SIMULAÇÃO PARALELA E VERIFICAÇÃO DE CIRCUITOS DE AUTÔMATOS CELULARES COM PONTOS QUÂNTICOS

LUIZ HENRIQUE BORGES SARDINHA

SIMULAÇÃO PARALELA E VERIFICAÇÃO DE CIRCUITOS DE AUTÔMATOS CELULARES COM PONTOS QUÂNTICOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Omar Paranaíba Vilela Neto

Belo Horizonte

Dezembro de 2017

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Sardinha, Luiz Henrique Borges

S244s Simulação Paralela e Verificação de Circuitos de Autômatos Celulares com Pontos Quânticos / Luiz Henrique Borges Sardinha. — Belo Horizonte, 2017
xvi, 59 p.: il.; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais – Departamento de Ciência da Computação

Orientador: Omar Paranaíba Vilela Neto

1. Computação – Teses. 2. Autômatos Celulares com Pontos Quânticos – QCA. 3. Paralelização.
4. Nanocomputação. I. Orientador. II. Título.

CDU 519.6*17 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Simulação paralela e verificação de circuitos de autômatos celulares com pontos quânticos

LUIZ HENRIQUE BORGES SARDINHA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. OMAR PARANAÍBA VILELA NETO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. JOSÉ AUGUSTO MIRANDA NACIF
Departamento de Informática - UFV

PROF. LUIZ FILIPE MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

PROF. SÉRGIO VALE AGUIAR CAMPOS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 14 de Dezembro de 2017.

Resumo

No campo de pesquisa pós-sílicio, o paradigma Autômatos Celulares com Pontos Quânticos (*Quantum-dot Cellular Automata* - QCA) surgiu como candidato para substituir os circuitos tradicionais CMOS. O QCA se baseia na transmissão de informações entre células tirando proveito de interações Coulombianas. Neste trabalho apresentou-se técnicas que visam expandir e acelerar as possibilidades de validação de um circuito originalmente descrito usando a ferramenta QCADesigner. O circuito é dividido em várias partições menores independentes que podem ser verificadas por falhas comuns como malposição e falta de sincronia. Essas mesmas fatias também podem ser arranjadas em fatias simuladas em paralelo. O processo de simulação apresentado é aproximadamente cinco vezes mais veloz que simulações comuns e permite a criação de um modelo no qual proposições lógicas podem ser avaliadas. A utilização das citadas técnicas visa auxiliar a adoção deste paradigma ao reduzir drasticamente o tempo necessário para validação completa de um circuito.

Palavras-chave: Autômatos Celulares com Pontos Quânticos, QCA, Verificação, Simulação, Paralelização, Nanocomputação.

Abstract

In beyond silicon research, Quantum-dot Cellular Automata (QCA) emerged as a candidate for replacing the traditional CMOS logic circuits. QCA is a paradigm based on the exchange of information between cells that take advantage of Coulomb's law. In this work, we present techniques that expand and accelerate the validation of designs created using QCADesigner simulator. The circuit is split into small independent regions which could be verified for common mistakes such as cell misplacements and synchronization issues. Those small circuits chops can also be arranged for parallel simulation allowing speed ups of approximately five times faster. The output from such devices can be used to build a model where logical prepositions could be evaluated. The entire process can vastly reduce the time for full circuitry validation and help adoption of the paradigm.

Keywords: Quantum-dot Cellular Automata, QCA, Verification, Simulation, Paralelization, Nanocomputing.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Motivação

O contínuo progresso previsto na eficiência computacional de processadores tradicionais, previsto por Moore et al. [1965] encontrou barreiras devido aos limites físicos. Motivados por estas limitações, pesquisadores buscam continuamente alternativas em escala nanométricas (Shalf & Leland [2015]). O paradigma dos Autômatos Celulares com Pontos Quânticos (*Quantum-dot Cellular Automata*, ou *QCA*) (Lent et al. [1993]) se apresenta como uma das promissoras alternativas devido à sua teórica alta taxa de sincronização, na casa dos THz, e gasto extremamente baixo de energia. O funcionamento desta tecnologia se baseia no acoplamento de campo de células organizadas de forma a permitir a correta execução lógica.

As técnicas para criação de circuitos tradicionais podem ser adaptadas para utilização de circuitos QCA (Henderson et al. [2004]; Ottavi et al. [2006]), entretanto não conseguem expressar toda sua capacidade. O processo de criação de circuitos QCA em ferramentas especializadas de criação, tais como o QCADesigner (Walus et al. [2004]), atualmente requerem o desenho e criação de baterias de teste de forma tortuosa e situacional.

Existe necessidade real da criação de técnicas que possam agilizar o processo de validação de circuitos QCA. Os motores de simulação relevantes, como o do vetor de coerência do QCADesigner, não fazem uso de todos os recursos computacionais disponíveis. As técnicas de verificação já existentes (Henderson et al. [2004]) pecam ao exigir a descrição do funcionamento de circuitos QCA em modelos criados para abranger o comportamento de circuitos tradicionais.

1.2 Objetivos

Este trabalho tem como objetivo o desenvolvimento de técnicas que acelerem o processo de criação e validação de um circuito QCA descrito célula-a-célula. As técnicas propostas devem permitir paralelização da execução do tortuoso processo de simulação, juntamente com a criação automática de modelos que permitam a rápida identificação de erros.

1.3 Contribuições

Este trabalho apresenta e descreve 4 (quatro) diferentes técnicas inovadoras:

1. **Particionamento de circuitos**: Baseado nas características físicas e comportamento dos motores de simulação, esse processo visa a identificação de regiões independentes de um determinado circuito.

2. **Verificação de partição**: Permite a rápida detecção de erros simples de sincronização e inversão de sinal que podem comprometer o funcionamento de uma partição e do circuito como um todo.

3. **Simulação independente de partições**: Possibilita a realização da simulação de regiões independentes do circuito de forma paralela. Os resultados do circuito como um todo podem ser obtidos ao se combinar o resultado das pequenas fatias.

4. **Verificação lógica de circuitos**: Constrói-se modelos lógicos do circuito aproveitando as tabelas verdade obtidas pela simulação das partições. Assim, regras que descrevem o comportamento esperado do circuito podem ser propostas e comprovadas.

Por fim, a fim de atestar a funcionalidade, coesão e ganho de performance das técnicas de etapas anteriores, apresenta-se resultados de diversos experimentos, incluindo situações que se aproximam com condições reais, com circuitos largamente utilizados na literatura.

1.4 Organização da Dissertação

Esta dissertação está dividida em um total de 6 (seis) capítulos, incluindo este de Introdução, descritos a seguir:

O capítulo 2 apresenta o paradigma dos Autômatos Celulares com Pontos Quânticos (QCA), explicando suas funcionalidades e revisando os principais trabalhos relacionados.

O capítulo 3 apresenta a base teórica e ferramental da verificação de modelos.

O capítulo 4 apresenta detalhadamente a metodologia de cada uma das etapas necessárias para o processo de validação eficaz de circuitos QCA.

O capítulo 5 apresenta os resultados que corroboram o funcionamento do processo completo de simulação paralela e verificação interna.

O capítulo 6 conclui este trabalho, sugerindo possíveis aprimoramentos e trabalhos futuros.

Capítulo 2

QCA

Este capítulo apresenta o paradigma de Autômatos Celulares com Pontos Quânticos (***Q****uantum-dot* ***C****ellular* ***A****utomata* - QCA), descrevendo como as células deste modelo funcionam e interagem entre si para a formação de circuitos. Posteriormente, os dispositivos básicos serão explicados em detalhes. Por fim, serão apresentadas as principais técnicas e ferramentas de simulação de circuitos.

2.1 A célula QCA

O paradigma QCA se baseia essencialmente na transmissão de informação entre células nanométricas pela alternância de seus estados de polarização (Lieberman et al. [2013]). Essas células têm pontos quânticos em sua estrutura definidos como uma região fixa onde cargas elétricas podem estar localizadas. Uma célula convencional é quadrada, com quatro pontos quânticos posicionados em suas extremidades e com dois elétrons livres e móveis — representados por pontos escuros — que podem tunelar apenas entre os pontos quânticos de seu interior. Assume-se que o movimento destas cargas para o exterior da célula é proibido graças à grande barreira potencial existente.

Devido à repulsão Coulombiana, o par de elétrons ficará propenso a se posicionar em diagonais opostas, como demonstrado na figura 2.1. Uma célula tem dois estados equivalentes de energia que podem ser denominados *polarizações* P = -1 e P = +1. O arranjo das cargas entre células pode codificar informação binária: P = -1 pode ser equivalente ao binário 0 e P = +1 equivalente ao binário 1, por exemplo.

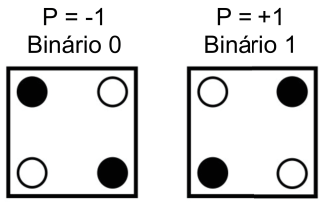

Figura 2.1. Diagrama de uma célula QCA tradicional composta de quatro pontos quânticos e dois elétrons móveis e livres em seus dois possíveis estados de *polarização*

2.2 Dispositivos Básicos

Se em determinado momento houver influência elétrica sob uma célula, elétrons irão se posicionar de forma a alcançar um estado estável de energia. Influências podem também vir de células próximas por conta da interação Coulombiana. Se duas células forem posicionadas lado a lado com uma delas polarizada, a outra alcançará estado estável de polarização, tendo a mesma distribuição espacial de elétrons que a primeira. A figura 2.2 mostra que essa propriedade pode ser aplicada de forma a permitir a propagação de informação em uma cadeia alinhada de células, construindo um fio.

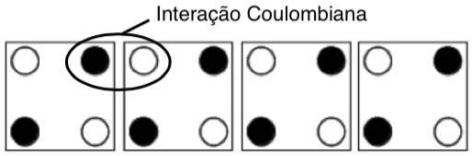

Figura 2.2. Um fio de células QCA.

Ao posicionar células diagonalmente umas das outras, há alta propensão a possuírem polarizações invertidas. A figura 2.3 mostra tal dispositivo e destaca as regiões de interação entre elétrons de diferentes células.

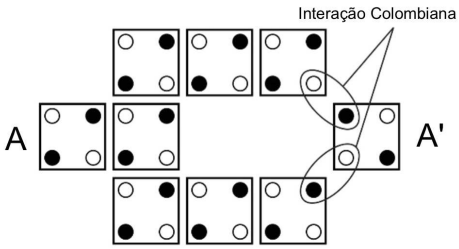

Figura 2.3. Inversor usando células QCA

A estrutura lógica básica de QCA é a *Porta da Maioria* de três entradas, apresentada na figura 2.4. Com este posicionamento, a célula central se polarizará tal como a maioria das entradas, pois essa é a configuração na qual a repulsão entre todos os elétrons dessas células é menor. O resultado será replicado, para a célula de saída completando seu funcionamento. Essa estrutura tem capacidade de representar as portas lógicas E e OU para duas entradas se a restante estiver fixada na polarização correspondente a 0 e 1, respectivamente.

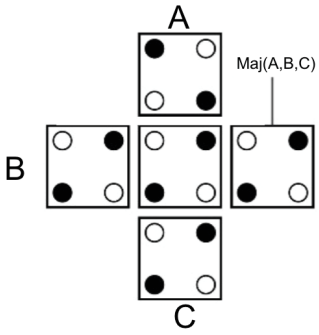

Figura 2.4. Porta da Maioria (*Majority Gate*) usando células em QCA

Por possuir capacidade de representar a porta da maioria e o inversor, bem como a capacidade de transmitir informação, é dito que o paradigma QCA é completo e pode representar qualquer lógica. Entretanto, é necessário atentar-se às características peculiares de sincronização. Se as influências ocorrerem em momentos diferentes do tempo, é possível que suas polarizações não sigam as características desejadas. Assim, é necessária a aplicação de técnicas de sincronização.

2.3 Clock

Em circuitos tradicionais, o clock é um sinal alternante capaz de sincronizar informação. Em QCA, no entanto, o clock é um campo elétrico controlável que nivela as barreiras de tunelamento dentro de uma ou mais células (Lent & Tougaw [1997]) sendo capaz de controlar a transmissão de informação. Uma zona de clock pode ser definida como um conjunto de células no qual um único potencial as modula.

O clock de QCA, definido por Lent et al. [1993], possui quatro fases cíclicas:

1. ***Switch***: as barreiras de tunelamento, inicialmente baixas, progressivamente se elevam permitindo a polarização da célula de acordo com o estado de influências externas, tais como células de entrada;

2. ***Hold***: as barreiras são mantidas de forma elevada e a polarização das células permanece fixa, como calculadas na fase anterior. Isso permite que os resultados sejam usados como entrada para uma outra zona em *Switch*;

3. ***Release***: o campo elétrico é reduzido gradualmente a fim de remover completamente as barreiras de potencial;

4. ***Relax***: Os elétrons estão completamente livres em virtude da ausência de barreiras.

Em síntese, o esquema de zonas de clock permite que células de QCA realizem certa computação (*Switch*), tenham seus estados congelados (*Hold*), se despolarizem (*Release*) e permaneçam inócuas e prontas para um novo ciclo (*Relax*). A figura 2.5 ilustra a transmissão controlada de informação por um fio no decorrer do tempo pelas diferentes fases de clock.

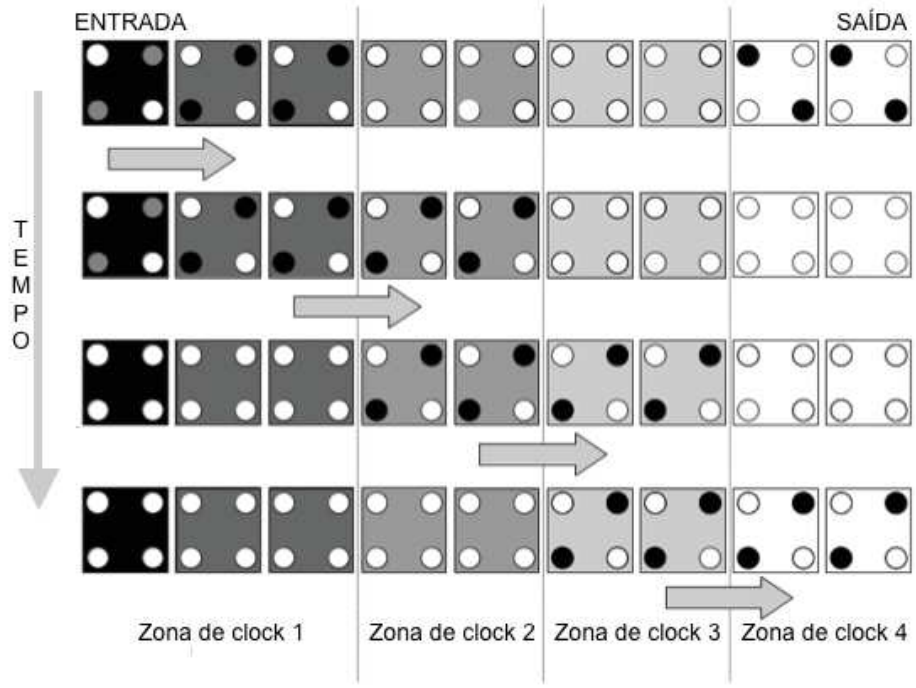

Figura 2.5. Funcionamento das zonas de clock

2.4 Simulação

QCA é considerado apenas um paradigma promissor, mas requer diversas condições para seu funcionamento físico correto. O tipo mais tradicional, de ponto metálico, requer temperaturas criogênicas para operar (Orlov et al. [1997]). Um outro tipo, conhecido como QCA Molecular, não exige condições térmicas tão extremas, mas ainda é de difícil fabricação e com desempenho inferior à circuitos CMOS tradicionais. Experimentos feitos por Orlov et al. [1997], Snider et al. [1999] e Li et al. [2003] comprovam o funcionamento em pequena escala dessa tecnologia que ainda permanece relevante devido a seu teórico baixo consumo de energia.

