT.

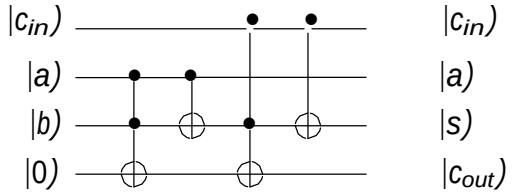


Figura 2.9: Esquema de um somador completo implementado com portas quânticas

2.3 Algoritmos Quânticos

Classicamente, um algoritmo pode ser definido como uma descrição de um padrão de comportamento, expresso em termos de um conjunto finito de ações (ZIVIANI et al., 2004). Na computação quântica essa definição se mantém. De fato, seja na computação clássica ou na quântica, ou ainda em qualquer outro modelo computacional, a definição de um algoritmo corresponde à descrição de como o sistema deve se comportar, em termos de suas operações suportadas, para realizar uma dada tarefa.

Muitos algoritmos quânticos têm sido propostos na literatura, e alguns deles superam a eficiência de seus equivalentes clássicos. O algoritmo de Shor (SHOR, 1994) e o algoritmo de Grover (GROVER, 1996), comentados a seguir, são de particular importância porque são utilizados como base de um grande conjunto de algoritmos quânticos que vieram após eles, como algoritmos de contagem, algoritmos estatísticos, dentre outros (NIELSEN; CHUANG, 2000).

2.3.1 Quantum Fourier Transform

A QFT (*Quantum Fourier Transform*) foi proposta por Peter Shor em 1994, como parte de uma solução proposta por ele para o problema de fatoração de números inteiros (SHOR, 1994). Ela é parte integrante de muitos algoritmos propostos posteriormente. O algoritmo de fatoração de Shor tornou-se conhecido principalmente porque ele precisa de tempo polinomial em relação ao número de bits das entradas para chegar à solução, se executado em um computador quântico real – $O(n^2 \log n \log\log n)$ – enquanto o melhor algoritmo clássico conhecido precisa de tempo exponencial – $exp(\Theta(n^{\frac{1}{3}}\log^{\frac{2}{3}} n))$ (NIELSEN; CHUANG, 2000). O problema de fatoração de números inteiros tem um grande campo de aplicação especialmente em sistemas de criptografia, como os que usam o algoritmo RSA no esquema de compartilhamento de chaves pública e privada, cuja segurança reside na dificuldade de resolver tal problema em computadores clássicos (KUROSE; ROSS, 2006).

2.3.2 Busca Quântica

O algoritmo de busca de Grover resolve o problema de buscar um elemento específico em uma base de dados desordenada (GROVER, 1996). Ele é provado ser ótimo e tomar tempo $O(\sqrt{n})$ se executado em um computador quântico real. Mais precisamente, o algoritmo de Grover precisa realizar m iterações para encontrar o valor buscado, onde m é dado por (2.18), N é o número total de elementos na base de dados, e r é o número de elementos na base que satisfazem o critério de busca.

$$m \leq \lfloor 0.25\pi(N/r)^{\frac{1}{2}} \rfloor \tag{2.18}$$

Seu desempenho, caso executado em um computador quântico real, é polinomialmente melhor que o melhor algoritmo clássico, que é provado levar tempo $O(n)$. Esse algoritmo ficou conhecido principalmente pela grande quantidade de instâncias reais desse problema. O exemplo detalhado de uma instância desse problema é apresentado no Apêndice A dessa dissertação.

3 TRABALHOS RELACIONADOS

Existe um grande número de simuladores quânticos cujo foco de aplicação varia desde a simulação dos detalhes físicos dos sistemas simulados até os voltados apenas para as visões estruturais e comportamentais do algoritmo quântico projetado, abstraindo todos os detalhes físicos. Eles também se diferenciam quanto ao modelo empregado para descrever o algoritmo quântico, quanto à forma como os resultados são apresentados, dentre outras características. Existem ainda simuladores que definem uma linguagem de programação específicas para descrever os algoritmos quânticos, chamadas linguagens de programação quântica.

Nesta revisão, nos concentramos em trabalhos relacionados que focam na lógica dos algoritmos quânticos, tratam apenas de lógica quântica binária e usam o modelo de circuitos para representar os algoritmos quânticos. Nós os dividimos em simuladores em software e simuladores em hardware.

3.1 Simuladores em Software

Consideramos simuladores em software aqueles que são executados em um processador de propósito geral de um computador pessoal comum.

A Libquantum (BUTSCHER; WEIMER, 2013) é uma biblioteca escrita em linguagem C voltada para a simulação da mecânica quântica geral, com um foco especial em computação quântica. Ela faz uso do modelo de circuitos para representar algoritmos quânticos, e é uma das mais rápidas e um dos simuladores mais usados no ensino e pesquisa. Apesar de não estar explicitamente declarado em sua documentação, a explicação sobre como é feita a implementação das portas quânticas suportadas, apresentada no manual da ferramenta, lembra o modelo Network of Butterflies proposto em (NEGOVETIC et al., 2002). Foi incluído suporte a paralelismo (*multithread*) a partir da versão 1.1.1.

Em (KARAFYLLIDIS, 2005) é apresentado o que seria o primeiro simulador de algoritmos quânticos com GUI (Interface Gráfica do Usuário, em inglês). Também faz uso do modelo de circuito para representação dos algoritmos, e suporta portas como Hadamard, Feynman e Toffoli. A GUI foi proposta para facilitar o uso da ferramenta, tanto na edição do algoritmo quanto na apresentação dos resultados. As saídas são apresentandas usando gráficos mostrando os valores complexos do registrador quântico. O mecanismo de simulação faz uso de matrizes como estruturas de dados básica, e faz uso de técnicas para computação de matrizes esparças para melhorar a implementação. Uma captura de tela da ferramenta, extraída do artigo (KARAFYLLIDIS, 2005), é apresentada na Fig. 3.1.

O JQuantum é um outro simulador de algoritmos quânticos com GUI nativa (VRIES, 2013). Trata-se de um simulador de código aberto escrito na linguagem de programação

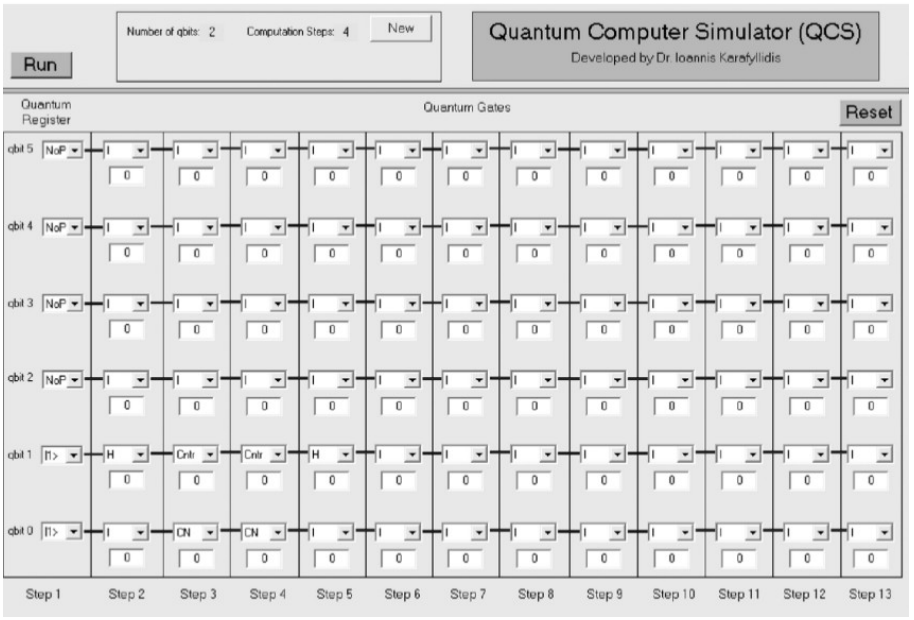


Figura 3.1: Primeiro simulador quântico com GUI

Java que também usa o modelo de circuitos para descrever o algoritmo quântico. Ao analisar o código-fonte da ferramenta, percebe-se que também faz uso do modelo *Network of Butterflies* para implementação das portas quânticas, apesar de, assim como na *libquantum*, não estar declarado em sua documentação. Seu código-fonte é muito bem documentado e sua GUI intuitiva, fatos que contribuíram para sua utilização nessa dissertação. Uma captura de tela de sua GUI é apresentada na Fig. 3.2.

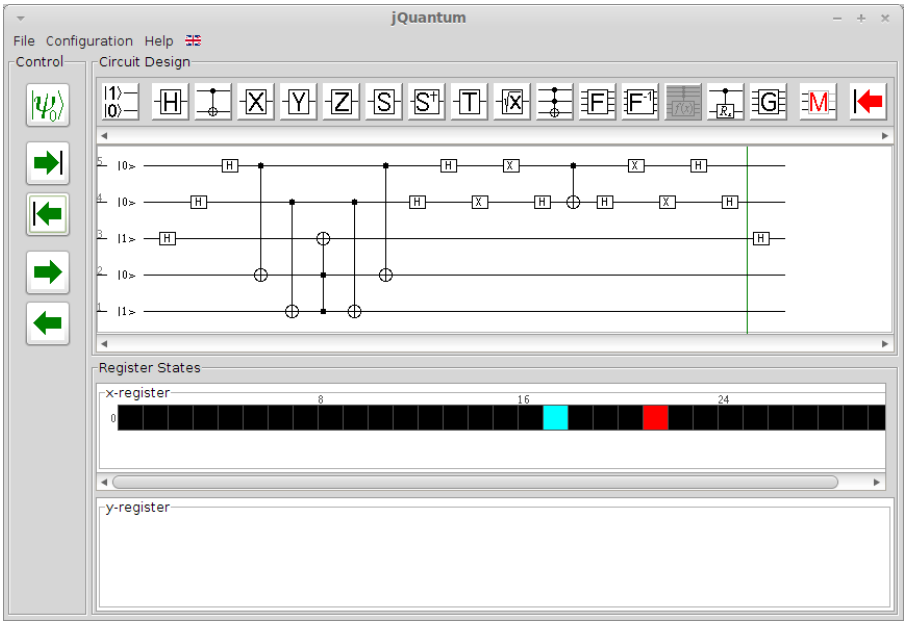


Figura 3.2: GUI of JQuantum

O mais rápido simulador de algoritmo quântico em software e com melhor complexidade espacial é o QuIDDPro desenvolvido por Viamontes, Markov e Hayes (VIAMONTES; MARKOV; HAYES, 2004). Inspirados em trabalhos anteriores que fazem uso de

variações de BDDs (*Binary Decision Diagrams*) para representar e realizar operações com matrizes, os autores propuseram uma estrutura derivada chamada QuIDD (*Quantum Information Decision Diagrams*) para armazenar o registrador quântico de forma compactada. Nela, a representação binária do índice da amplitude é usada para percorrer uma árvore binária da raíz até as folha, na qual é armazenado um índice para um vetor global de números complexos que contém o respectivo valor. A estrutura é compactada por meio da eliminação de redundâncias, como ilustrado na Fig. 3.3 extraída do artigo (VIAMONTES; MARKOV; HAYES, 2003). Na figura são representadas exemplos de três configurações das estruturas QuIDD, no pior, melhor e caso médio de compactação, juntamente com o vetor global de números complexos em cada caso.

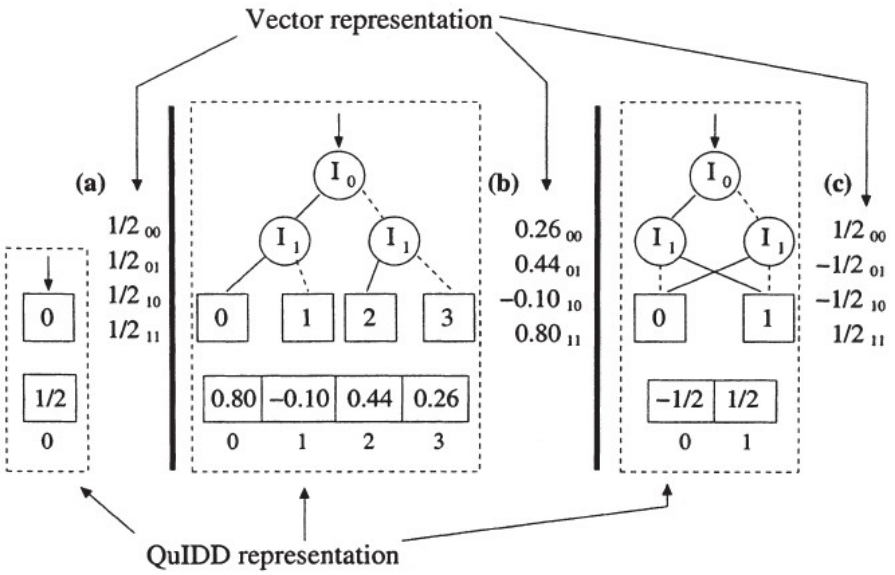


Figura 3.3: A estrutura de dados QuIDD com três graus de compactação: (a) melhor, (b) médio, (c) pior caso

As operações necessárias, como produto interno e produto tensorial, são realizadas diretamente sobre o registrador quântico codificado como um QuIDD, sem necessidade de descompactá-lo. Adaptações foram feitas nas operações originais dos BDDs, a exemplo da operação *apply*, para comportar particularidades da estrutura QuIDD.

Em uma versão estendida do artigo (VIAMONTES; MARKOV; HAYES, 2003), os autores apresentam uma análise aprofundada da complexidade das estruturas e algoritmos propostos. A estrutura de dados QuIDD é adequada para o domínio do problema, uma vez que as matrizes dos operadores são sempre quadradas, quase sempre esparsas, e apresentam estrutura recursiva devido à operação de produto tensorial, o que favorece a compactação da estrutura QUIDD. Com isso, o simulador apresenta complexidade espacial polinomial, diferente dos trabalhos anteriores, nos quais a complexidade espacial é exponencial. Contudo, a complexidade temporal continua sendo exponencial assim como nos demais simuladores disponíveis.

3.2 Simuladores em Hardware

Numa tentativa de lidar com o tempo de execução gasto na simulação, alguns trabalhos fazem uso de hardware clássico nativamente paralelo, tais como FPGAs e GPUs, para simular algoritmos quânticos, alcançando com isso tempos de simulação cerca de três or-

dens de magnitude melhores que os alcançados com o simulador *libquantum* (AMINIAN et al., 2008) (KHALID; ZILIC; RADECKA, 2004). Nesta revisão, nos concentramos nas iniciativas que fazem uso de FPGA e/ou propõem uma arquitetura de processador específico para esse fim.

No trabalho de Fujishima (FUJISHIMA et al., 2003), um processador para computação quântica chamado *Logic Quantum Processor* (LQP) é proposto. Ele faz uso de um operador de soma de produtos elementar como o elemento básico de hardware para realizar uma transformação quântica unitária, o qual é replicado para implementar as duas únicas operações suportadas, a operação Hadamard e Pauli X.

Com somente essas duas portas, os únicos valores que as amplitudes do vetor de estado poderão assumir em qualquer momento do processamento serão 0 ou $1/\sqrt{m}$. O autor se vale disso para usar apenas um único bit para representar os dados valores das amplitudes. Contudo, isso limita que o número de portas suportadas pelo LQP seja ampliado, uma vez que a inserção de outras portas pode fazer com que dois bits não sejam mais suficientes para representar todos os valores presentes no vetor de estado do sistema em um dado momento. A arquitetura foi testada em FPGA e alcançou tempos de simulação cerca de 275 vezes mais rápidos que simulação em software. Essa é provavelmente a primeira proposta de um processador específico para esse fim.

Uma arquitetura híbrida de um computador quântico denominada SQRAM (*Sequential Quantum Random Access Machine*) é proposta no trabalho de Nagarajan et al (NAGARAJAN; PAPANIKOLAOU; WILLIAMS, 2007). A SQRAM foi projetada com o objetivo de simplificar a geração de código a partir da linguagem funcional de programação quântica também proposta no artigo. A arquitetura consiste de um componente clássico, o qual controla o funcionamento do componente puramente quântico. A Fig. 3.4, extraída desse artigo, ilustra a visão geral da máquina SQRAM proposta.

