UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Tese

Uma Metodologia de Síntese Automática com o Foco na Tecnologia Quantum-Cellular-Automata

Julio Saraçol Domingues Júnior

Pelotas, 2020

Julio Saraçol Domingues Júnior

Uma Metodologia de Síntese Automática com o Foco na Tecnologia Quantum-Cellular-Automata

Tese apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Prof. Dr. Felipe de Souza Marques
Coorientador: Prof. Dr. Leomar Soares Da Rosa Júnior

Pelotas, 2020

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

J11m Domingues Júnior, Julio Saraçol

Uma metodologia de síntese automática com o foco na tecnologia quantum-cellular-automata / Julio Saraçol Domingues Júnior ; Felipe de Souza Marques, orientador ; Leomar Soares da Rosa Junior, coorientador. — Pelotas, 2020.

154 f. : il.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2020.

1. Síntese lógica. 2. Tecnologias emergentes. 3. Quantum-cellular-automata. 4. Ferramentas de automação de projeto eletrônico. I. Marques, Felipe de Souza, orient. II. Rosa Junior, Leomar Soares da, coorient. III. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Julio Saraçol Domingues Júnior

Uma Metodologia de Síntese Automática com o Foco na Tecnologia Quantum-Cellular-Automata

Tese aprovada, como requisito parcial, para obtenção do grau de Doutor em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 14 de Setembro de 2020

Banca Examinadora:

Prof. Dr. Felipe de Souza Marques (orientador).
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Rafael Iankowski Soares.
Doutor em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul.

Prof. Dr. José Augusto Miranda Nacif.
Doutor em Ciência da Computação pela Universidade Federal do Minas Gerais.

Prof. Dr. Paulo Francisco Butzen.
Doutor em Microeletrônica pela Universidade Federal de Rio Grande do Sul.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a meus pais Julio e Vera pela educação e os valores morais que me ensinaram e pelos exemplos de pessoas que são. Da mesma forma, gostaria de estender os agradecimentos a minha madrasta, Iolanda e a minha irmã Stéfani que sempre me apoiaram nas minhas escolhas. Dedico essa conquista aos meus avós Jorge, Luci, Orozimbo (in memoriam) e Geni (in memorian).

Em especial, gostaria de agradecer a minha parceira de vida e noiva, Alice Fonseca Finger. Muito obrigado por me apoiar em todos os momentos, além de ser a minha companheira na infinita quilometragem da ponte Alegrete/Bagé/Pelotas. Sou muito feliz ao teu lado e, tu és uma parte essencial dessa conquista, TE AMO. Também quero agradecer aos meus sogros Nóris e Francisco, por sempre me abrigarem durante os inúmeros "bate-volta"que fiz nesses quase cinco anos. Obrigado por me proporcionarem esse apoio fundamental neste momento.

A Universidade Federal do Pampa, pela oportunidade de poder contribuir com a formação de futuros engenheiros de computação, tenho muito orgulho de colaborar com a expansão ao acesso ao ensino superior público na região da campanha. Aos colegas do curso de Engenharia de Computação do campus Bagé, por todo o suporte nos momentos que precisei, em especial, aos colegas de sala Leonardo Pinho, Érico Amaral e Sandro Camargo, por sempre me apoiarem e sempre estarem dispostos a ouvir e colaborar na solução de algum problema. Ao analista de tecnologia da informação da computação Leandro Béria, por todo o pronto atendimento comigo toda vez que tive alguma dificuldade com os servidores dos experimentos, mesmo em finais de semana.

A Universidade Federal de Pelotas, a qual abriu as portas para que eu tivesse a oportunidade de ser bacharel, mestre, professor substituto, e atualmente, aluno de doutorado. Tenho muito orgulho de ter construído minha vida acadêmica e profissional nesta instituição. Em especial gostaria de agradecer aos colegas Renato Souza e Maicon Cardozo que tiveram participação direta ou indireta no meu trabalho. Sem as conversas e discussões com vocês certamente a execução desse trabalho teria sido mais difícil ainda.

Por fim, gostaria de agradecer ao meu orientador Felipe e co-orientador Leomar, por toda a paciência que tiveram nesses quase 5 anos de doutorado. Eu agradeço pelos exemplos de orientadores e profissionais que vocês sempre foram comigo. Espero um dia retribuir aos meus alunos um pouco do que vocês fizeram por mim todos esses anos. Em especial agradeço ao meu orientador Felipe, obrigado por ser o grande colaborador na concepção deste trabalho, e por ser sempre essa referência técnica que me auxíliou nos momentos mais incertos. Concluindo, agradeço a todos que de alguma forma ou de outra me ajudaram a executar esse trabalho. Thanks!!!

RESUMO

DOMINGUES JÚNIOR, Julio Saraçol. **Uma Metodologia de Síntese Automática com o Foco na Tecnologia Quantum-Cellular-Automata**. Orientador: Felipe de Souza Marques. 2020. 154 f. Tese (Doutorado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2020.

A tecnologia *Complementary Metal-Oxide-Semiconductor* (CMOS) é a mais utilizada no projeto de circuitos integrados nas últimas décadas. Entretanto, esta tecnologia está atingindo seus limites físicos. Dessa forma, muitos novos dispositivos estão sendo propostos para a substituição dos transistores, como a tecnologia *Field-Coupled Nanocomputing* (FCN). Muitos dos paradigmas FCN estão sendo estudados, incluindo as abordagens *Quantum-dot Cellular Automata* (QCA) e *Nanomagnetic Logic* (NML). O presente trabalho propõe uma nova metodologia de síntese a tecnologia QCA considerando diferentes ferramentas tanto para síntese lógica, quanto para síntese física, preenchendo uma lacuna no projeto de circuitos QCA, propondo um novo método e integrando novos elementos de *design*. Foi proposta uma nova versão do método estado-da-arte de síntese automática QCA, o qual foi denominado *Migortho*. Além disso, foi adotada uma nova estrutura de dados a ser utilizada, neste caso a *Majority Inverter Graph* (MIG), além de novos elementos de *design*, como por exemplo: *MajX*, *Doble Wire*, *Double Wire Lent*. Adicionalmente, foram implementados uma série de comandos que invocam métodos de otimização de MIG através da biblioteca *EPFL Logic Libraries* denominada *Mockturtle*. Em suma, foi possível avaliar o impacto dos novos elementos propostos nesse trabalho, assim como a nova estrutura de dados para representação da lógica. Da mesma forma, foi possível identificar qual comando implementado pode impactar na maior redução dos circuitos de entrada, atingindo uma taxa média de otimização de $15,8\%$. Além disso, vários circuitos foram avaliados através da síntese pelo método *Migortho*, o qual de forma geral apresentou ganhos em relação ao resultado da versão do método *Ortho*. Ademais, duas versões dos *benchmarks* utilizados foram propostos para futuras avaliações de síntese QCA. Por fim, foram efetuadas avaliações com o *benchmark* específico estado-da-arte para a estrutura MIG, o qual foi possível obter tanto a estimativa do leiaute QCA final, quanto aplicar otimizações através dos comandos *Mockturtle*, otimizando os circuitos descritos pelo método estado-da-arte.

Palavras-chave: Síntese Lógica. Tecnologias Emergentes. Quantum-Cellular-Automata. Ferramentas de Automação de Projeto Eletrônico.

ABSTRACT

DOMINGUES JÚNIOR, Julio Saraçol. **An Automatic Synthesis Methodology with a Focus on Quantum-Cellular-Automata Technology**. Advisor: Felipe de Souza Marques. 2020. 154 f. Thesis (Doctorate in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2020.

The complementary metal-oxide-semiconductor (CMOS) has been used as the main technology to design integrated circuits in the last decades. However, this technology is near to its physical limits. This way, several new devices have been proposed for the replacement of the transistors, such as the technology Field-Coupled Nanocomputing (FCN). Many FCN paradigms are being studied, including quantum-dot cellular automata (QCA) and nanomagnetic logic (NML). This work proposes a new methodology for QCA synthesis, which attaches different logic and physical synthesis tools, filling gaps, and proposing new methods and design elements for integration. A new version of the state-of-the-art QCA automatic synthesis method has been proposed, which has been called *Migortho*. In addition, a new data structure was adopted to be used, in this case, the *Majority Inverter Graph* (MIG). In the same way, new design elements were proposed, such as *MajX*, *Double Wire*, and *Double Wire Lent*. Furthermore, a series of commands were implemented that invoke MIG optimization methods through the *EPFL Logic Libraries* library called *Mockturtle*. In short, it was possible to evaluate the impact of the new elements proposed in this work, as well as the new data structure for logic representation. Likewise, it was possible to identify which command implemented could impact the greatest reduction in input circuits, reaching an average optimization rate of $15,8\%$. Besides, several circuits were evaluated through synthesis by the *Migortho* method, which generally presented gains concerning the result of the version of the *Ortho* method. In addition, two versions of the used benchmarks were proposed for further evaluations of the QCA synthesis. Finally, evaluations were made with the *benchmark* specific state-of-the-art for the MIG structure, which made it possible to obtain both the estimate of the final QCA layout and to apply optimizations through the *Mockturtle* commands, that it optimized the circuits described by the state-of-the-art method.

Keywords: Logic Synthesis. Emergent Technologies. Quantum-Cellular-Automata. Electronic Design Automation Tools.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CMOS	*Complementary Metal-Oxide-Semiconductor*
FCN	*Field-Coupled Nanocomputing*
QCA	*Quantum-dot Cellular Automata*
NML	*Nanomagnetic Logic*
EDA	*Electronic Design Automation*
BDD	Diagramas de Decisão Binária
AIG	*And Inverter Graph*
MIG	*Majority Inverter Graph*
OGD	*Orthogonal Graph Drawing*
FPGA	*Fields Programmable Gate Arrays*
RTL	*Register Transfer Level*
DAG	*Directed Graph Acyclic*
SOP	*Sum of Products*
AOIG	*And-Or-Inverter Graph*
VLSI	*Very Large Scale Integration*
EPFL	*École Polytechnique Fédérale de Lausanne*
HDL	*Hardware Description Language*
AOI	*And-Or-Inverter*
BDS	*Binary Diagram System*
ASIC	*Application Specific Integrated Circuits*
LUT	*Look-Up Table*
TLG	*Threshold Logic Gates*
TLF	*Threshold Logic Functions*
USE	*Universal, Scalable and Efficient Clocking*
SAT	*Satisfiability*
SMT	*Satisfiability Modulo Theories*

SUMÁRIO

1 INTRODUÇÃO

Muitos dos avanços da indústria de semicondutores se devem ao aprimoramento das técnicas e dos processos de construção de circuitos digitais. Neste sentido, muitos desses avanços estão diretamente relacionados a capacidade de miniaturização dos elementos, o que possibilita que mais elementos sejam inseridos em uma mesma área de Silício. Dessa forma, por muitos anos essa técnica permitiu a diminuição do consumo de energia por operação.

Contudo, esse avanço na escala de miniaturização em tecnologias nanométricas também acarretou no surgimento de alguns problemas intrínsecos, os quais não eram considerados em tecnologias micrômicas, como por exemplo, as limitações causadas no processo de litografia. Da mesma forma, outro problema intrínseco são as limitações atreladas ao consumo de energia estático em células lógicas.

Essas novas limitações são problemas gerados pelo alcance dos limites físicos da tecnologia *Complementary metal-oxide-semiconductor* (CMOS). Neste sentido, embora ainda existam esforços para manter a utilização da tecnologia CMOS, como por exemplo abordagens baseadas em transistores mais complexos como os da tecnologia *FinFET*, existem diversos estudos para conceber tecnologias que venham a substituir a tecnologia CMOS. Todas essas novas tecnologias estão preocupadas, principalmente, com a redução do consumo de energia. Essa preocupação se torna ainda mais evidente quando se trata do projeto de sistemas embarcados (MARWEDEL, 2010) (WOLF, 2012).

Nesse sentido, é importante destacar alguns aspectos que ainda são limitantes para a consolidação dessas novas tecnologias. Dentre eles, cabe ressaltar que todas as ferramentas de síntese tradicionais, consideram características da tecnologia CMOS para a realização da síntese dos circuitos. Assim, diversos trabalhos tentam realizar adaptações dos fluxos tradicionais para considerarem especificidades das novas tecnologias, como por exemplo (ZHANG et al., 2004) (SHANG, 2010) e (MOMENZADEH et al., 2005). Dessa forma, assim como são consideradas novas tecnologias para substituir a tecnologia CMOS, também existe a necessidade da concepção de novas técnicas e ferramentas de síntese para lidar tanto com as novas limitações da

tecnologia CMOS, quanto com as características das novas tecnologias.

A evolução da engenharia de materiais cria novas possibilidades, e algumas dessas tecnologias estão sendo estudadas nas últimas décadas, como a *Field-Coupled Nanocomputing* (FCN). Nos circuitos FCN, todos os operadores lógicos são baseados em interações locais de campo entre blocos construídos em nanoescala, os quais são organizados no formato de vetor (ANDERSON; BHANJA, 2014) (Formigoni; Vilela Neto; Nacif, 2018). Os principais paradigmas dentro da FCN são as tecnologias: *Quantum-dot Cellular Automata* (QCA) e a tecnologia *Nanomagnetic Logic* (NML), onde uma característica interessante é que a maioria dessas tecnologias possui como elemento básico a ideia da abstração de um votador booleano, ou porta majoritária como elemento básico para a criação da lógica. Embora essas tecnologias tenham demonstrado bons resultados em relação a alto desempenho e baixo consumo de energia, ainda estão sendo desenvolvidas, ou seja, não possuem maturidade em seus processos de manufatura e síntese. Desta forma, essas limitações se tornam os novos desafios para a área de processos de fabricação e a área de ferramentas para automação de projetos eletrônicos (*Electronic Design Automation* - EDA), na execução tanto da etapa de síntese lógica, quanto na etapa de síntese física.