Dada a difícil implementação física, o uso de simulações em projetos de circuitos é imprescindível. Simuladores não precisam somente demonstrar resultados, mas também permitir a criação de circuitos de forma interativa.

Desenvolvido na Universidade de Notre Dame por Orlov et al. [1997], o AQUINAS foi o primeiro simulador de QCA. Ele tinha como objetivo calcular a polarização de pequenos circuitos utilizando modelos de mecânica quântica; prosperou até que se mostrou lento para circuitos um pouco maiores.

Almejando solucionar tal limitação, Niemier [2000] apresentou o Q-BERT (*Quantum Logic Based Engine Rules Tool*) que se destacou por permitir a simulação de circuitos maiores, mas pecava por sua interface gráfica complicada.

Para tentar resolver os problemas enfrentados anteriormente, Vilela Neto et al. [2007] apresentou um modelo ágil de simulador que faz uso de redes neurais artificiais.

O *QCADesigner*, desenvolvido por Walus et al. [2004], destaca-se por trazer de forma interativa e gráfica para o usuário todo o aparato para criação de circuitos QCA, dos mais simples aos mais complexos, como processadores. A figura 2.6 mostra sua interface gráfica. Em sua versão pública mais recente (2.0.3), permite o uso de dois métodos de simulação distintos:

- ***Bi-estável***: Simplifica o cálculo de polarização ao utilizar princípios de lógica digital. Considera a célula como unidade base.
- ***Vetor de Coerência***: Baseado em cálculos físicos complexos que levam em conta cada um dos pontos quânticos das células. Mais lento que o anterior, porém mais coeso.

Existe também um terceiro método, ainda não público, descrito por Karim & Walus [2014] que é um meio termo entre os dois anteriores, mas que alcança precisão suficiente.

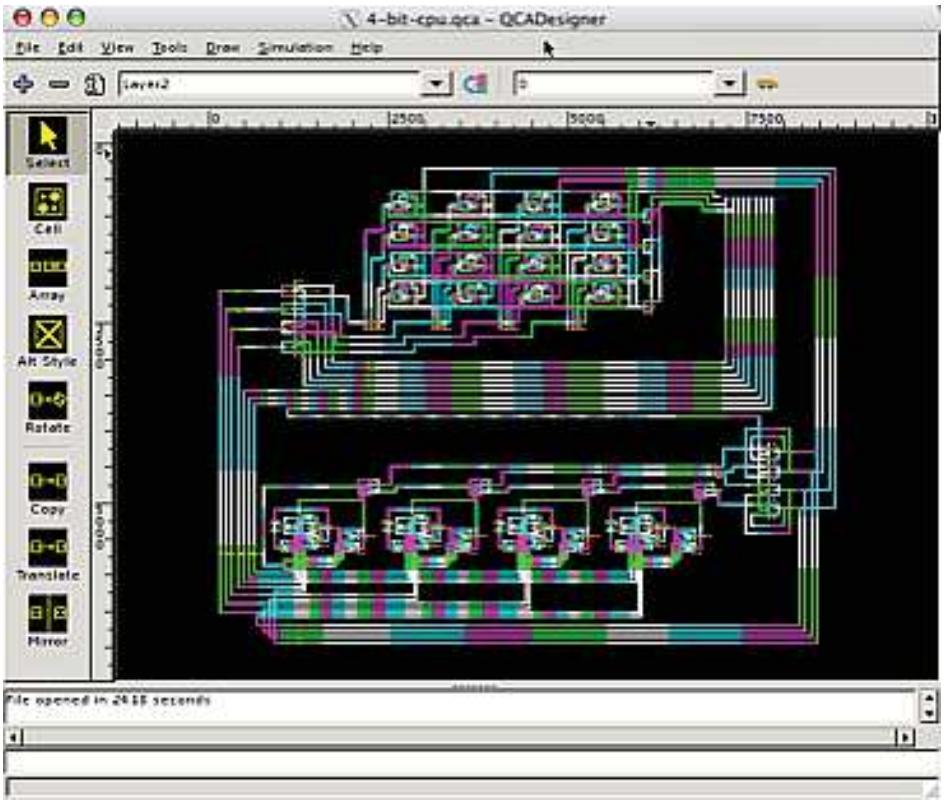

Figura 2.6. Interface gráfica do simulador QCADesigner demonstrando um processador de 4-bits

Neste trabalho, será apresentado uma forma mais ágil de simular grandes circuitos. A simulação de certos circuitos pode exigir horas ou dias e são situacionais, ou seja, levam em conta uma sequência finita de entradas e executam apenas por um determinado número de ciclos de clock. A técnica a ser descrita utilizará os métodos de cálculo já consagrados pelo QCADesigner de forma localizada, permitindo a paralelização de execução.

Capítulo 3

Verificação de Modelos

O termo verificação é utilizado como técnica ou procedimento capaz de comprovar propriedades de um sistema. Pode ser obtida com a utilização de observações, mas não gera tanta confiabilidade pois se baseia na falta de alcance de estados proibidos nas sequências predeterminadas. Quando feita formalmente, exige que o sistema seja bem modelado para que propriedades sejam comprovadas matematicamente.

Neste capítulo serão descritos os princípios básicos de verificação de modelo utilizados para modelar o funcionamento de circuitos QCA.

3.1 Lógica Temporal

Pnueli [1977] descreveu a lógica temporal como base para a prova de propriedades sobre sistemas reativos, ou seja, aqueles que interagem com o seu ambiente ao invés de simplesmente produzir um resultado final.

Normalmente, sistemas reativos podem ser capturados por estruturas de *Kripke* que são representadas por um conjunto de estados, transições e uma função que rotula cada estado pelo seu conjunto de propriedades verdadeiras. Formalmente é representada pela quádrupla $M = (S, S_0, R, \lambda)$, onde:

- $S = \{s_0, ..., s_n\}$: Conjunto de estados.
- S_0: Conjunto de estados iniciais, com $S_0 \subseteq S$.
- R: Conjunto de transições válidas entre estados de S.
- λ: Função capaz de rotular um estado s de acordo com as propriedades verdadeiras nele. $\lambda(s_i)$ deve ser único para qualquer valor de i.

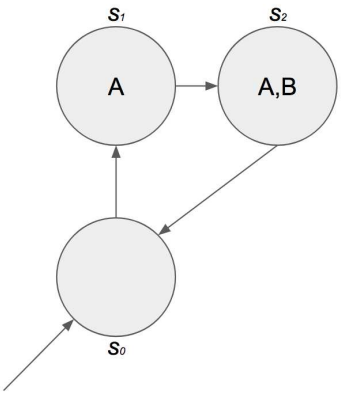

Figura 3.1. Exemplo de estrutura de *Kripke*

A figura 3.1 mostra um exemplo de estrutura de *Kripke* simples, definida por:

- $S = \{s_0, s_1, s_2\}$
- $S_0 = \{s_0\}$
- $R = \{(s_0, s_1), (s_1, s_2), (s_2, s_0)\}$
- $\lambda = \{(s_0, \{\}), (s_1, \{A\}), (s_2, \{A, B\})\}$, com A e B sendo propriedades distintas do sistema

A lógica convencional, por exemplo, não consegue exprimir sequências de acesso aos estados de um sistema reativo, assim, necessita-se da adição de operadores temporais no artefato formal que descreve propriedades. Devido à complexidade de expressão, duas diferentes técnicas de caracterização se destacaram: a lógica de tempo linear e a lógica de tempo ramificado.

3.1.1 Lógica de tempo linear - LTL

A lógica linear, denominada LTL (*Linear Temporal Logic*), consegue construir fórmulas que especifiquem propriedades de cada uma das possíveis trajetórias de sistemas especificados por estruturas de *Kripke*. Seu uso se baseia na discretização do tempo pelas transições, ou seja, um passo de tempo é contado a cada transição. Nela, os principais operadores são:

- Fp: A proposição p será válida em algum futuro;
- Gp: A proposição p é sempre válida no futuro;

- Xp: A proposição p é válida no próximo estado;
- pUq: A proposição p é válida até que q seja válida.

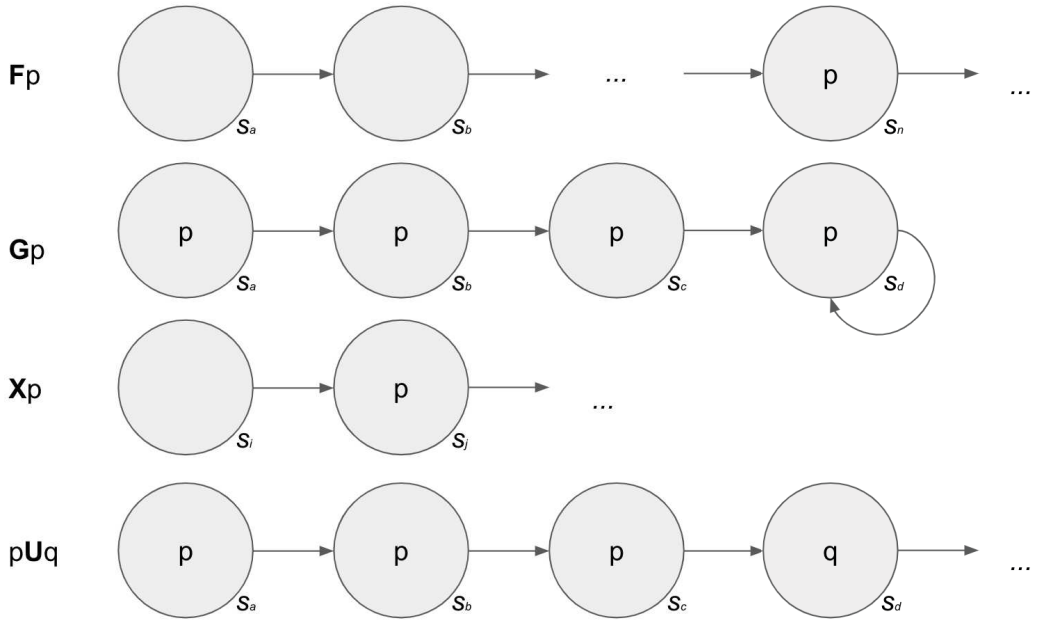

Figura 3.2. Operadores *LTL*

A figura 3.2 mostra exemplos de trajetórias em estruturas de *Kripke*, onde os operadores LTL são válidos. Proposições não precisam se basear em estados iniciais.

3.1.2 Lógica de tempo ramificado - CTL

A CTL (*Computing Tree Logic*) é baseada em estados ao invés das trajetórias de um sistema. Dessa forma, consegue especificar proposições que caracterizam todas as possibilidades alcançáveis.

Os principais operadores CTL são:

- **AF**p: A proposição p é eventualmente válida em todas as possíveis ramificações futuras;
- **EF**p: A proposição p é válida em pelo menos um estado futuro;
- **AG**p: A proposição p é válida continuamente, isto é, p é válido tanto atualmente quanto em todos os estados de todas as possíveis ramificações futuras;
- **EG**p: A proposição p é continuamente válida em pelo menos um possível estado futuro;

- **AX**p: A proposição p é válida em todos os possíveis estados imediatamente posteriores;
- **EX**p: A proposição p é válida em pelo menos um dos possíveis estados imediatamente posteriores;
- **A**p**U**q: A proposição p é válida em um ponto até que q seja válida em todas as ramificações futuras;
- **E**p**U**q: A proposição p é válida em um ponto até que q seja válida em pelo menos uma ramificação futura.

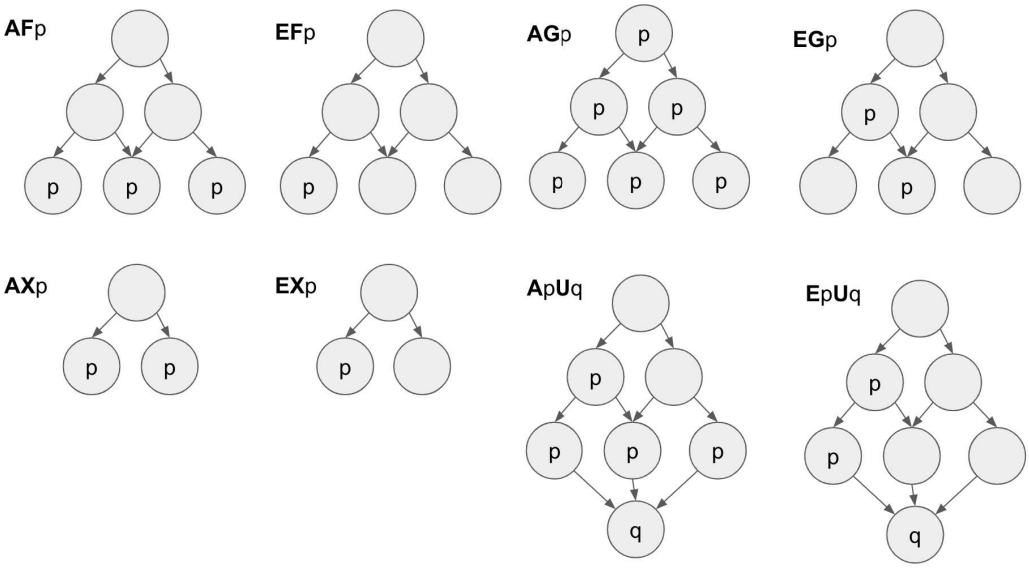

Figura 3.3. Operadores *CTL*

A figura 3.3 demonstra os operadores CTL e clarifica a diferença desse tipo de lógica com a LTL. Apesar de ser sintaticamente mais complexa, a expressividade de cada um dos tipos de lógica é diferente. Existem casos que só podem ser expressos em CTL e outros que só podem ser expressos em LTL.

3.2 Propriedades de um modelo

Propriedades são características desejadas descritas por proposições lógicas. Neste contexto, a LTL e CTL tem particularidades que fazem ambas serem igualmente importantes e necessárias. Emerson & Halpern [1986] tentaram definir um tipo novo de lógica temporal, denominada *CTL**, que abrange os operadores de ambas. Entretanto, o suporte a essa nova lógica em ferramentas é baixa, tornando seu uso desaconselhado.

Normalmente, quando propostas, propriedades seguem as seguintes finalidades:

- **Segurança**: Caracterizam estados e/ou sequências de estados proibidos;
- **Vivacidade**: Expressas de forma a garantir que o sistema é capaz de alcançar um estado desejado.

Propriedades de segurança podem ser expressas em ambas as linguagens, entretanto costuma-se utilizar CTL em situações onde um estado é sempre proibido e LTL quando o objetivo é detectar trajetórias proibidas. Propriedades de vivacidade são descritas com o uso de CTL, visto que o operador E (eventualmente) é exclusivo dessa.