O componente clássico, baseado na arquitetura Harvard, é composto essencialmente por uma CPU (Unidade Central de Processamento, do inglês) e duas memórias, uma para armazenar programas e outra para dados, além de outros elementos para controle do fluxo de execução. Os dados são armazenados na memória de dados numa estrutura de pilha, por se adequar melhor ao modo de operação do paradigma funcional. Ele dispõe de um conjunto registradores internos e suporta instruções específicas para o controle do componente quântico.

O componente quântico, por sua vez, dispõe de um registrador quântico e de uma interface de hardware denominada QHI (Quantum Hardware Interface), o qual é responsável por receber instruções da CPU e manipular os qubits de acordo. Um conjunto específico de instruções é definido para esse componente, consistindo das operações quânticas que o sistema suporta. Essas são modeladas como matrizes, e detalhes de sua realização são abstraídas no artigo.

No trabalho de Negovertic et al é feita uma abordagem diferente para realização da computação de algoritmos quânticos em FPGA (NEGOVETIC et al., 2002). O método foi batizado de *Network of Butterflies*, no qual as operações são realizadas por meio da troca de valores entre os pares de coeficientes de menor distância de Hamming. O trabalho não apresenta resultados de validação em FPGA. Tomamos esse trabalho como base nessa dissertação para implementar as portas quânticas.

No trabalho de Khalid (KHALID; ZILIC; RADECKA, 2004), um modelo totalmente paralelo para simulação de algoritmos quânticos é proposto, onde cada soma e multiplicação necessária para a operação de multiplicação de matrizes são dispostas e executadas em paralelo, a fim de economizar tempo de processamento. Uma mantissa de tamanho fixo

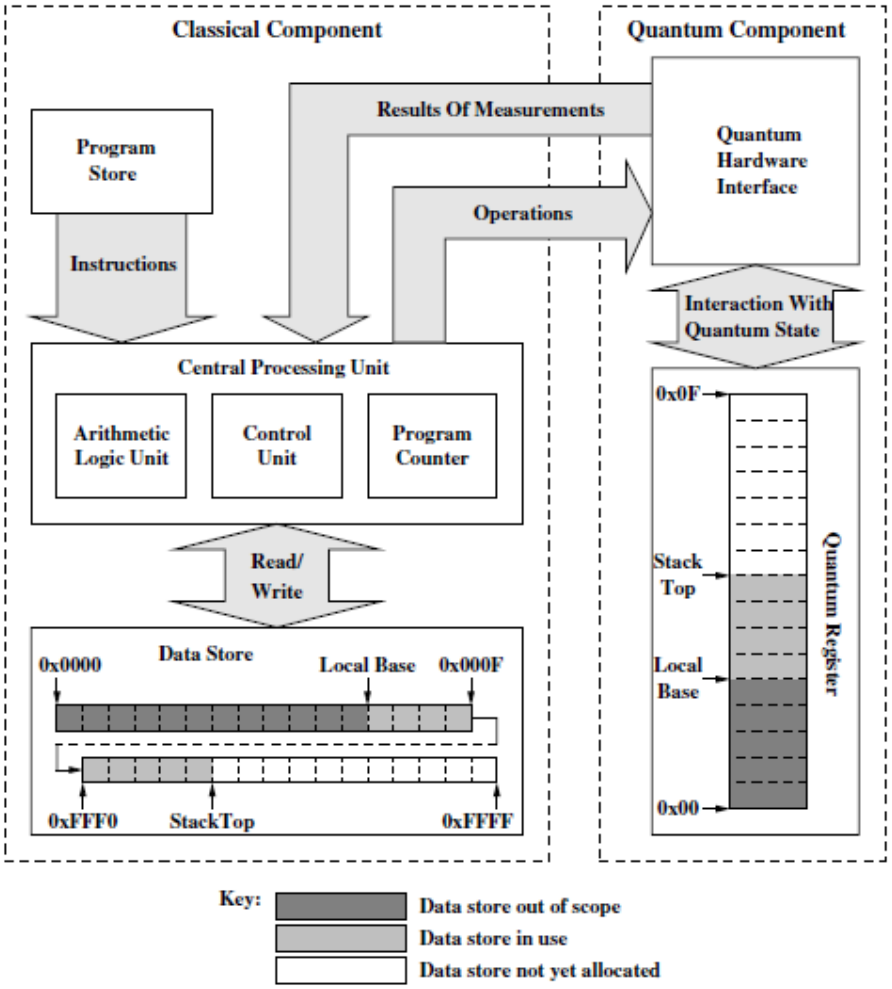


Figura 3.4: Visão geral da máquina SQRAM proposta por Nagarajan et al.

é usada para armazenar valores reais e imaginários para cada coeficiente do registrador quântico. Para evitar problemas com atraso de propagação, essa estrutura é instanciada entre cada porta do algoritmo quântico, conforme ilustração apresentada na Fig. 3.5, retirada do artigo. Uma frequência de relógio de 50 MHz é utilizada para sequenciar a execução. Com isso são alcançados tempos médios de simulação três ordens de magnitude melhores que os resultados obtidos com simulações com o *libquantum*. Contudo, essa solução não apresenta um bom gerenciamento de recursos da FPGA.

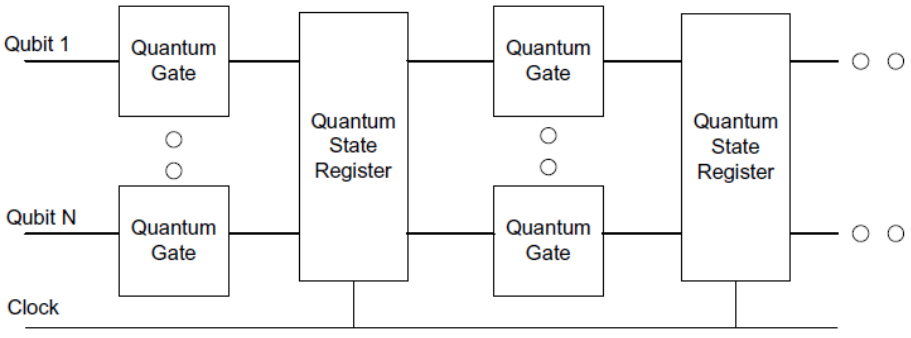


Figura 3.5: Visão geral da implementação proposta por Khalid et al.

Já no trabalho de Aminian et al (AMINIAN et al., 2008) é proposto um outro método para execução de algoritmos quânticos em FPGA. Nele são mantidas algumas estruturas propostas no trabalho de Khalid, mas eles dividem o conjunto de portas quânticas suportadas em dois grupos. O primeiro grupo contém somente portas cuja implementação pode ser feita apenas com troca de coeficientes, a exemplo das portas PauliX e CNot. O segundo grupo contém portas onde é necessário realizar somas e multiplicações na sua realização, a exemplo da porta Hadamard.

O artigo apresenta implementações diferenciadas para cada porta. Com isso, o artigo apresenta que a melhoria no tempo de processamento em relação à simulação em computadores clássicos alcançado no trabalho de Khalid é repetida, mas com resultados cerca de 70% melhores em uso de elementos lógicos da FPGA, tomando como base de comparação um *benchmark* de circuitos binários reversíveis disponível em (MASLOV, 2013).

Uma abordagem integrada de hardware e software para simulação de algoritmos quânticos em FPGA é proposta no trabalho de Mohamed et al (MOHAMED; BADAWY; JULLIEN, 2009). A proposta apresentada sugere o uso de LUTs (*Look-up Table*) para armazenar os valores das funções seno e cosseno. Esses são consultados durante as operações quânticas realizadas sobre os qubits, os quais são considerados individualmente e representados em sua forma polar. Blocos extras de hardware são inseridos para monitorar a ocorrência de estados emaranhados e então ajustar os valores das amplitudes na memória que guarda o vetor de estado.

Na Fig. 3.6, composta por figuras extraídas do artigo original, são ilustrados o fluxo de execução proposto e o diagrama de blocos sugerido para realização do processamento. No trabalho é sugerido que a descrição VHDL seja automaticamente gerada e carregada na plataforma FPGA alvo, contudo nenhum resultado experimental, tanto da ferramenta em software quanto da implementação em hardware, é apresentado.

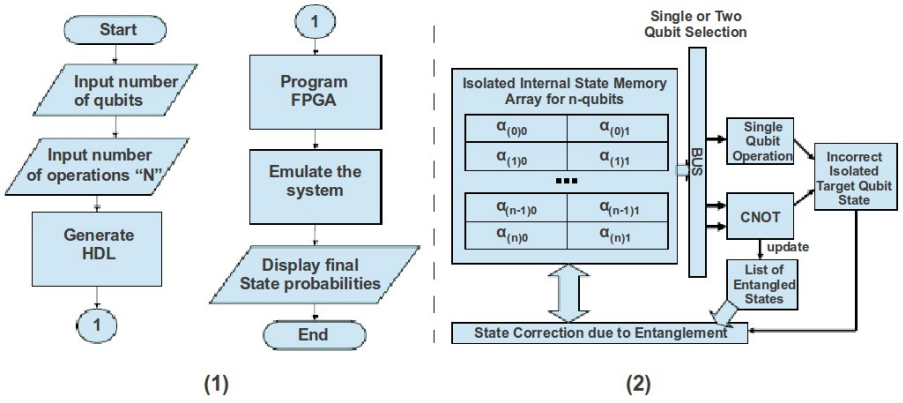


Figura 3.6: Fluxo (1) e arquitetura (2) propostos por Mohamed et al.

Ao se utilizar FPGA para realizar tais simulações, transfere-se a complexidade do problema de simular algoritmos quânticos do domínio do tempo para o domínio do espaço, uma vez que os elementos de hardware necessários ao processamento das operações básicas são replicados para execução paralela, para que cada operador quântico que constitui o algoritmo seja executado de maneira atômica.

Acontece que essa abordagem recai em três problemas principais: (1) o consumo exponencial das células lógicas disponíveis na FPGA com o aumento do sistema simulado, seja em número de qubits, no tamanho da mantissa, ou no número de portas do algoritmo,

(2) aumento também exponencial do tempo de síntese da descrição HDL do sistema de acordo com seu tamanho, e ainda (3) é necessário conhecer o fluxo de projeto para FPGA para que esta seja utilizada.

Para abordar esses problemas é proposta nessa dissertação uma arquitetura de co-processador tipo SIMD (*Single instruction, Multiple Data*), específico para simulação de algoritmos quânticos. Tal arquitetura, que trata do dado como um vetor e que normalmente opera sobre todo ele em um único ciclo de clock, foi apresentada pela Intel na linha de produtos Pentium, compreendendo um conjunto de instruções altamente otimizadas para o processamento de tarefas multimedia (STALLINGS, 2010). A descrição HDL do co-processador é automaticamente gerada pela ferramenta que desenvolvemos. Não encontramos trabalhos relacionados que tratem especificamente da geração automática da descrição de hardware voltado para este fim.

4 CO-PROCESSADOR

Neste capítulo apresentamos a arquitetura que propomos, juntamente com uma implementação alternativa que também desenvolvemos ao longo deste trabalho.

4.1 Portas Quânticas

A arquitetura proposta para o co-processador está baseada no modelo *Network of Butterflies* para realização das portas quânticas, proposta em (NEGOVETIC et al., 2002). Nesse modelo, as operações das portas quânticas são descrita com base nas permutações geradas sobre os coeficientes que descrevem o vetor de estado do sistema. Um exemplo do funcionamento desse modelo é mostrado na Fig. 4.1, retirada do artigo original.

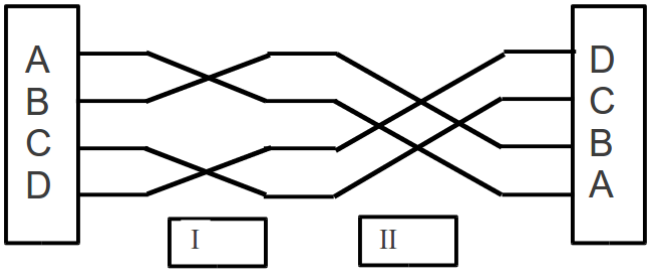


Figura 4.1: O modelo *Network of Butterflies*, aplicando duas portas Pauli X em sequência

4.1.1 Pauli X

Como explicado no capítulo 2, a funcionalidade da porta Pauli X consiste em trocar os coeficientes associados aos vetores base do sistema. Quando aplicada a um sistema com um único qubit, o resultado de sua aplicação é a troca dos coeficientes associados aos dois vetores base $|0\rangle$ e $|1\rangle$ do sistema. Uma vez que a matriz do operador Pauli X tem somente constantes complexas 0 e 1, não é necessário realizar soma ou multiplicação alguma.

Contudo, ao ser aplicada um sistema composto, apesar de continuar sendo aplicada a um único qubit, a funcionalidade da Pauli X deve ser refletida sobre todo o vetor de estado do sistema. Como ilustrado na Fig. 4.2, o comportamento consiste em selecionar os pares corretos de coeficientes para trocar os seus valores, a fim de formar o novo vetor de estado. A seleção é feita com base no índice do qubit sobre o qual porta está sendo aplicada, indicado pelo parâmetro *target*.

Por exemplo, em um sistema com dois qubits cujo estado é $|\psi\rangle$, se a porta Pauli X é aplicada ao qubit 0 – ou seja, *target* é igual a 0 – os pares de coeficientes que terão seus

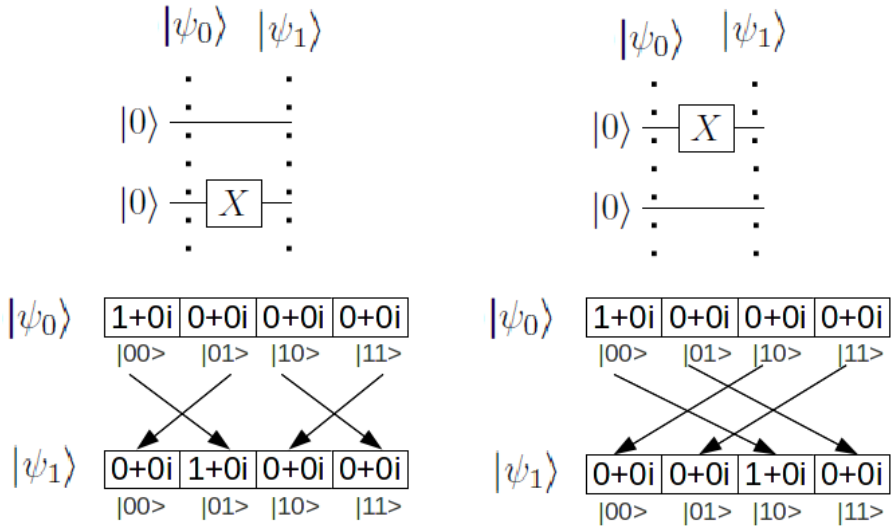


Figura 4.2: Detalhamento do modelo *Network of Butterflies*, usando a porta Pauli X

valores trocados serão: α_{00} e α_{01}; α_{10} e α_{11}. Do mesmo modo, considerando o mesmo sistema mas dessa vez aplicando a porta Pauli X sobre o qubit 1 – *target* igual a 1 – os pares formados serão α_{00} e α_{10}; α_{01} e α_{11}. Assim, o coeficiente α'_{00} do vetor de estados $|\psi')$ resultante da aplicação da porta Pauli X conterá o valor de α_{10} ou de α_{01}, dependendo apenas do valor *target*.