1.1 Motivação

Considerando o exposto anterior, existem diversos trabalhos que apresentam métodos de síntese de circuitos para o paradigma FCN, como por exemplo (SHANG, 2010)(MOMENZADEH et al., 2005) e (REIS et al., 2016). Contudo, em sua maioria são propostas com abordagens que possuem etapas executadas manualmente. Isso se deve a alta complexidade que envolve a proposição de algoritmos de síntese, e as regras de projeto dessas novas tecnologias. Entretanto, para lidar com essas restrições e, com projetos de grande escala em um cenário mais realístico, são necessárias as ferramentas de EDA. Nesse sentido, os trabalhos de Teodósio (TEODOSIO; SOUSA, 2007), Fontes (FONTES et al., 2018) e Walter (WALTER et al., 2019), apresentam abordagens de síntese automática para circuitos FCN, especialmente para as tecnologias NML e QCA.

Portanto, considerando o estado da arte, existem trabalhos que apresentam abordagens focadas nas diferentes etapas do projeto de construção de circuitos digitais, em especial, utilizando o paradigma FCN. Assim, por exemplo na etapa de síntese lógica, pode-se citar os trabalhos baseados na investigação da melhor estrutura de dados para a representação da lógica nessas tecnologias, a quais utilizam arranjos de células majoritárias.

Esse tema se torna de alta importância, visto que a estrutura de dados pode impactar diretamente no custo computacional da heurística na busca por soluções. Neste caso, o foco está na porta majoritária, a qual é o elemento básico para a construção de lógica dessas tecnologias. O diferencial deste tipo de arranjo, é a capacidade de representar tanto o comportamento de uma porta lógica primitiva and, quanto o comportamento lógico de uma porta primitiva or. Atualmente, um dos problemas já identificados é que as estruturas de dados tradicionais, como por exemplo os Diagramas de Decisão Binária (BDD) (AKERS, 1978) e Grafos de portas *And* e Inversores (*And Inverter Graphs* - AIG) (AIGER, 2012), não representam de forma eficiente arranjos majoritários.

Contudo, os trabalhos propostos por Amarú, em: (AMARÚ; GAILLARDON; MICHELI, 2013a), (AMARÚ; GAILLARDON; MICHELI, 2013b) (AMARÚ; GAILLARDON; MICHELI, 2014a) (AMARÚ; GAILLARDON; MICHELI, 2014b) (AMARÚ; GAILLARDON; MICHELI, 2014c) (AMARÚ; GAILLARDON; MICHELI, 2015) (AMARÚ; GAILLARDON; MICHELI, 2016) apresentam novas estruturas de dados para a representação de lógica baseada em portas majoritárias. Essas novas estruturas permitiram o desenvolvimento de novas possibilidades de representação da lógica baseada em majoritária, além de viabilizarem novas abordagens de otimização das representações, o que pode resultar em novas técnicas para otimização de circuitos.

Do ponto de vista de síntese de circuitos, esses trabalhos consideram a primeira etapa, onde o foco está em, a partir de uma descrição de alto nível, encontrar uma configuração otimizada da estrutura de dados para representar a lógica do circuito em questão. Neste caso, representando a lógica do circuito em lógica majoritária por exemplo. Essas abordagens apresentam resultados promissores para a representação da lógica através de uma das estruturas de dados propostas, denominada *Majority Inverter Graph* (MIG). Além disso, os algoritmos de otimização baseados no MIG, propostos por Amarú apresentam ótimos resultados, sendo aplicados tanto para a síntese para tecnologia CMOS, quanto para QCA.

Em contrapartida, diversos trabalhos são desenvolvidos considerando outras etapas da síntese, onde a partir de uma descrição lógica, busca-se, por exemplo, como conectar elementos de uma dada tecnologia para representar a lógica em questão. Neste sentido, essas propostas exploram diretamente a questão de posionamento e roteamento (*place and route*) de circuitos QCA a partir de uma descrição de alto nível. Considerando esse modelo de síntese automática é possível citar os trabalhos de Trindade (TRINDADE et al., 2016), Fontes (FONTES et al., 2018) e Walter (WALTER et al., 2019). Walter (WALTER et al., 2019) destaca-se, pois propõe uma ferramenta com o foco na síntese automática de circuitos FCN, tanto para a tecnologia QCA, quanto para NML. A ferramenta é denominada Fiction e têm como foco as etapas de posionamento e roteamento, considerando diferentes esquemas de *clock*, utilizando

duas abordagens distintas para gerar os leiautes finais. A primeira baseada em um método que utiliza SMT *solver* (*Satisfiability Modulo Theories*) para gerar a solução final (Walter et al., 2018), e a segunda baseada no Algoritmo de Biedl (Walter et al., 2019) utilizando a estrutura de dados de grafos ortogonais (*Orthogonal Graph Drawing* - OGD)(BIEDL, 1998). Entretanto, embora as abordagens da ferramenta Fiction sejam o estado da arte, quando se refere a síntese para circuitos FCN, ela apenas trata das etapas finais do projeto de circuitos FCN, ou seja, não existe nenhuma preocupação com a etapa de síntese lógica.

Considerando este panorama, apesar de existirem várias abordagens tanto nas etapas de descrição de circuitos, quanto nas etapas de roteamento e posicionamento, ainda não foi proposta uma metodologia de síntese para circuitos QCA, que contemple um fluxo com todas as etapas bem definidas como já acontece no caso da tecnologia CMOS. Por exemplo, o que acontece na metodologia de células padrão (*Standard Cell*), iniciando desde a descrição de alto nível e, logo após, aplicando otimizações de síntese lógica e, por fim, gerando a descrição do leiaute final do circuito através das etapas da síntese física. Neste sentido, este trabalho propõe contribuir com a proposição de uma metodologia de síntese com o foco na tecnologia QCA.

1.2 Problema de Pesquisa

É possível contribuir no desenvolvimento do processo de síntese automática de circuitos para a tecnologia QCA, com a proposição de uma metodologia de síntese?

Considerando o exposto anterior, onde foi definido o problema de pesquisa, também foi possível delinear o objetivo geral do trabalho, assim como foram traçados alguns objetivos específicos a serem atingidos, os quais são apresentados abaixo.

1.2.1 Objetivo Geral

Objetivo Geral: Propor uma nova metodologia de síntese automática de circuitos focada na tecnologia Quantum-Cellular-Automata.

1.2.2 Objetivos Específicos

Neste sentido, a partir do objetivo geral, foram definidos alguns objetivos específicos, os quais são elencados abaixo:

- Conceber uma metodologia de síntese automática para circuitos na tecnologia QCA com base nos trabalhos propostos na literatura, a luz dos fluxos tradicionais e bem estabelecidos;
- Prototipar a metodologia de síntese automática utilizando a estrutura de dados MIG e seus métodos de otimização com foco em majoritárias;

- Explorar a capacidade de otimização dos métodos propostos por Amarú no processo de síntese utilizando a estrutura de dados MIG, além do impacto dessa estrutura no *place and route* de circuitos nas tecnologia QCA;
- Análisar quais são os melhores métodos a serem aplicados na descrição de MIG, para viabilizar otimizações na geração de leiaute QCA;
- Contribuir com desenvolvimento da especificação de novos elementos de *design* na tecnologia QCA, a fim de viabilizar a implementação de circuitos mais complexos e com maior número de elementos;

Dessa forma, espera-se ao atingir os objetivos específicos e, por consequência, o objetivo geral, contribuindo com a concepção de uma nova metodologia de síntese automática para a tecnologia QCA.

1.3 Contribuições da Tese

De forma mais sintetizada, é possível elencar as principais contribuições deste trabalho, são elas:

- Proposição de uma nova metodologia de síntese automática de circuitos com o foco na tecnologia QCA;
- Implementação de uma nova versão do método estado-da-arte em síntese QCA que utiliza a estrutura e otimizações em MIG, o qual foi denominado como *Migortho*;
- Modelagem e adição de novos elementos de *design*, no contexto da ferramenta Fiction para lidar com restrições impostas na utilização da estrutura de dados MIG juntamente com o método *Ortho* e seu sistema de *clock*;
- Implementação de diversos métodos de síntese lógica MIG, a partir da biblioteca *Mockturtle*, no formato de comandos no ambiente da ferramenta Fiction;
- Experimentação, estudo e análise das diferentes variáveis possíveis de configuração e o impacto dos comandos implementados para síntese QCA;
- Proposição de duas configurações alternativas otimizadas para geração de *benchmarks* para síntese QCA, partir de *benchmarks* conhecidos da comunidade de síntese lógica;
- Nova versão da ferramenta Fiction, denominada Fiction 2.1, que possui todas as modificações descritas acima para síntese QCA através do método *Migortho*, o qual provê resultados melhores que a versão do método *Ortho*;

- A otimização entre 26% até 42% dos leiautes finais, da nova versão do único método escalável disponível na literatura para síntese QCA;

1.4 Organização do Trabalho

O texto a seguir está dividido da seguinte forma: o capítulo dois apresenta a fundamentação teórica sobre os principais conceitos para o melhor entendimento desta tese. No terceiro capítulo é apresentada à discussão sobre os trabalhos correlatos da literatura, relacionados tanto com majoritárias quanto com a utilização da estrutura de dados MIG, além dos principais trabalhos sobre síntese para a tecnologia QCA, os quais colaboraram com a proposta de metodologia desta tese. O quarto capítulo é apresentada a proposta de metodologia de síntese automática para a tecnologia *Quantum-Cellular-Automata*, baseada no fluxo *Standard Cell*. No quinto capítulo serão apresentadas as principais análises sobre a prototipação da metodologia proposta, assim como os resultados e discussões sobre os experimentos realizados utilizando o ambiente da ferramenta Fiction 2.1. Por último, no sexto capítulo, serão apresentadas as conclusões sobre o estudo, assim como as referências bibliográficas.

2 FUNDAMENTAÇÃO TEÓRICA

Considerando que este trabalho apresenta uma proposta de metodologia de síntese automática com foco na tecnologia *Quantum Cellular Automata*, é preciso, primeiramente, apresentar uma revisão bibliográfica dos principais conceitos das tecnologias trabalhadas. Esses conceitos serão utilizados como base de conhecimento para melhor compreensão da abordagem desenvolvida.

2.1 Síntese de Circuitos e a Metodologia *Standard Cell*

A síntese de um circuito tradicionalmente inicia a partir de uma descrição mais genérica e abstrata, sem preocupações com as restrições físicas de manufatura. Em um segundo momento, uma descrição do circuito é gerada, considerando aspectos físicos (geométricos) e restrições da tecnologia alvo. Esse processo de síntese pode ser executado através de diferentes metodologias. Entretanto, de forma geral duas merecem destaque: *custom* e *semicustom*.

Na metodologia *custom* o projetista desenvolve todo o projeto de forma manual sem automatização de etapas. Essa metodologia pode resultar em circuitos otimizados e com grande desempenho quando executada por projetistas experientes. Por outro lado, a complexidade das tarefas em circuitos de larga escala aliado ao tempo de projeto cada vez menor, torna inviável à prática desta metodologia.

Já na metodologia *semicustom* algumas etapas são automatizadas através das ferramentas de EDA, o que possibilita a redução do tempo de projeto. Essa metodologia pode ser executada de diversas formas, porém, existem duas abordagens que se destacam: baseada em matrizes e a baseada em bibliotecas. Nos fluxos baseados em matrizes, geralmente utilizam-se elementos reconfiguráveis, os quais podem representar um comportamento lógico de uma função booleana arbitrária. Como exemplo, pode-se citar os *Fields Programmable Gate Arrays (FPGAs)*.

Por outro lado, o fluxo baseado na metodologia de células padrão, destaca-se na tecnologia CMOS, onde considera-se o uso de uma biblioteca de células. Neste fluxo, o objetivo é encontrar a correspondência de partes do circuito com os elementos da

biblioteca de células. O diferencial desta abordagem, em relação a baseada em matrizes, é a necessidade de um bom algoritmo para encontrar as correspondências possíveis dos elementos da biblioteca com as porções de lógica do circuito. Dessa forma, cabe aos algoritmos de síntese, juntamente com a função custo a ser minimizada, encontrar o melhor conjunto de células para construir o circuito. A Figura 1 apresenta a organização dos fluxos de síntese de um circuito digital.

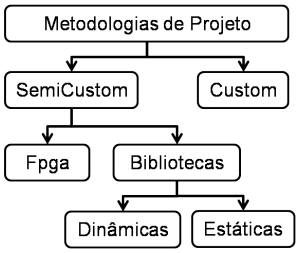


Figura 1 – Metodologias de síntese de circuitos.
Fonte: Próprio Autor.

Usualmente, as metodologias baseadas na abordagem *semicustom* dividem-se em três macro-etapas: síntese de alto-nível, síntese lógica e síntese física. A síntese de alto-nível ou estrutural, têm como objetivo lidar com uma descrição a fim de gerar uma descrição no nível de registradores (*Register Transfer Level (RTL)*). Essa etapa consiste em gerar uma visão do nível arquitetural do circuito (nível macroscópico). Isso corresponde a determinar algumas características e critérios do circuito, como por exemplo, interconexões, atraso e funcionalidades lógicas (DE MICHELI, 1994).