3.3 *NuSMV*

3.3.1 A Linguagem *SMV*

Criada por McMillan [1993], o SMV (*Symbolic Model Verifier*) tem por objetivo facilitar a criação de modelos a partir da descrição de um sistema por uma máquina de estados finitos de forma bem similar às linguagens de descrição de hardware, como *Verilog* e *VHDL*. O seu entendimento é simples já que traz consigo conceitos consagrados como blocos (descritos pela palavra-chave *MODULE*), variáveis (*VAR*) que podem ser enumerativas e/ou numerais, estado inicial (*initial*) e transições discretas (*next*).

Abaixo um pequeno trecho de código SMV que descreve um semáforo de trânsito que troca seu estado a cada passo de tempo:

```
MODULE main
...
VAR COR: {VERDE, AMARELO, VERMELHO};
...
ASSIGN
initial(COR) := VERDE;
...
next(COR) := case
(COR = VERDE) : AMARELO;
(COR = AMARELO) : VERMELHO;
(COR = VERMELHO) : VERDE;
TRUE: COR;
esac;
...
```

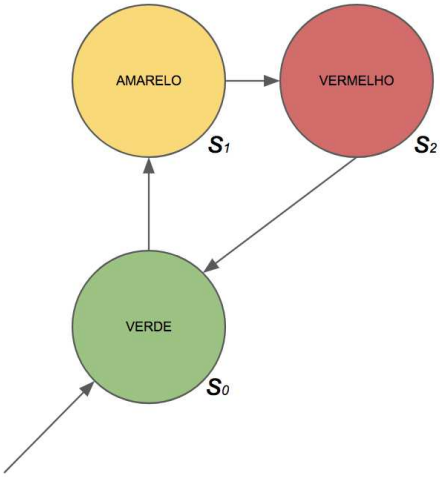

Figura 3.4. Estrutura de Kripke de um semáforo criada por um modelo SMV

A figura 3.4 mostra a estrutura de Kripke que pode ser internamente construída e utilizada para os mais diversos fins, tais como comprovações de propriedades.

3.3.2 A ferramenta *NuSMV* em si

NuSMV (Cimatti et al. [1999]), por sua vez, é uma ferramenta de código aberto que reimplementa e extende o ferramental SMV. Nela, propriedades podem ser expressas em CTL ou LTL.

Uma propriedade CTL pode ser adicionada usando a palavra-chave *SPEC* em um bloco (*MODULE*) descrito em SMV. Para o exemplo de um semáforo, podemos escrever a seguinte propriedade de segurança:

```
SPEC COR = VERDE -> AX COR = AMARELO
```

Especificamente, neste caso também podemos expressar essa mesma restrição em LTL, usando a palavra-chave *LTLSPEC*:

```
LTLSPEC COR = VERDE -> X COR = AMARELO
```

Em ambos os casos, o objetivo é garantir que após a cor verde, necessariamente virá a cor amarela.

A ferramenta mostra-se extremamente útil também em resultados negativos. Ela é capaz de demonstrar ao usuário um contra-exemplo da propriedade em detalhes, em outras palavras, ela mostra o passo-a-passo das transições e estados seguidos para que o estado proibido seja alcançado ou demonstrar que certo estado é inalcançável.

A seguinte expressão CTL, por exemplo, tenta expressar agora que a cor seguinte à verde deverá ser a vermelha:

SPEC COR = VERDE -> **AX** COR = VERMELHA

Se a aplicarmos no mesmo modelo do semáforo, a ferramenta dirá que ela é falsa e demonstrará para o usuário um contra-exemplo de forma textual:

-> State: 1.1 <-
COR = VERDE
-> State: 1.2 <-
COR = AMARELA

3.4 Verificação de modelos aplicada em QCA

O trabalho de Olson [2011] apresenta um técnica de tradução de circuitos QCA em linguagem de descrição de hardware. Por sua vez, seu objetivo final é a criação de hardware especializado para emular o funcionamento de circuitos baseando-se em resultados obtidos no QCADesigner. Não foi demonstrado melhorias por conta da grande relação de dependência entre células. O tempo de execução de testes chegou a ser mais lento que o próprio simulador base.

Com a tradução de circuitos em linguagem de hardware, é possível o uso de ferramentas especializadas de verificação formal, como o *JasperGold* (Cadence Design Systems [1999]) e o *VCFormal* (Synopsys [2015]). Entretanto, tais ferramentas são comerciais e fortemente especializadas em circuitos tradicionais de silício fazendo que o uso em QCA tenha que ser adaptado, causando prováveis erros e dificuldades de interpretação.

Dada a natureza lógica de QCA e a facilidade de expressão em SMV de sistemas lógicos, serão demonstradas técnicas capazes de modelar circuitos QCA de forma a permitir verificações lógicas e de sincronização. O uso da ferramenta NuSMV se justifica nesse trabalho pelo seu amplo uso pela literatura, juntamente com seu vasto suporte pela comunidade e custo nulo.

Capítulo 4

Metodologia

Neste capítulo serão descritos métodos que têm por objetivo acelerar o processo de verificação de circuitos criados usando o QCADesigner. Esta última é uma ferramenta que se mostra como confiável e é amplamente utilizada como comprovação para diversos tipos de projetos devido à sua fácil utilização e acesso.

O processo completo é dividido em quatro etapas correlatas que se iniciam no tratamento de um arquivo contendo a descrição de um circuito feito no QCADesigner. Cada fase é individualmente importante, inclusive sendo necessário para as posteriores. A Figura 4.1 mostra cada um desses passos e suas respectivas dependências. Em seguida, cada uma destas etapas serão descritas em detalhe.

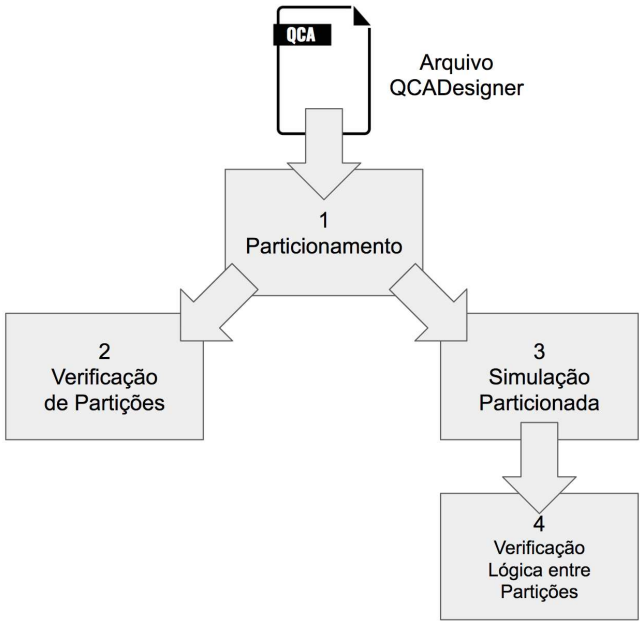

Figura 4.1. Passos do modelo de verificação

4.1 Particionamento

O processo de particionamento visa dividir o circuito em Vizinhanças. Estas, por sua vez, são definidas como blocos de indivíduos que podem se influenciar em uma simulação, ou seja, são constituídas por células adjacentes de uma mesma zona de clock e das diretamente adjacentes de uma zona de clock imediatamente anterior e posterior.

Agrupar assim se mostra aceitável pois mudanças de sinais só acontecerão na fase de *Switch* do clock, onde apenas células da zona de clock atual e anterior (que estarão em *Hold*) estarão polarizadas. Células distantes tem influência menosprezável (devido ao decaimento de ordem quadrática pela distância). Quaisquer outras células próximas estarão despolarizadas (em *Release* ou *Relax*). A zona de clock imediatamente posterior é adicionada apenas como referência para passos futuros.

4.1.1 Sintetização em forma de matriz

O arquivo do QCADesigner contendo informações de um design é de complicada organização. Operações, como busca, são custosas (normalmente exaustivas). Para resolver estes problemas inerentes, os dados ali presentes são sintetizados em uma matriz equivalente. Tal método é uma extensão do proposto e utilizado por Ribeiro et al. [2016].

No QCADesigner, células normalmente são posicionadas obedecendo uma grade (*grid*), ou seja, têm coordenadas discretrizadas e a superposição é proibida. A utilização de valores fracionados, mesmo que possível na ferramenta original, não será aceita por esta representação simplificada.

Uma posição na matriz corresponderá a uma fração unitária da grade original e terá um valor numérico de dois caracteres que caracterizará a célula que pode ali estar presente: o primeiro representará seu modo, apresentado na tabela 4.1, e o segundo sua zona de *clock* (de 0 a 3). Um valor negativo (e.g. *-1*) denota uma posição vazia (onde não há célula).

A matriz obtida na sintetização do circuito de um Multiplexador 4-para-1 (Silva et al. [2015]) está demonstrada na Figura 4.2. Este tipo de representação traz vantagens na identificação de células por posição que, por consequência, facilitam a identificação de adjacência. Possíveis problemas relativos a excessivo gasto de memória não se mostram preocupantes em decorrência de limitações de tamanho de circuitos suportados do próprio QCADesigner.

Valor	Descrição
0	Comum
1	Entrada
2	Saída
3	Fixo +1
4	Fixo -1
5	Vertical
6	Crossover

Tabela 4.1. Modos de célula suportados e seus valores numéricos correspondentes na representação por matriz

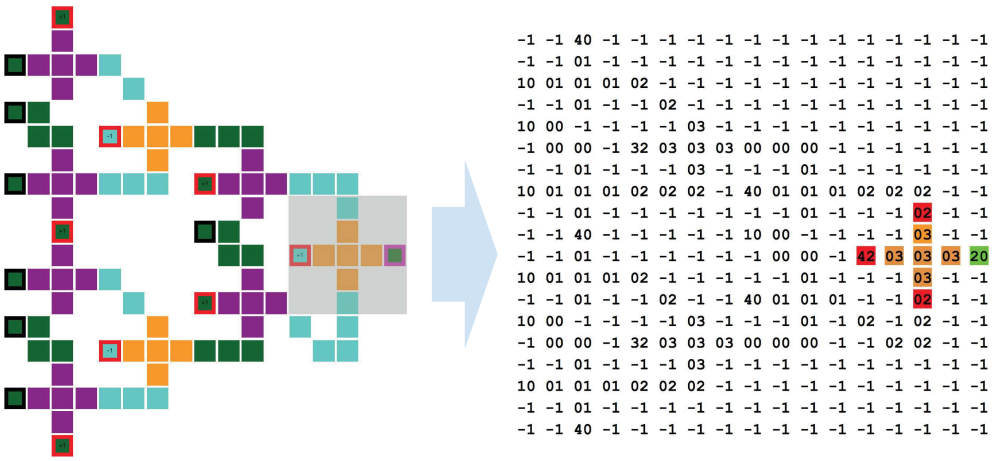

Figura 4.2. Circuito de Multiplexador 4-para-1 em QCA e sua respectiva representação simplificada em matriz

4.1.2 Vizinhanças

O processo de identificação de vizinhança se inicia em cada uma das células de saída e se repete para cada uma das células conectadas de zonas de clock anteriores. Se ao final determinada célula não fizer parte de nenhum agrupamento, pode-se dizer que esta célula é desnecessária.

Na própria figura 4.2 pode-se observar células destacadas que então representam uma das vizinhanças do circuito em questão. Na representação por matriz, a cor laranja identifica células da zona de clock em questão, enquanto as em vermelho são de uma zona de clock imediatamente anterior. Por fim, células em verde são de uma zona de clock imediatamente posterior.

4.2 Verificação de Partições

A verificação interna de partições tem como principal objetivo identificar problemas de sincronização para computação. Problemas desse tipo são muito comuns em todos os estágios de desenvolvimento de circuitos, principalmente com o uso do QCADesigner. A simples modificação equivocada das propriedades de uma célula pode produzir resultados finais inesperados e de complexa detecção.

4.2.1 Modelo

Problemas de sincronização são de mais difícil detecção quando acontecem dentro de uma zona de clock. Assim, é criado um modelo para cada zona baseando-se em cada uma das células, incluindo as de zonas de clock anterior e posterior, que são transformadas em variáveis numéricas que quantificam a distância (em células) delas com a zona de clock anterior referencial dessa vizinhança. Células que fazem parte da zona de clock anterior tem valor fixo zero (nulo) pois são, por definição, referenciais.

Distâncias nesse modelo devem estar limitadas entre 0 e 7, a fim de possibilitar a detecção da inversão termodinâmica.

Assim como observado por Lent & Tougaw [1997], em sequências de propagação muito longas, normalmente de tamanho superior a 8 células, há a ocorrência de adversidades que podem causar resultados inesperados, denominado como problema da inversão termodinâmica. Assim, o projetista de um circuito deve ser avisado de tal situação para que analise cuidadosamente o funcionamento de tal bloco. No modelo proposto, portanto, as distâncias serão limitadas a valores entre 0 e 7, de forma a prevenir, imediatamente, a existência de tais revés.

A adjacência direta de células posicionadas diagonalmente umas às outras somente será considerada se não houver influência maior horizontal ou vertical. A figura 4.3 exemplifica tal situação. A célula verde e a amarela são consideradas adjacentes umas as outras em ambas as situações. As células vermelhas, no entanto, não são consideradas adjacentes à verde visto que a amarela está mais próxima e a influenciará mais.

Um bloco é considerado correto se todas as suas células são adjacentes a uma outra célula que tenha valor numérico imediatamente superior ao valor da primeira. Células de outras zonas de clock (anterior e posterior) não devem ser verificadas pois são apenas utilizadas como referência.

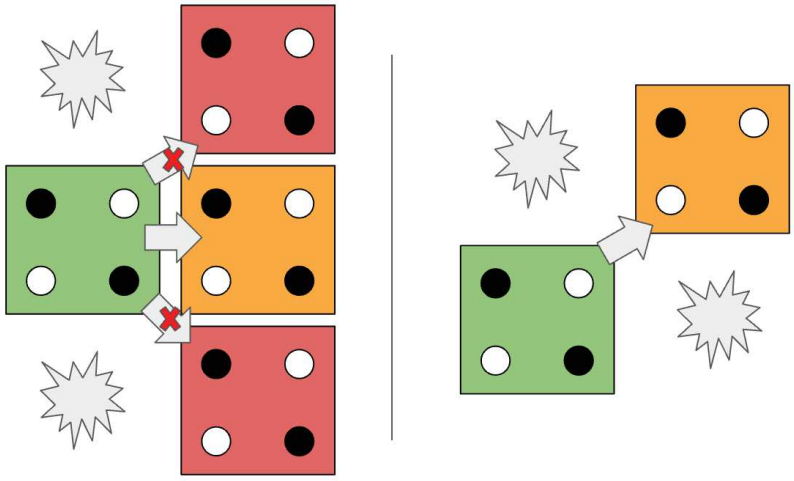

Figura 4.3. Tratamento de influência diagonal

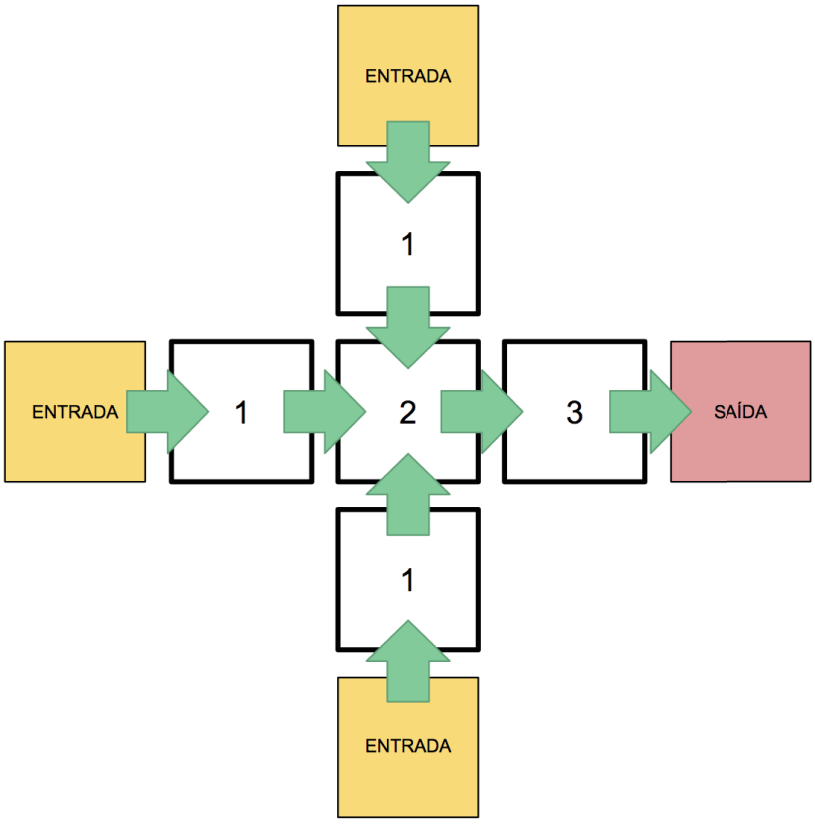

Figura 4.4. Verificação interna de uma Porta da Maioria

4.2.2 Funcionalidade

O método descrito funciona perfeitamente em designs tradicionais. A figura 4.4 demonstra a funcionalidade correta na vizinhança de uma Porta da Maioria. O valor

numérico no interior de cada célula, como descrito antes, representa a menor distância (em células) para qualquer entrada. O fluxo de informação pode ser visualizado pelo correto incremento dos valores das entradas em direção às saídas. A convergência simultânea de valores na célula central demonstra que a computação está sendo realizada corretamente.