Multiplexadores controlados pelo parâmetro *target* são utilizados para implementar essa funcionalidade no co-processador proposto, como ilustra o esquema apresentado na Fig. 4.3. Formam pares os coeficientes cujos índices binários possuem distância de Hamming igual a 1 e diferem apenas pelo bit de ordem igual ao parâmetro *target*. Com isso, a porta Pauli X pode ser implementada sem a necessidade de realizar soma ou multiplicação. O esquema de funcionamento da porta Pauli X é modelado no Algoritmo 1. O par de um dado coeficiente de índice ***i*** é calculado pela função PAIR em Algoritmo 2.

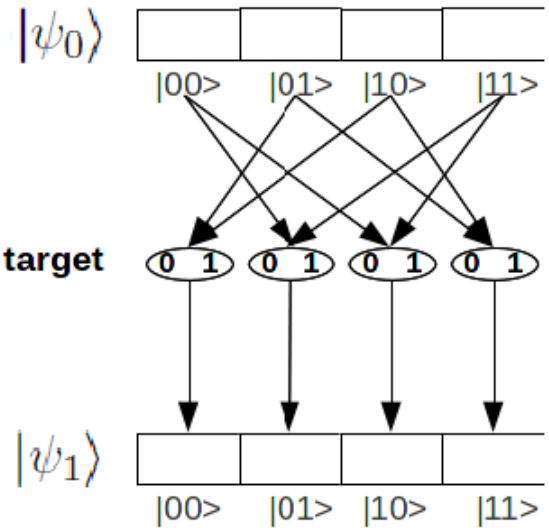


Figura 4.3: Esquema de implementação da porta Pauli X em um sistema com dois qubits

Um exemplo de código SystemVerilog do módulo que implementa o funcionamento da porta Pauli X é apresentado no Código-Fonte 4.1.

Algorithm 1 Função da porta PauliX

```
1: function PAULIX(|ψ), target)
2:     for i = 0 → 2^N − 1 do          ▷ N é o número de qubits
3:         α'_i = α_PAIR(i,target)     ▷ α' são coeficientes de |ψ'), e α de |ψ)
4:     end for
5:     return |ψ')
6: end function
```

Algorithm 2 Função PAIR

```
1: function PAIR(i, target)
2:     return i ⊕ (1 ≪ target)        ▷ Retorna o índice i após inverter o bit na posição target
3: end function
```

Código-Fonte 4.1: Módulo que implementa a porta Pauli X

```
module pauli_x( input [1-1:0]target,
                input qreg in,
                output qreg out
              );
  mux21 mux21_0(
      .ctrl(target),
      .in ({in.at[2],in.at[1]}),
      .out(out.at[0])
  );
  mux21 mux21_1(
      .ctrl(target),
      .in({in.at[3],in.at[0]}),
      .out(out.at[1])
  );
  mux21 mux21_2(
      .ctrl(target),
      .in({in.at[0],in.at[3]}),
      .out(out.at[2])
  );
  mux21 mux21_3(
      .ctrl(target),
      .in({in.at[1],in.at[2]}),
      .out(out.at[3])
  );
endmodule
```

4.1.2 CNOT

O comportamento da porta CNOT pode ser entendido como uma extensão da Pauli X, no entanto condicionada pelo valor do qubit de controle, indicado pelo parâmetro $ctrl_0$. Na implementação, o bit de ordem igual a $ctrl_0$ da representação binária do índice do coeficiente do vetor de estado é levado em consideração para realizar o controle. Somente se este é igual a 1, os valores dos pares de coeficientes selecionado no primeiro nível de

multiplexadores são trocados. Na Fig. 4.4 é mostrado o esquema de implementação dessa porta.

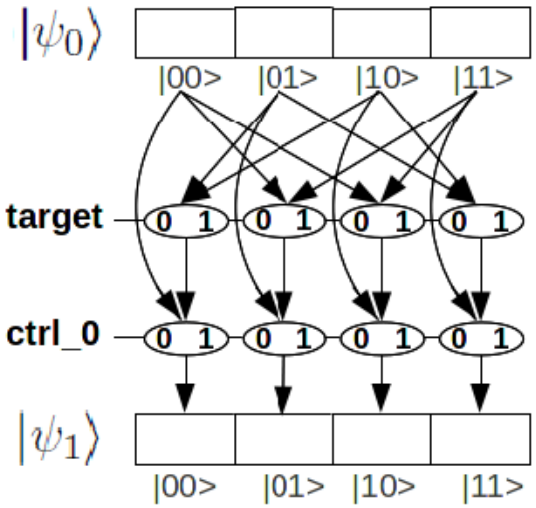


Figura 4.4: Esquema de implementação da porta CNOT em um sistema com dois qubits

Como exemplo, considere novamente um sistema composto de dois qubits cujo vetor de estado é indicado por $|\psi)$. Uma porta CNOT é aplicada sobre o sistema, com o qubit de índice 1 controlando o qubit de índice 0 – $ctrl_0$ é igual a 1 e *target* igual a 0. O primeiro nível de multiplexadores implementa as mesmas regras de seleção de pares já discutida para implementação da porta Pauli X, formando assim os pares: α_{00} e α_{10}; α_{01} e α_{11}. O segundo nível de multiplexadores garante que somente os pares selecionados cujo bit de ordem $ctrl_0$ da representação binária é igual a 1 serão de fato trocados. Os demais pares terão seus valores mantidos. Assim, nesse exemplo, somente o par de coeficientes α_{01} e α_{11} terão seus valores trocados.

O comportamento dessa porta é modelado no Algoritmo 3. A função AT apenas retorna o valor do bit do índice binário em uma dada posição, como mostrado no Algoritmo 4.

Algorithm 3 Função da porta CNOT

```
1:  function CNOT(|ψ), target, ctrl_0)
2:      for i = 0 → 2^N − 1 do
3:          if AT(i, ctrl_0) == 1 then
4:              α'_i = α_PAIR(i,target)
5:          else
6:              α'_i = α_i
7:          end if
8:      end for
9:      return |ψ')
10: end function
```

4.1.3 Toffoli

O esquema de implementação da porta Toffoli segue diretamente da implementação da porta CNOT, pela inserção de uma terceira camada de multiplexadores, essa, por sua vez, controlada pelo parâmetro $ctrl_1$. Seu comportamento é modelado no Algoritmo 5.

Algorithm 4 Função AT

```
1: function AT(i, target)
2:     return i&(1 ≪ target)
3: end function
```

Algorithm 5 Função da porta Toffoli

```
1: function TOFFOLI(|ψ), target, ctrl_0, ctrl_1)
2:     for i = 0 → 2^N − 1 do
3:         if (AT(i, ctrl_0) == 1)&&(AT(i, ctrl_1) == 1) then
4:             α'_i = α_PAIR(i,target)
5:         else
6:             α'_i = α_i
7:         end if
8:     end for
9:     return |ψ')
10: end function
```

4.1.4 Hadamard

A porta Hadamard também tem um esquema de implementação muito similar ao da Pauli X. A diferença é que nessa porta os valores dos coeficientes de $|\psi)$ devem ser transformados antes de serem escritos em $|\psi')$. As operações realizadas são descritas nas equações (4.1) e (4.2), derivadas de representação matricial da porta Hadamard,

$$\alpha_1 = \frac{\alpha_0 + \beta_0}{\sqrt{2}} \tag{4.1}$$

$$\beta_1 = \frac{\alpha_0 - \beta_0}{\sqrt{2}} \tag{4.2}$$

onde α e β formam um par de coeficientes de $|\psi)$, e α' e β' formam um par de coeficientes de $|\psi')$. É também o valor do parâmetro *target* que controla os multiplexadores que selecionam os pares corretos de coeficientes. Uma ilustração do esquema de implementação da porta Hadamard em um sistema de dois qubits é mostrado na Fig. 4.5. O comportamento da porta Hadamard é modelado pelo Algoritmo 6.

Algorithm 6 Função da porta Hadamard

```
1: function HADAMARD(|ψ), target)
2:     for i = 0 → 2^N − 1 do
3:         if i < PAIR(i, target) then
4:             α'_i = (α_i + α_PAIR(i,target))/√2
5:         else
6:             α'_i = (α_PAIR(i,target) − α_i)/√2
7:         end if
8:     end for
9:     return |ψ')
10: end function
```

Em um sistema composto de N qubits, são necessários multiplexadores de N entradas para realizar a seleção dos pares de coeficientes em cada porta, enquanto os multiplexadores que implementam o controle nas portas CNOT e Toffoli são sempre de duas entradas.

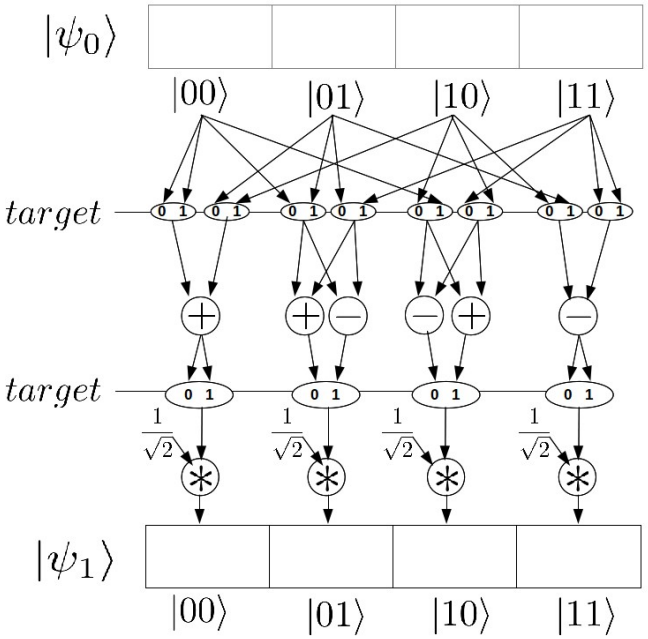


Figura 4.5: Esquema de implementação da porta Hadamard em um sistema com dois qubits

Os controles de todos esses multiplexadores possuem $log_2(N)$ bits. Na elaboração do código RTL do co-processador, é necessário conectar os coeficientes do vetor de estado às entradas corretas de cada multiplexador.

4.2 Representação dos Dados

O registrador quântico, que representa o vetor de estados do sistema, é implementado como dois bancos de registradores de 2^N posições (N é o número de qubits), um para armazenar a parte real e o outro para armazenar a parte imaginária dos coeficientes complexos. O índice dos coeficientes no vetor de estado corresponde diretamente ao endereço do registrador quântico. A vantagem de usar essa estrutura é poder armazenar qualquer estado possível, até mesmo os estados emaranhados.

Por conta da relação de completude, o valor máximo que um dado coeficiente pode assumir tem módulo 1. O dado é representado em ponto fixo, com mantissa de precisão M, onde os primeiros dois bits são suficientes para representar o sinal e a parte inteira, e os demais são usados para representar a parte fracionária do valor do coeficiente.

Uma vez que na computação quântica o vetor de estado inicial é usualmente preparado em um estado puro (sem superposição), apenas uma cadeia de 2^N bits como entrada é suficiente para carregar o estado inicial do sistema. Cada bit controla um único multiplexador, o qual é conectado a um dos 2^N endereços do registrador quântico para carregar o valor complexo correspondente. Por outro lado, o vetor de estado resultante pode estar em um estado de superposição. Portanto, ao final da computação é necessário transmitir todos os $M.2^{N+1}$ bits armazenados no registrador quântico.

Uma alternativa seria realizar uma medição sobre o vetor de estado resultante antes de devolver o dado de saída. Assim, como uma medição faz o estado do sistema colapsar para um de seus estados base, seria necessário transmitir apenas uma cadeia de 2^N bits. Para tanto, seria necessário integrar um gerador de números aleatórios ao sistema, para

simular a medição. Essa abordagem é adotada nos trabalhos (AMINIAN et al., 2008) (KHALID; ZILIC; RADECKA, 2004). Optamos por não implementar dessa forma e realizar a medição externamente, somente quando necessário. Assim podemos acessar o resultado da computação na íntegra.

4.2.1 Organização

O co-processador proposto é uma máquina SIMD (*Single Instruction, Multiple Data*) onde o número de módulos de memória é 2^N, com a diferença que as operações atuam sobre os dados de diferentes módulos de memória, selecionados de acordo com os parâmetros *target*, $ctrl_0$ e $ctrl_1$ da instrução (STALLINGS, 2010). O diagrama de blocos do co-processador é mostrado na Fig. 4.6.

O conjunto de portas de um dado algoritmo é armazenado como instruções na *Instruction Memory*, cujo formato é mostrado na Fig. 4.7. O número de portas do algoritmo quântico – que corresponde diretamente ao número de instruções para o co-processador – é armazenado em um registrador interno quando a *Instruction Memory* é carregada.

A *Control Unit* endereça essa memória em ordem, da primeira à última instrução, começando quando o sinal START é enviado, e gerando o sinal FINISHED ao final, indicando o final da execução.

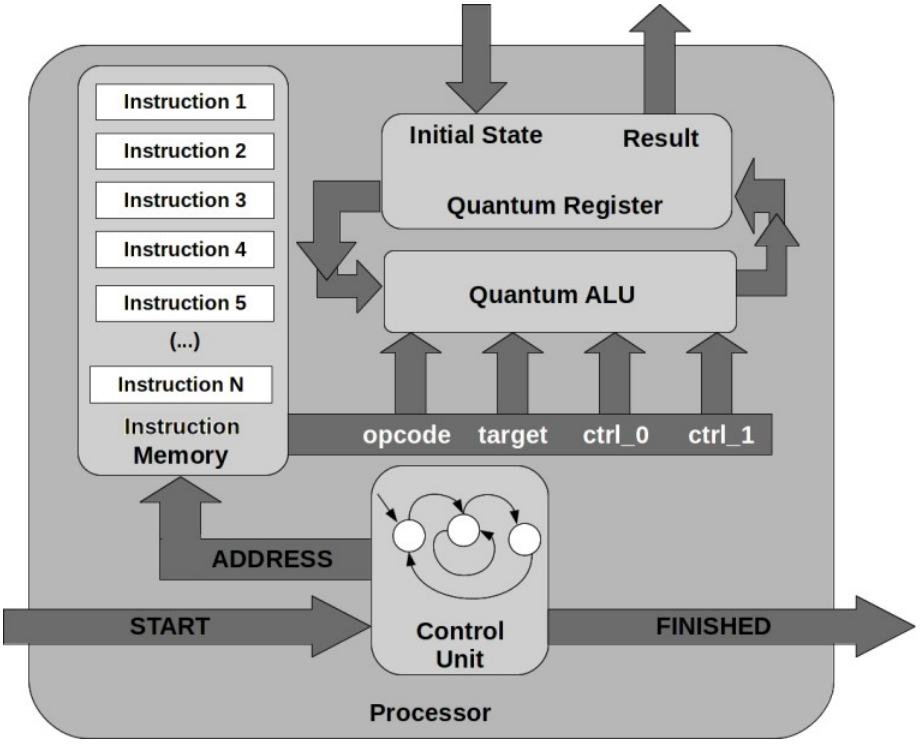


Figura 4.6: Diagrama de blocos do co-processador

Data	$ctrl_1$	$target$	$ctrl_0$	$opcode$
Size (bits)	$\lceil log_2 N \rceil$	$\lceil log_2 N \rceil$	$\lceil log_2 N \rceil$	2

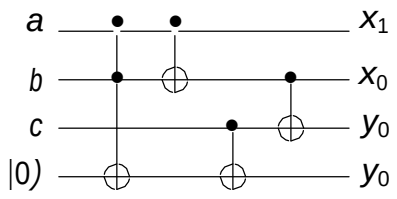
Figura 4.7: Formato da instrução do co-processador

O *Quantum Register* é composto por dois bancos de registradores de tamanho $M.2^N$

bits cada (*M* é a precisão da mantissa, *N* é o número de qubits), um para guardar a parte real, o outro para guardar a parte complexa dos coeficientes do vetor de estado do sistema. Os dois bancos são acessados diretamente e em paralelo durante a execução do co-processador. Existe uma associação direta entre o índice do coeficiente e o endereço onde seus dados encontram-se armazenados em ambos os bancos de registradores.