A próxima etapa, denominada síntese lógica, tem a tarefa de transformar essa descrição estrutural em uma descrição de nível lógico (booleano). O objetivo geral é a manipulação de especificações lógicas, a fim de criar uma descrição de elementos lógicos primitivos interligados. Em outras palavras, a síntese lógica determina o nível microscópico do circuito, ou seja, nível de portas lógicas (DE MICHELI, 1994).

Assim, a síntese lógica é aplicada sobre a descrição *RTL*, com o objetivo de traduzir a lógica descrita sobre registradores para uma estrutura de dados. A primeira etapa visa minimizar através de métodos (normalmente iterativos) alguma característica esprecífica, como por exemplo: número de elementos (em geral representam ganho em área), reduzir profundidade lógica (com o objetivo de reduzir atraso) ou remover redundâncias da representação lógica. Essa primeira etapa da síntese lógica é denominada fase independente de tecnologia. Esse pré-processamento é importante, pois pode viabilizar otimizações nas fases posteriores.

Após esta fase independente de tecnologia, se inicia a fase de mapeamento tecnológico, o qual tenta identificar padrões do circuito em questão (porções), que são equivalentes aos elementos de alguma tecnologia alvo, por exemplo, com células de uma biblioteca de células. O mapeamento tecnológico visa gerar um circuito minimizado, o qual pretende atender algum critério específico, como por exemplo: atraso,

consumo de energia ou área. O resultado desta fase é a descrição de uma rede de elementos interconectados da biblioteca de células, a qual é comumente denominada de *Netlist* de portas lógicas (DE MICHELI, 1994).

Esta descrição *Netlist* é utilizada nas etapas de síntese física, a qual é aplicada já considerando características físicas da tecnologia alvo. Dentre as etapas da síntese física pode-se citar, o posicionamento e roteamento. O posicionamento têm como objetivo central definir a localização de cada instância de uma das células do circuito, assim como o roteamento definirá suas interconexões. Ao final do processo de síntese física, obtém-se como o resultado, o projeto de leiaute do circuito. A Figura 2 apresenta o diagrama das etapas de síntese de um circuito conforme a metodologia de células padrão.

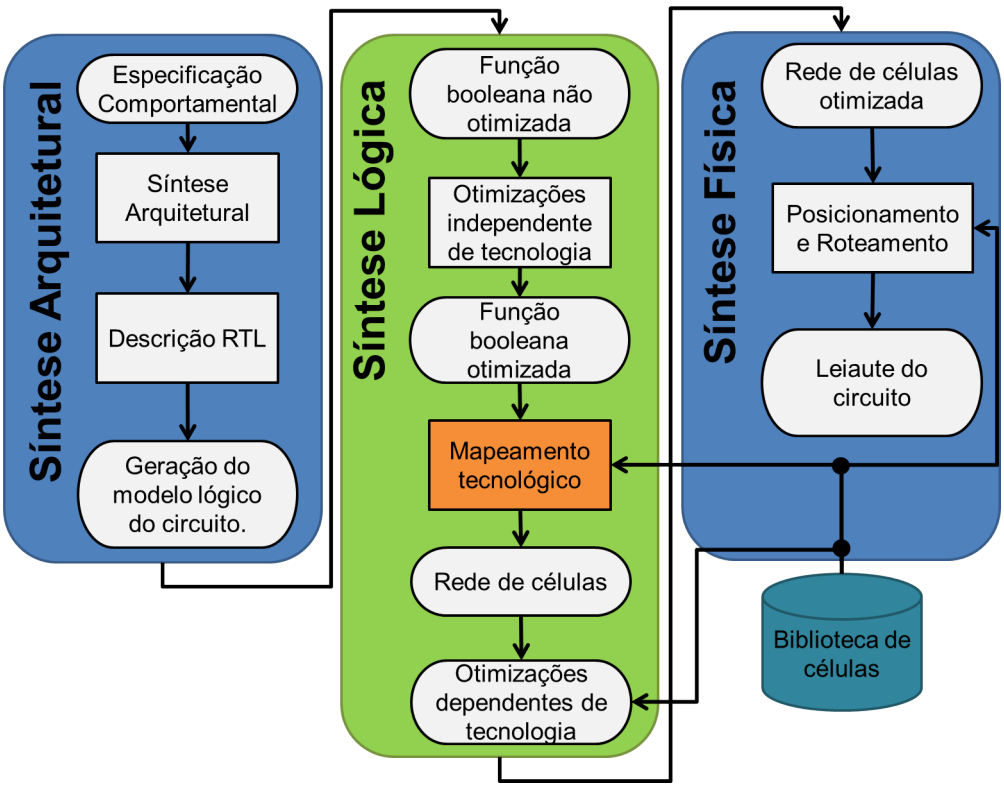


Figura 2 – Metodologia *Standard Cell* para a tecnologia CMOS.
Fonte: Próprio Autor.

2.2 Field-coupled Nanocomputing Circuits

A tecnologia *Field-coupled Nanocomputing* funciona baseada em interações locais entre células. Dessa forma, a informação é propagada através do circuito. As mais populares tecnologias baseadas no paradigma FCN são a NML (ANDERSON; BHANJA, 2014) e a QCA (Lent; Tougaw, 1997).

As células da tecnologia NML utilizam nanomagnetos, os quais são de um domínio único e podem assumir unicamente um dos dois estados possíveis de magnetização, são eles: $M = -1$ e $M = +1$, como apresenta a Figura 3(a), onde são representados o valor binário 0 e o valor binário 1 (Walter et al., 2019).

Por outro lado, as células da tecnologia QCA são compostas por quatro poços quânticos que armazenam uma carga elétrica, e estão dispostos no cantos como um quadrado (ZHANG et al., 2004). Esses elétrons tendem a ocupar lados antipodais como resultado de sua força de repulsão eletrostática mútua. Assim, existem duas configurações de elétrons que são energeticamente equivalentes nas células QCA, como apresenta a Figura 3(b). Essas duas configurações são definidas como a representação da polarização da célula $P = +1$ e $P = -1$, as quais são utilizadas para codificar a representação da lógica para os valores 1 e 0, respectivamente.

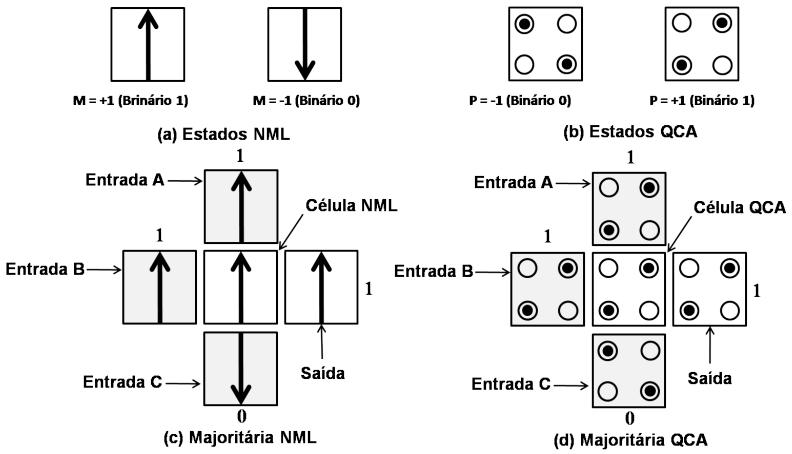


Figura 3 – Elementos essenciais das tecnologias baseadas no paradigma FCN e seus estados.
Fonte: Próprio Autor.

O principal elemento para a representação de funções lógicas no paradigma FCN é a porta majoritária. A porta majoritária implementa uma função lógica de até três entradas. Assumindo que as entradas são A, B, e C, a função lógica da porta majoritária é apresentada na Equação (1).

As Figuras 3(c) e 3(d) apresentam as representações do porta majoritária nas tecnologias NML e QCA, respectivamente. Esse elemento possibilita a implementação de operações lógicas básicas como a operação AND e a operação OR, através da fixação da polarização de uma de suas entradas com a polarização 0 ou a polarização 1, como descreve as equações (2) e (3).

$$M(A, B, C) = (A * B) + (A * C) + (B * C) \tag{1}$$

$$M(A, B, 0) = (A * B) \tag{2}$$

$$M(A, B, 1) = (A + B) \tag{3}$$

Um dos principais pontos tanto da tecnologia QCA, quanto da tecnologia NML, é o esquema de relógio (*clocking scheme*). Sinais de *clock* externos devem ser aplicados para controlar o mecanismo de propagação da informação, que por sua vez, permitem que as mudanças nos níveis de polarização viabilizem a troca da polarização das células de forma adiabática, e nos momentos corretos (REIS et al., 2016). Assim, o *clock* do circuito realiza a função relacionada ao *timing* e ao sincronismo da lógica. Neste sentido, diferentes abordagens para implementação dos esquemas de *clock* tem sido propostos na literatura. O modelo de *clocking scheme* mais utilizado pela literatura considera um conjunto de quatro zonas (estados ou fases) de *clock* na tecnologia QCA e três zonas de *clock* para a tecnologia NML.

De forma geral, um dos principais desafios no projeto de circuitos do paradigma FCN, é que logo após ser decomposta em um conjunto de portas interconectadas (independente da tecnologia, seja QCA ou NML), uma função a ser implementada precisa realizar algumas etapas, como posicionar os elementos (bem como possíveis fios necessários) e, ao mesmo tempo, satisfazer as seguintes restrições de design FCN:

1. **Exclusividade**: Cada porta deve estar localizada em uma única zona *clock* (exceto os fios que podem atravessar outros fios dentro da mesma zona de *clock*, mas não devem atravessar outras portas).

2. **Adjacência**: As portas conectadas devem estar localizadas em zonas de *clock* adjacentes ou interligadas por fios.

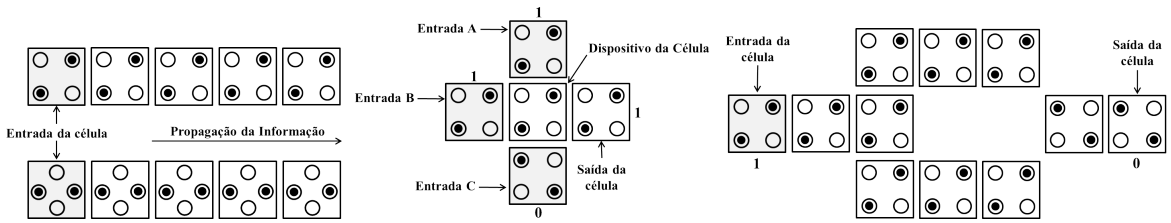
3. **Zonas do *Clock***: a cada zona do relógio (*clock*) deve ser atribuído um identificador entre 1 e c, onde c é o número máximo de *clock*, ou seja, 4 para QCA e 3 para NML, de modo que caminhos através de zonas de *clock* consecutivas sejam possíveis (por exemplo, 1 seja vizinho de 2 ou c seja vizinho de 1, etc.).

Além disso, geralmente, é desejável que o comprimento de todos os caminhos que levam a qualquer porta de entrada múltipla (contabilizando o número de zonas de *clock* ultrapassados) difira em não mais que 3 zonas em QCA ou 2 zonas em NML. Como essa restrição opcional afeta apenas o desempenho (mas não a aplicabilidade geral), ela é considerada uma restrição de *design* opcional da FCN (denominada comprimento do caminho).

2.2.1 Quantum Cellular Automata - QCA

Nesta tese, o estudo foi concentrado sobre a tecnologia QCA, a qual não apenas oferece uma solução em escala nanométrica, mas também um novo método de computação e transformação de informação. Uma característica interessante dessa nova tecnologia é o fato de que ela não trabalha com níveis lógicos baseados em tensão, mas sim baseados no posicionamento de elétrons individuais. Assim, a informação binária é codificada pela configuração de cargas elétricas.

Devido à natureza da interação de *Coulomb* nas células, a corrente não flui entre as células, no *QCA* os elétrons repulsam elétrons das células vizinhas provocando uma pertubação que, por consequência, propagam o nivel lógico de um ponto de entrada para outro de saída. Além disso, estudos indicam que essa tecnologia pode atingir frequência de operação em TeraHertz e, também, devido ao processo de fabricação molecular, é possivel atingir a densidade de 10^{12} dispositivos por cm^2 (MOMENZADEH et al., 2005). Através dos QCAs, esses benefícios podem ser alcançados com uma dissipação de potência ultra baixa em comparação com circuitos *CMOS* convencionais (SHANG, 2010). Quando as células estão dispostas perto uma da outra, elas podem formar estruturas mais complexas, são elas: *Wire*, *Majority Gate* e *Inverter* (WALUS et al., 2004). A Figura 4 apresenta os elementos básicos da tecnologia QCA.



(a) Wire (b) Majority Gate (c) Inverter

Figura 4 – Elementos básicos da tecnologia QCA.
Fonte: Próprio Autor.

O *clocking scheme* mais utilizado pela literatura considera um conjunto de quatro zonas a qualquer momento, são elas: *Switch*, *Hold*, *Release* e *Relax*. Na fase *Switch* as barreiras entre pontos adjacentes são levemente levantadas e, as células são polarizadas de acordo com o estado que estão definidas para assumir. A computação começa durante a fase de *Switch* e, mantém o valor durante a fase *Hold*. Já a liberação da célula é efetuada na fase que o *clock* está em *Release*, ficando inativa durante o estado de *Relax* (SHANG, 2010). Assim, a célula é atualizada a cada ciclo, sendo o caso de ser utilizada como um elemento de memória, um *loop* de células é utilizado com *clock* em série.