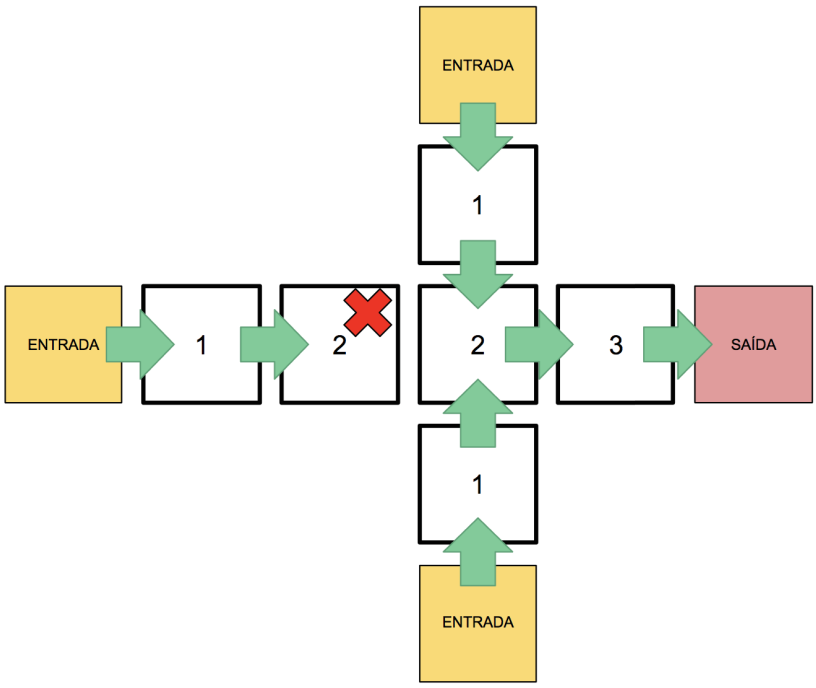

Figura 4.5. Verificação interna de uma Porta da Maioria defeituosa (assíncrona).

Problemas de sincronização são facilmente detectados por este método. A figura 4.5 mostra um exemplo dessa situação onde uma célula extra fora adicionada em um dos caminhos das entradas de uma Porta da Maioria. O X vermelho demonstra a célula que fere as regras por não ter uma outra célula adjacente de valor imediatamente superior.

4.2.3 Indução de falhas

O mesmo método pode também ser aplicado para detectar problemas pontuais que podem acontecer durante a criação de um circuito. Mesmo que os resultados da simulação pareçam estar corretos, existe a possibilidade de problemas aparecerem em futuras adaptações e fusões.

De forma a demonstrar tais funcionalidades, três diferentes falhas foram induzidas no circuito demonstrado na figura 4.6. O esquema de cores visa ilustrar as diferentes zonas de clock que se inicia-se na cor magenta-escuro e progride sucessivamente às zonas de cor ciano, laranja e verde-escuro.

Figura 4.6. Circuito básico correto

4.2.3.1 Desconexão

Nessa situação, a célula de zona de clock laranja tem sua zona de clock erroneamente trocada para a posterior (verde-escuro). A figura 4.7 mostra esta situação e o respectivo modelo falho resultante da vizinhança cor ciano. A não existência de uma saída causa tal falha.

Tal situação pode também ser detectada na etapa de particionamento (descrita na seção 4.1) já que vizinhanças somente seriam construídas se diretamente conectadas (sem saltos de zona de clock) a saídas. Seu caso é aqui analisado a fim de demonstrar a robustez do modelo, já que situações semelhantes complexas podem existir.

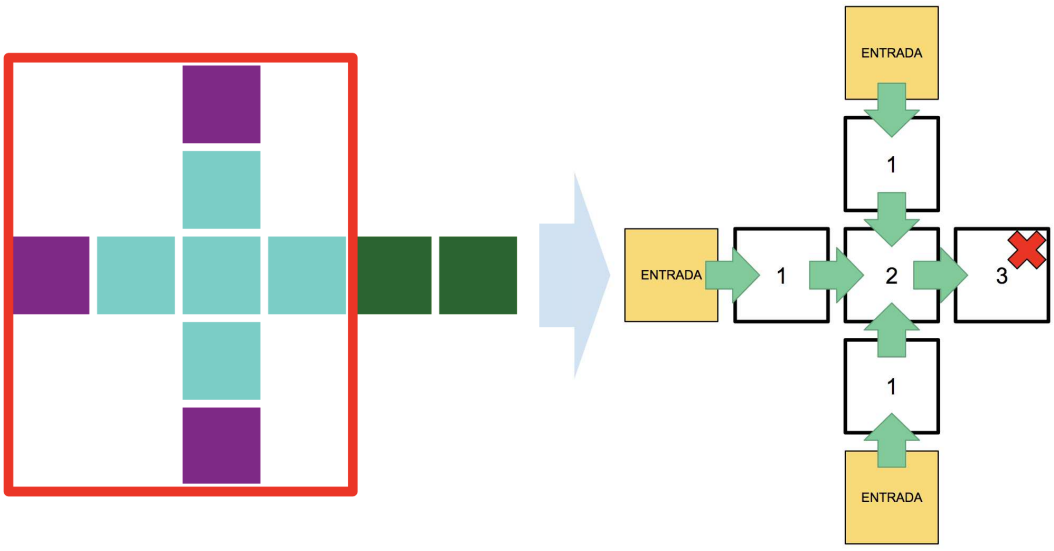

Figura 4.7. Induzindo um erro de desconexão

4.2.3.2 Falha de sincronização

Essa situação, demonstrada na figura 4.8, se define pela mudança errônea da zona de clock de uma célula pertencente à uma Porta da Maioria que ocasione um problema de sincronização. Diferentemente da situação demonstrada na figura 4.5, um dos caminhos fora encurtado.

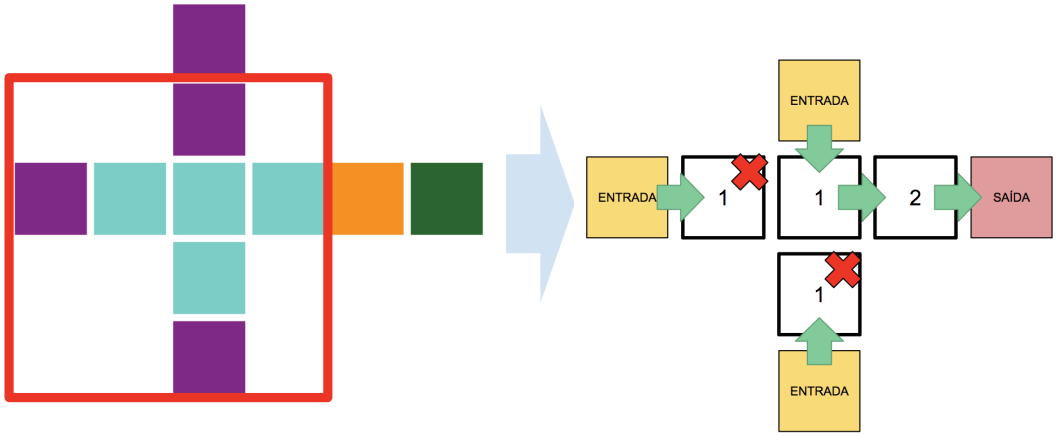

Figura 4.8. Induzindo um erro de sincronização

4.2.3.3 Célula adjacente desnecessária

Uma célula comum laranja é adicionada desnecessariamente como demonstrado na figura 4.9. Devido às definições de adjacência diagonal, não haverá nenhuma outra célula adjacente que não a outra célula laranja, causando a quebra de regra ao não permitir que o seu fluxo alcance diretamente uma saída.

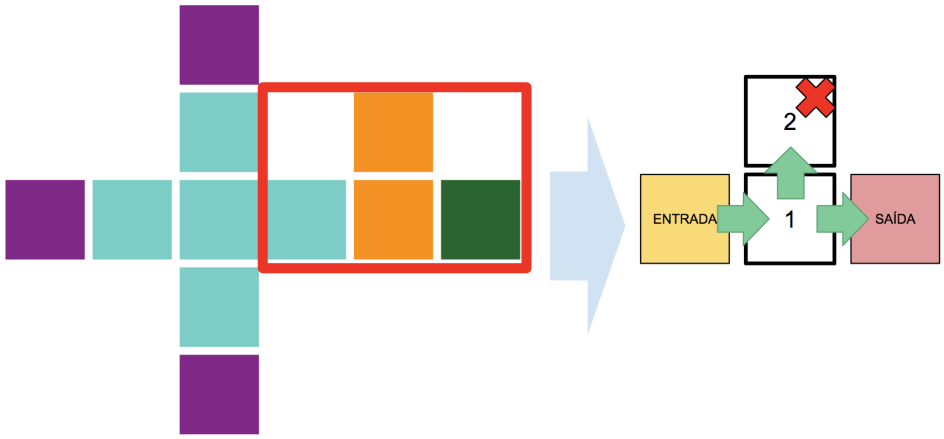

Figura 4.9. Induzindo um erro de célula desnecessária

4.2.4 Modelagem em SMV

De forma a permitir a automatização do processo de verificação geral de um circuito, um modelo SMV é criado para cada partição obtida pelos passos descritos na seção 4.1 seguindo a especificação anteriormente descrita nessa seção.

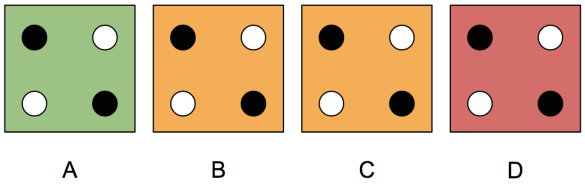

Figura 4.10. Exemplo simples de vizinhança

A figura 4.10 demonstra uma vizinhança de quatro células (A, B, C e D) onde duas (B e C) fazem parte da zona de clock de referência. O modelo SMV construído automaticamente para ele será:

```
1  MODULE main

2  VAR A: {0};
3  VAR B: 0..7;
4  VAR C: 0..7;
5  VAR D: 0..7;

6  initial(B) := 7;
7  initial(C) := 7;
8  initial(D) := 7;

9  ASSIGN next(B) := case
10 (A < C & A < B) : A + 1;
11 (C < B) : C + 1;
12 TRUE: B;
13 esac;
```

```
14  ASSIGN next(C) := case
15  (B < C) : B + 1;
16  TRUE: C;
17  esac;

18  ASSIGN next(D) := case
19  (C < D) : C + 1;
20  TRUE: D;
21  esac;

22  LTLSPEC G F ((B = A - 1) | (B = C - 1))
23  LTLSPEC G F (C = B - 1)
```

A linha 1 apenas caracteriza o módulo principal e é necessário para todo arquivo SMV. As linhas 2 a 5 declaram os possíveis valores que as variáveis (correlacionadas a células) podem obter. A célula A, por ser de uma zona de clock anterior, terá seu valor fixo nulo. As linhas 6 a 8 declaram o valor inicial para as variáveis não constantes. Ele será máximo de forma a permitir a execução da lógica de minimização.

As linhas seguintes declaram a evolução dos valores para cada uma das variáveis de forma correlacionada com a adjacência das células relacionadas:

- B é adjacente a A e C (linhas 9 a 13);
- C é adjacente apenas a B pois D se encontra em zona de clock posterior (linhas 14 a 17);
- D é adjacente apenas a C (linhas 18 a 21).

As linhas 22 e 23 declaram as regras de segurança utilizadas para as células B e C, respectivamente. A utilização de LTL se mostra como mais eficiente para este caso, já que os valores numéricos não são suscetíveis a variações. Toda execução trará sempre o mesmo valor, sendo a trajetória única.

Por fim, o modelo gerado automaticamente pode ser carregado na ferramenta NuSMV para que as preposições ali contidas sejam comprovadas ou não. Uma falha indicará claramente a célula que não as obedece e que provavelmente causará falhas de sincronização ou de inversão termodinâmica.

4.3 Simulação Paralela de Partições

Normalmente, simulações no QCADesigner são feitas usando o método de Vetor de Coerência. Este motor apresenta resultados mais coesos ao se basear em varrer todas as células calculando sua polarização de forma fracionada a cada passo de tempo. Cálculos são feitos mesmo que sua zona de clock esteja inativa e não cause impacto nenhum nos resultados finais.

O processo de simulação precisa ser atualizado a fim de permitir as partições independentes de um circuito sejam paralelamente simuladas. Entretanto, não se pode fazer uso direto dos métodos de cálculo do QCADesigner já que eles fazem uso de polarizações fracionadas que podem causar dependência direta de dados. Propõe-se aproximar seu funcionamento a lógica usada em circuitos CMOS a fim de resolver tal problema que impossibilita a execução paralela. Entradas e resultados serão discretizados: valores positivos de polarização representam o binário *1*, enquanto valores negativos representam o *0*.