O módulo *Quantum ALU* tem apenas uma única instância de cada módulo que implementa uma dada porta quântica suportada. Os módulos das portas quânticas operam em paralelo sobre o dado que vem do *Quantum Register*, e somente a saída do módulo que implementa a porta quântica correta é selecionada de acordo com o parâmetro *opcode* da instrução. O dado então é re-escrito no *Quantum Register*. Desse modo, somente uma memória *Quantum Register* é necessário para realizar as operações do algoritmo quântico. Cada instrução do co-processador, que codifica a operação de uma única porta do algoritmo, é realizada em um ciclo de clock.

Uma vez definidos os principais parâmetros dessa arquitetura, que são o número de qubits, a precisão da mantissa, e o tamanho da *Instruction Memory*, a síntese da descrição RTL do co-processador precisa ser realizada uma única vez e carregada na FPGA alvo. Novos algoritmos quânticos podem ser simulados sem necessidade de re-síntese do código RTL, desde que se encaixem nos parâmetros já definidos. Para tanto, faz-se necessário apenas carregar o conjunto de instruções que descreve o algoritmo quântico na *Instruction Memory* do co-processador. Como exemplo, a figura 4.8 apresenta a tradução de um algoritmo quântico descrito no modelo de circuitos para o conjunto de instruções suportados.



Endereço	Instrução [a]			
	$ctrl_1$	*target*	$ctrl_0$	*opcode*
0	11	00	10	11
1	*XX*	10	11	10
2	*XX*	00	01	10
3	*XX*	01	10	10

[a] Valores representados em binário. X significa *don't care*

Figura 4.8: Tradução da descrição do circuito RD32 do modelo de circuitos para o conjunto de instruções do co-processador

4.3 Caso Especial

O conjunto de portas implementado nesse trabalho representa um caso especial, e permite que uma implementação alternativa seja proposta. Observamos que para esse conjunto de portas, sempre que dois estados estiverem em superposição, seus coeficientes associados terão a mesma amplitude. Observação similar é apresentada no trabalho de Fujishima (FUJISHIMA et al., 2003), no qual é comentado a respeito dos valores possíveis dos coeficientes do vetor de estado de circuitos quânticos contendo somente portas Hadamard e Pauli X.

A fim de entender melhor essas observações, considere o circuito quântico apresentado na Fig. 4.9. Na tabela 4.1 é apresentada a sequência de valores de cada um dos $2^3 = 8$ coeficientes do vetor de estado do sistema quântico de três qubits após a aplicação de cada uma das portas no circuito quântico em questão.

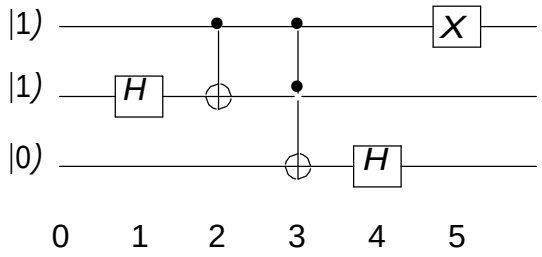


Figura 4.9: Esquemático de um circuito quântico

Tabela 4.1: Sequência de valores dos coeficientes do vetor de estado a cada passo do circuito de exemplo

Passo	Índice do coeficiente do registrador quântico							
	0	1	2	3	4	5	6	7
0	$0+0i$	$0+0i$	$0+0i$	$0+0i$	$0+0i$	$0+0i$	$1+0i$	$0+0i$
1	$0+0i$	$0+0i$	$0+0i$	$0+0i$	$\frac{1}{\sqrt{2}}+i$	$0+0i$	$-\frac{1}{\sqrt{2}}+i$	$0+0i$
2	$0+0i$	$0+0i$	$0+0i$	$0+0i$	$-\frac{1}{\sqrt{2}}+i$	$0+0i$	$\frac{1}{\sqrt{2}}+i$	$0+0i$
3	$0+0i$	$0+0i$	$0+0i$	$0+0i$	$-\frac{1}{\sqrt{2}}+i$	$0+0i$	$0+0i$	$\frac{1}{\sqrt{2}}+i$
4	$0+0i$	$0+0i$	$0+0i$	$0+0i$	$-\frac{1}{2}+i$	$-\frac{1}{2}+i$	$\frac{1}{2}+i$	$-\frac{1}{2}+i$
5	$-\frac{1}{2}+i$	$-\frac{1}{2}+i$	$\frac{1}{2}+i$	$-\frac{1}{2}+i$	$0+0i$	$0+0i$	$0+0i$	$0+0i$

Note que em qualquer passo da execução, no máximo quatro valores complexos distintos aparecem nos coeficientes do vetor de estado, a saber: 0, 1, $\frac{1}{\sqrt{2^k}}$ e $-\frac{1}{\sqrt{2^k}}$, para um dado valor k. De fato, considerando esse conjunto de portas quânticas, esses são os únicos valores possíveis dos coeficientes assumirem. Considerando nH_i como o número de portas Hadamard aplicadas a um dado qubit i, e % o operador que retorna o resto da divisão inteira entre dois números inteiros, o valor k pode ser calculado pela equação 4.3.

$$\sum_{i}^{N} nH_i \% 2 \tag{4.3}$$

Em outras palavras, o valor do expoente k é dado pelo número de qubits em superposição em um dado ponto da execução. A divisão por dois na equação 4.3 vem do fato da porta Hadamard ser reversível, de modo que a sua dupla aplicação sobre um qubit equivale à operação identidade, ou seja, mantém o qubit em seu estado anterior.

Em um sistema com N qubits, o valor k varia de 0 a N; $k = 0$ indicando que não há qubits em superposição, e $k = N$ indicando que todos qubits estão em superposição. Pode-se considerar o valor 1 de um dado coeficiente como o caso particular em que $k = 0$. Vale ressaltar que a aplicação das portas Pauli X, CNOT e Toffoli apenas resultam em troca de valores entre os coeficientes, e não alteram o conjunto de valores armazenados nos coeficientes do vetor de estado em um dado momento.

4.3.1 Implementação

Observamos que era possível diminuir a precisão da mantissa para apenas dois bits, a qual passaria a armazenar tão somente os índices de uma LUT de quatro valores. Foi necessário adequar somente a implementação da porta Hadamard, considerando que ela é a única porta, dentre as portas suportadas, que é capaz de alterar o conjunto de valores contidos no registrador quântico em um dado momento. Associamos os índices da LUT ao respectivos valores da seguinte forma:

$$00 \rightarrow 0$$
$$01 \rightarrow 1$$
$$10 \rightarrow \frac{1}{\sqrt{2^k}}$$
$$00 \rightarrow -\frac{1}{\sqrt{2^k}}$$

Relembrando as equações (4.1) e (4.2), que definem a operação da porta Hadamard, foi possível mapear as combinações de índices dos operandos e os índices dos resultados, conforme apresentado na Tabela 4.2. Vale ressaltar que após a aplicação da porta Hadamard, os valores dos coeficientes do vetor de estado resultante serão 0 e $\pm\frac{1}{\sqrt{2^{k-1}}}$, ou serão 0 e $\pm\frac{1}{\sqrt{2^{k+1}}}$.

Tabela 4.2: Combinações de operandos e resultados possíveis da porta Hadamard em sua implementação compactada

Operandos		Resultados	
α	β	+	−
00	00	00	10
00	01	10	11
00	10	10	11
00	11	11	10
01	00	10	10
01	01	*xx*	*xx*
01	10	*xx*	*xx*
01	11	*xx*	*xx*
10	00	10	10
10	01	*xx*	*xx*
10	10	10	00
10	11	00	11
11	00	11	11
11	01	*xx*	*xx*
11	10	00	11
11	11	11	00

Com isso, é possível operar a porta Hadamard diretamente sobre os índices da LUT, sem necessidade de recuperar os valores. Mais ainda, a LUT nem mesmo precisa ser armazenada no co-processador. Também não é necessário guardar o número de portas Hadamard que são aplicadas sobre o qubit, pois, para saber quantos e quais são os qubits que estão em superposição, basta considerar o número de coeficientes cujos valores são diferentes de 0 (índice 00). Esse processo pode ser feito externamente, pelo processador principal da plataforma computacional.

O algoritmo 7 modela a nova implementação da porta Hadamard, onde as funções $HADAMARD\_TABLE\_PLUS$ e $HADAMARD\_TABLE\_MINUS$ codificam a os valores apresentados na tabela 4.2 respectivamente para operação de adição e subtração, recebendo valores α e β como parâmetro, e retornam o resultado correspondente. Desse modo, não é mais necessário instanciar somadores e multiplicadores na implementação da porta Hadamard.

Algorithm 7 Implementação alternativa da porta Hadamard

```
1:  function HADAMARD(|ψ), target)
2:      for i = 0 → 2^N − 1 do
3:          if i < PAIR(i, target) then
4:              α'_i = HADAMARD_TABLE_PLUS(α_i, α_PAIR(i,target))
5:          else
6:              α'_i = HAMAMARD_TABLE_MINUS(α_PAIR(i,target), α_i)
7:          end if
8:      end for
9:      return |ψ')
10: end function
```

Outra melhoria possível de ser feita foi integrar os módulos que implementam as portas Hadamard, Pauli X, CNot e Toffoli em um único módulo, conforme ilustração apresentada na Fig. 4.10. Uma instância desse módulo integrado é feita para cada um dos 2^N coeficientes do *Quantum Register*. As instâncias são feitas no módulo *Quantum ALU*, o qual permanece responsável por selecionar a saída correta de acordo com o valor do parâmetro *opcode*.

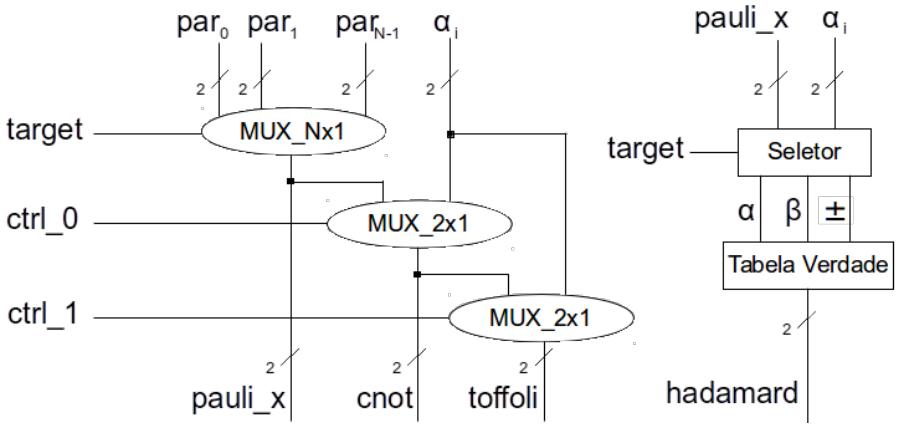


Figura 4.10: Esquema da implementação do módulo integrado das portas suportadas

Essa abordagem alternativa também diminui a quantidade de dados que precisa trafegar pelo link de comunicação. Em contrapartida, essa compactação sobre os dados da mantissa é feita considerando que as portas Pauli X, CNOT, Hadamard e Toffoli são as únicas suportadas. A adição do suporte à outras portas, especialmente as que necessitam de soma e multiplicação, irá ampliar consideravelmente a quantidade de valores possíveis, ampliando assim o tamanho da LUT e, por consequência, o tamanho da mantissa. A abordagem inicial, apresentada anteriormente no capítulo, suporta a inserção de novas portas quânticas, bastando para isso codificar e incluir o módulo correspondente.

5 RESULTADOS

Neste capítulo são apresentados os resultados obtidos com a arquitetura proposta em suas duas alternativas de implementação. Também são apresentadas as comparações com os resultados de trabalhos similares. Apresentamos ainda a ferramenta que desenvolvemos para auxiliar no uso do co-propocessador que propomos em simulações de algoritmos quânticos em FPGA.

5.1 Sobre a ferramenta

A ferramenta que desenvolvemos usa como base o código-fonte da ferramenta de código aberto JQuantum para simulação de algoritmos quânticos. Os resultados gerados pela implementação original da ferramenta foram usados como modelo de referência durante a implementação e testes do co-processador proposto, e em todos os testes realizados nossos resultados corresponderam com os da ferramenta. Algumas modificações na GUI original da JQuantum foram feitas para suportar a execução em FPGA. Uma captura de tela da interface gráfica modificada é apresentada na Fig. 5.1.

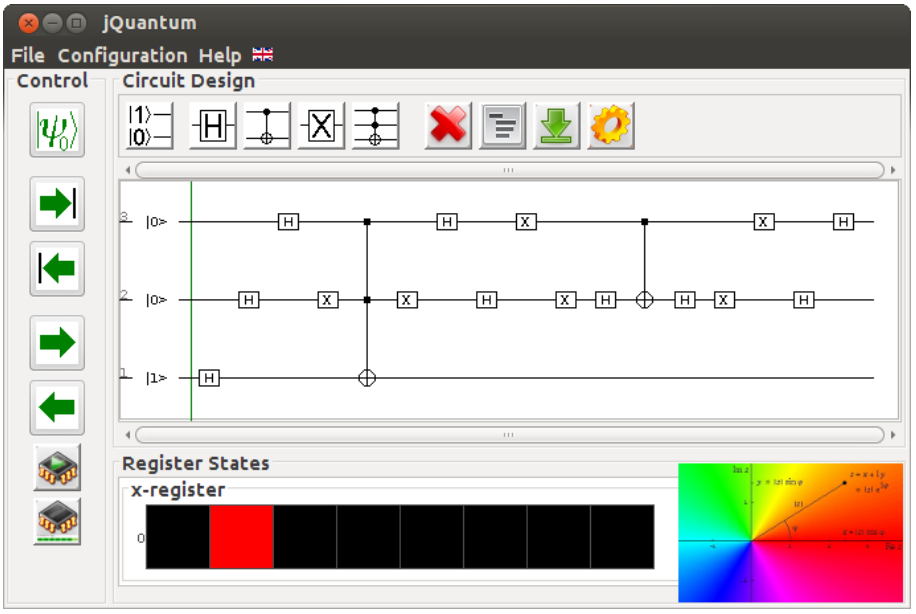


Figura 5.1: GUI modificada da ferramenta que desenvolvemos

O algoritmo quântico é projetado usando o modelo de circuitos e seu esquemático é mostrado no painel central da ferramenta. No painel de baixo da tela é mostrado o conteúdo do vetor de estado do sistema, onde cada coeficiente é representado pelo endereço

do registrador quântico. As cores representam valores complexos de acordo com o mapa de cores mostrado – uma funcionalidade original do simulador JQuantum.

Uma vez que os parâmetros principais do co-processador são informados, como número de qubits, tamanho da mantissa e tamanho da *Instruction Memory*, a ferramenta gera automaticamente uma descrição RTL (*Register Transfer Level*) sintetizável do co-processador, independente de biblioteca de terceiros, em linguagem SystemVerilog. Qualquer ferramenta de terceiros pode ser utilizada para sintetizar o código e prototipá-lo na FPGA. Uma vez prototipado, a comunicação entre a estação de trabalho e o co-processador pode ser feita na própria ferramenta, para carregar o algoritmo projetado na *Instruction Memory* do co-processador e para coletar e mostrar o resultado obtido no processamento.

Para validação dos experimentos realizados, utilizamos uma interface serial RS-232 para comunicação entre a FPGA e a estação de trabalho. A estrutura montada compreende ainda duas máquinas de estados, uma para recebimento dos dados enviados pela estação de trabalho (FSM_Rx), outra para devolução dos resultados gerados(FSM_Tx). Uma ilustração do esquema é apresentado na Fig. 5.2. Uma foto da estação de trabalho usada no desenvolvimento deste trabalho é apresentada na Fig. 5.3.