Se a informação não estiver sincronizada no circuito, é possível que sejam criados erros no processamento da lógica, os quais implicam na confiabilidade do circuito (*Reliability*) (CAMPOS et al., 2016). Muitos trabalhos sobre *clocking scheme* para a tecnologia QCA foram propostos nos últimos anos como o trabalho de Caio (CAMPOS et al., 2016) e Vankamamidi (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008). A Figura 5 apresenta as quatro fases consideradas no processo de *clock* em um elemento fio (*Wire*).

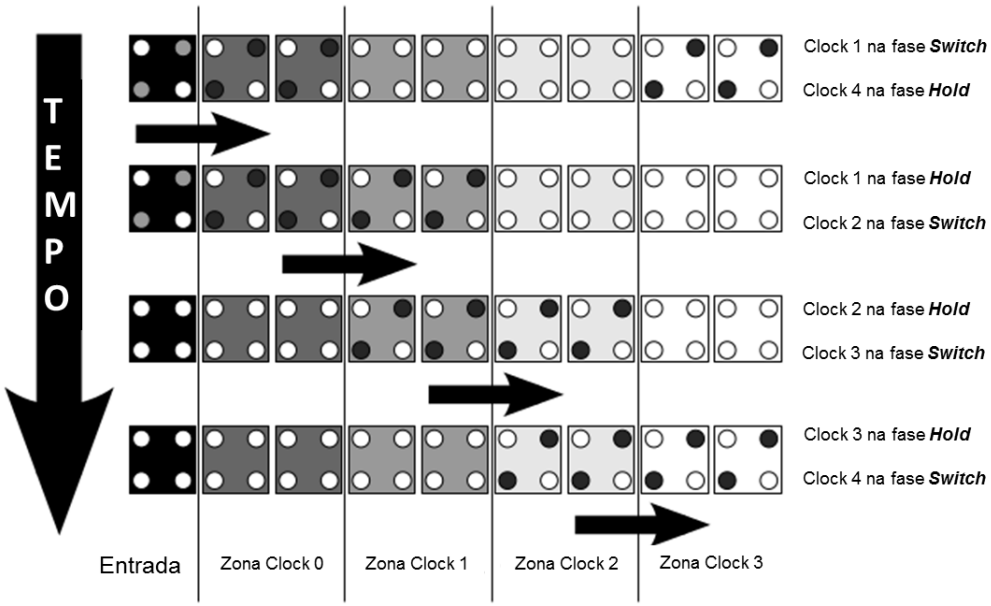


Figura 5 – Fases do processo de *clock* em um fio (*Wire*) em QCA.
Fonte: Próprio Autor, adaptado de (TEIXEIRA, 2016).

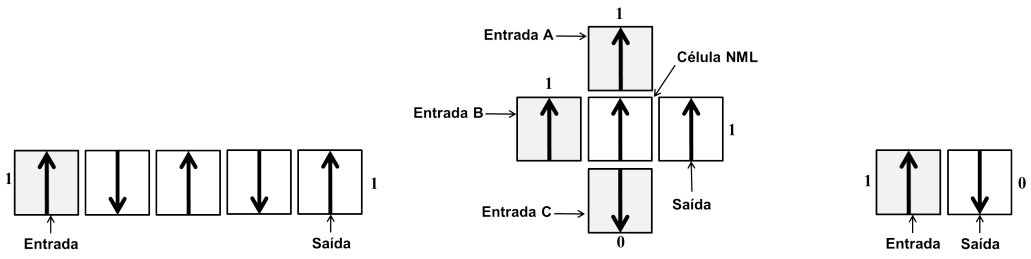
2.2.2 Nanomagnetic Logic - NML

Outra tecnologia que se destaca considerando o paradigma FCN é a tecnologia NML, essa tecnologia utiliza células nanomagnéticas e realizam a propagação da informação e da lógica através do acoplamento de campo magnetostático (*Magnetostatic Field-Coupling*). As células da tecnologia NML, quando não há interferência magnética externa, tendem a ter sua magnetização paralela ao eixo mais longo do retângulo.

Quando a configuração de uma célula NML está com o apontamento para Norte-Sul ou Sul-Norte ela tem a mesma energia desmagnetizante. A tecnologia NML utiliza dois estados associados a representação do 0 lógico e 1 lógico como codificação binária como descreve a Figura 3(a).

Como os circuitos NML são baseados em magnetos, eles têm algumas vantagens específicas sobre outras implementações em QCA. Eles não têm consumo de energia estático e, potencialmente, uma dissipação de potência muito baixa. Os elementos NML têm uma alta resistência ao calor e à radiação, tornando-os ideais para aplicações em ambientes hostis. Finalmente, devido à sua natureza magnética, combinam elementos de lógica e memória no mesmo dispositivo, abrindo possibilidades completamente novas no desenvolvimento de circuitos lógicos. Sua única desvantagem em relação a tecnologia QCA, é sua velocidade de operação relativamente baixa, em torno de 50 a $100\ MHz$.

Outra característica interessante desta tecnologia, é que a transmissão da informação é possível através da interação magnética. Como barras magnéticas, as células magnéticas possuem um campo que é capaz de influenciar as células em seu entorno, essa influência é capaz de realizar a propagação da informação. É importante observar que essa configuração de vizinhança promove células nanomagnéticas paralelas umas às outras para serem influenciadas com polaridade oposta, ou seja, se a primeira célula for polarizada para Norte-Sul, a célula seguinte será polarizada para Sul-Norte, e a próxima para Norte-Sul e, assim por diante. Essa configuração cria automaticamente uma das portas elementares: o inversor (*Inverter*). A Figura 6 apresenta os elementos básicos da tecnologia NML, são eles: *Wire*, *Majority Gate* e *Inverter*.



(a) Wire (b) Majority Gate (c) Inverter

Figura 6 – Elementos básicos da tecnologia NML.
Fonte: Próprio Autor, adaptado de (ANDERSON; BHANJA, 2014).

Assim como na tecnologia QCA, na tecnologia NML foi preciso a definição de um *clocking scheme* para sincronizar o funcionamento de suas células para a construção de circuitos mais complexos.

Entretanto, diferentemente da tecnologia QCA, a tecnologia NML funciona apenas com três zonas de *clock*, com as seguintes fases: *Reset*, *Hold* e *Switch*. A Figura 7 apresenta as três fases consideradas no processo de *clock* em um *Wire*.

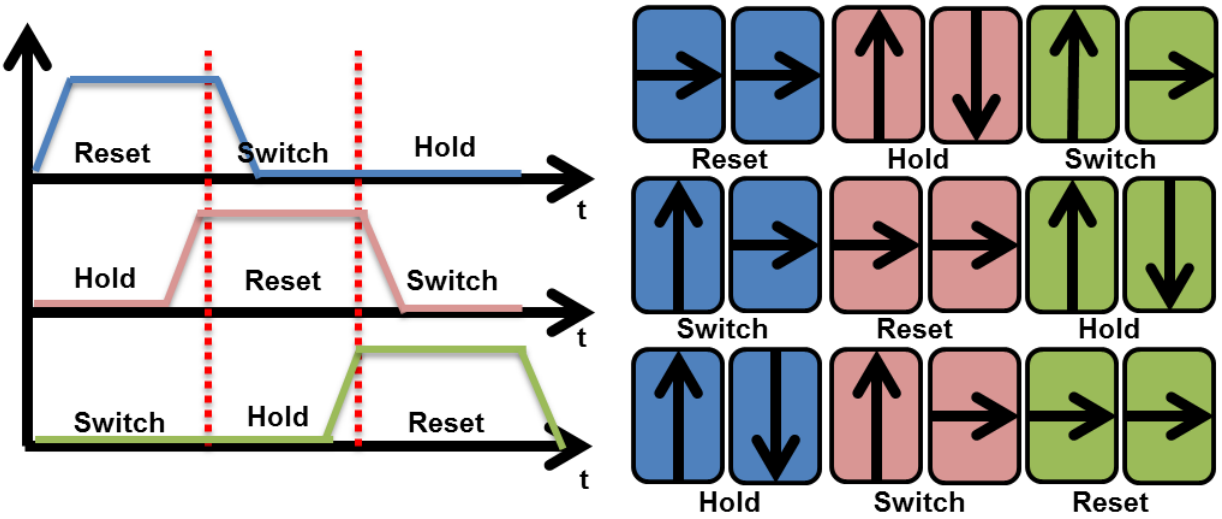


Figura 7 – Fases do processo de *clock* em um *Wire* em NML.
Fonte: Próprio Autor, adaptado de (ANDERSON; BHANJA, 2014).

2.3 Estrutura de Dados

A estruturas de dados são elementos essenciais no que diz respeito a efetividade das heurísticas na otimização de soluções. Em outras palavras, elas podem facilitar ou inviabilizar a busca por uma solução ótima. Neste sentido, existem diversas estruturas de dados que permitem a execução de diferentes métodos de otimização. A representação de circuitos digitais mais comum na etapa de síntese lógica são os grafos. Diferentes tipos de grafos podem ser utilizados para esta finalidade. Dentre as estruturas mais utilizadas na representação podemos citar: Árvores, Grafos Acíclicos Direcionados (*Directed Graph Acyclic* - DAG), em especial algumas especializações de DAGs, como por exemplo: Diagramas de Decisão Binária, Grafo de And e Inversores. A seguir serão apresentadas algumas estruturas de dados que foram utilizadas neste trabalho, são elas: Grafo de Inversores e Majoritárias e Grafos Ortogonais.

2.3.1 Majority Inverter Graph - MIG

Tanto as metodologias avançadas de otimização de DAG, quanto as ferramentas associadas utilizam métodos algébricos e booleanos para aplicar otimizações. Quando os nós do DAG são restritos a apenas um tipo de função, o procedimento de otimização pode ser muito mais eficaz (Amarú, 2015). Isso ocorre porque as transformações lógicas são projetadas especificamente para direcionar a funcionalidade (função booleana) do nó escolhido. Por exemplo, nos AIGs, transformações lógicas como balanceamento, refatoração e reescrita geral, são muito eficazes (Amarú, 2015). Neste caso, o balanceamento é baseado no axioma da associatividade da álgebra booleana tradicional (Mishchenko; Chatterjee; Brayton, 2006) (Mishchenko; Brayton, 2006).

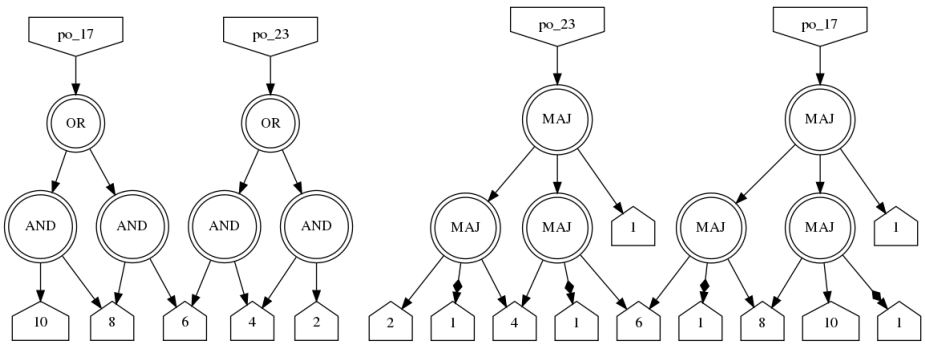
No caso da refatoração, é aplicado um procedimento no subgrafo AIG, o qual é primeiro transformado em uma Soma de Produtos (*Sum of Products* - SOP), onde logo após é aplicada a fatoração (Amarú, 2015). A reescrita de AIGs, conceitualmente, inclui balanceamento e refatoração, onde seu objetivo é a substituição dos subgrafos AIG por implementações AIG pré-calculadas equivalentes e otimizadas, reduzindo o número de nós e níveis (Mishchenko; Chatterjee; Brayton, 2006).

Aplicando transformações locais efetivas, muitas vezes durante a otimização do AIG, é possível obter resultados com qualidade razoavelmente satisfatórios. A restrição da função nos AIGs facilita a avaliação da qualidade intermediária e o desenvolvimento dos algoritmos, mas em geral é mais propensa a mínimos locais. No entanto, os métodos booleanos ainda podem complementar a otimização no AIG resultante para obter maior qualidade dos resultados (Amarú, 2015).

Neste sentido, tentando explorar as mesmas propriedades para viabilizar otimizações, Amarú propõe uma nova estrutura de dados baseada em conceitos semelhantes aos da estrutura de AIG, porém, com o foco na representação de portas majoritárias, onde toda a lógica é expressa considerando apenas este elemento. Assim, todas as técnicas citadas anteriormente para otimizações de AIGs podem ser exploradas. Esta nova estrutura foi denominada Grafo de Majoritárias e Inversores.

Esta subseção apresenta esta nova forma de representação, assim como também apresenta alguns axiomas de sua álgebra booleana nativa. São apresentadas sucintamente os axiomas que compõem a base das técnicas de otimização algébrica e booleana propostas por Amarú nessa estrutura de dados, as quais desbloquearam novos pontos no espaço de *design*. É interessante notar que tentativas iniciais de lógica majoritária já foram relatadas nos anos 60 (Amarú, 2015), mas, devido à sua complexidade inerente, não conseguiram ganhar impulso mais tarde na síntese automatizada.