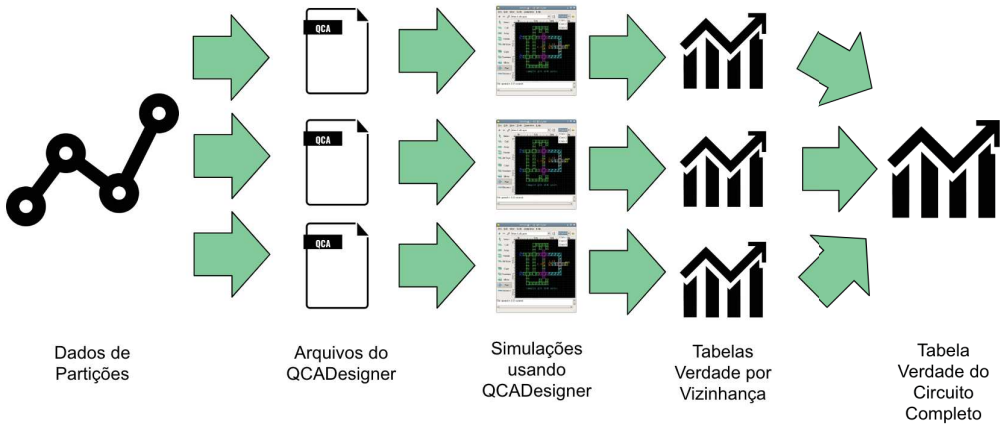

Figura 4.11. Representação esquemática do processo de simulação paralela de partições

A figura 4.11 resume o processo de simulação paralela a ser descrita nessa seção. O funcionamento destes novos procedimentos tem boa viabilidade de uso em estágios iniciais de criação, porém pode se distanciar de resultados reais em alguns casos. O projetista deve se atentar a também executar baterias de testes de forma tradicional (não paralela) confirmando as funcionalidades esperadas.

4.3.1 Tabela verdade de cada partição

As mesmas vizinhanças descritas na seção 4.1 podem ser reconstruídas em diferentes pequenos circuitos. As células da zona de clock em questão são caracterizadas como saídas (para que tenham suas polarizações gravadas), enquanto células da zona de clock anterior são marcadas como entradas. Células da próxima zona de clock, nesta etapa, são descartadas.

A figura 4.12 mostra um circuito de uma Porta da Maioria alimentado por dois fios tradicionais e um fio inversor. Cada partição desse circuito está demarcada em azul e pode ser simulada separadamente. As células presentes em mais de uma região podem ser utilizadas para conectar os valores de polarização dado um conjunto de entrada a fim de obter a polarização correta final das saídas (neste caso, demarcadas em vermelho).

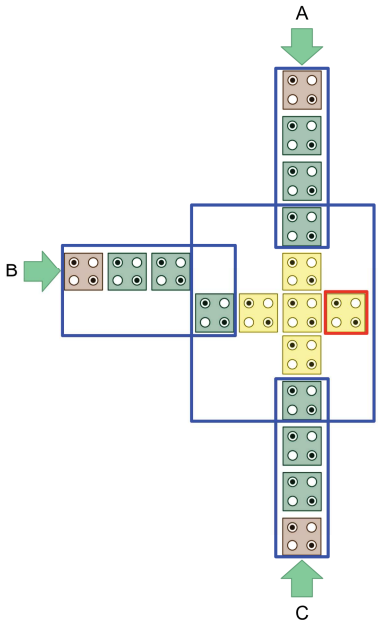

Figura 4.12. Circuito de uma Porta da Maioria alimentado por três diferentes fios

Os pequenos circuitos então serão simulados usando o motor de Vetor de Coerência do QCADesigner. Visando garantir a cobertura, todos as possíveis combinações (discretizadas) de entrada são utilizadas na simulação. Ademais, visando evitar problemas que podem ser gerados por uma ordem específica, a sequência de combinações é invertida e usada em uma segunda simulação obrigatória para fins comprobatórios.

O resultado final de cada simulação será utilizado para a construção de uma tabela verdade dessa partição. Ela relaciona o valor da polarização das entradas com os respectivos valores de cada uma das outras células relevantes.

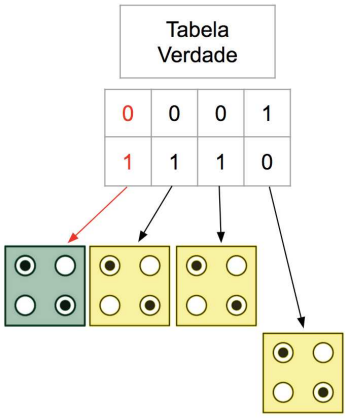

Figura 4.13. Tabela Verdade de uma partição com os valores relacionados

A figura 4.13 ilustra e relaciona os valores de uma tabela verdade gerada a partir da simulação da partição que representa o fio inversor do circuito apresentado na figura 4.12. Células de clock anterior (em verde) são consideradas entradas enquanto células amarelas, mesmo sendo comuns no circuito original, saídas. O resultado obtido que pode ser visualizado na tabela verdade comprova a funcionalidade de inversor na última célula da direita desse bloco.

Estruturas muito simples, de fácil detecção e de resultados previamente conhecidos podem evitar passar pelo processo de simulação. Um fio unidimensional, por exemplo, terá a polarização de todas as células, no próximo passo de tempo, sempre igual ao valor da polarização da célula de entrada.

4.3.2 Resultados por saltos em Tabelas Verdade

Uma simulação sempre levará em consideração uma sequência de valores para as entradas a fim de obter os valores das saídas. Dessa forma, se as tabelas verdade das vizinhanças forem consultadas, partindo das entradas, pode-se obter os valores de saída.

A figura 4.14 exemplifica a técnica baseada em saltos de tabela verdade de um circuito composto de quatro partições. Com os valores de entrada A, B e C fixados em 0, a propagação de valores causará, em um passo de tempo, a ocorrência do valor 0 para a célula de saída (D3). Qualquer outra combinação de entradas pode ser utilizada para obtenção de qualquer polarização relevante, mesmo que não seja uma saída.

Esta técnica, no entanto, suporta apenas circuitos combinacionais. Circuitos sequenciais trariam a existência de termos X (*don't care*) na resolução de saltos, inviabilizando seu uso.

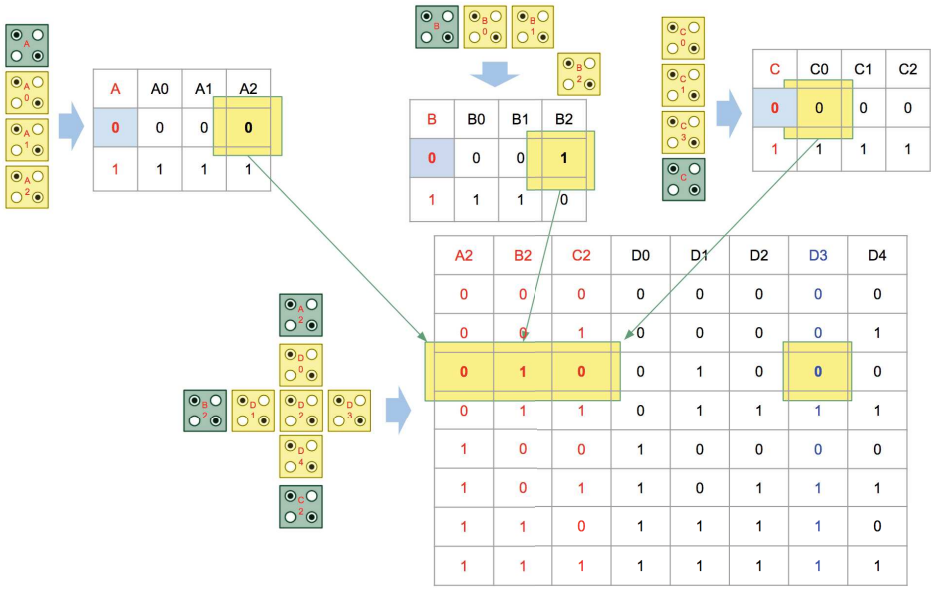

Figura 4.14. Demonstração de obtenção de valores de simulação por salto em tabela verdade

4.4 Verificação Lógica entre partições

Um modelo SMV pode ser automaticamente construído baseado nas tabelas verdade criadas na seção 4.3. Cada vizinhança terá um módulo correspondente construído de forma a correlacionar os valores de entrada com a polarização de cada uma das células ali contidas. O módulo principal instanciará cada um desses módulos e sinais de entrada, conectando-os de forma a respeitar os atrasos de sincronização.

Por representar logicamente um circuito, o comportamento obtido deve ser ramificado, ou seja, deve permitir que diferentes valores de entrada sejam utilizados na obtenção de diferentes valores de saída. Um modelo de trajetória limitaria as possibilidades do projetista.

Assim, seria construído um modelo SMV que representaria logicamente o circuito como um todo. O circuito de uma Porta da Maioria e de fios os alimentando — o mesmo utilizado mostrado na figura 4.12 e utilizado também como exemplo na seção 4.3 — teria a seguinte estrutura:

```
1  MODULE D(A2,B2,C2)
2      VAR
3      D0: {X,0,1};
   ...
4      D3: {X,0,1};

5      initial(D0) := X;
   ...
6      initial(D3) := X;

7      next(D0) := case
8          TRUE & A2 = 0 & B2 = 0 & C2 = 0: 0;
9          TRUE & A2 = 0 & B2 = 0 & C2 = 1: 0;
10         TRUE & A2 = 0 & B2 = 1 & C2 = 0: 0;
11         TRUE & A2 = 0 & B2 = 1 & C2 = 1: 0;
12         TRUE & A2 = 1 & B2 = 0 & C2 = 0: 1;
13         TRUE & A2 = 1 & B2 = 0 & C2 = 1: 1;
14         TRUE & A2 = 1 & B2 = 1 & C2 = 0: 1;
15         TRUE & A2 = 1 & B2 = 1 & C2 = 1: 1;
16         TRUE: D0;
17     esac;

18     next(D3) := case
19         TRUE & A2 = 0 & B2 = 0 & C2 = 0: 0;
20         TRUE & A2 = 0 & B2 = 0 & C2 = 1: 0;
21         TRUE & A2 = 0 & B2 = 1 & C2 = 0: 0;
22         TRUE & A2 = 0 & B2 = 1 & C2 = 1: 1;
23         TRUE & A2 = 1 & B2 = 0 & C2 = 0: 0;
24         TRUE & A2 = 1 & B2 = 0 & C2 = 1: 1;
25         TRUE & A2 = 1 & B2 = 1 & C2 = 0: 1;
26         TRUE & A2 = 1 & B2 = 1 & C2 = 1: 1;
27         TRUE: D3;
28     esac;
```

```
...
MODULE main
VAR
    A: {0,1};
    B: {0,1};
...
    VIZA: A(A);
    VIZB: B(B);
    VIZC: C(C);
    VIZD: C(VIZA.A2,VIZB.B2,VIZC.C2);
...
```

A linha 1 exemplifica a criação de um módulo para cada vizinhança e declara cada uma das suas entradas.

A declaração das variáveis que armazenam o valor lógica de cada uma das células internas é feita logo em seguida (linhas 3 e 4).

O valor X será utilizado como valor inicial de células (demonstrado nas linhas 5 e 6). As linhas 7 a 28 demonstram os novos valores dessas em relação às diferentes combinações de entrada (usando as tabelas verdades).

Neste modelo, todas as células terão valor a cada momento devido ao fluxo de informação ser diferente do utilizado nas simulações tradicionais: cada passo de tempo representa um ciclo completo de clock.

As linhas 16 e 27, especialmente, garantem que o valor X perseverá enquanto as entradas não tenham valores válidos. Isso implica que, como na simulação particionada, este modelo não tratará a lógica com termos *don't care*.

A linha 29 declara o módulo principal seguido das declarações das reais entradas do circuito nas linhas 31 e 32. O modelo é finalizado com a declaração de cada vizinhança com suas respectivas conexões (linhas 33 a 36).

4.4.1 Regras

Circuitos úteis têm saídas que variam dependendo dos valores de entrada. Uma saída que sempre tiver valor fixo pode ser substituída, no caso de QCA, por células de polarização fixa. Assim, para cada saída, duas regras CTL de vivacidade podem ser automaticamente adicionadas ao modelo SMV. Uma buscará a cobertura (ou alcance) do valor binário *1* e outra a do *0*. A comprovação de apenas uma dessas regras signi-

ficará que essa saída é inútil. A falha de ambas é impossível pois representaria que a saída é inalcançável. O particionamento inicial (descrito na seção 4.1) já detectaria e evitaria a propagação dessa falha para passos futuros. Por exemplo, para o circuito com uma célula de saída D3 pertencente à vizinhança D, teremos as seguintes preposições:

SPEC **EF** VIZD.D3 = 0
SPEC **EF** VIZD.D3 = 1

O projetista, da mesma forma, pode querer demonstrar como alcançar uma determinado valor de saída. A regra deverá especificamente tentar comprovar a negação desta combinação — se atentando a não considerar o valor *X* — pois o contra-exemplo que pode ser gerado demonstrará claramente uma sequência capaz de alcançá-la. A título de exemplo, a seguinte regra geraria um contra-exemplo capaz de demonstrar como o valor *1* pode ser alcançado em VIZD.D3:

SPEC **AF** VIZD.D3 = X | VIZD.D3 != 1

Proposições lógicas com intuito de conferir características individuais do circuito, no entanto, não podem ser geradas automaticamente. O projetista pode as propor utilizando o valor de qualquer uma das células e ainda relacionar o atraso esperado de sincronização ao utilizar a noção de passo de tempo introduzida pelo modelo.

No mesmo exemplo descrito anteriormente, pode-se querer verificar que a saída D3 funcione realmente como uma Porta da Maioria. Dois valores positivos de entrada, em A e C, implicariam que a saída D3, no próximo passo de tempo (próxima transmissão de informações) também tenha valor positivo. Assim, a seguinte regra pode ser redigida:

SPEC A = 1 & C = 1 -> **AX** VIZD.D3 = 1

Capítulo 5

Resultados

Neste capítulo serão apresentados resultados que comprovam o funcionamento das técnicas de validação de circuitos descritas no Capítulo 4. Inicialmente se descreverá a coesão e desempenho do uso em circuitos de funcionalidade real e em seguida será feita uma análise quantitativa das etapas e em comparação com a performance do QCADesigner.

5.1 Parâmetros de execução

Todos os resultados apresentados foram obtidos com a utilização de um computador que executa o sistema operacional macOS 10.13 com processador Intel Core i7 2.9 GHz 7820HQ (4 núcleos/8 threads) e 16 GB de memória RAM 2133 MHz LPDDR3. Todos os testes doravante apresentados foram executados 5 (cinco) vezes e o tempo de execução, quando exibido, refere-se ao tempo médio desses experimentos.

Quando realizadas, simulações do QCADesigner foram feitas utilizando o motor de vetor de coerência com configurações padrões. Estes parâmetros podem ser visualizados na tabela 5.1.

Parâmetro	Valor
Dimensões das células	18 nm x 18nm
Diâmetro do ponto quântico	5nm
Tolerância de convergência	0.001
Raio de efeito	80 nm
Permissividade relativa	12.9
Energia máxima do clock	9.8E-22J
Energia mínima do clock	3.8E-23J
Fator de amplitude do clock	2
Distância entre camadas	11.5 nm
Máximo de iterações por amostra	100

Tabela 5.1. Parâmetros padrões de simulação do modelo de Vetor de Coerência em uso no QCADesigner

5.2 Aplicações em circuitos reais relevantes

A boa execução do processo deve ser corroborada por testes que se aproximem com utilizações reais, ou seja, devem ser realizados em circuitos reais de utilidade comprovada. Assim, elegeu-se os circuitos do Multiplexador e do Gerador de código de Hamming para esta função:

- **Multiplexador (4-para-1)**

  Multiplexadores tem como objetivo receber diferentes sinais e escolher, por meio de sinais de seleção, quais deles terá seu valor redirecionado para a saída.