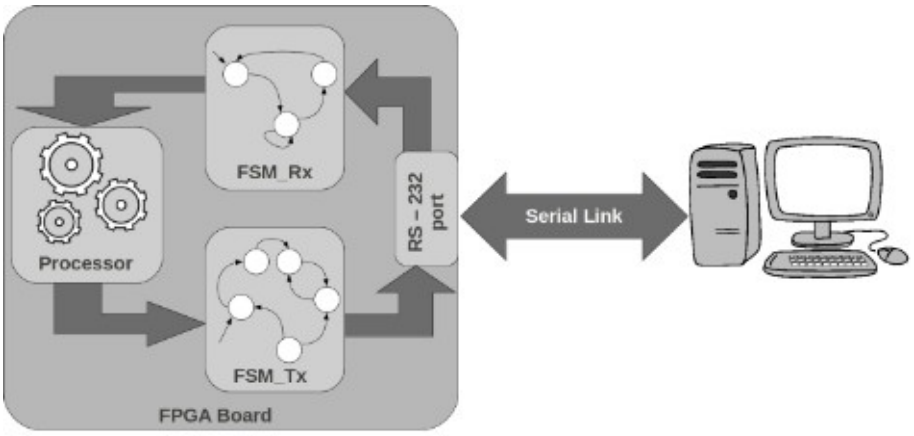


Figura 5.2: Estrutura da comunicação entre a FPGA e a estação de trabalho

A ferramenta funciona gerando o código RTL sintetizável em linguagem SystemVerilog do co-processador, de acordo com os parâmetros definidos pelo usuário, como número de qubits, precisão da mantissa, e a placa FPGA alvo. Isso reduz a necessidade do desenvolvedor do algoritmo quântico conhecer uma HDL para realizar suas simulações em FPGA. Durante a geração do código, a ferramenta basicamente realiza as conexões dos componentes seguindo os passos descritos no capítulo anterior, poupando assim tempo de codificação.

5.2 Nossos resultados

Nos testes reportados a seguir nós prototipamos o co-processador gerado por nossa ferramenta em uma placa DE2 com um chip FPGA Altera Cyclone II EP2C35F672C6 (ALTERA, 2001), e ajustamos sua frequência de funcionamento em 50 MHz. Para sintetizar o código HDL usamos o software Quartus II v.13 da Altera, com a compilação paralela habilitada, em um computador com CPU Intel i7-3770 3.40GHz e 16 GB de RAM rodando Linux Ubuntu 64 bits. Inicialmente são apresentados os resultados com a

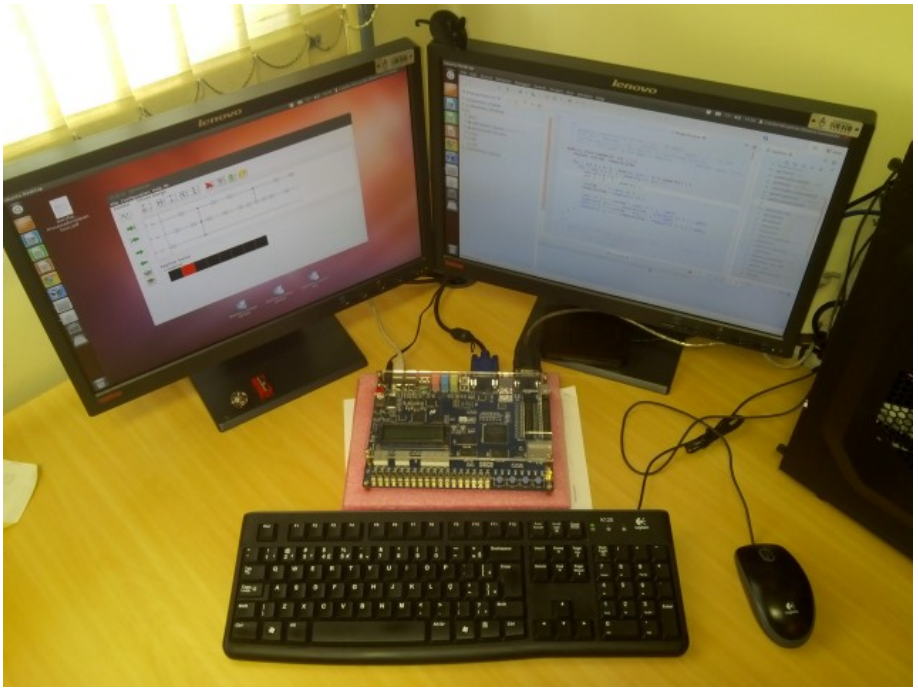


Figura 5.3: Foto da estação de trabalho usada no desenvolvimento deste trabalho

primeira proposta apresentada no capítulo anterior.

5.2.1 Tempo de síntese

Na Fig. 5.4 é mostrado que o tempo necessário para realizar a síntese da descrição do co-processador aumenta exponencialmente ao aumentar o número de qubits do sistema. Por outro lado, nossa proposta demanda re-síntese somente se modificações no número de qubits ou no tamanho da mantissa são feitas.

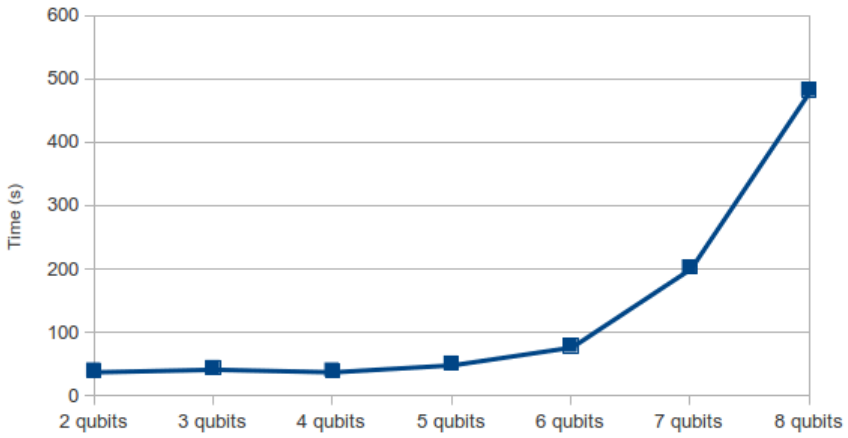


Figura 5.4: Gráfico do tempo de síntese em função do número de qubits. Mantissa fixa em 8 bits

Tabela 5.1: Uso de células lógicas variando-se o número de Qubits e tamanho da Mantissa

Módulo[1]	1 qubit	2 qubit	3 qubit			4 qubit	5 qubits
			8 bits	16 bits	32 bits		
Quantum ALU	229	531	1402	3526	9408	2982	6869
Quantum Register	32	64	128	256	512	256	512

[1]A mantissa tem tamanho default de 8 bits. Os valores do Quantum Register são expressos em número de flip-flops

5.2.2 Número de elementos lógicos

Esse crescimento exponencial no tempo de síntese é consequência direta do crescimento exponencial do número de células lógicas necessárias à medida que o sistema quântico simulado cresce em número de qubits ou em tamanho da mantisa, como pode ser verificado na Tabela 5.1. O tamanho da *Instruction Memory* não é mostrado porque possui um tamanho constante, configurável antes de gerar a descrição HDL do processador.

Os resultados indicam que a Quantum ULA é o principal responsável pelo aumento exponencial do tamanho do sistema em número de células lógicas. Nossa solução, contudo, é invariante quanto ao número de portas do algoritmo quântico simulado, uma vez que as portas quânticas são armazenadas como instruções na *Instruction Memory* do co-processador, o qual é de tamanho fixo, configurado na ferramenta antes da síntese.

Tentamos ainda descobrir o maior co-processador que poderia ser prototipado em nossa FPGA, a qual contém cerca de 35 mil elementos lógicos. Na Fig. 5.5 é mostrada a curva exponencial do uso de células lógicas em função do número de qubits suportados pelo co-processador. Os elementos disponíveis na FPGA são rapidamente consumidos, e um co-processador de tamanho máximo de 6 qubits com precisão de 8 bits na mantissa pode ser prototipado.

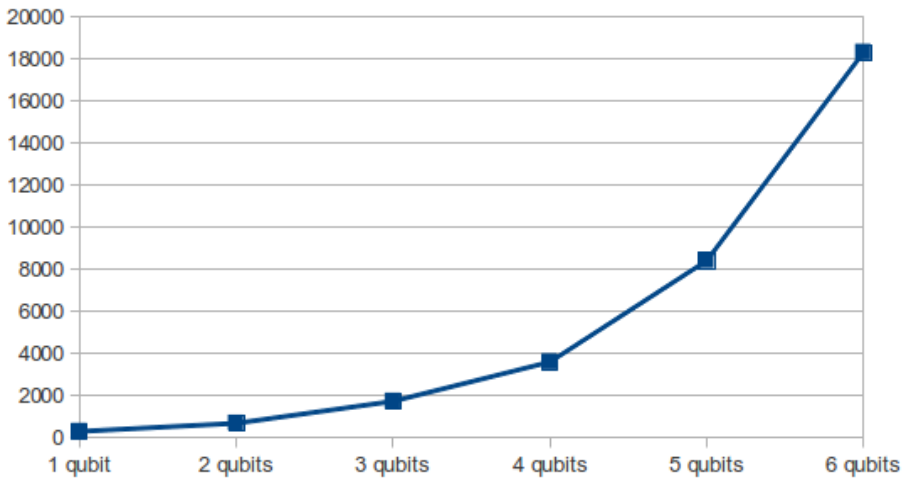


Figura 5.5: Uso de elementos lógicos variando-se o número de qubits e a precisão da mantissa

Por outro lado, no co-processador proposto o número de qubits e a precisão mantissa são fixados antes da síntese, e então é sintetizado somente uma vez para poder simular qualquer circuito quântico que use até o número de qubits suportado. Isso abre precedente para uma investigação da implementação do co-processador seguindo o fluxo de projeto

para ASIC (*Application Specific Integrated Circuit*), no qual o co-processador poderia ser implementado com o número de elementos lógicos necessários, sem estar limitado pelo tamanho da FPGA.

5.2.2.1 Resultados da síntese para o caso especial

Realizamos também a síntese do co-processador fazendo uso da implementação alternativa apresentada no capítulo anterior. Como era de se esperar, devido às características do domínio do problema, a utilização de recursos manteve o aumento exponencial à medida que é aumentado o número de qubits do sistema simulado. Contudo, percebe-se uma diminuição acentuada da utilização de elementos lógicos em relação à abordagem inicial. Como é mostrado no gráfico da Fig. 5.6, seguindo a implementação alternativa dessa vez foi possível simular um sistema com nove qubits na mesma placa FPGA DE2.

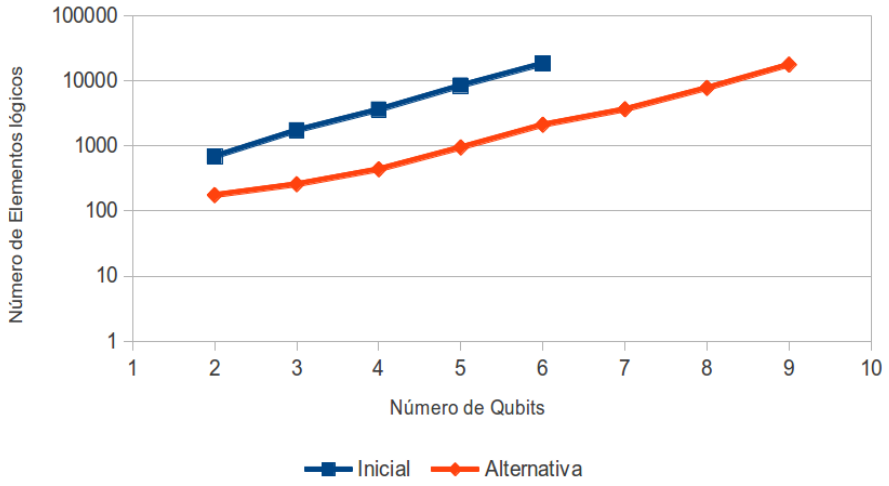


Figura 5.6: Comparação entre a utilização de elementos lógicos da implementação inicial e da implementação alternativa

5.2.3 Tempo de execução

O tempo de execução de qualquer algoritmo quântico no co-processador que propomos é linear com o número de portas do circuito quântico que o descreve. Uma vez que a operação de cada porta é realizada em um único ciclo de relógio, o tempo de execução pode ser calculado diretamente como a multiplicação do período de relógio pelo número de portas quânticas no algoritmo.

Uma característica negligenciada nas discussões apresentadas nos trabalhos relacionados é que o tempo gasto com a comunicação entre a placa FPGA e a estação de trabalho pode ser o gargalo de todo o processo. Isso de fato aconteceu nos testes realizados, uma vez que a interface de comunicação escolhida para realizar os testes de validação deste trabalho, a serial RS-232, toma boa parte do tempo total de processamento, como apresentado na Tabela 5.2 para alguns circuitos.

5.3 Resultados comparados

Nos resultados apresentados a seguir, comparamos nossos dados apenas com os resultados dos trabalhos de Aminian et al e Khalid et al por serem mais próximos ao nosso

Tabela 5.2: Comparativo dos tempos de execução

Circuito	No. de Portas	JQuantum	FPGA s/ link	FPGA c/ link
3_17*c*	6	4.7ms	0.12us	5.6ms
Ham3tc	5	3.1ms	0.1us	5.5ms
Hwb4-11-23	11	3.4ms	0.22us	7.1ms
rd32	4	3.1ms	0.08us	7.9ms

Tabela 5.3: Uso de células lógicas em trabalhos anteriores

	# de Qubits	# de Portas	(AMINIAN et al)		(KHALID et al)	
			8 bits	16 bits	8 bits	16 bits
Ham3tc	3	5	24	24	800	1440
3_17tc	3	6	24	24	960	1728
rd32	4	4	48	48	1280	2304
HWb4-11-23	4	11	64	64	3520	6336
xor5d1	5	4	128	128	2560	4608
Full Adder	5	6	128	128	3840	6912
Mod5d1	5	8	224	224	5129	9216
graycode6	6	5	320	320	6400	11520
Mod5adder-15	6	15	576	576	19200	34560
Rd53d2	8	12	3072	3072	-	-

[1]dados compilados das tabelas 4 e 5 de (AMINIAN et al., 2008)

trabalho proposto, e os únicos dentre a bibliografia pesquisada que apresentam resultados quantitativos (AMINIAN et al., 2008) e (KHALID; ZILIC; RADECKA, 2004).

5.3.1 Fatores na utilização de células lógicas

O número de elementos lógicos da FPGA utilizados nas soluções apresentadas por Aminian et al e Khalid et al é dependente de três parâmetros: número de qubits, precisão da mantissa, e número de portas quânticas no circuito quântico simulado, como pode ser conferido na tabela 5.3. Nela, os resultados de ambas as abordagens foram ordenados respectivamente por número de qubits e por número de portas quânticas no algoritmo.

Pode-se perceber também o aumento exponencial no número de células lógicas utilizadas ao aumentar esses parâmetros. Em nosso trabalho, em contrapartida, o grau de utilização de elementos lógicos da FPGA independe do número de portas quânticas, como discutido anteriormente.

5.3.2 Tempo de síntese

Na figura 5.7 é apresentado que os tempos de síntese nos trabalhos de Aminian et al e Khalid et al seguem a tendência exponencial da utilização de recursos da FPGA (AMINIAN et al., 2008) (KHALID; ZILIC; RADECKA, 2004). Nas abordagens propostas por Aminian et al e Khalid et al, uma nova síntese é necessária a cada modificação no algoritmo simulado. Nosso trabalho, como já discutido, apenas demanda uma nova síntese quando mudanças maiores são feitas, como no número de qubits ou na precisão da mantissa. Mudanças no algoritmo são simplesmente carregadas na memória do co-processador já sintetizado e prototipado em FPGA.

