Descrição em VHDL de Algoritmos Quânticos

Pedro Henrique G. Marchi **Renata H. S. Reiser** **Maurício L. Pilla**
CDTEC – Universidade Federal de Pelotas (UFPEL) – Pelotas – RS – Brasil
{phgmarchi,reiser,pilla}@inf.ufpel.edu.br

RESUMO

Relevantes aplicações do VHDL (*VHSIC Hardware Description Language)* [4] estão no campo da programação de dispositivos lógicos como FPGA e das aplicações específicas de circuitos integrados ASICs (*Application Specific Integrated Circuits*).

Ao contrário da programação essencialmente baseada em computações sequenciais, no VHDL a programação é concorrente. Ao evitar a limitação gerada pelo acesso à memória sequencial, torna-se uma linguagem atrativa para execução de algoritmos paralelos. Pela geração do código VHDL e aplicação de FPGAs, tem-se uma opção para gerar um aumento significativo na velocidade e no número de qubits quando da simulação quântica, fornecendo uma alternativa de análise e desenvolvimento de algoritmos quânticos.

Assim, a motivação para uso de VHDL neste trabalho é buscar uma descrição dos circuitos quânticos a partir dos padrões estabelecidos para os circuitos clássicos. A principal contribuição na atual etapa de trabalho, é viabilizar a simulação do paralelismo quântico associada às portas unitárias que compõem um conjunto universal para manipulação de circuitos quânticos. Em particular, uma biblioteca de métodos para especificação de coeficientes, os quais são definidos por números complexos normalizados, está em desenvolvimento incluindo operações aritméticas e considerando a representação polar, baseada no módulo e ângulo.

O principal objetivo é o desenvolvimento de uma extensão da biblioteca qEx-VHDL, pela construção de um módulo qEx-VHDL-FP, contendo funções específicas para manipulação de qubits e portas lógicas unidimensionais [1], [2].

Se um bit de informação pode armazenar um dos dois valores 0 ou 1, um registrador clássico de n bits pode armazenar um conjunto de 2n elementos ({0, 1, ..., 2n-1}) por vez. No computador quântico, a unidade básica de informação é o qubit, interpretado por um vetor bidimensional do espaço de Hilbert, o qual é definido como um espaço vetorial complexo munido do produto interno [Nielsen e Chuang 2000]. Além de armazenar 0 ou 1, o qubit também armazena a superposição de ambos 0 e 1. Assim, a representação de um qubit na representação de Dirac é dada pela expressão $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, onde α e β são as amplitudes dos estados básicos, números complexos normalizados ($|\alpha|^2 + |\beta|^2 = 1$). Mais especificamente, $\alpha=(\rho_{\alpha},\theta_{\alpha})$ e $\beta=(\rho_{\beta},\theta_{\beta})$ são a representação polar definida pelo módulo e ângulo dos respectivos coeficientes do qubit $|\Psi\rangle$.

A transformação dos possíveis estados de registradores quânticos pode ser modelada por operadores unitários, referidos como portas quânticas. Tais dispositivos quânticos podem executar uma operação unitária fixada, sobre qubits selecionados, em período determinado no tempo. Um operador unitário U é interpretado como uma matriz unitária U. Operadores quânticos de um qubit são denominados portas elementares. Outras operações não unitárias são operadores de controle e medida (neste caso, interpretadas por projeções do vetor correspondente ao qubit sobre um par de subespaços ortogonais) [3].

DOI: 10.5540/03.2015.003.01.0090

Para a realização dos cálculos, consideramos o número complexo a na notação polar, $a = |\rho|(\cos\theta + i\, sen\, \theta)$, sendo ρ indicando o módulo e θ o ângulo correspondente. Essa notação reduz significativamente o número de cálculos em algumas operações se comparada com a notação algébrica. Nas operações de multiplicação e exponenciação, tem-se:

$$ab = |\rho_a||\rho_b|(\cos\,(\theta_a + \theta_b) + i\, sen\,(\theta_a + \theta_b));$$