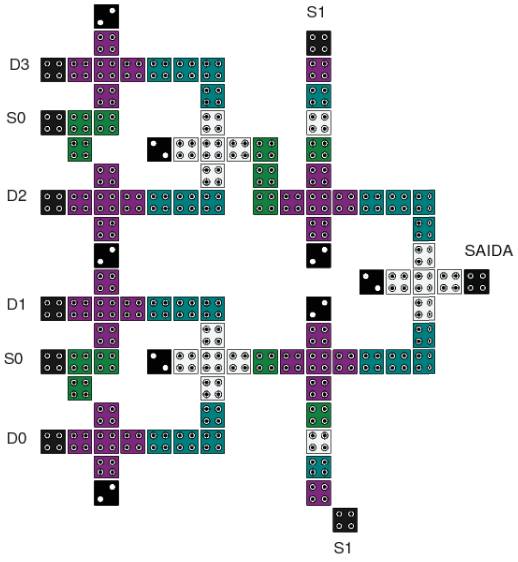

Figura 5.1. Circuito do Multiplexador 4-para-1 (Sardinha et al. [2013])

A variante a ser utilizada é baseada na proposta por Sardinha et al. [2013]. A figura 5.1 demonstra esta implementação. O valor em SAÍDA será escolhido, pelos sinais S0 e S1, dentre os valores D0 a D3. A coloração das células indica a zona de clock a que pertencem ou, quando em preto, que são células em modo especial: células de valor fixo tem polarização claramente visível enquanto outras são entradas ou saídas.

- **Gerador de código de Hamming (4 bits)**:

  O código de Hamming se mostra relevante para aplicações de comunicação. Ele, eficientemente, permite a transmissão segura de informações ao permitir a correção automática de até um bit de cada bloco.

  Propõe-se utilizar como base de análise o circuito capaz de gerar o código de Hamming para conjuntos de informações de 4 bits. Neste caso, a saída será composta de 3 (três) bits que garantem sua resiliência. A figura 5.2 mostra a estrutura utilizada para os testes que é uma versão adaptada da apresentada por Silva et al. [2015]. O código de cores da representação das células é o mesmo anteriormente descrito.

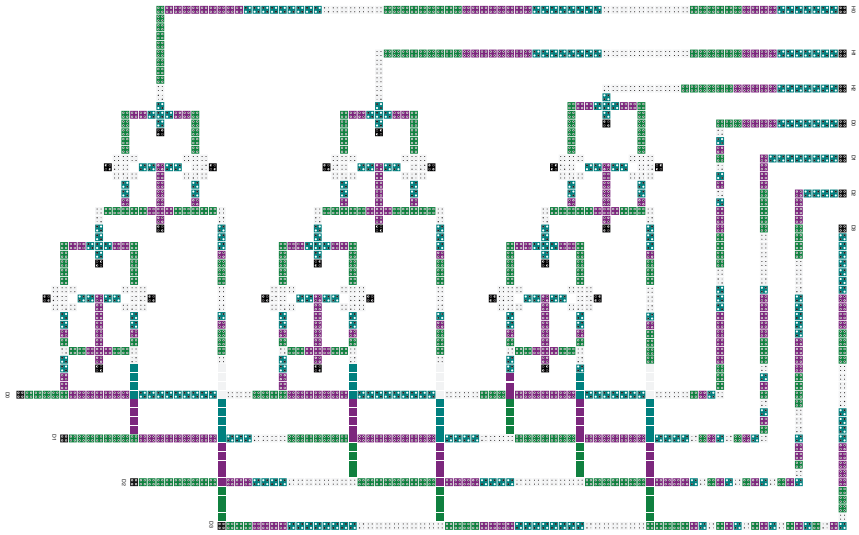

Figura 5.2. Circuito do Gerador de código de Hamming para 4 bits (Silva et al. [2015])

5.2.1 Particionamento Automático

A primeira fase a se realizar e comprovar é a do particionamento. Esta etapa é básica e necessária para todas as outras. Ela baseia-se em arquivos do QCADesigner, criando suas representações por matriz e dividindo automaticamente o circuito em regiões independentes.

Característica	Multiplexador	Hamming
Número de células	102	1163
Número de partições	42	352
Número de partições multidimensionais	15	101
Tempo de execução da etapa	1.1s	1.2s

Tabela 5.2. Análise de desempenho do processo de particionamento em circuitos reais relevantes

A execução desta fase nos circuitos relevantes em questão não ocasionou em nenhum problema. A classificação de partições como unidimensionais, que sempre executam a função de fio, ou multidimensionais. A tabela 5.2 mostra os resultados obtidos.

5.2.2 Verificação interna de partições

A etapa de verificação interna de partições é a segunda a ser executada, procedendo a etapa de particionamento. Ela deverá ser executada em todas as partições, inclusive unidimensionais, em busca de problemas de sincronização e de inversão térmica.

Característica	Multiplexador	Hamming
Número de partições analisadas	42	352
Tempo médio de execução individual por partição	< 0.1s	< 0.1s
Tempo total de execução da etapa	1.4s	1.6s

Tabela 5.3. Análise numérica isolada do processo de verificação interna de partições em circuitos reais relevantes

A verificação automática de cada região pode ser feita de forma completamente paralela já que não há dependência de informações entre elas. A tabela 5.3 mostra os resultados obtidos que, como esperado, não resultaram em detecção de problemas nos aclamados circuitos. O processo em si, mesmo bastante veloz, se apresentou um pouco mais lento que o esperado devido à necessidade de escrita de arquivos *SMV* e a coordenação de processos do *NuSMV*.

5.2.3 Simulação paralela

A etapa da simulação paralela visa a obtenção da polaridade de determinadas células de forma mais veloz do que quando feita completamente no QCADesigner. Ela, como a etapa de verificação de partições, acontecerá em cada uma das distintas partições independentes do circuito, preenchendo as tabelas verdade (ou de correlação de valores entrada-saída). Dado que cada região é independente das outras, a execução das simulações destas pode ser realizada de forma paralela.

A análise de desempenho desse estágio deve se atentar ao atraso — representando o número de zonas de clock entre as entradas e saídas mais distantes — e o número de partições multidimensionais do circuito. Fios unidimensionais são de fácil detecção e tem execução desnecessária, já que apenas replicam os valores de entrada.

Característica	Multiplexador	Hamming
Número total de partições	42	352
Número total de partições multidimensionais	15	101
Atraso total para obtenção de resultados	10	37
Tempo de execução		
Particionamento	1.4s	1.6s
Simulação	8.7s	39.6s
Total	10.1s	41.2s
Tempo de simulação no QCADesigner	49.6s	205.4s

Tabela 5.4. Análise de desempenho do processo de simulação paralela em circuitos reais relevantes

A tabela 5.4 mostra o tempo de execução das etapas necessárias para a simulação (desconsiderando o processo de verificação, a ser aferido posteriormente) nos referidos circuitos comparando-os com o desempenho utilizado ao utilizar o QCADesigner de forma legada. Uma aceleração aproximada de 5 vezes pode ser notada ao se comparar o tempo necessário de execução das duas etapas relevantes em comparação com o necessário pelo QCADesigner.

5.2.3.1 Coesão de resultados

A simulação paralela também deve obter resultados coesos, ou seja, devem ser os mesmos obtidos pela simulação completa no QCADesigner.

Em ambos os circuitos testados houve coerência nos resultados. Entretanto, por motivos didáticos, serão apresentados apenas os resultados do circuito do Gerador de código de Hamming para 4 bits. A maior quantidade de entradas no circuito do Multiplexador tornaria difícil a distinção dos resultados nas tabelas e gráficos construídos e qualquer outra abordagem seria seletiva e não comprovaria completamente o funcionamento.

A tabela 5.5 mostra extensivamente os resultados obtidos pelo processo de simulação paralela, enquanto a figura 5.3 mostra os resultados da mesma simulação realizada completamente no QCADesigner. As três primeiras combinações da tabela estão coloridas de forma coerente com a mesma combinação apresentada nos resultados do QCADesigner. O atraso de inicialização presente condiz com o atraso (número de partições consultadas) da simulação paralela fazendo os ajustes necessários, já que cada ciclo completo de completo equivaleria a 4 atrasos.

Entradas				Saídas		
D0	D1	D2	D3	H0	H1	H3
0	0	0	0	0	0	0
1	0	0	0	1	1	0
0	1	0	0	1	0	1
1	1	0	0	0	1	1
0	0	1	0	0	1	1
1	0	1	0	1	0	1
0	1	1	0	1	1	0
1	1	1	0	0	0	0
0	0	0	1	1	1	1
1	0	0	1	0	0	1
0	1	0	1	0	1	0
1	1	0	1	1	0	0
0	0	1	1	1	0	0
1	0	1	1	0	1	0
0	1	1	1	0	0	1
1	1	1	1	1	1	1

Tabela 5.5. Tabela verdade obtida pelo processo de simulação paralela do circuito do Gerador de código de Hamming para 4 bits

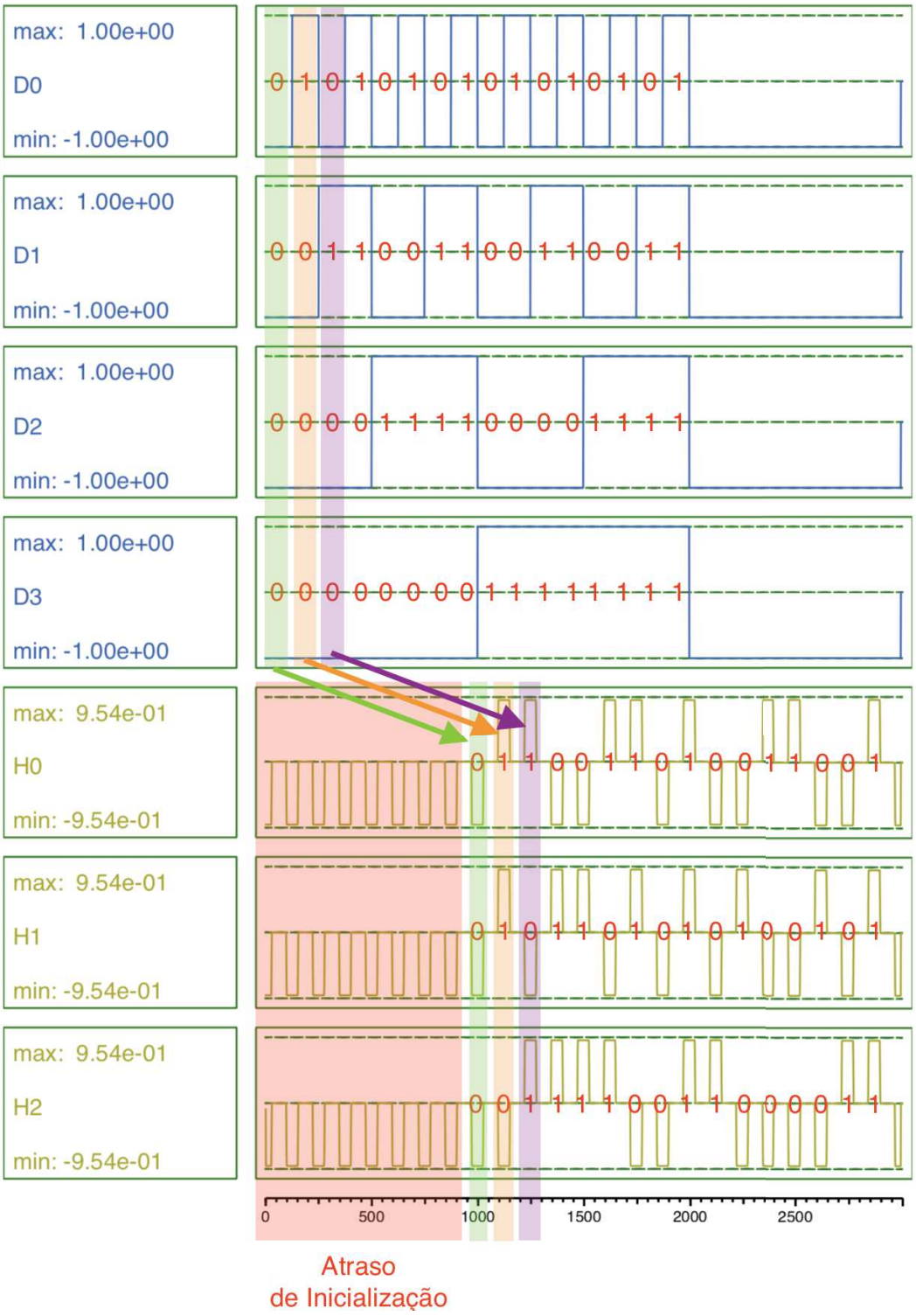

Figura 5.3. Resultados de simulação extensiva no QCADesigner do circuito do Gerador de código de Hamming para 4 bits

5.2.4 Verificação de cobertura

Com os resultados condizentes da simulação paralela, podemos utilizar as tabelas verdade geradas para cada partição na geração de um modelo lógico que consegue representar o circuito de forma completa.

O processo inicial a ser feito deve comprovar a utilidade das saídas, assim, descrevendo um processo de verificação de cobertura destas.

Para permitir comparações, mesmo que um circuito tenha múltiplas saídas, cada uma delas terá a cobertura verificada sozinha em cada um dos possíveis valores lógicos. Deste modo, cada situação de teste teve uma, e apenas uma, preposição adicionada da seguinte forma:

SPEC EF ***SAIDA*** = ***VALOR***

Onde:

- ***SAIDA***: Nome da saída a ser verificada
- ***VALOR***: Valor lógico esperado pelo teste (0 ou 1)

Característica	Multiplexador	Hamming
Número total de saídas	1	3
Número total de testes	2	6
Tempo de verificação de cobertura		
Média	10s	57814s (∼16h)
Mínimo	9s	57802s (∼16h)
Máximo	11s	57832s (∼16h)

Tabela 5.6. Análise de desempenho do processo de verificação de cobertura em circuitos reais relevantes

A tabela 5.6 mostra o tempo necessário para a correta verificação de cobertura. É notável que o tamanho do circuito influencia bastante o tempo de verificação. Enquanto um circuito mediano, como o do Multiplexador, consegue ser verificado em segundos, circuitos como o do Gerador de código de Hamming exigem horas de execução. O crescimento exponencial das possibilidades do modelo causou um gasto muito elevado de memória, inclusive, estourando o limite de memória RAM disponível. O amplo espaço e a provável paginação da memória deterioraram o desempenho.

5.2.5 Verificação lógica

A verificação lógica funcional dos circuitos é o passo seguinte para a validação completa dos circuitos. Dada a elevada complexidade do modelo lógico do Gerador de código de Hamming (observada na etapa de verificação de cobertura) decidiu-se apenas pela análise de regras para o circuito do Multiplexador.

A verificação do circuito do Multiplexador deve comprovar que os sinais seletores funcionam corretamente, assim, pode-se propor regras que seguem a seguinte estrutura:

LTLSPEC D**i** = **v** & S0 = $\mathbf{i_0}$ & S1 = $\mathbf{i_1}$ -> F SAIDA = **v**

Onde:

- **i**: Valor de 0 a 3
- **v**: Valor lógico esperado pelo teste (0 ou 1)
- $\mathbf{i_0}$: Bit 0 da representação binária de $\boldsymbol{i}$
- $\mathbf{i_1}$: Bit 1 da representação binária de $\boldsymbol{i}$

Pode-se também propor regras que levam em conta a demora esperada para recebimento de informação. Como o atraso esperado é de 10 saltos, podemos reescrever as fórmulas propostas da seguinte forma:

LTLSPEC D**i** = **v** & S0 = $\mathbf{i_0}$ & S1 = $\mathbf{i_1}$ -> X X X X X X X X X X SAIDA = **v**

É esperado que a verificação com atraso exato seja um pouco mais lenta do que a que faz uso do operador de futuro. A execução de todas as situações não ocasionaram na detecção de erros. A tabela 5.7 mostra os resultados obtidos. Ambas as regras, neste caso, são equivalentes, entretanto quantificar o atraso trás um nível a mais de segurança para as regras, o que pode evitar possíveis surpresas em modificações futuras.