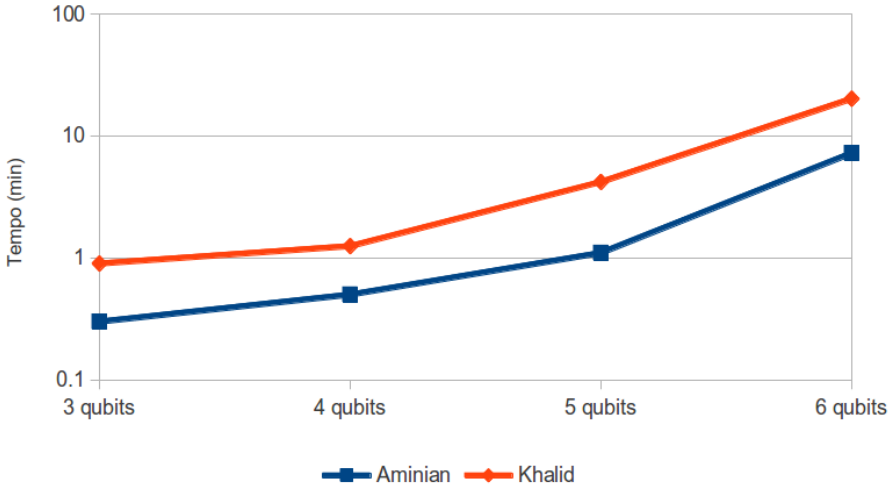


Figura 5.7: Tempos de síntese nos trabalhos relacionados

Tabela 5.4: Comparação do uso de células lógicas

Circuit	Nosso		(AMINIAN et al)		(KHALID et al)	
	8 bits	16 bits	8 bits	16 bits	8 bits	16 bits
ham3tc	140	140	24	24	800	1440
3_17tc	150	150	24	24	960	1728
rd32	191	191	48	48	1280	2304
hwb4-11-23	230	230	64	64	3520	6336
xord51	416	409	128	128	2560	4608
Mod5d1	571	573	224	224	5120	9216
Greycode6	903	905	320	320	6400	11520
rd53d2	4019	4019	3072	3072	-	-

5.3.3 Elementos lógicos em relação a um benchmark

Também comparamos nossos resultados em número de células lógicas para o mesmo benchmark de circuitos reversíveis usado no trabalho de Aminian et al e disponíveis em (MASLOV, 2013). Para tornar a comparação mais justa, mantivemos na *Quantum ALU* somente instâncias de portas que eram utilizadas em cada circuito do benchmark. Adicionalmente, nós reduzimos o tamanho da *Instruction Memory* para caber somente o número de portas em cada circuito. Uma nova síntese foi necessária para cada circuito do benchmark. Essas mudanças temporárias tornaram nossa implementação também dependente do número de portas no circuito quântico. Os resultados são mostrados na Tabela 5.4. Nossos resultados escalam de maneira semelhante aos resultados em (AMINIAN et al., 2008) e superam os resultados em (KHALID; ZILIC; RADECKA, 2004).

Na Tabela 5.4 não há variação no uso de células lógicas quando o tamanho da mantissa é incrementado, diferindo dos resultados apresentados na Tabela 5.1. Isso acontece porque nenhum dos circuitos do benchmark adotado faz uso da porta Hadamard, única porta dentre as demais suportadas por nosso co-processador que é sensível a esse parâmetro.

Em (KHALID; ZILIC; RADECKA, 2004) um módulo que implementa uma dada porta quântica é instanciado no circuito para cada ocorrência desta no circuito quântico, e ainda um registrador quântico é instanciado entre cada um desses módulos, de forma

que não há reuso. Nossos resultados superam os deles porque temos apenas uma única instância do registrador quântico, o qual é realimentado com a saída da Quantum ALU a cada ciclo de relógio. Além disso, somente uma única instância de cada porta presente no circuito é feita na Quantum ALU.

Em (AMINIAN et al., 2008), o conjunto de portas suportadas foi dividido em dois subconjuntos, um contendo portas como a Hadamard, que envolvem adição e multiplicação, e o outro contendo portas que apenas realizam trocas de amplitudes como as portas quânticas Pauli X e Feynman. Cada subconjunto contém sua própria representação de dados. Os circuitos do benchmark fazem uso somente de portas do segundo subconjunto, e isso pode explicar o resultado melhor alcançado por eles, já que em nossa proposta há somente uma representação de dados única para todas os tipos de porta.

Por outro lado, pode-se considerar a utilização média de células lógicas dos circuitos, organizados por número de qubits. No gráfico apresentado na Fig. 5.8 é mostrado a razão entre as médias dos nossos resultados e as médias dos resultados em (AMINIAN et al., 2008). A utilização de células lógicas em nossa proposta decresce de 6 para 1.2 vezes a utilização média no trabalho de Aminian et al, ao aumentar o número de qubits do sistema de 3 para 8, o que leva a crer que nossa proposta tende a ser melhor para circuitos com maior número de qubits.

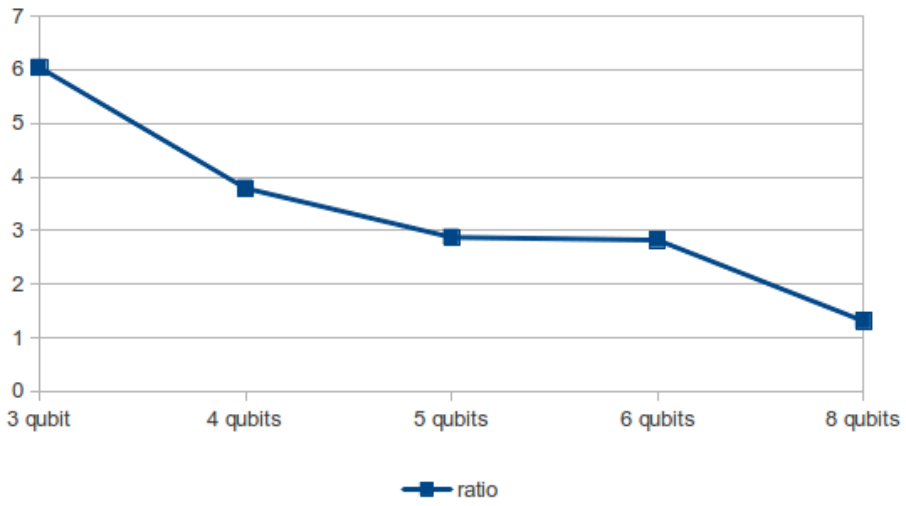


Figura 5.8: Razão entre a média de utilização de células lógicas da nossa abordagem e a de (AMINIAN et al., 2008).

O aumento exponencial visto da utilização de elementos lógicos da FPGA é inerente ao domínio do problema. De fato, tal característica é percebida nos demais trabalhos relacionados. Contudo, a nossa abordagem ameniza essa característica ao ter esse resultado indiferente quanto ao número de portas no circuito quântico simulado. Além disso, nós diminuímos em relação aos demais trabalhos a quantidade de vezes que o sistema precisa ser sintetizado durante o uso prático de FPGAs para simular algoritmos quânticos. Por fim, a ferramenta apresentada facilita consideravelmente a utilização de FPGAs nessas simulações.

6 CONCLUSÕES

Neste trabalho apresentamos um co-processador cuja estrutura se assemelha à de uma máquina SIMD (*Single Instruction, Multiple Data*), especificamente voltado à simulação de algoritmos quânticos em FPGA. Sua descrição RTL é gerada automaticamente por uma ferramenta que desenvolvemos.

Analisando os trabalhos já desenvolvidos, percebemos que três problemas principais permeiam a simulação de algoritmos quânticos utilizando FPGAs, a saber: (1) crescimento exponencial do número de células lógicas necessárias à simulação em FPGA com o aumento do número de qubits, da precisão da mantissa ou do número de portas do algoritmo quântico, (2) o tempo de síntese também cresce exponencialmente com essas três variantes e, (3) o esforço extra de projeto para escrever o algoritmo quântico em uma HDL. Neste trabalho eliminamos dois desses problemas e amenizamos o terceiro.

O modelo utilizado, adaptado do modelo *Network of Butterflies* para implementação em FPGA, permite que o comportamento das portas quânticas possa ser programado, eliminando assim a necessidade de sintetizar novamente a descrição a cada mínima modificação no algoritmo quântico simulado. Apenas modificações no número de qubits ou na precisão da mantissa demandam re-síntese.

A configuração dos circuitos que implementam as portas quânticas é feita por meio de uma instrução ao co-processador, as quais ficam armazenadas em uma memória interna de tamanho fixo. Com isso, eliminamos a influência da quantidade total de portas quânticas do algoritmo sobre o número de elementos lógicos consumidos, amenizando assim o crescimento exponencial do consumo de recursos da FPGA.

Por último, ao propor uma ferramenta que gere automaticamente o código RTL sintetizável do co-processador, a qual é independente de bibliotecas de terceiros, eliminamos boa parte do esforço extra de projeto do simulador em HDL para FPGA. De fato, caso o desenvolvedor do algoritmo quântico disponha da placa FPGA suportada, utilizando a nossa ferramenta ele nem mesmo precisa conhecer uma linguagem de descrição de hardware para realizar suas simulações em FPGA.

Adicionalmente, nos deparamos com o problema da comunicação entre a FPGA e a estação de trabalho, o qual é normalmente negligenciado na literatura relacionada, mas que deve ser considerado caso seja desejado fazer uso prático de FPGAs para esse tipo de simulação. A quantidade de dados que deve ser devolvida para a estação de trabalho ao final da execução do algoritmo na FPGA também cresce exponencialmente, e caso não haja um canal de comunicação eficiente esse pode ser um gargalo de todo o processo.

Os resultados referentes ao uso de células lógicas pelo co-processador se aproxima dos melhores resultados de descrições geradas manualmente disponíveis na literatura. Além disso, considerando o consumo médio de células lógicas à medida que o sistema simulado aumenta em número de qubits, pode-se afirmar que nossos resultados são melhores que

os demais disponíveis na literatura para circuitos suficientemente grandes.

Além disso, a implementação alternativa proposta para o caso especial, fazendo uso de LUTs, permitiu diminuir o número de bits na mantissa, e por consequência, diminuiu consideravelmente o número de elementos lógicos utilizados pelo co-processador. Essa abordagem alternativa, contudo, limita o número de portas quânticas que pode ser suportada pelo co-processador, mas abre espaço para uma investigação futura mais aprofundada sobre a quantidade de valores que de fato são representados dependendo do conjunto de portas suportados, e sobre o *trade-off* entre utilizar LUTs ou representar os dados diretamente na mantissa.

6.1 Trabalhos Futuros

Planeja-se ampliar o número de portas quânticas suportadas pelo co-processador a fim de simular uma variedade maior de algoritmos quânticos. As portas implementadas até então são suficientes para validar a proposta, mas ainda insuficientes para simular alguns dos algoritmos quânticos mais notórios, a exemplo da Transformada Quântica de Fourier.

Está também entre nossos interesses futuros a investigação de outros conjuntos de portas quânticas que permitem realizar implementações otimizadas no processador quântico. Isso, aliado à funcionalidade que pretendemos incluir na nossa ferramenta, de poder escolher a quais portas o co-processador dará suporte, permitirá realizar otimizações na descrição do co-processador de acordo com a demanda do usuário da ferramenta.

Planeja-se também incluir no co-processador suporte à instruções de desvio condicional do fluxo de execução, a fim de suportar repetições de instruções, o que permitirá que blocos de instruções já suportadas pelo co-processador proposto sejam repetidos. Isso resultará em um uso mais otimizado da *Instruction Memory* do co-processador.

Além disso, pretende-se pesquisar uma forma de embutir um instrumento de medida no chip. Com isso, pretende-se dar a opção de medir o estado do sistema ao final da computação, e enviar esse resultado para a estação de trabalho. Deste modo, o sistema estará colapsado em uma de suas bases, reduzindo assim, caso se queira, a quantidade de dados que irá trafegar pelo canal de comunicação e aumentando a eficiência do processo.

Pretende-se também fazer uso de outros canais de comunicação mais eficientes entre a placa FPGA e a estação de trabalho, a exemplo do canal USB e da interface PCI express, disponível em placas mais recentes. Pretende-se ainda simular esses dados em placas FPGAs de maior capacidade, e usá-las para investigar a utilização de algoritmos quânticos no fluxo de projeto de circuitos integrados.

A descrição RTL do co-processador é a mesma independe do algoritmo quântico a ser simulado. A lista de portas que descreve o algoritmo é carregada na memória interna do co-processador já prototipado para então o algoritmo quântico ser simulado. Essa flexibilidade abre precedente para uma investigação futura sobre a viabilidade de implementação desse co-processador em um ASIC (Circuito Integrado de Aplicação Específica, em inglês). Como a principal limitação desse trabalho é o consumo exponencial de recursos, dado o número limitado de células lógicas na FPGA, é possível que uma migração para o fluxo ASIC confira a viabilidade de uso prático para essa abordagem. O interessante é que enquanto a lei de Moore continuar ativa, as chances dessa abordagem ser viável é ainda maior, e as possibilidades de aplicação para esse trabalho tendem a aumentar.

REFERÊNCIAS

ALTERA. **DE2 Development and Education Board - User Manual**. [S.l.]: Altera Corporation, 2001.

AMINIAN, M. et al. FPGA-Based Circuit Model Emulation of Quantum Algorithms. In: SYMPOSIUM ON VLSI, 2008. ISVLSI '08. IEEE COMPUTER SOCIETY ANNUAL. **Anais...** [S.l.: s.n.], 2008. p.399 –404.

BAMPI, S.; REIS, R. Challenges and Emerging Technologies for System Integration beyond the End of the Roadmap of Nano-CMOS. **VLSI-SoC: Technologies for Systems Integration**, [S.l.], p.21–33, 2011.

BUTSCHER, B.; WEIMER, H. **Libquantum**: the C library for quantum computing and quantum simulation. Disponível em: <http://www.libquantum.de/>. Acesso em: Set. 2013.

CHUANG, I. L.; GERSHENFELD, N.; KUBINEC, M. Experimental Implementation of Fast Quantum Searching. **Phys. Rev. Lett.**, [S.l.], v.80, p.3408–3411, Apr 1998.

D-WAVE. **D-Wave - The Quantum Computing Company @ONLINE**. Disponível em: <http://www.dwavesys.com/>. Acesso em: Set. 2013.

FEYNMAN, R. P. Quantum Mechanical Computers. **Optics News**, [S.l.], v.11, n.2, p.11–20, 1985.

FUJISHIMA, M. et al. High-speed processor for quantum-computing emulation and its applications. In: CIRCUITS AND SYSTEMS, 2003. ISCAS '03. PROCEEDINGS OF THE 2003 INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2003. v.4, p.IV–884–IV–887 vol.4.

GEREZ, S. H. **Algorithms for VLSI Design Automation**. 1st.ed. New York, NY, USA: John Wiley & Sons, Inc., 1999.

GROVER, L. K. A fast quantum mechanical algorithm for database search. In: STOC '96: PROCEEDINGS OF THE TWENTY-EIGHTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, New York, NY, USA. **Anais...** ACM, 1996. p.212–219.

HSU, J. D-Wave's Quantum Computing Claim Gets Boost in Testing. **IEEE Spectrum**, [S.l.], Mai 2013. Disponível em: <http://spectrum.ieee.org/tech-talk/computing/hardware/dwave-quantum-computer-shows-promise-in-tests>. Acesso em: Set. 2013.

INTEL. **Excerpts from A Conversation with Gordon Moore**: moore's law. Disponível em: <http://large.stanford.edu/courses/2012/ph250/lee1/docs/Excepts_A _Conversation_with_Gordon_Moore.pdf>. Acesso em: Set. 2013.

KARAFYLLIDIS, I. Quantum Computer Simulator Based on the Circuit Model of Quantum Computation. **Circuits and Systems I: Regular Papers, IEEE Transactions on**, [S.l.], v.52, n.8, p.1590 – 1596, aug. 2005.