Definição 1: *Um MIG é uma rede lógica homogênea com grau de entradas em cada nodo igual a* 3*, e com cada nó representando a função majoritária. Em um MIG, as arestas são marcadas por um atributo regular ou complementado (o qual representa a presença de um inversor). Para esse fim, observe que o operador majoritário* $M(a, b, c)$ *se comporta como o operador de conjunção* $AND(a, b)$ *quando* $c = 0$ *e, como operador de disjunção* $OR(a, b)$ *quando* $c = 1$*. Portanto, a majoritária pode ser vista como uma generalização da conjunção e disjunção (AMARÚ; GAILLARDON; MICHELI, 2014a).* A Figura 8 apresenta a forma de representação de uma descrição Grafo E-OU-Inversor (*And-Or-Inverter Graph*- AOIG) sendo transformada em uma representação MIG.



(a) Descrição AOIG (b) Descrição MIG

Figura 8 – Circuito C17 representado em AOIG sendo transformado em MIG.
Fonte: Próprio Autor.

Analisando a Figura 8 é possível observar que a transição da representação de uma lógica AOI é simples, visto que os nodos do tipo OR e AND são substituídos por nodos MIG, onde existem 3 entradas. Assim, para representar a porta lógica OR, é necessário além das duas entradas primárias da estrutura AOIG, apenas inserir o valor lógico constante 1 na terceira entrada do novo nodo MIG. Da mesma forma, quando é preciso representar a lógica AND, além das duas entradas originais da estrutura AOIG, é necessário inserir o valor lógico constante 0 (representados aqui pela adição do inversor na aresta da constante 1).

Como esta nova estrutura possui características de comportamento diferentes da álgebra booleana tradicional, foi preciso específicar um novo conjunto de axiomas, e a construção da nova álgebra booleana para a estrutura MIG. Estes axiomas estão apresentados nos trabalhos propostas por Amarú em (AMARÚ; GAILLARDON; MICHELI, 2014a) (AMARÚ; GAILLARDON; MICHELI, 2014b) (AMARÚ; GAILLARDON; MICHELI, 2014c) (AMARÚ; GAILLARDON; MICHELI, 2015) (AMARÚ; GAILLARDON; MICHELI, 2016). A seguir serão apresentadas as propriedades mais básicas da álgebra booleana, denominadas por Amarú como *Ômega* (Ω), são elas: comutatividade (*commutativity*), associatividade (*associativity*), distributividade (*distributivity*), propa-

gação da inversão (*inverter propagation*). Além dessas propriedades tradicionais, também é adicionada a propriedade referente a característica da porta majoritária, denominada *majority*.

A Figura 9 apresenta as propriedades *Ômega*, assim como as Figuras 10(a), 10(b), 10(c), 10(d) e 10(e) apresentam as representações gráficas de cada uma das propriedades *Ômega*.

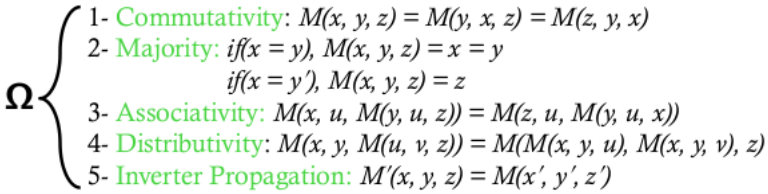


Figura 9 – Propriedades *Ômega*.
Fonte: Amarú, retirado de (Amarú, 2015).

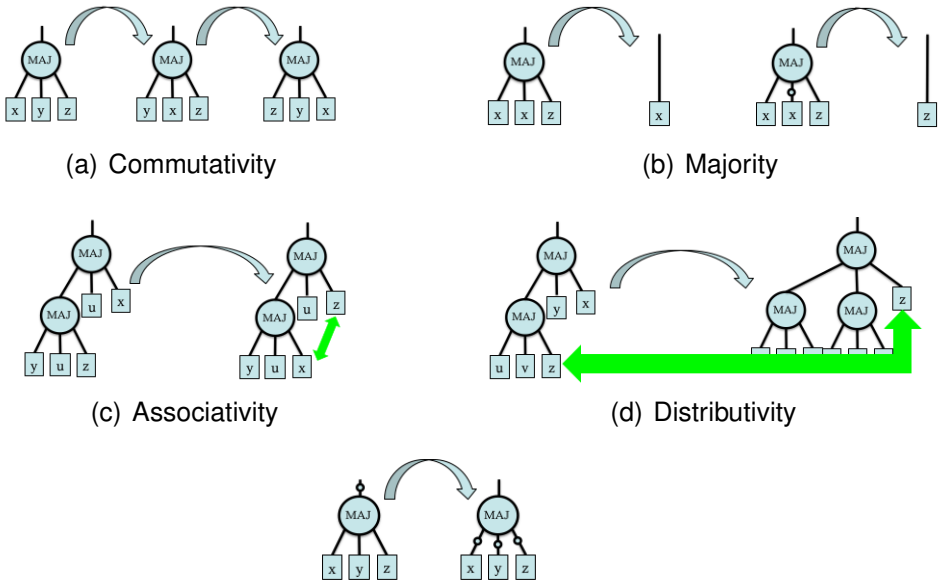
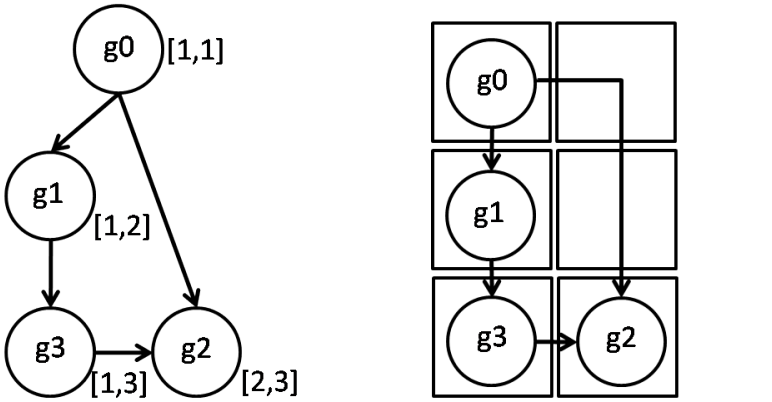


Figura 10 – Representação gráfica das propriedades *Ômega*.
Fonte: Amarú, retirado de (Amarú, 2015).

2.3.2 Orthogonal Graphs

Um desenho de grafo ortogonal (*Orthogonal Graph Drawing* - OGD) é uma estrutura de dados especializada de um caso especial de grafos. Ele foi intensivamente estudado e utilizado em diversos trabalhos sobre síntese lógica nos últimos anos (BIEDL, 1998), muitas soluções para circuitos VLSI (*Very Large Scale Integration*) foram propostos utilizando OGD. Um OGD pode ser definido como um grafo $G = (V, E)$ onde os vértices (V) são desenhados com pontos, e cada aresta (E) é representado

como uma sequência de segmentos, os quais mapeiam a disposição dos elementos de forma horizontal ou vertical alternadamente, como apresenta a Figura 11(a) (BIEDL, 1998). Um OGD possui muitas características que são consideradas muito úteis para problemas na tecnologia FCN, principalmente, para projeto FCN dispostos em *grid*, como apresenta o exemplo de um OGD (Figura 11(a)) sendo mapeado em um *grid* (Figura 11(b)).



(a) Grafo OGD Assinalado (b) Representação em *grid* do OGD

Figura 11 – Exemplo de mapeamento da estrutura OGD para síntese FCN.
Fonte: Próprio Autor.

Essa estrutura de dados se torna importante para este trabalho, pois é a estrutura de dados utilizada para a implementação do algoritmo da etapa de Posicionamento e Roteamento, proposta por Biedl (BIEDL, 1998). O algoritmo de Biedl foi proposto para grafos com grau máximo 3, sendo capaz de gerar uma solução de disposição planar dos nodos, sendo necessário apenas adequar as características da tecnologia alvo em questão (esquema *clock* entre outros).

2.4 EPFL Logic Synthesis Libraries

Outro elemento importante na execução desta tese foi o *framework* disponibilizado pela Escola Politécnica Federal de Lausanne (*École polytechnique fédérale de Lausanne* - EPFL) na Suiça. O *framework* é uma coleção modular *open-source* de bibliotecas escritas em *C++* para o desenvolvimento de aplicações em síntese lógica (SOEKEN et al., 2018). Todas as bibliotecas são bem documentadas e bem testadas, e viabilizam um componente base para o desenvolvimento de soluções complexas para síntese lógica. Esse *framework* foi desenvolvido com o objetivo de poupar tempo do desenvolvedor com programação de algoritmos básicos e padronizados, revertendo o tempo de desenvolvimento para a criação de soluções mais complexas em síntese lógica (SOEKEN et al., 2019a).

Este *framework* é composto por uma série de bibliotecas disponíveis com propósitos específicos. Por exemplo, a biblioteca *Alice*, permite uma implementação simples e padronizada para a interface com o usúario através de um terminal de comandos, e a integração com linguagens de *scripts*. Por outro lado, a biblioteca *Lorina* viabiliza de forma fácil um modelo de carregamento (*parser*) de descrições lógicas ou redes booleanas nos mais diversos formatos aceitos pela comunidade de síntese lógica. Já o módulo *Mockturtle* é uma biblioteca de redes lógicas, onde possui uma grande gama de algoritmos clássicos e do estado-da-arte de síntese lógica (dentre eles, os métodos de otimização MIG), os quais podem ser aplicados sobre diferentes estruturas de dados (SOEKEN et al., 2018). Desta forma, esse conjunto de bibliotecas forneceram a base sólida para a implementação e prototipação da metododologia proposta com base nos métodos de otimização MIG presentes na biblioteca *Mockturtle*. A Figura 12 apresenta as diferentes bibliotecas do *framework EPFL Logic Synthesis Libraries*.

Figura 12 – EPFL Logic Synthesis Libraries.
Fonte: Próprio Autor, adaptado de (SOEKEN et al., 2019a)

2.5 Fiction Tool

Como mencionado anteriormente, a ferramenta Fiction (WALTER et al., 2019) proposta por Walter, é um *framework* para tecnologias FCN implementado em C++, desenvolvido pela universidade de Bremen na Alemanha. A ferramenta utiliza as bibliotecas do *framework EPFL logic synthesis libraries* (SOEKEN et al., 2019a), como a biblioteca *Lorina* e a biblioteca *Alice*. Esta ferramenta é considerada o estado da arte em síntese para tecnologias FCN, e possui o enfoque na etapa de projeto físico dessas nanotecnologias, como por exemplo, a tecnologia QCA em suas diferentes formas (ou seja, atômica ou molecular) e dispositivos NML.

Como as pesquisas sobre a possível tecnologia que venha a substituir a tecnologia CMOS ainda está em andamento, o ambiente da ferramenta é o mais genérico possível, sendo capaz de executar tarefas de *design* físico para leiautes de circuitos FCN em uma estrutura de dados que abstrai determinada tecnologia ou *design* de célula. Assim, é possível utilizar um conjunto extensível de bibliotecas de portas, tecnologias e tipos de células, os quais podem ser facilmente compilados para qualquer tecnologia FCN desejada (WALTER et al., 2019).

Neste sentido, a ferramenta Fiction provê duas abordagens de síntese para estas tecnologias de forma distinta. A primeira é baseada em utilização de resolvedores SAT (*Satisfiability*), onde através dessa abordagem a ferramenta consegue gerar leiautes que atendem as restrições impostas. Esse método é denominado *Exact* e pode ser invocado no ambiente da ferramenta através do comando **exact**.

Além disso, o ambiente da ferramenta Fiction também apresenta um método escalável baseado em OGDs, denominado *Ortho*, o qual pode ser invocado no ambiente da ferramenta pelo comando **ortho**. Essa abordagem apresenta uma vantagem significativa de tempo de execução em comparação com a abordagem *exact* conforme a entrada aumenta de tamanho. Embora os leiautes gerados por esse método não sejam considerados ótimos em termos de área, essa técnica é aplicável mesmo para circuitos maiores e fornece resultados em tempo de execução razoável (Walter et al., 2019).

Quanto aos esquemas de *clock*, existem vários sistemas de *clock* regulares para o paradigma FCN, em especial a tecnologia QCA. O ambiente da ferramenta permite a utilização de vários esquemas bem estabelecidos no estado-da-arte nativamente, como por exemplo o esquema 2DDWave (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008) e o esquema USE (CAMPOS et al., 2016) (os quais serão discutidos no capítulo de trabalhos relacionados). Esses esquemas de *clock* podem ser utilizados nas etapas de posicionamento e roteamento dos circuitos. No entanto, em algumas ocasiões pode ser necessário a modificação por esquemas de *clock* com mais zonas ou um grau de liberdade maior. Neste sentido, a ferramenta Fiction suporta esquemas de relógio regulares e irregulares com quantidades variáveis de zonas de *clock*.

Em relação a bibliotecas de células, existe apenas uma disponível, a QCA-ONE baseada na proposta de Campos (CAMPOS et al., 2016), contudo, com algumas configurações diferentes para algumas células. Como saídas, a ferramenta Fiction permite gerar um arquivo para visualização do circuito na ferramenta QCADesigner (WALUS et al., 2004), onde é possível realizar simulações. Sendo assim, embora o método *Exact* apresente melhores resultados finais em área de leiaute, o mesmo não é escalável para grandes circuitos, o que inviabiliza avaliações mais robustas com *benchmarks* bem estabelecidos da comunidade de síntese lógica. Nesse sentido, o ambiente da ferramenta Fiction se tornou ideal para a rápida prototipação desta pro-

posta de metodologia, onde foi estabelecido a utilização do método *Ortho* como base para o desenvolvimento da solução, pois mesmo não obtendo os leiautes mais otimizados, é capaz de sempre entregar uma solução em tempo razoável de computação. Além disso, o método *Ortho* é o primeiro método capaz de aplicar síntese automática para a tecnologia QCA, sem restrições de tamanho do *designs*, em outras palavras, ele é aplicável a circuitos com grande número de elementos.