Característica	Valor
Atraso (em saltos)	10
Número de testes por tipo de verificação	8
Tempo de execução da verificação com atraso inexato	
Médio	14.9s
Máximo	15.1s
Mínimo	14.8s
Tempo de execução da verificação com atraso exato	
Médio	15.2s
Máximo	15.3s
Mínimo	15.1s

Tabela 5.7. Análise de execução do processo de verificação lógica funcional no circuito do Multiplexador

5.3 Indução de Erros

A confirmação de regras é importante, mas a identificação de falhas é ainda mais. É necessário existir a possibilidade de identificar facilmente causadores de erros. Pode-se induzir prováveis erros que devem ser prontamente identificados pelos métodos descritos anteriormente.

5.3.1 Erro interno de partição

Erros mais simples, como a adição errônea de uma célula podem ser detectados pela verificação interna de partições. A célula adicionada, por exemplo, pode causar problemas de sincronização.

A figura 5.4 mostra um exemplo de erro induzido desta forma no circuito do Multiplexador 4-para-1. A tabela 5.8 mostra a análise temporal da execução do processo neste circuito.

Etapa	Tempo
1. Particionamento	1.1s
2. Verificação de Partições	1.5s
Total	2.6s

Tabela 5.8. Análise temporal de execução no circuito do Multiplexador 4-para-1 com erro induzido por adição de célula

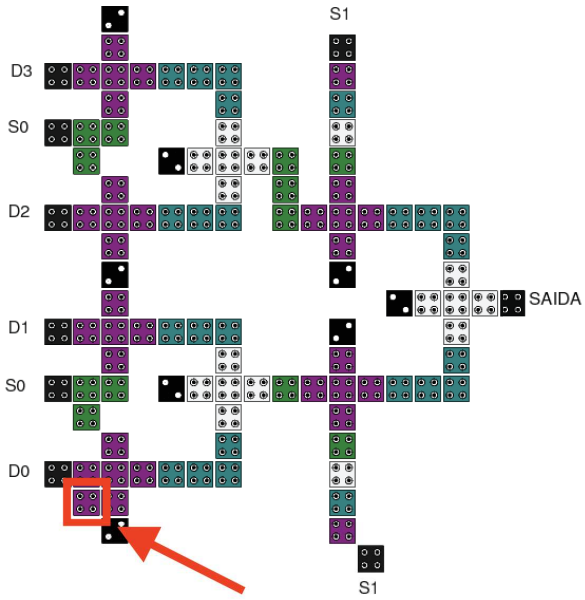

Figura 5.4. Indução de erro por adição de célula no circuito do Multiplexador 4-para-1

O erro pode ser verificado ao ter a regra, exatamente desta célula, violada. A execução do processo no NuSMV demonstrou o rastro que pode ser visualizado na figura 5.5. A célula extra adicionada fere as regras ao não estar transmitir informação a nenhuma outra, impedindo o fluxo.

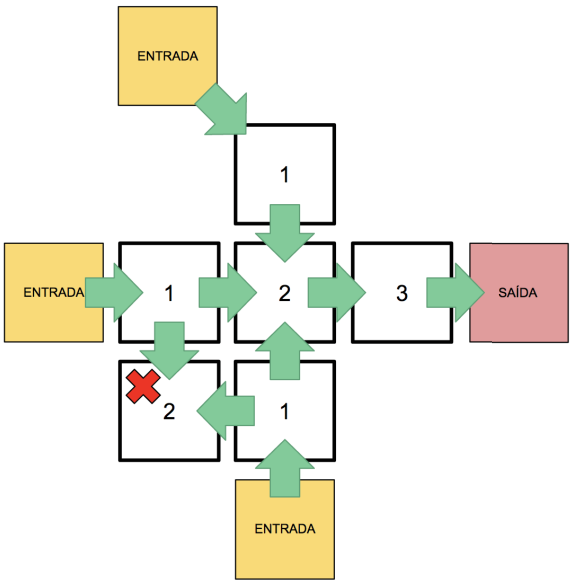

Figura 5.5. Rastro de verificação interna da partição com erro induzido por adição de célula

5.3.2 Erro lógico

O projetista de circuitos também pode querer estabelecer regras lógicas que comprovam o funcionamento correto de seu circuito. Em caso de violação, o modelo lógico criado do circuito deve ser capaz de gerar um contra-exemplo destas regras.

Para o circuito do Multiplexador, por exemplo, regras podem ser estabelecidas para garantir que certas combinações de entradas resultem em valores de saída específico: não importa qual combinação de valores de seletores, se todos os sinais de entrada tiverem valor 1, a saída necessariamente deve ter valor 1 (ou não ter valor 0, já que pode conter um valor ainda não determinado). Pode-se, assim, escrever a seguinte regra de segurança:

```
SPEC D0 = 1 & D1 = 1 & D2 = 1 & D3 = 1 -> AF SAIDA != 0
```

A aplicação desta regra no circuito normal, como esperado, não ocasiona na detecção de erros. Com o intuito de modificar o circuito sem causar outros problemas, pode-se inverter o valor lógico de uma das células fixas a fim de trocar o funcionamento de uma das portas lógicas de "E" para "OU", causando invalidação da lógica ali necessária.

Etapa	Tempo	Tempo Total
1. Particionamento	1.1s	1.1s
2. Simulação Particionada	9.5s	10.6s
3. Verificação Lógica	15.2s	25.8s

Tabela 5.9. Desempenho de execução no circuito do Multiplexador 4-para-1 com erro por inversão de célula fixa

A figura 5.6 destaca o circuito do Multiplexador com a célula de polarização fixa invertida. A tabela 5.9 demonstra o tempo utilizado para a execução de cada uma das etapas do processo de validação aplicadas nele. A etapa de simulação particionada, neste caso, é um pouco mais veloz devido à não necessidade da utilização da técnica de saltos em tabelas verdade. Apenas os dados para montagem do modelo de verificação lógica eram necessários.

O projetista, ao visualizar a mensagem de erro de descumprimento de regra pode visualizar o contra-exemplo gerado. A figura 5.6 também mostra o traço (polarização de cada uma das células) gerado pela ferramenta que é justamente capaz de demonstrar a existência de um valor 1, proibido, para a saída. Ele pode então confirmar que o valor esperado para as células em destaque estão incorretas e concluir que o valor da célula fixa está incorreto.

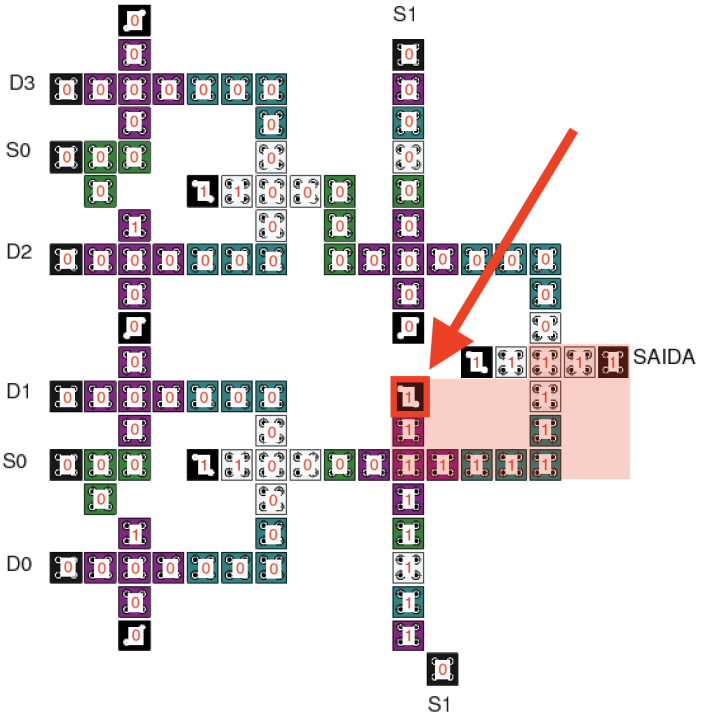

Figura 5.6. Multiplexador 4-para-1 com erro por inversão de célula fixa e respectivo rastro de execução

5.4 Análise quantitativa de desempenho

Por fim, é necessário avaliar o desempenho do processo completo de validação correlacionando-a com o tamanho de circuitos. Desta forma, é necessário a utilização de circuitos modulares simples que não tenham especifidades que facilitem ou compliquem a execução dos procedimentos. A utilização em circuitos reais é muito particular e não permite uma análise quantitativa completa.

O fio em QCA é uma estrutura que atende a esses fins por ser modular, tendo o objetivo simples de retransmitir controladamente informações da entrada para a saída. A utilização de sua variação simples (unidimensional) ocasiona na simplificação indesejada do processo de simulação, assim, decidiu-se pelo uso da variação multidimensional.

A figura 5.7 mostra a estrutura modular utilizada. A fácil criação de fios multidimensionais de tamanhos distintos pode ser realizada com a replicação da região delineada de forma horizontal com a devida atenção à modificação da zona de clock. O tamanho de um fio é considerado como o número de estruturas (zonas de clock) pelas quais o fluxo de dados deve propagar.

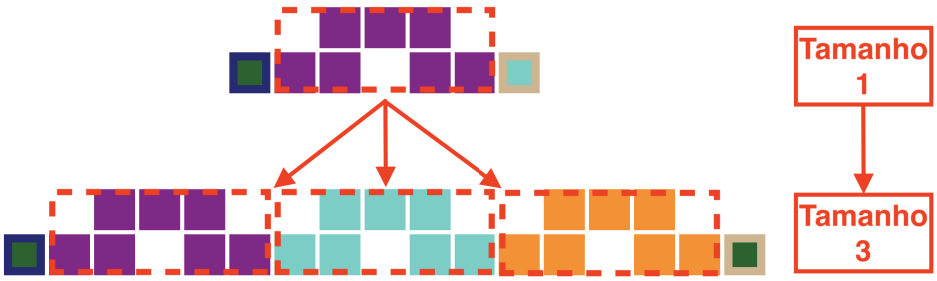

Figura 5.7. Estrutura modular de um fio multidimensional.

Para os testes, foram gerados 7 (sete) circuitos representantes de fios multidimensionais de tamanhos de 4 até 128, progredindo potências de 2 (dois). Em cada um deles executou-se cada uma das etapas separadamente. Para a etapa de verificação lógica verificou-se a correta transmissão de valores da entrada para a saída com uso das seguintes preposições:

```
SPEC ENTRADA = 0 -> AF SAIDA != 1
SPEC ENTRADA = 1 -> AF SAIDA != 0
```

O tempo de execução isolado de cada uma das etapas pode ser visualizado na tabela 5.10 e na figura 5.8. O comportamento de todas as etapas, com exceção da verificação lógica, se mostra como aproximadamente linear. O processo de simulação, mesmo tendo como maior custo o uso externo do QCADesigner, não influencia muito por conta de se realizar em pequenos blocos de tamanhos parecidos. Apenas o número de simulações feitas se altera, causando comportamento próximo ao linear. A verificação lógica, no entanto, apresenta comportamento exponencial devido à sua necessidade de lidar com cada possível combinação de estados de cada uma das células do circuito a cada passo de tempo.

Tamanho	4	8	16	32	64	128	256
Particionamento	96ms	104ms	140ms	246ms	550ms	1029ms	1915ms
Verificação Interna	760ms	1172ms	2419ms	4711ms	8842ms	16242ms	31840ms
Simulação Paralela	698ms	1071ms	2109ms	4374ms	9273ms	18474ms	39324ms
Verificação Lógica	29ms	71ms	91ms	307ms	1285ms	4089ms	108742ms

Tabela 5.10. Tempo de execução das etapas em fios multidimensionais de diferentes tamanhos

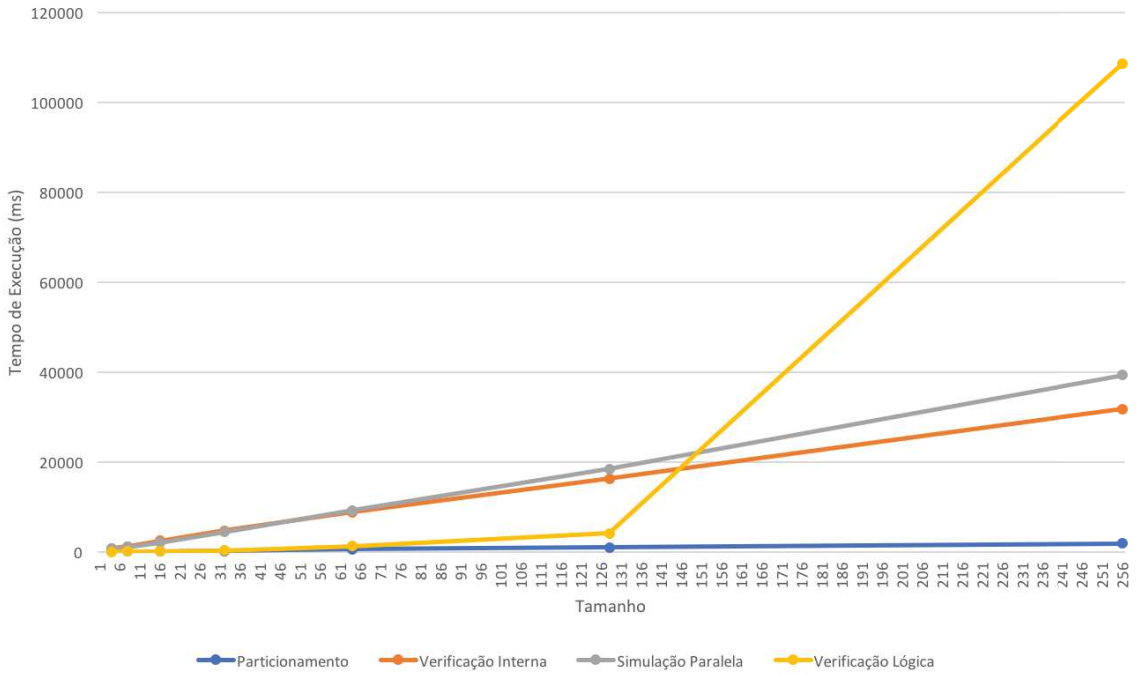

Figura 5.8. Gráfico de análise de desempenho em fios multidimensionais de diferentes tamanhos

5.4.1 Comparação com QCADesigner

Os mesmos circuitos testados anteriormente foram também submetidos ao processo de simulação completa pelo QCADesigner. A tabela 5.11 e o gráfico da figura 5.9 mostram a comparação do desempenho para estes casos. As condições de execução dos testes sempre ocorreram nas mesmas condições (utilizando o mesmo computador descrito na seção 5.1). O tempo referencial da simulação paralela também leva em conta o processo de particionamento.