KHALID, A.; ZILIC, Z.; RADECKA, K. FPGA emulation of quantum circuits. In: COMPUTER DESIGN: VLSI IN COMPUTERS AND PROCESSORS, 2004. ICCD 2004. PROCEEDINGS. IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2004. p.310 – 315.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet**: uma abordagem top-down. Trad. 3 ed..ed. São Paulo: Addison Wesley, 2006.

MACK, C. Fifty Years of Moore's Law. **Semiconductor Manufacturing, IEEE Transactions on**, [S.l.], v.24, n.2, p.202–207, 2011.

MASLOV, D. **Reversible Logic Synthesis Benchmarks**. Disponível em: <http://webhome.cs.uvic.ca/~ dmaslov/>. Acesso em: Set. 2013.

MCBRIDE, S. Quantum computing firm D-Wave gets 30 mln investment. **Reuters**, [S.l.], Oct 2012. Disponível em: <http://www.reuters.com/article/2012/10/04/dwave-funding-quantumcomputing-idUSL1E8L405P20121004>. Acesso em: Set. 2013.

MCGEOCH, C. C.; WANG, C. Experimental evaluation of an adiabatic quantum system for combinatorial optimization. In: ACM INTERNATIONAL CONFERENCE ON COMPUTING FRONTIERS, New York, NY, USA. **Proceedings...** ACM, 2013. p.23:1–23:11. (CF '13).

MISZCZAK, J. **Models of quantum computation and quantum programming languages**. Disponível em: <http://arxiv.org/abs/1012.6035v2>. Acesso em: Out. 2013.

MOHAMED, T.; BADAWY, W.; JULLIEN, G. On using FPGAS to accelerate the emulation of quantum computing. In: ELECTRICAL AND COMPUTER ENGINEERING, 2009. CCECE '09. CANADIAN CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.175 –179.

MOORE, G. Progress in digital integrated electronics. In: ELECTRON DEVICES MEETING, 1975 INTERNATIONAL. **Anais...** [S.l.: s.n.], 1975. v.21, p.11 – 13.

MOORE, G. E. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. **Solid-State Circuits Society Newsletter, IEEE**, [S.l.], v.11, n.5, p.33–35, 2006.

NAGARAJAN, R.; PAPANIKOLAOU, N.; WILLIAMS, D. Simulating and Compiling Code for the Sequential Quantum Random Access Machine. **Electronic Notes in Theoretical Computer Science**, [S.l.], v.170, n.0, p.101 – 124, 2007. Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005).

NEGOVETIC, G. et al. Evolving quantum circuits and an FPGA-based Quantum Computing Emulator. In: INTERN. WORKSHOP ON BOOLEAN PROBLEMS. **Proceedings...** [S.l.: s.n.], 2002. p.15–22.

NEVEN, H. et al. NIPS 2009 demonstration: binary classification using hardware implementation of quantum annealing. **Quantum**, [S.l.], p.1–17, 2009.

NIELSEN, M. A.; CHUANG, I. L. **Quantum Computation and Information Quantum**. [S.l.]: Bookman, 2000.

POWELL, J. The Quantum Limit to Moore's Law. **Proceedings of the IEEE**, [S.l.], v.96, n.8, p.1247 –1248, aug. 2008.

RIEFFEL, E.; POLAK, W. An Introduction to Quantum Computing for Non-Physicists. **ACM Comput. Surv.**, New York, NY, USA, v.32, n.3, p.300–335, 2000.

SHOR, P. Algorithms for quantum computation: discrete logarithms and factoring. In: FOUNDATIONS OF COMPUTER SCIENCE, 1994 PROCEEDINGS., 35TH ANNUAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 1994. p.124 –134.

STALLINGS, W. **Computer organization and architecture (8th ed.)**: designing for performance. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2010.

STEFFEN, M. et al. Toward quantum computation: a five-qubit quantum processor. **Micro, IEEE**, [S.l.], v.21, n.2, p.24 –34, mar/apr 2001.

UDRESCU, M.; PRODAN, L.; VLADUTIU, M. Using HDLs for describing quantum circuits: a framework for efficient quantum algorithm simulation. In: COMPUTING FRONTIERS, 1., New York, NY, USA. **Proceedings...** ACM, 2004. p.96–110. (CF '04).

VIAMONTES, G.; MARKOV, I.; HAYES, J. Improving Gate-Level Simulation of Quantum Circuits. **Quantum Information Processing**, [S.l.], v.2, n.5, p.347–380, 2003.

VIAMONTES, G.; MARKOV, I.; HAYES, J. High-performance QuIDD-based simulation of quantum circuits. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, 2004. PROCEEDINGS. **Anais...** [S.l.: s.n.], 2004. v.2, p.1354–1355 Vol.2.

VRIES, A. de. **jQuantum - a Quantum Computer Simulator**. Disponível em: <http://jquantum.sourceforge.net/>. Acesso em: Set. 2013.

WARREN, P. The future of computing - new architectures and new technologies. **Nanobiotechnology, IEE Proceedings -**, [S.l.], v.151, n.1, p.1 – 9, 5 2004.

Wu, J. et al. A NANO enhancement to Moore's law. In: SOCIETY OF PHOTO-OPTICAL INSTRUMENTATION ENGINEERS (SPIE) CONFERENCE SERIES. **Anais...** [S.l.: s.n.], 2012. (Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, v.8401).

ZIVIANI, N. et al. **Projeto de Algoritmos**: com implementações em pascal e c. [S.l.]: Thomson, 2004. v.2.

APÊNDICE A EXEMPLO DO ALGORITMO DE GROVER

Com a finalidade de realizar uma analise passo a passo de um algoritmo quântico, considere o circuito a seguir. Trata-se de uma versão simples do algoritmo de busca quântica de Grover, projetado para encontrar em uma base de dados D com quatro elementos qual deles corresponde a um dado critério. Neste caso estamos buscando por $x \in D|x = 10$. O esquema desse algoritmo é apresentado na Fig. A.1.

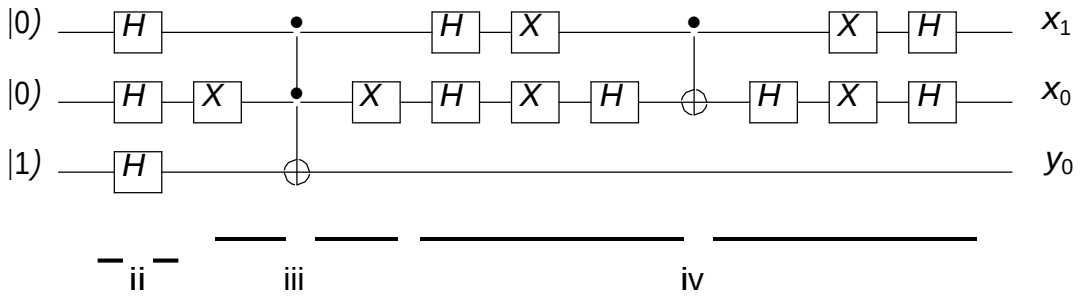


Figura A.1: Exemplo do algoritmo de busca de Grover

Este algoritmo quântico consiste em cinco passos fundamentais descritos a seguir:

i Preparação do estado inicial do sistema;

ii Superposição de todos os valores da base de dados;

iii Sub-circuito do oráculo;

iv Subcircuito de difusão;

v A iteração de Grover, que repete os passos iii e iv m vezes, onde m é dado pela Eq. 2.18.

Para entender o comportamento desse algoritmo, vamos seguir a sua evolução em termos de multiplicação de matrizes.

O sistema tem seu vetor de estado $|\psi_0)$ inicializado como o resultado do produto tensorial de cada representação vetorial dos qubits de entrada individuais. Os dois qubits superiores codificam os valores de x, enquanto o último é o qubit auxiliar, conhecido como qubit do oráculo. Daqui pra frente, considere todas as constantes usadas como pertencentes ao conjunto dos números complexos. Temos então que:

$$|\psi_0\rangle = |0\rangle \otimes |0\rangle \otimes |1\rangle = |001\rangle = \begin{matrix}1\\0\end{matrix} \otimes \begin{matrix}1\\0\end{matrix} \otimes \begin{matrix}0\\1\end{matrix} = \begin{matrix}0\\1\\0\\0\\0\\0\\0\\0\end{matrix}$$

o que significa que o vetor de estado inicial $|\psi_0\rangle$ é representado como:

$$|\psi_0\rangle = \begin{matrix}r\\\\\end{matrix} 0+0i \quad 1+0i \quad 0+0i \quad 0+0i \quad 0+0i \quad 0+0i \quad 0+0i \quad 0+0i \; \begin{matrix}\mathbf{l}_t\\\\\end{matrix}$$

onde $\boldsymbol{t}$ é a operação transposta.

A.1 Passo 1

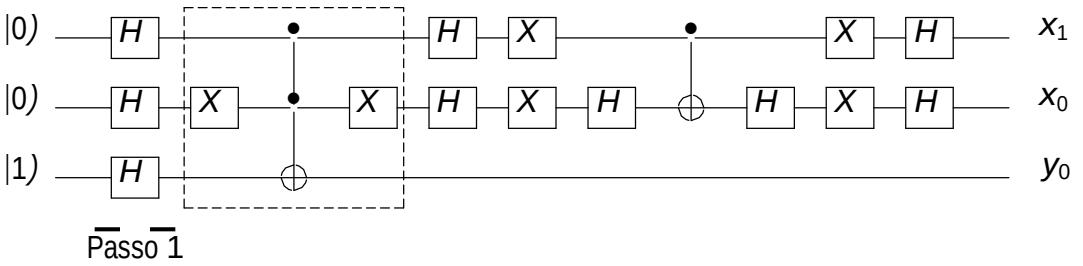


Figura A.2: Exemplo do algoritmo de Grover - Passo 1

O primeiro passo do algororitmo é colocar o vetor de estado do sistema em uma superposição de todos os possíveis valores. Isto é feito por meio da aplicação do operador Hadamard sobre cada qubit da base e sobre o qubit do oráculo, para obter o vetor de estado $|\psi_1\rangle$. Isso é feito combinando-se os três operadores Hadamard de um único qubit por meio da operação produto tensorial e multiplicando esse resultado pelo vetor de estado anterior $|\psi_0\rangle$. Assim,

$$|\psi_1\rangle = (H \otimes H \otimes H)|\psi_0\rangle = \frac{1}{\sqrt{2}^4} \begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{matrix} \begin{matrix}0\\1\\0\\0\\0\\0\\0\\0\end{matrix}$$

portanto:

$$|\psi_1\rangle = \begin{matrix}\text{í}\\\\\end{matrix} \frac{\sqrt{2}}{4}+0i \quad -\frac{\sqrt{2}}{4}+0i \quad \frac{\sqrt{2}}{4}+0i \quad -\frac{\sqrt{2}}{4}+0i \quad \frac{\sqrt{2}}{4}+0i \quad -\frac{\sqrt{2}}{4}+0i \quad \frac{\sqrt{2}}{4}+0i \quad -\frac{\sqrt{2}}{4}+0i \; \begin{matrix}\mathbf{l}_t\\\\\end{matrix}$$

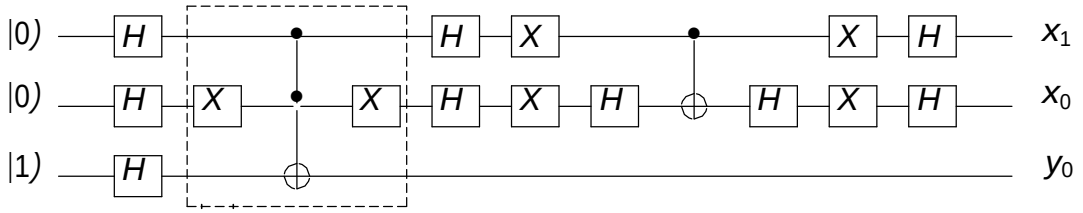


Figura A.3: Exemplo do algoritmo de Grover - Passo 2

A.2 Passo 2

Este é o primeiro passo do sub-circuito **oráculo**, que é responsável por codificar a função de consulta. Neste passo, onde somente uma porta Pauli X aparece no esquemático, o vetor de estado resultante $|\psi_2)$ é obtido combinando-se essa porta com o operador da porta Pauli I (Identidade) por meio do produto tensorial. Então,

$$|\psi_2) = (\boldsymbol{I} \otimes X \otimes \boldsymbol{I})|\psi_1) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}$$

portanto:

$$|\psi_2) = \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}^t$$

A.3 Passo 3

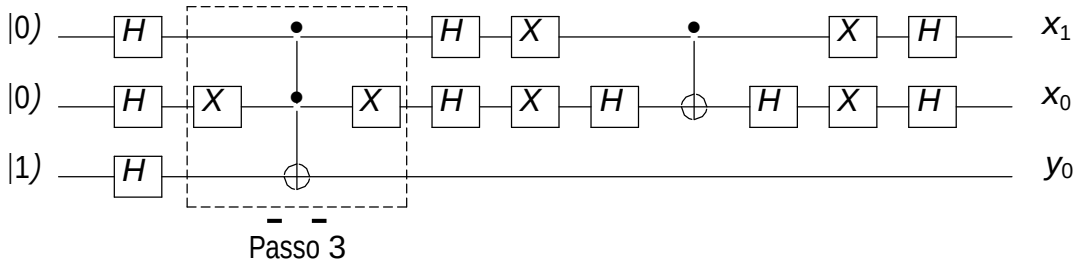


Figura A.4: Exemplo do algoritmo de Grover - Passo 3

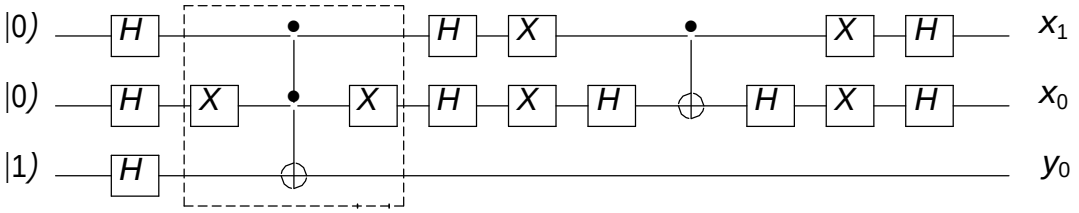
Neste passo, o operador T da porta Toffoli é aplicado sobre o vetor de estado $|\psi_2)$ do sistema para gerar o vetor de estado $|\psi_3)$.

$$|\psi_3) = T|\psi_2) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \\ \frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \\ \frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{2}}{4} + 0i \\ \frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \end{bmatrix}$$

resultando:

$$|\psi_3) = \begin{bmatrix} \frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i & \frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i & \frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i & \frac{\sqrt{2}}{4} + 0i \end{bmatrix}^t$$

A.4 Passo 4



Passo 4

Figura A.5: Exemplo do algoritmo de Grover - Passo 4

Este é o último passo do sub-circuito **oráculo**. Novamente, a porta Pauli X é combinada com duas portas Pauli I por meio do produto tensorial entre elas, e aplicado ao vetor de estado $|\psi_3)$ para gerar o vetor de estado $|\psi_4)$.

$$|\psi_4) = (I \otimes X \otimes I)|\psi_3) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \\ \frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \\ \frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \\ -\frac{\sqrt{2}}{4} + 0i \\ \frac{\sqrt{2}}{4} + 0i \end{bmatrix}$$

resultando:

$$|\psi_4) = \begin{bmatrix} \frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i & \frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i & \frac{\sqrt{2}}{4} + 0i & \frac{\sqrt{2}}{4} + 0i & -\frac{\sqrt{2}}{4} + 0i \end{bmatrix}^t$$

Note que somente as amplitudes dos coeficientes α_{100} e α_{101} estão invertidos em comparação com as amplitudes dos coeficientes de $|\psi_1)$ antes da aplicação do oráculo.