2.6 Conclusões

Este capítulo apresentou uma introdução sobre os principais conceitos relacionados a esta tese. Neste sentido, foram apresentadas brevemente as informações sobre o processo de síntese de circuitos e suas metodologias tradicionais, assim como o paradigma FCN e suas duas principais tecnologias QCA e NML. Da mesma forma, foram discutidas a importância das estruturas de dados na área de síntese lógica. Foram discutidas as duas principais estruturas de dados utilizadas neste trabalho, MIG e OGD. É importante salientar a importância da estrutura MIG, visto que ela viabilizou novos desafios e métodos de otimização focados unicamente na porta majoritária. Assim, vários dos métodos que foram utilizados neste trabalho são oriundos desse conjunto de axiomas. Por último, também é apresentado o *framework* da EPFL, o qual viabilizou uma estrutura bem testada e validada para a proposição deste trabalho. Além disso, é importante destacar que o *framework* é o estado-da-arte em ferramentas e algoritmos para síntese lógica, muitos dos algoritmos que serão apresentados na próxima seção estão disponíveis no *framework*, evitando assim, o trabalho de reimplementação dessas heurísticas. Ademais, também foi apresentada a ferramenta Fiction, considerada o estado da arte na etapa de síntese física para tecnologias FCN. Esta ferramenta foi utilizada como ambiente de prototipação da metodologia proposta neste trabalho. No próximo capítulo serão apresentados os trabalhos correlatos, os quais foram utilizados como a base de estudo para a proposição desta tese.

3 TRABALHOS RELACIONADOS

Embora as estruturas de dados e métodos de síntese tenham evoluído nos últimos anos, ainda grande parte são baseados no modelo de construção de lógica da tecnologia CMOS. Porém, com a evolução dos processos e a miniaturização da tecnologia, diversos problemas emergiram em grande parte sobre as limitações físicas. Desta forma, muito se considera a possível substituição da tecnologia CMOS. Entretanto, possivelmente, tanto os métodos atuais de síntese, quanto suas estruturas de dados, não estarão aptos a realizarem a transição de tecnologia de forma eficiente.

Assim, existem diversos trabalhos no estado da arte no que diz respeito a estruturas de dados e métodos de síntese com foco nessas novas tecnologias. Dentre as tecnologias, merecem destaque as baseadas no paradigma FCN, dentre elas a tecnologia QCA, a qual é baseada em computação por efeito de campo e foco deste trabalho. Essa tecnologia implementa sua lógica através de um elemento básico, neste caso, a porta majoritária. Portanto, este capítulo tenta propor uma discussão a cerca dos principais trabalhos correlatos sobre estruturas de dados propostas para síntese QCA, e os métodos de síntese baseados na porta majoritária publicados recentemente. Além disso, também é apresentado uma pequena discussão sobre outra possível estrutura a ser utilizada na síntese lógica para FCN, a *Threshold Logic*. Por último, também são abordados alguns trabalhos propostos recentemente, os quais são considerados o estado-da-arte para o desenvolvimento sobre síntese automática de circuitos FCN, considerando diferentes temas da síntese, como por exemplo: esquemas de *clock* e abordagens para posicionamento e roteamento.

3.1 Métodos baseados em Portas Majoritárias

Um dos primeiros trabalhos considerando síntese automática de majoritárias focada na tecnologia QCA foi o trabalho proposto por Zhang (ZHANG et al., 2004), denominado *A Method of Majority Logic Reduction for Quantum Cellular Automata*. Neste trabalho, Zhang propõe a implementação de 13 funções básicas utilizando a porta majoritária. Além disso, Zhang demonstra que a biblioteca de 13 funções pode mapear todas as funções de 3 entradas possíveis e, somado a isso, é apresentado o método de conversão de uma SOP tradicional para o arranjo de majoritárias.

Os experimentos para avaliar o conjunto de funções implementados em QCA, foram efetuados realizando a síntese de um somador de 1 bit. A partir da SOP do somador foi efetuada a síntese pelo método proposto, e logo após, a lógica resultante foi descrita através de instâncias das células da biblioteca proposta. Por fim, foi executada a simulação do circuito na ferramenta QCADesigner (WALUS et al., 2004). Os resultados sobre área obtiveram uma taxa de otimização de 25% em relação a implementações manuais publicadas na literatura. A Figura 13 apresenta as 13 funções definidas por Zhang.

Função	Função Majoritária	Porta Lógica
$F = ABC$	$M(M(A,B,0),C,0)$	
$F = AB$	$M(A,B,0)$	
$F = ABC + \bar{A}\bar{B}\bar{C}$	$M(M(M(A,B,0),C,0),$ $M(M(\bar{A},\bar{B},0),\bar{C},0),1)$	
$F = AB + \bar{A}\bar{B}C$	$M(M(A,B,0),$ $M(M(\bar{A},\bar{B},0),C,0),1)$	
$F = A$	$M(A,0,1)$	
$F = AB + \bar{B}C$	$M(M(A,B,0),$ $M(\bar{B},C,0),1)$	

(a)

Função	Função Majoritária	Porta Lógica
$F = AB + \bar{A}\bar{B}$	$M(M(A,B,0),$ $M(\bar{A},\bar{B},0),1)$	
$F = AB + BC$	$M(B,M(A,C,1),0)$	
$F = AB + BC + \bar{A}\bar{B}\bar{C}$	$M(M(B,M(A,C,1),0),$ $M(\bar{A},M(\bar{B},\bar{C},0),0),1)$	
$F = ABC + A\bar{B}\bar{C}$	$M(M(A,B,\bar{C}),$ $M(A,\bar{B},C),0)$	
$F = ABC + \bar{A}B\bar{C} +$ $A\bar{B}\bar{C}$	$M(M(A,C,0),$ $M(A,B,\bar{C}),M(\bar{A},\bar{B},\bar{C}))$	
$F = AB + BC + AC$	$M(A,B,C)$	
$F = ABC + \bar{A}\bar{B}C +$ $A\bar{B}\bar{C} + \bar{A}B\bar{C}$	$M(M(\bar{A},B,C),$ $M(A,B,C),\bar{C})$	

(b)

Figura 13 – As 13 funções propostas por Zhang
Fonte: Próprio Autor, adaptado de (ZHANG et al., 2004).

Embora o trabalho apresente grandes contribuições para os primeiros modelos de síntese com o foco na tecnologia QCA, todo seu fluxo ainda é executado de forma manual. Além disso, não foram efetuadas avaliações quanto a escalabilidade do método, sendo consideradas apenas funções de 3 entradas.

Um dos principais trabalhos sobre arranjos majoritários com foco na tecnologia QCA, é o trabalho de Momenzadeh (MOMENZADEH et al., 2005), denominado *Characterization, Test, and Logic Synthesis of And-Or-Inverter (AOI) Gate Design for QCA Implementation*. Neste trabalho é demonstrado que as ferramentas de síntese atuais CMOS não fazem bom uso da lógica majoritária para o projeto de circuitos. Dessa forma, é proposta uma porta AOI (*And-Or-Inverter*) utilizando a porta majoritária na tecnologia QCA.

Assim, o trabalho apresenta um conjunto de configurações dessa estrutura de AOI para a implementação das funções lógicas propostas por Zhang em (ZHANG et al., 2004). Os resultados dos experimentos são apresentados e o método proposto com AOI atingiu taxas de até 23.9% de redução de elementos majoritários em relação a Zhang, e 33.4% em relação a atraso. A Figura 14 apresenta a biblioteca de 13 células proposta por Zhang na abordagem com AOI.

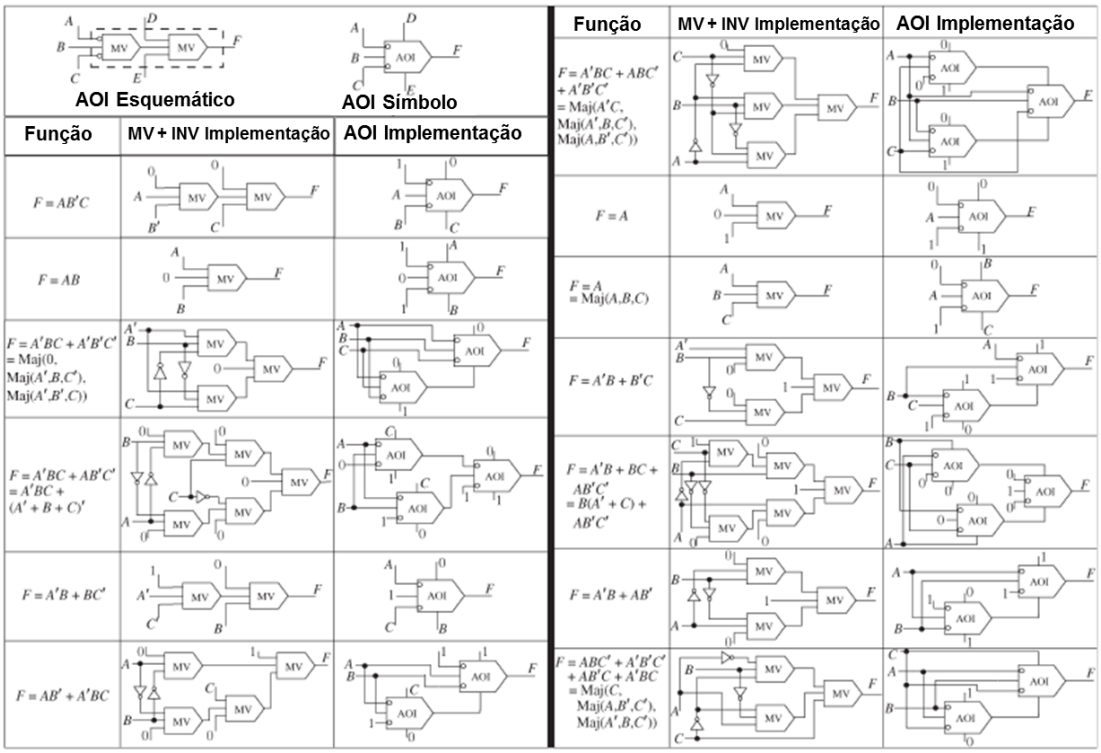


Figura 14 – 13 funções propostas por Zhang utilizando a AOI
Fonte: Próprio Autor, adaptado de (MOMENZADEH et al., 2005).

O trabalho proposto se torna relevante pois apresenta análises sobre a aplicabilidade de métodos e estruturas de dados bem estabelecidos para a tecnologia CMOS na síntese de tecnologias baseadas em portas majoritárias. Além disso, o método proposto precisa de um passo extra para realizar a decomposição da lógica descrita na estrutura AOI, nos elementos da biblioteca de majoritárias. Da mesma forma que acontece com Zhang (ZHANG et al., 2004), o trabalho não apresenta uma análise de escalabilidade, considerando circuitos maiores.

Outro trabalho importante no sentido de aplicar síntese automática para majoritárias foi proposto por Shang (SHANG, 2010). Shang propõe uma abordagem utilizando um conjunto de *scripts* da ferramenta de síntese tradicional na área de síntese lógica, denominada SIS (BERKELEY, 2014a). O trabalho é intitulado *An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata*. Dessa forma, é efetuada uma decomposição com os métodos da ferramenta SIS, considerando a lógica do elemento da tecnologia *QCA*, a porta majoritária. Logo após, é proposta uma biblioteca descrita manualmente com 40 funções, as quais podem implementar qualquer função de até 3 entradas com arranjos majoritários. Por fim, é efetuada a tradução das porções do circuito para uma das células propostas. Além de realizar a tradução para os elementos propostos, o método apresentado tem um passo extra para eliminar algumas redundâncias da rede resultante. São efetuadas avaliações em comparação ao método de Zhang (ZHANG et al., 2004) e os resultados apresentados atingem otimizações em média de 8.1%, 28.9% e 29.0% para atraso, contagem de células e área, respectivamente. A Figura 15 apresenta as 40 funções propostas com arranjos de majoritárias.

Função Nº	S: Função Primitiva	S1: Expressão Majoritária	Função Nº	S: Função Primitiva	S1: Expressão Majoritária
1	0	0	21	a'b'	M(a,1,b)'
2	1	1	22	a + b	M(a,1,b)
3	a	a	23	a' c'	M(a,1,c)'
4	a'	a'	24	a + c	M(a,1,c)
5	b	B	25	b'c'	M(b,1,c)'
6	b'	b'	26	b + c	M(b,1,c)
7	c	C	27	a'b	M(a',0,b)
8	c'	c'	28	a + b'	M(a,1,b')
9	ab	M(a,0,b)	29	a' c	M(a',0,c)
10	a' + b'	M(a,0,b) '	30	a + c'	M(a,1,c')
11	Ac	M(a,0,c)	31	b' c	M(b',0,c)
12	a' + c'	M(a,0,c)'	32	b + c'	M(b,1,c')
13	Bc	M(b,0,c)	33	ab + ac + bc	M(a,b,c)
14	b' + c'	M(b,0,c)'	34	a'b' + a'c' + b'c'	M(a,b,c)'
15	ab'	M(a,0,b')	35	a'b + a'c + bc	M(a',b,c)
16	a' + b	M(a',1,b)	36	ab' + ac + b'c	M(a,b',c')
17	ac'	M(a,0,c')	37	ab' + ac + b'c	M(a,b',c)
18	a' + c	M(a',1,c)	38	a'b + a'c' + bc'	M(a',b,c')
19	bc'	M(b,0,c')	39	ab + ac' + bc'	M(a,b,c')
20	b' + c	M(b',1,c)	40	a'b' + a'c + b'c	M(a',b',c)

Figura 15 – 40 funções propostas por Shang
Fonte: Próprio Autor, adaptado de (SHANG, 2010).