Tamanho	4	8	16	32	64	128	256
Simulação Paralela	796ms	1174ms	2249ms	4621ms	9822ms	19504ms	41240ms
QCADesigner	3072ms	6145ms	12420ms	27102ms	57401ms	122350ms	272560ms

Tabela 5.11. Análise de desempenho em fios multidimensionais de diferentes tamanhos

O gasto de tempo, nestes casos, é aproximadamente 6 (seis) vezes menor que o do QCADesigner. O desempenho superior ao encontrada em circuitos reais se deve ao fato de que todas as partições só possuírem uma entrada e uma saída, tornando o processo saltos de tabela verdade muito simples e eficaz.

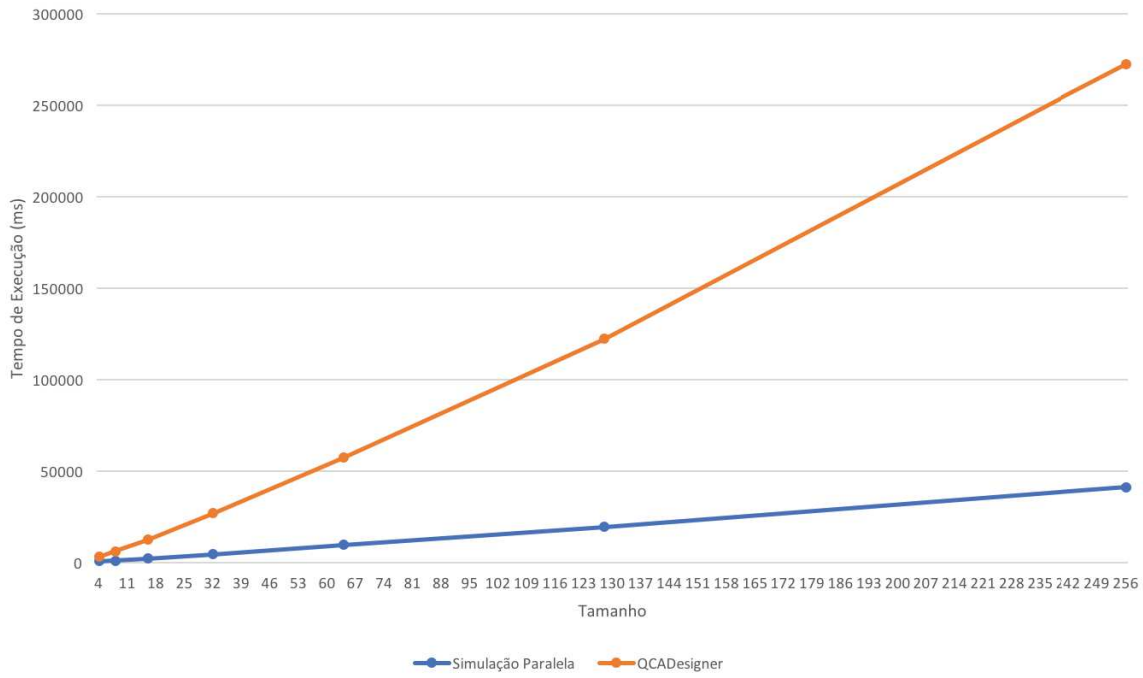

Figura 5.9. Gráfico de comparação de desempenho do processo de simulação em fios multidimensionais de diferentes tamanhos

5.4.2 Efeito da quantidade de unidades de processamento

Devido ao fato de que o algoritmo de simulação é sensível de forma não linear à quantidade de células, foram feitas simulações variando-se a quantidade de unidades de processamento disponíveis.

Para realizar tais experimentos, devido à grande dificuldade de diretamente limitar o número de unidades de processamento no sistema operacional macOS, fora utilizado o sistema de máquinas virtuais VMWare Fusion 10 (VMware, Inc [2017]). A máquina virtual fora criada com exatamente as mesmas especificações da máquina principal que a executa (ou seja, com o mesmo sistema operacional e sem outras limitações que não sejam no número de unidades de processamento).

A tabela 5.12 e o gráfico da figura 5.10 demonstram o tempo de execução para cada um dos casos analisados. O tempo apresentado leva em consideração todo o processo necessário para a simulação completa dos circuitos citados, incluindo o tempo de particionamento.

De forma clara, com esses dados, nota-se que o processo de simulação proposto não é completamente paralelízavel. Isso é esperado, já que o processo de particionamento e de coleta de resultados são sequenciais. De qualquer forma, o ganho temporal é notável em qualquer situação, sendo considerável em circuitos de calibre maior.

Tamanho	4	8	16	32	64	128	256
1 (uma) unidade de processamento	3481ms	6843ms	15376ms	27415ms	54289ms	119744ms	243356ms
2 (duas) unidades de processamento	1913ms	4006ms	8281ms	15167ms	29975ms	66203ms	126622ms
4 (quatro) unidades de processamento	1284ms	2509ms	4982ms	9693ms	20060ms	39985ms	81118ms
8 (oito) unidades de processamento	1270ms	1724ms	3427ms	6626ms	13462ms	27553ms	56035ms

Tabela 5.12. Análise de desempenho em fios multidimensionais utilizando diferentes quantidades de unidades de processamento

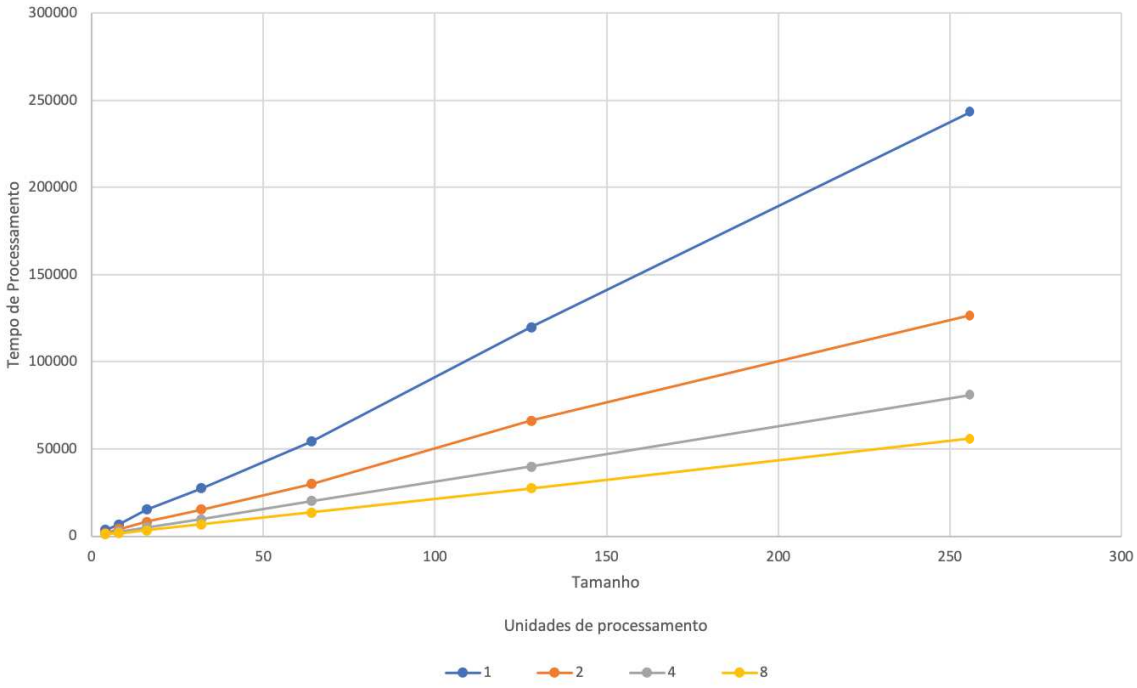

Figura 5.10. Gráfico de comparação de desempenho do processo de simulação variando o tamanho de um fio multidimensional e o número de unidades lógicas de processamento

Ao se comparar os dados obtidos contando com apenas 1 (uma) unidade de processamento com os dados obtidos em uma simulação completa no QCADesigner (conforme tabela 5.11), pode-se concluir que, em um circuito grande (nesse caso, iniciando-se com fios multidimensionais de tamanho 128), há respeitosos ganhos de desempenho mesmo com pouco poder de paralelização. Esse fato corrobora ainda mais com a ideia de que, quando usando o algoritmo do vetor de coerência, é melhor simular vários circuitos pequenos do que um grande.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões

Nesta dissertação, foram descritos métodos capazes de acelerar o processo de criação e validação de circuitos QCA. O passo inicial, necessário para todas as outras etapas, consiste na identificação de regiões independentes do circuito. Células de partições diferentes tem influência desconsiderável umas com as outras.

A fase seguinte, de verificação interna, identifica erros básicos de sincronização e de inversão térmica que podem prejudicar o desenvolvimento de etapas posteriores. Uma única célula incorreta pode comprometer todo o circuito.

A terceira etapa demonstra procedimentos capazes de calcular resultados do circuito de forma completa baseando-se na simulação de cada uma de suas partições menores. Este processo ainda se demonstra ainda mais eficiente em circuitos maiores, com grande incidência de fios.

Os resultados das simulações de cada uma das pequenas fatias do circuito também podem ser utilizados para a construção de um modelo lógico do circuito que proporciona a realização de sua validação lógica. O projetista tem à sua disposição o ferramental necessário para confirmar características específicas do circuito utilizando proposições de lógica temporal.

Em seguida, resultados da aplicação em circuitos reais foram apresentados consolidando a metodologia descrita. A verificação interna de partições executa-se em todo circuito em tempo inferior a 1% do tempo de simulação usando o QCADesigner: uma simples execução deste passo em segundos pode economizar horas em circuitos maiores. O processo de simulação paralela apresentou ganhos aproximados de 5 (cinco) vezes nos circuitos testados ainda garantindo a coesão dos resultados. Apresentou-se também que o uso dessas técnicas leva a ganhos mesmo quando executadas em um

ambiente que force a execução sequencial (com uma única unidade de processamento). Por fim, a verificação lógica, mesmo demorada em circuitos maiores, permite a criação de regras formais que irão garantir seu funcionamento em qualquer situação.

Assim, com os excelentes resultados apresentados, espera-se que todos os métodos aqui descritos consigam auxiliar na prosperidade do paradigma QCA. A maturidade de tecnologias avança em conjunto com a qualidade e eficiência do ferramental disponível para ela.

6.2 Trabalhos futuros

Os bons resultados em QCA também podem ser reproduzidos em quaisquer outras tecnologias baseadas também em acoplamento de campo, tais como a NML (Bernstein et al. [2005]). Mesmo com a imaturidade de todas, elas compartilham características de funcionamento que permitem a aplicação quase completa das técnicas inovadoras apresentadas.

No geral, circuitos eletrônicos são modulares, ou seja, compostos de diversas réplicas de um mesmo bloco. A identificação desses conjuntos pode permitir diminuir ainda mais o número de simulações físicas, permitindo a obtenção ainda mais rápida de resultados.

As tabelas verdade de cada uma das partições permitem também o cálculo de perda de energia do circuito, aumentando as possibilidades para a computação reversível e de baixíssimo consumo de energia.

Reconhecendo a possibilidade de integração dessas técnicas em outras ferramentas, o código-fonte das técnicas apresentadas está disponível de forma aberta e gratuita em `https://github.com/amagus/QCAFast`. Assim, espera-se permitir ainda mais o florescimento do promissor paradigma QCA.

Referências Bibliográficas

Bernstein, G. H.; Imre, A.; Metlushko, V.; Orlov, A.; Zhou, L.; Ji, L.; Csaba, G. & Porod, W. (2005). Magnetic qca systems. *Microelectronics Journal*, 36(7):619--624.

Cadence Design Systems (1999). JasperGold. `https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html`.

Cimatti, A.; Clarke, E.; Giunchiglia, F. & Roveri, M. (1999). Nusmv: A new symbolic model verifier. Em *International conference on computer aided verification*, pp. 495--499. Springer.

Emerson, E. A. & Halpern, J. Y. (1986). "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM (JACM)*, 33(1):151--178.

Henderson, S. C.; Johnson, E. W.; Janulis, J. R. & Tougaw, P. D. (2004). Incorporating standard cmos design process methodologies into the qca logic design process. *Nanotechnology, IEEE Transactions on*, 3(1):2--9.

Karim, F. & Walus, K. (2014). Efficient simulation of correlated dynamics in quantum-dot cellular automata (qca). *Nanotechnology, IEEE Transactions on*, 13(2):294--307.

Lent, C. S. & Tougaw, P. D. (1997). A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85(4):541--557.

Lent, C. S.; Tougaw, P. D.; Porod, W. & Bernstein, G. H. (1993). Quantum cellular automata. *Nanotechnology*, 4(1):49.

Li, Z.; Beatty, A. M. & Fehlner, T. P. (2003). Molecular qca cells. 1. structure and functionalization of an unsymmetrical dinuclear mixed-valence complex for surface binding. *Inorganic chemistry*, 42(18):5707--5714.

Lieberman, M.; Fehlner, T.; Niemier, M. & Kogge, P. (2013). Quantum-dot cellular automata. *Electron Transport in Quantum Dots*, p. 397.

McMillan, K. L. (1993). Symbolic model checking. Em *Symbolic Model Checking*, pp. 25--60. Springer.

Moore, G. E. et al. (1965). Cramming more components onto integrated circuits.

Niemier, M. T. (2000). *Designing digital systems in quantum cellular automata*. Tese de doutorado, University of Notre Dame.

Olson, A. (2011). Towards fpga hardware in the loop for qca simulation.

Orlov, A.; Amlani, I.; Bernstein, G.; Lent, C. & Snider, G. (1997). Realization of a functional cell for quantum-dot cellular automata. *Science*, 277(5328):928--930.

Ottavi, M.; Schiano, L.; Lombardi, F. & Tougaw, D. (2006). Hdlq: a hdl environment for qca design. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2(4):243--261.

Pnueli, A. (1977). The temporal logic of programs. Em *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pp. 46--57. IEEE.

Ribeiro, M. A.; Chaves, J. F. & Vilela Neto, O. P. (2016). Field-coupled nanocomputing energy-information analysis tool. *Sforum, Microelectronics Students Forum*.

Sardinha, L. H. B.; Costa, A. M. M.; Vilela Neto, O. P.; Vieira, L. F. M. & Vieira, M. A. M. (2013). Nanorouter: A quantum-dot cellular automata design. *IEEE Journal on Selected Areas on Communication*, 31(12):825--834.

Shalf, J. M. & Leland, R. (2015). Computing beyond moore's law. *Computer*, 48(12):14--23.

Silva, D. S.; Sardinha, L. H.; Vieira, M. A.; Vieira, L. F. & Vilela Neto, O. P. (2015). Robust serial nanocommunication with qca. *Nanotechnology, IEEE Transactions on*, 14(3):464--472.

Snider, G.; Orlov, A.; Amlani, I.; Zuo, X.; Bernstein, G.; Lent, C.; Merz, J. & Porod, W. (1999). Quantum-dot cellular automata: Review and recent experiments. *Journal of Applied Physics*, 85(8):4283--4285.

Synopsys (2015). VCFormal. `https://www.synopsys.com/verification/static-and-formal-verification/vc-formal.html`.

Vilela Neto, O. P.; Barbosa, C. R. et al. (2007). Neural network simulation and evolutionary synthesis of qca circuits. *Computers, IEEE Transactions on*, 56(2):191--201.

VMware, Inc (2017). VMWare Fusion 10. `https://www.vmware.com/fr/products/fusion.html`.

Walus, K.; Dysart, T. J.; Jullien, G. A. & Budiman, R. A. (2004). Qcadesigner: A rapid design and simulation tool for quantum-dot cellular automata. *Nanotechnology, IEEE Transactions on*, 3(1):26--31.