$$a^n = |\rho_a|^n(\cos\,(n\theta_a) + i\, sen\,(n\theta_a)).$$

Nesta representação são realizados cálculos apenas sobre os valores do módulo e do ângulo de cada um dos complexos. Estas vantagens que se aplicam na etapa de modelagem dos dados também são estendidas na especificação em VHDL, pois é possível fazer uma descrição simplificada, e consequentemente, a geração de hardware mais simples para executar as operações (aritméticas). Na Figura 1, análoga às demais operações, tem-se os cálculos da multiplicação, que foram otimizados em apenas um operador e a implementação dos valores utiliza um sistema de Ponto Flutuante, com precisão de 32 bits, 1 bit de sinal, expoentes de 8 bits e mantissa de 23 bits.

```
1   library ieee;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4   use IEEE.std_logic_unsigned.all;
5
6
7
8   entity multiplica is
9   port (
10              modulo1, cos1, modulo2, cos2: in std_logic_vector (31 downto 0);
11              mods, coss: out std_logic_vector (31 downto 0)
12  );
13  end multiplica;
14  architecture arc_multiplica of multiplica is
15  begin
16      process (modulo1, cos1, modulo2, cos2)
17              variable modtemps, costemps: std_logic;
18              variable modtempexp, costempexp: std_logic_vector (7 downto 0);
19              variable costempman, sentempman: std_logic_vector (22 downto 0);
20              variable mancos1, mancos2: std_logic_vector (22 downto 0);
21              variable modtempman: std_logic_vector (45 downto 0);
22              variable expcos1, expcos2: std_logic_vector (7 downto 0);
23              variable temp: std_logic_vector (6 downto 0);
24              variable temp2: integer;
25              begin
26              modtemps:= (modulo1 (31)) xor (modulo2 (31));
27              if (modulo1(30) = '0' and modulo2(30) ='1') then
28                  if (modulo1(29 downto 23) > modulo2(29 downto 23))then
29                      modtempexp:= '0' & (modulo1(29 downto 23) - modulo2(29 downto 23));
30                  end if;
31                  if (modulo1(29 downto 23) < modulo2(29 downto 23))then
32                      modtempexp:= '1' & (modulo2(29 downto 23) - modulo1(29 downto 23));
33                  end if;
34                  if (modulo1(29 downto 23) = modulo2(29 downto 23))then
35                      modtempexp:="00000000";
36                  end if;
37              elsif (modulo1(30) = '1' and modulo2(30) ='0') then
38                  if (modulo1(29 downto 23) > modulo2(29 downto 23))then
39                      modtempexp:= '1' & (modulo1(29 downto 23) - modulo2(29 downto 23));
40                  end if;
41                  if (modulo1(29 downto 23) < modulo2(29 downto 23))then
42                      modtempexp:= '0' & (modulo2(29 downto 23) - modulo1(29 downto 23));
43                  end if;
44                  if (modulo1(29 downto 23) = modulo2(29 downto 23))then
45                      modtempexp:="00000000";
46                  end if;
47              else
48              modtempexp := modulo1(30) & (modulo1(29 downto 23) + modulo2(29 downto 23));
49              end if;
50              modtempman:= (modulo1 (22 downto 0)) * (modulo2 (22 downto 0));
```

Figura 1: Biblioteca qEx-VHDL-FP – Multiplicações de Complexos

Referências

[1] Monteiro, E. , Jaccottet, D. , Reiser, R., Costa, E., Pilla, M. “Simulação Quântica em VHDL: Um Estudo de Caso Baseado no Algoritmo de Deutsch” In: XXXV CLEI, 2009, Pelotas.

[2] Monteiro, E. , Jaccottet, D. , Reiser, R., Costa, E., Pilla, M. “Simulação Quântica em VHDL: Um Estudo de Caso Baseado no Algoritmo Quântico de Grover” In: WSCAD-SSC 2009, SP.

[3] Nielsen, M. A. and Chuang, I. L. (2000). “Quantum Computation and Quantum Information”. Cambridge University Press.

[4] Pedroni, V. A. (2004). “Circuit Design with VHDL”. MIT Press.