Como nos trabalhos anteriores, Shang acrescenta passos extras onde a decomposição da lógica é efetuada com o foco nas 40 funções descritas na biblioteca de majoritárias. Somado a isso, o método possui um *overhead* computacional, com o passo de exploração de decomposições pelos métodos de síntese lógica da ferramenta SIS. Entretanto, como discutido no trabalho de Zhang (MOMENZADEH et al., 2005), esses métodos não fazem bom uso da lógica da majoritária.

O primeiro trabalho proposto por Amarú (AMARÚ; GAILLARDON; MICHELI, 2013a), intitulado *BDS-MAJ: A BDD-based Logic Synthesis Tool Exploiting Majority Logic Decomposition*, apresenta um relaxamento na estrutura de BDD clássica para lidar com arranjos majoritários. Esta proposta utilizou o ambiente da ferramenta BDS (*Binary Diagram System*) (YANG; CIESIELSKI, 2002), a qual é capaz de representar a decomposição da lógica em nodos tipicamente primitivos, como por exemplo, nodos que representam a lógica *And/Or*. Embora o BDS seja capaz de representar uma lógica mais complexa, como por exemplo, a lógica da porta lógica *Xor* e de um *Multiplexador-MUX*, essa representação é efetuada através de métodos de decomposição lógica. Assim, a representação desse tipo de estrutura não é efetuada diretamente.

Este trabalho propõe uma nova estrutura de BDD, a qual pode representar estruturas de majoritárias diretamente. O método BDD baseado em decomposição para majoritária (BDD-MAJ) foi implementado na ferramenta BDS, a qual foi renomeada para BDS-MAJ. A ferramenta recebe uma rede booleana e tenta aplicar a decomposição baseada em portas majoritárias. Neste sentido, Amarú acrescenta uma etapa intermediária no algoritmo da ferramenta BDS, o qual tenta aplicar transformações da lógica para a representação da porta majoritária, este método é apresentado em (AMARÚ; GAILLARDON; MICHELI, 2013a). A Figura 16 ilustra um fluxograma com a sequência de passos do método na ferramenta BDS-MAJ.

Os resultados da abordagem de síntese considerando a nova versão BDD-MAJ apresentam um ganho em área entre 28,8% e 26,4% e atraso entre 12,8% e 20,9% em relação a ferramenta acadêmica ABC (BERKELEY, 2014b). Em relação a ferramenta comercial Synopsys Design Compiler (SYNOPSYS, 2011) para a tecnologia de 22 *nm* e considerando uma biblioteca padrão das seguintes portas: *MAJ-3, XOR-2, XNOR-2, NAND-2, NOR-2 e INV* os ganhos foram de 6% e 7,8% para área e atraso, respectivamente. Dessa forma, os resultados apresentados reforçam a importância de estudos sobre novas abordagens para síntese de circuitos digitais que considerem portas majoritárias, além de evidenciar a eficiência da nova estrutura de BDD, a qual considera arranjos majoritários.

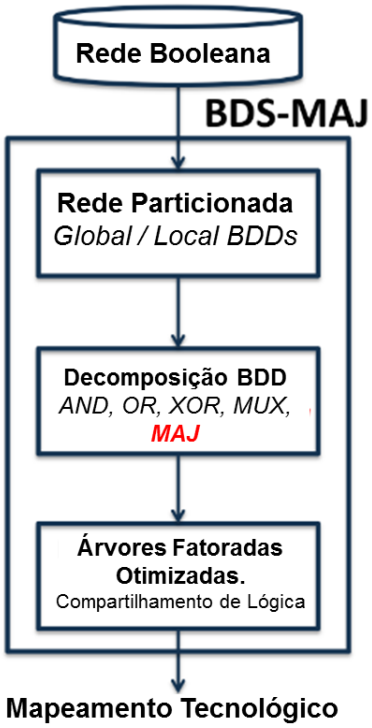


Figura 16 – Fluxograma das etapas da ferramenta BDS-Maj
Fonte: Próprio Autor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2013a)

.

Considerando as evidências reportadas na literatura que as estruturas tradicionais de síntese lógica aplicadas a tecnologia CMOS não são eficientes para síntese de portas majoritárias, Amarú apresenta uma série de artigos propondo novas soluções para representação da lógica diretamente em majoritárias. Assim, neste contexto a estrutura *Majority-Inverter Graph* (MIG) foi proposta. Neste artigo intitulado *Majority-Inverter Graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization*, é apresentada a prova da equivalência entre as estruturas de dados MIG e AOIG. Neste sentido, considerando que a estrutura de dados AIG é uma especialização da classe AOIG, pode-se afirmar que $AIGs \subset AOIGs \subset MIGs$. Assim, é possível concluir que $AIGs \subset MIGs$.

Este trabalho apresenta a estrutura de dados MIG e um conjunto de teoremas para uma nova forma de álgebra booleana baseada na estrutura MIG (AMARÚ; GAILLARDON; MICHELI, 2014a). Umas das provas apresentadas é que a estrutura de MIG é uma representação universal, sendo assim, a Figura 17(a) apresenta a transformação da função lógica Xor entre as variáveis x, y, z na estrutura de AOIG para a representação em MIG. Da mesma forma, a Figura 17(b) apresenta a transformação da função lógica $x * (y + u * v)$ na estrutura de AOIG em uma representação equivalente na estrutura de MIG.

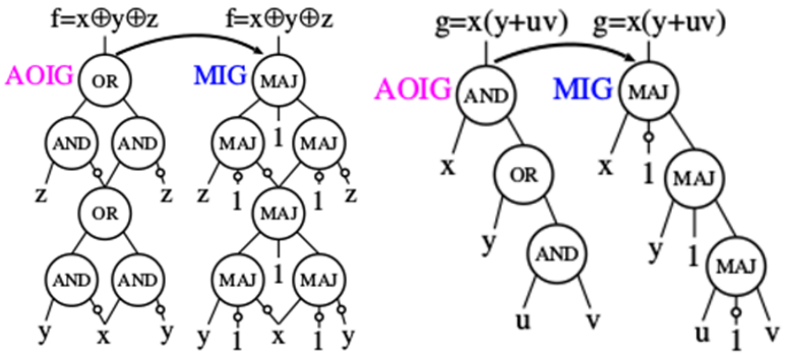


Figura 17 – Transformação da representação das funções *Xor* e $x * (y + u * v)$ em AOIG para uma representação em MIG.
Fonte: Próprio Autor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

Em conjunto com a nova representação, o trabalho apresenta uma prova do sistema axiomático (denominado *Ômega* (Ω)), demonstrando as principais propriedades da lógica booleana tradicional. A Figura 18(a) apresenta estas propriedades. Entretanto, além das propriedades do sistema axiomático *Ômega*, as quais também são apresentadas em (AMARÚ; GAILLARDON; MICHELI, 2015), este trabalho apresenta novas propriedades que auxiliam e permitem a aplicação de otimizações na estrutura MIG, denominado sistema axiomático *Psi* (Ψ). A Figura 18(b) apresenta as propriedades *Psi*.

$$\Omega \begin{cases} \textbf{Commutativity} - \Omega.C \\ M(x,y,z) = M(y,x,z) = M(z,y,x) \\ \textbf{Majority} - \Omega.M \\ \begin{cases} \text{if}(x=y)\text{: } M(x,y,z) = x = y \\ \text{if}(x=y')\text{: } M(x,y,z) = z \end{cases} \\ \textbf{Associativity} - \Omega.A \\ M(x,u,M(y,u,z)) = M(z,u,M(y,u,x)) \\ \textbf{Distributivity} - \Omega.D \\ M(x,y,M(u,v,z)) = M(M(x,y,u),M(x,y,v),z) \\ \textbf{Inverter Propagation} - \Omega.I \\ M'(x,y,z) = M(x',y',z') \end{cases}$$

(a) Propriedades Ômega.

$$\Psi \begin{cases} \textbf{Relevance} - \Psi.R \\ M(x,y,z) = M(x,y,z_{x/y'}) \\ \textbf{Complementary Associativity} - \Psi.C \\ M(x,u,M(y,u',z)) = M(x,u,M(y,x,z)) \\ \textbf{Substitution} - \Psi.S \\ M(x,y,z) = \\ M(v,M(v',M_{v/u}(x,y,z),u),M(v',M_{v/u'}(x,y,z),u')) \end{cases}$$

(b) Propriedades Psi.

Figura 18 – Axiomas propostos através das propriedades Omega e Psi.
Fonte: Amarú, retirado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

Com a apresentação das novas propriedades do sistema axiomático, são propostos dois algoritmos, os quais utilizam essas propriedades para aplicar otimizações na descrição inicial do MIG. O primeiro algoritmo possui como objetivo a otimização em área, desta forma, a Figura 19 apresenta o pseudocódigo descrito no trabalho.

O segundo algoritmo possui o foco na otimização da profundidade lógica da estrutura de MIG, resultante em uma abordagem para diminuir o caminho crítico da descrição. A Figura 20 apresenta o pseudocódigo descrito no trabalho.

Algoritmo 1: Pseudo-código MIG otimização área

Entrada: MIG α **Saída:** MIG α Otimizado

for (cycles=0; cycles<*effort*; cycles++) **do**

$\Omega.M_{L\rightarrow R}(\alpha)$; $\Omega.D_{R\rightarrow L}(\alpha)$;
$\Omega.A(\alpha)$; $\Psi.C(\alpha)$;
$\Psi.R(\alpha)$; $\Psi.S(\alpha)$; } reshape } eliminate
$\Omega.M_{L\rightarrow R}(\alpha)$; $\Omega.D_{R\rightarrow L}(\alpha)$;

end for

Figura 19 – Algoritmo de otimização em área.
Fonte: Próprio AUtor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

Algoritmo 2: Pseudo-código MIG otimização profundidade

Entrada: MIG α **Saída:** MIG α Otimizado

for (cycles=0; cycles<*effort*; cycles++) **do**

$\Omega.M_{L\rightarrow R}(\alpha)$; $\Omega.D_{L\rightarrow R}(\alpha)$; $\Omega.A(\alpha)$; $\Psi.C(\alpha)$;
$\Omega.A(\alpha)$; $\Psi.C(\alpha)$;
$\Psi.R(\alpha)$; $\Psi.S(\alpha)$; } reshape } push-up
$\Omega.M_{L\rightarrow R}(\alpha)$; $\Omega.D_{L\rightarrow R}(\alpha)$; $\Omega.A(\alpha)$; $\Psi.C(\alpha)$;

end for

Figura 20 – Algoritmo de otimização em profundidade lógica.
Fonte: Próprio Autor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

Estes algoritmos são, na verdade, uma sequência de otimizações efetuadas a partir dos sistemas Ω e Ψ. As Figuras 21 e 22 apresentam cada uma das otimizações propostas nos algoritmos. Na Figura 21 é apresentada a sequência de manipulações utilizadas no algoritmo para otimização em área. Por outro lado, na Figura 22(a) e 22(b) são apresentadas as otimizações referentes ao algoritmo com foco em otimização de profundidade lógica.

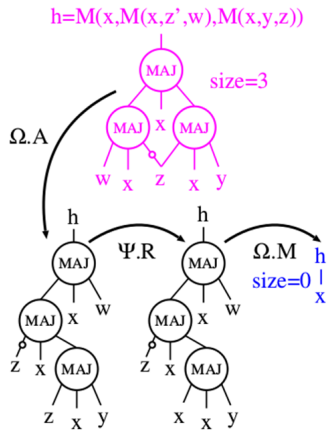


Figura 21 – Otimizações propostas através das propriedades Ω e Ψ para área.
Fonte: Próprio Autor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

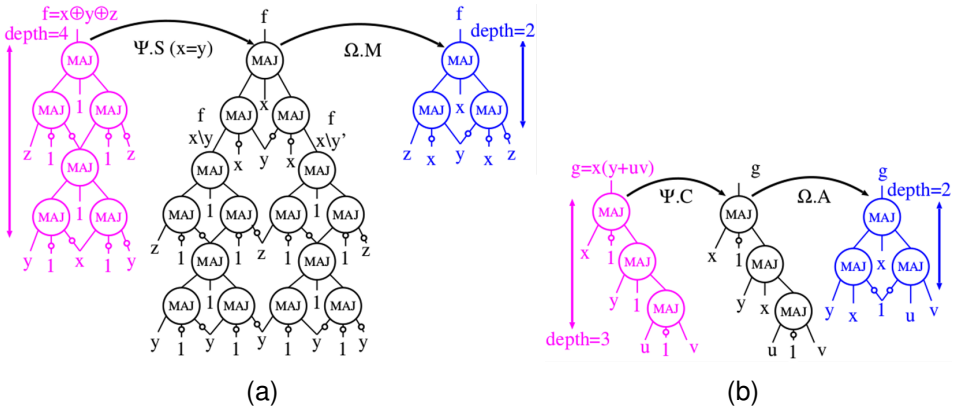


Figura 22 – Otimizações propostas através das propriedades Ω e Ψ para profundidade.
Fonte: Próprio Autor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

A implementação desses métodos resultaram no pacote denominado *MIGhty* o qual é uma ferramenta para manipulação e síntese de circuitos em MIG. Assim, as avaliações foram efetuadas de duas formas diferentes. A primeira avaliação possuía o objetivo de comparar a estruturas de dados, desta forma, foram comparados os resultados da síntese das ferramentas *MIGhty*, ABC (BERKELEY, 2014b), BDS (YANG; CIESIELSKI, 2002), as quais utilizam as estrutura de dados MIG, AIG, BDD, respectivamente. O *benchmark* selecionado foi o conjunto de circuitos MCNC. Os resultados demonstraram que a estrutura de MIG alcança 18.6% de otimização em relação a estrutura de AIG, e 23.7% em relação a estrutura de BDD quanto a profundidade lógica. Quanto a área os resultados foram equivalentes a estrutura de AIG, sendo 0.9% diferente e chegando a 2.1% em relação a BDDs. Os resultados para a atividade de chaveamento produziram o mesmo comportamento que as avaliações para área. A Figura 23 apresenta os resultados em um gráfico de forma sumarizada.