A.5 Passo 5

Este é o primeiro passo do sub-circuito de **difusão** ou **diferenciação de fase** do algoritmo de Grover. Juntamente com o sub-circuito **oráculo**, este sub-circuito forma a

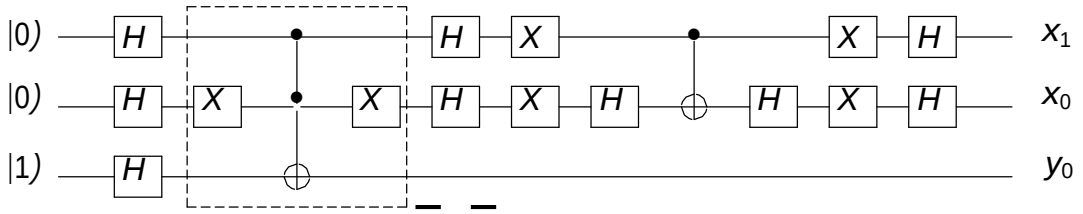


Figura A.6: Exemplo do algoritmo de Grover - Passo 5

Iteração de Grover, significando as partes do algoritmo de busca de Grover que devem ser repetidas para encontrar a solução em $O(\sqrt{N})$ iterações. Nessa instância do algoritmo, é necessário executar a **Iteração de Grover** apenas uma vez, de acordo com a Eq. 2.18.

As duas portas Hadamard são combinadas com a porta Pauli I por meio do produto tensorial e aplicada sobre o vetor de estado $|\psi_4\rangle$ para gerar o vetor de estado $|\psi_5\rangle$, como segue:

$$|\psi_5\rangle = (H \otimes H \otimes \boldsymbol{I})|\psi_4\rangle =$$

$$\frac{1}{4}\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}$$

resultando:

$$|\psi_5\rangle = \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}^t$$

A.6 Passo 6

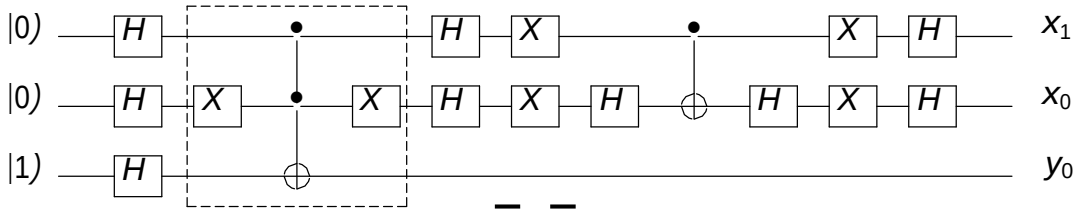


Figura A.7: Exemplo do algoritmo de Grover - Passo 6

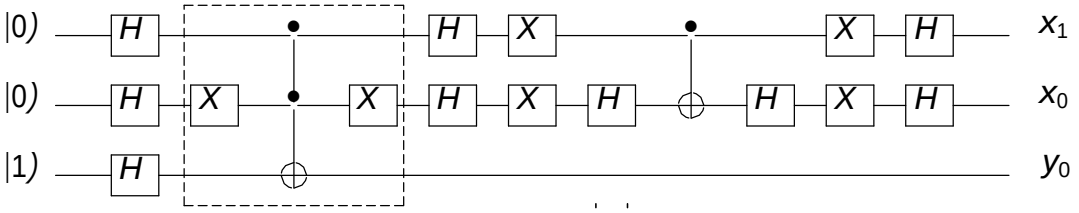
Nesse passo essas duas portas Pauli X são combinadas com a porta Pauli I por meio do produto tensorial e multiplicadas pelo vetor de estado $|\psi_5\rangle$ para gerar o vetor de estado $|\psi_6\rangle$.

$$|\psi_6\rangle = (X \otimes X \otimes \boldsymbol{I})|\psi_5\rangle = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}$$

resulting:

$$|\psi_6\rangle = \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & \frac{\sqrt{2}}{4} + 0\boldsymbol{i} & -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}^t$$

A.7 Passo 7



Passo 7

Figura A.8: Exemplo do algoritmo de Grover - Passo 7

Nesse passo, a porta Hadamard única é combinada com duas portas Pauli I por meio do produto tensorial e o resultado é multiplicado pelo vetor de estado $|\psi_6\rangle$ para gerar o vetor de estado $|\psi_7\rangle$.

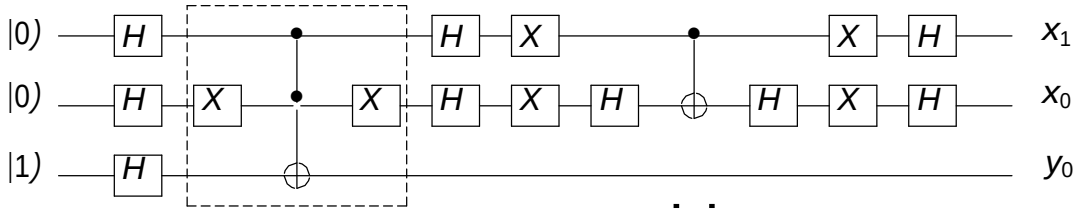
$$|\psi_7\rangle = (\boldsymbol{I} \otimes H \otimes \boldsymbol{I})|\psi_6\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}$$

resultando:

$$|\psi_7\rangle = \begin{bmatrix} \frac{\sqrt{1}}{2} + 0\boldsymbol{i} & -\frac{\sqrt{1}}{2} + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} & -\frac{\sqrt{1}}{2} + 0\boldsymbol{i} & \frac{\sqrt{1}}{2} + 0\boldsymbol{i} \end{bmatrix}^t$$

A.8 Passo 8

Nesse passo, a porta Feynman combinada com a porta Pauli X pelo produto tensorial é aplicada sobre o vetor de estado $|\psi_7\rangle$ para gerar o vetor de estado $|\psi_8\rangle$ como segue.



Passo 8

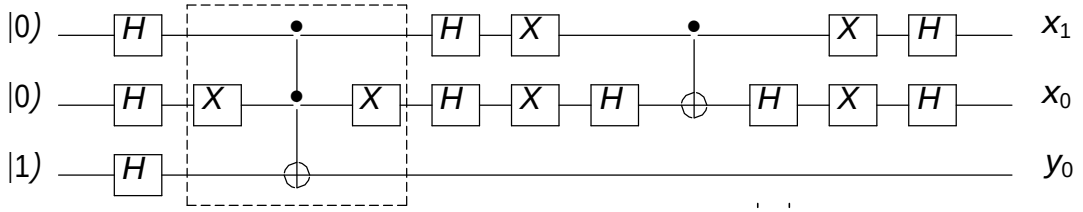
Figura A.9: Exemplo do algoritmo de Grover - Passo 8

$$|\psi_8\rangle = (CNOT \otimes \boldsymbol{I})|\psi_7\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{1}}{2\sqrt{}} + 0\boldsymbol{i} \\ -\frac{1}{2} + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \\ -\frac{\sqrt{1}}{\sqrt{2}} + 0\boldsymbol{i} \\ \frac{1}{2} + 0\boldsymbol{i} \end{bmatrix}$$

resultado:

$$|\psi_8\rangle = \begin{bmatrix} \frac{\sqrt{1}}{2} + 0\boldsymbol{i} & -\frac{\sqrt{1}}{2} + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} & -\frac{\sqrt{1}}{2} + 0\boldsymbol{i} & \frac{\sqrt{1}}{2} + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} & 0 + 0\boldsymbol{i} \end{bmatrix}^t$$

A.9 Passo 9



Passo 9

Figura A.10: Exemplo do algoritmo de Grover - Passo 9

Novamente nesse passo, a porta Hadamard única é combinada com duas portas Pauli I por meio do produto tensorial e aplicada sobre o vetor de estado $|\psi_8\rangle$ para gerar o vetor de estado $|\psi_9\rangle$, como segue.

$$|\psi_9\rangle = (\boldsymbol{I} \otimes H \otimes \boldsymbol{I})|\psi_8\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{1}}{2\sqrt{}} + 0\boldsymbol{i} \\ -\frac{1}{2} + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \\ -\frac{\sqrt{1}}{\sqrt{2}} + 0\boldsymbol{i} \\ \frac{1}{2} + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \\ 0 + 0\boldsymbol{i} \end{bmatrix}$$

resultando:

$$|\psi_9\rangle = \begin{bmatrix} \frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & \frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & \frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & \frac{\sqrt{2}}{4}+0i \end{bmatrix}^t$$

Algo importante a ser destacado nesse ponto é que a sequência de passos 7, 8 e 9 correspondem à aplicação da operação mudança de fase controlada (ver figura A.11). Assim como foi feito com a porta de mudança de fase controlada, frequentemente é possível substituir alguma porta quântica por um conjunto de outras portas quântica que geram o mesmo resultado.

Figura A.11: Porta controlled phase shift implementada com portas Hadamard e CNOT.

A.10 Passo 10

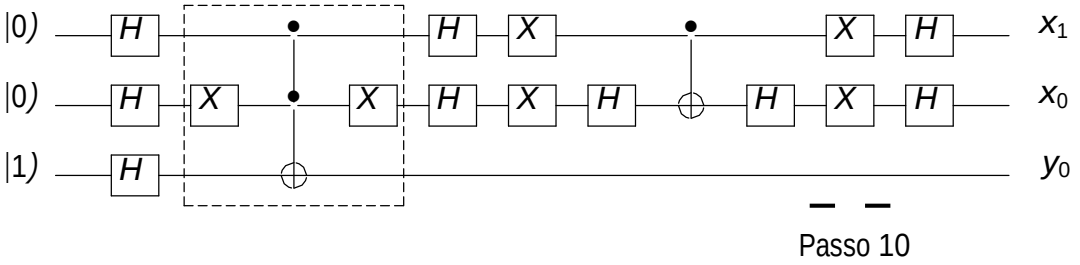


Figura A.12: Exemplo do algoritmo de Grover - Passo 10

Nesse passo, as duas portas Pauli X são combinadas com a porta Pauli I por meio da operação produto tensorial e então multiplicadas pelo vetor de estado $|\psi_9\rangle$ para gerar o vetor de estado $|\psi_{10}\rangle$.

$$|\psi_{10}\rangle = (X \otimes X \otimes I)|\psi_9\rangle = \begin{bmatrix} 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4}+0i \\ -\frac{\sqrt{2}}{4}+0i \\ \frac{\sqrt{2}}{4}+0i \\ -\frac{\sqrt{2}}{4}+0i \\ -\frac{\sqrt{2}}{4}+0i \\ \frac{\sqrt{2}}{4}+0i \\ -\frac{\sqrt{2}}{4}+0i \\ \frac{\sqrt{2}}{4}+0i \end{bmatrix}$$

resultando:

$$|\psi_{10}\rangle = \begin{bmatrix} \frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & \frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & \frac{\sqrt{2}}{4}+0i & -\frac{\sqrt{2}}{4}+0i & \frac{\sqrt{2}}{4}+0i \end{bmatrix}^t$$

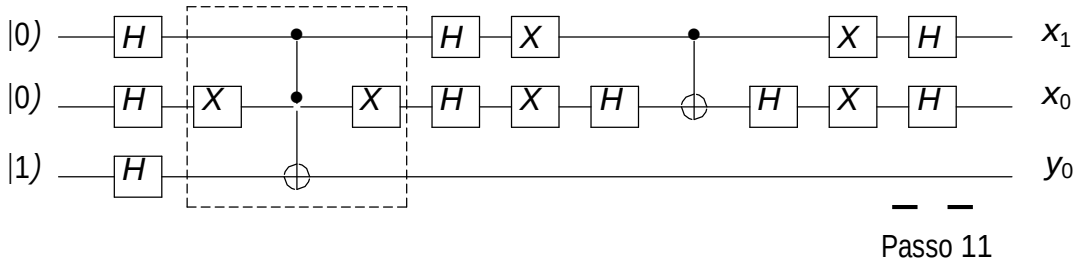


Figura A.13: Exemplo do algoritmo de Grover - Passo 11

A.11 Passo 11

Este é o último passo desse exemplo do algoritmo de Grover, onde duas portas Hadamard são combinadas com uma porta Pauli I por meio do produto tensorial e multiplicadas pelo vetor de estado $|\psi_{10}\rangle$ para gerar o vetor de estado final $|\psi_{11}\rangle$. Este passo também é o último da chamada Iteração de Grover, a qual deve ser aplicada somente uma vez neste caso. Detalhes dessa operação são mostrados a seguir.

$$|\psi_{11}\rangle = (H \otimes H \otimes \boldsymbol{I})|\psi_{10}\rangle =$$

$$\frac{1}{2}\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ -\frac{\sqrt{2}}{4} + 0\boldsymbol{i} \\ \frac{\sqrt{2}}{4} + 0\boldsymbol{i} \end{bmatrix}$$

resultando:

$$|\psi_{11}\rangle = \begin{bmatrix} 0+0\boldsymbol{i} & 0+0\boldsymbol{i} & 0+0\boldsymbol{i} & 0+0\boldsymbol{i} & -\frac{1}{\sqrt{2}}+0\boldsymbol{i} & \frac{1}{\sqrt{2}}+0\boldsymbol{i} & 0+0\boldsymbol{i} & 0+0\boldsymbol{i} \end{bmatrix}^t$$

Com esse passo a passo apresentado, espera-se ter ficado claro que a evolução de um algoritmo quântico é – para um entendimento básico – o resultado de uma série de multiplicações de matrizes (operadores, gerados por meio do produto tensorial entre as matrizes dos operadores das portas individuais) por vetor (o vetor de estado corrente do sistema).

Pode-se perceber que após a aplicação do sub-circuito de difusão as amplitudes α_{100} e α_{101} aumentaram, enquanto as demais decresceram para 0. Isso significa que se uma medição é feita no sistema nesse ponto, o sistema quântico irá colapsa para o estado $|100\rangle$ ou para o estado $|101\rangle$ com 50% de probabilidade cada, o que é 100% de probabilidade dos qubits que codificam x serem o valor buscado 10.

Desta forma, essa versão do algoritmo de busca de Grover pode ser modificada para implementar buscas mais interessantes, bastando para isso trocar o sub-circuito oráculo por outro que implementa a nova consulta, o qual deve ser reversível e pode usar outros qubits auxiliares. Também, dependendo do número de qubits na base de dados, e do número de valores na base que satisfazem a busca, o número de repetições da iteração de Grover pode variar.

APÊNDICE B LISTA DE PUBLICAÇÕES

- CONCEICAO, C. ; REIS, R. . “Quantum Algorithms Simulation on FPGAs.”, VIII Southern Programmable Logic Conference, 2012, Bento Gonçalves. Designer Forum, 2012. SPL - Designer Forum - 2012
- FLACH, G. ; CONCEICAO, C. ; JOHANN, M. ; REIS, R. . “Revisiting Atari 2600 on an FPGA”, 2012. VIII Southern Programmable Logic Conference, 2012, Bento Gonçalves. SPL, 2012
- MIRANDA, M. ; CONCEICAO, C. ; REIS, R. . Demonstrating Grover’s Quantum Search Algorithm. South Symposium on Microeletronics, 2012, São Miguel das Missões. SIM, 2012.
- CONCEICAO, C. ; REIS, R. . “Geração Automática de Processadores Dedicados à Simulação de Algoritmos Quânticos em FPGA”. IBERCHIP Workshop, 2013, Cusco, Peru. IBERCHIP XIX Workshop, 2013.
- CONCEICAO, C. ; REIS, R. . “Automatic Generation of Co-Processor for Simulation of Quantum Algorithms on FPGA”. Simpósio Sul de Microeletrônica, 2013, Porto Alegre, RS. SIM, 2013
- MIRANDA, M. ; CONCEICAO, C. ; REIS, R. . “Demonstrando o Algoritmo de Busca Quântica de Grover.” IBERCHIP Workshop, 2013, Cusco, Peru. IBERCHIP XIX Workshop, 2013.
- CONCEICAO, C. ; REIS, R., “A Co-Processor Architecture for Simulation of Quantum Algorithms on FPGA” (EM REVISÃO), Latin American Symposium on Circuits and Systems, 2014, Santiago, Chile. LASCAS 2014