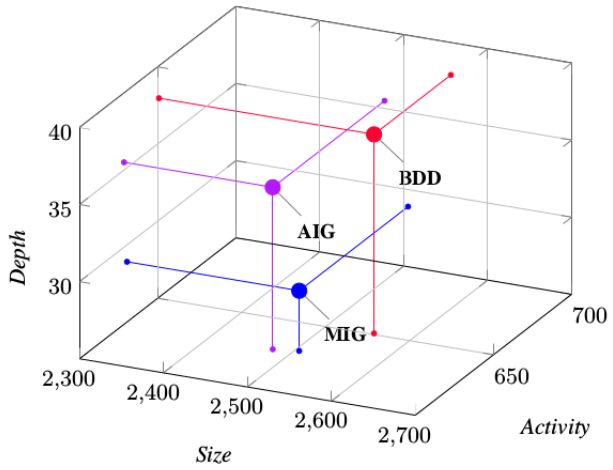


Figura 23 – Resultados comparativos entre a representação da lógica nas estruturas de dados MIG, AIG e BDD.
Fonte: Amarú, retirado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

Como conclusões gerais, considerando os fatores de profundidade lógica, área e atividade de chaveamento, desta avaliação foi possível afirmar que a estrutura de MIG é 17.5% melhor que a estrutura de AIG, e 27.7% melhor que a estrutura de BDD.

A segunda avaliação foi referente a efetividade de otimização no mapeamento tecnológico, foram efetuados experimentos com a estrutura de MIG e AIG na ferramenta acadêmica ABC, e também, foram efetuados experimentos utilizando uma ferramenta comercial para síntese. Para isso, foi descrita uma biblioteca de células padrão, considerando as seguintes células: *MIN-3, MAJ-3, XOR-2, XNOR-2, NAND-2, NOR-2, INV*, as quais foram caracterizadas para a tecnologia de 22 *nm*. Foram efetuadas as sínteses com as diferentes ferramentas e estruturas de dados e os resultados demonstraram que, em média, o fluxo utilizando MIG foi capaz de atingir graus de otimizações de 22%, 14% e 11% para área, atraso e consumo de energia, respectivamente, em relação as ferramentas/estruturas de dados acadêmicas e comerciais.

A Figura 24 apresenta o gráfico que sumariza esses resultados para área, atraso e dissipação de potência.

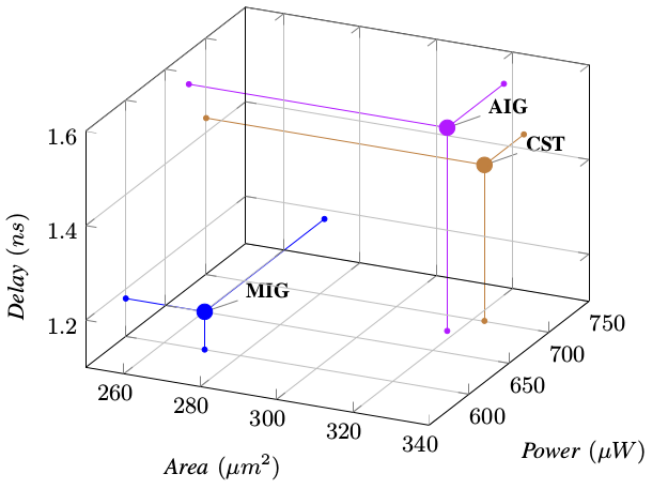


Figura 24 – Resultados comparativos da síntese entre as estruturas MIG, AIG e Fluxo comercial.
Fonte: Amarú, retirado de (AMARÚ; GAILLARDON; MICHELI, 2014a).

Após a modelagem da solução de representação da lógica diretamente em majoritárias, Amarú (AMARÚ; GAILLARDON; MICHELI, 2015) estende a discussão da nova estrutura de dados, a qual considera uma estrutura similar ao AIG descrito no formato AIGER (AIGER, 2012). Assim, esta nova estrutura considera um grafo de nodos que representam uma porta majoritária e arestas que se interligam de forma direta ou com a inversão da lógica, representando inversores.

Neste sentido, o artigo intitulado *Boolean Logic Optimization in Majority-Inverter Graphs* discute as especificações da representação da majoritária apresentando o passo a passo para a transformação da representação de um grafo AOIG para a estrutura de MIG. A Figura 25 apresenta a representação gráfica da transformação de um AOIG para um MIG. Além disso, também é efetuada uma discussão em uma possível transformação na representação, a fim de otimizar a estrutura final, a qual foi denominada MIGOpt (MIG Otimizado).

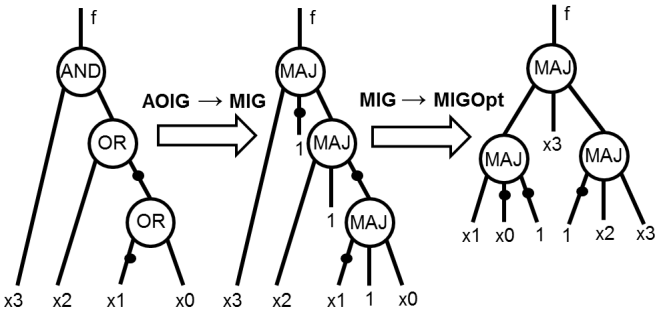


Figura 25 – Transformação da lógica em AOIG em uma representação em MIG.
Fonte: Próprio Autor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2015)

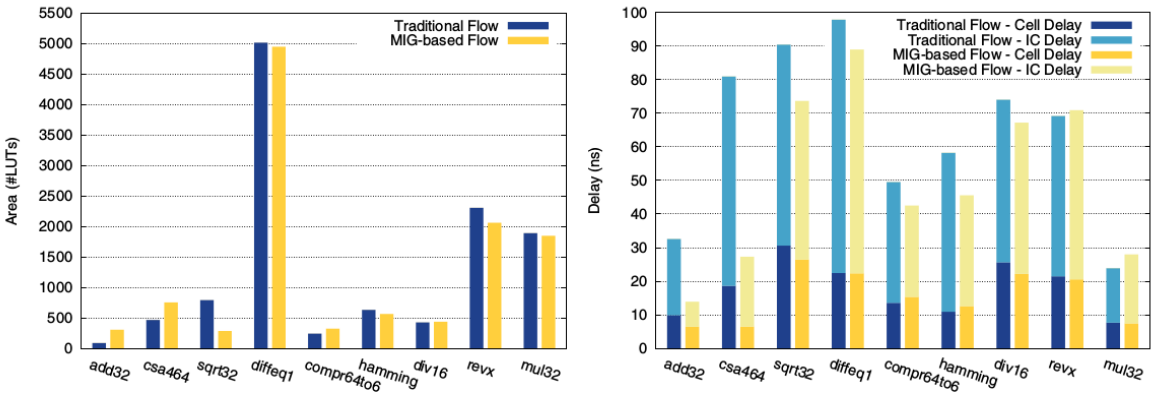
Em conjunto com a nova representação, o trabalho também faz a apresentação do sistema axiomático denominado *Ômega* (Ω), demonstrando as principais propriedades da lógica booleana tradicional. Além de uma discussão em torno das possíveis otimizações inserirem erros denominados *"Safe Errors"* ou em português, Erros Seguros, no processo de otimização. Neste sentido, o artigo apresenta uma avaliação da criticidade das otimizações em relação ao resultado final, em outras palavras, qual a efetividade de determinadas porções da descrição, as quais foram otimizadas, na geração do resultado final.

A fim de validar o sistema axiomático e a nova estrutura de dados, são propostas duas abordagens para mapeamento considerando a estrutura de MIG. A primeira abordagem considerando a síntese voltada para a redução de área e a segunda abordagem com o foco em redução da profundidade lógica dos circuitos. Foram efetuados experimentos com o conjunto de circuitos do *benchmark* IWLS'05 e outro conjunto de circuitos aritméticos descritos em HDL, os resultados referentes a otimização na profundidade lógica chegaram a 17.98$\%$ e 26.69$\%$, respectivamente, porém, com o aumento em área e dissipação de potência.

O resultado das otimizações na profundidade lógica no caso de somadores *ripple carry* foram equivalentes a profundidade lógica da estrutura de somadores *carry look ahead*. Também foram efetuados experimentos com fluxo ASIC (*Application Specific Integrated Circuits*) para a tecnologia de 22 *nm*. Neste caso, as otimizações com a estrutura de MIG atingiram 15.07$\%$, 4.93$\%$, 1.93$\%$ para delay, área e dissipação de potência, respectivamente.

Além de explorar a capacidade de representação otimizada baseada em majoritárias dos MIGs, e avaliar a capacidade dessas otimizações representarem otimizações nos circuitos finais na tecnologia CMOS, Amarú também explora a aplicação da estrutura de MIG para a síntese da tecnologia FPGA. Assim, neste trabalho intitulado *Majority-Inverter Graph for FPGA Synthesis*, foram efetuados experimentos considerando a tecnologia FPGA comercial de 28 *nm*, considerando o elemento básico da tecnologia, a *look-up table - (LUT)* com até 6 entradas (AMARÚ et al., 2015). Inicialmente, é apresentado um estudo de caso para um circuito específico *f51m* do *benchmark* MCNC, os resultados de otimização com a quantidade de elementos para representar a lógica entre AIG através da ferramenta ABC em relação a ferramenta *MIGhty* com MIG chegaram a 13%. Quanto a profundidade lógica foi possível perceber que o resultado com a estrutura de MIG reduziu 10 níveis de lógica.

Quanto as avaliações para o *benchmark*, os resultados finais apresentaram redução quanto a profundidade lógica do circuito mapeado em até 21%, porém, com o acréscimo em área de até 60%. A Figura 26 apresenta os resultados em gráficos dos experimentos, na Figura 26(a) é apresentada a relação da área final (número de LUTs) entre o método tradicional de síntese para FPGA e a abordagem utilizando MIG. Na Figura 26(b) é apresentada a relação de atraso para as duas abordagens (tradicional e com MIG), além de relacionar o atraso médio das células e do circuito completo.



(a) Area (b) Delay

Figura 26 – Gráfico de resultados da abordagem tradicional versus MIG para a tecnologia de FPGA.
Fonte: Amarú, retirado de (AMARÚ et al., 2015).

Por fim, as diferentes avaliações, juntamente com os diferentes axiomas propostos para a estrutura MIG culminou na proposta de um novo modelo de álgebra booleana com o foco em portas majoritárias e com a utilização de MIG como estrutura de dados para a representação da lógica, no trabalho intitulado *Majority-Inverter Graph for FPGA Synthesis* (AMARÚ; GAILLARDON; MICHELI, 2016). Além das propriedades *Omega* (Ω) e *Psi* (Ψ) os autores discutem o conjunto de propriedades básicas da álgebra booleana denominada *Delta* ($\triangle$), e logo após apresentam a relação de $\triangle$ com as propriedades Ω com o foco nos elementos majoritários. A Figura 27 apresenta as propriedades Delta.

$$
\boldsymbol{\Delta}\begin{cases}
\textbf{Identity : } \boldsymbol{\Delta.I} \\
x \vee 0 = x \\
x \wedge 1 = x \\
\textbf{Commutativity : } \boldsymbol{\Delta.C} \\
x \wedge y = y \wedge x \\
x \vee y = y \vee x \\
\textbf{Distributivity : } \boldsymbol{\Delta.D} \\
x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \\
x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \\
\textbf{Associativity : } \boldsymbol{\Delta.A} \\
x \wedge (y \wedge z) = (x \wedge y) \wedge z \\
x \vee (y \vee z) = (x \vee y) \vee z \\
\textbf{Complement : } \boldsymbol{\Delta.Co} \\
x \vee x' = 1 \\
x \wedge x' = 0
\end{cases}
$$

Figura 27 – Propriedades Delta.
Fonte: Próprio Autor, adaptado de (AMARÚ; GAILLARDON; MICHELI, 2016).

Da mesma forma, além da avaliação dos métodos já propostos anteriormente, os autores apresentam uma avaliação da nova abordagem para FPGAs proposta em (AMARÚ et al., 2015) utilizando a estrutura de MIG.

As avaliações foram efetuadas considerando os fluxos de síntese para ASIC e FPGAs, e os resultados demonstraram que para células padrão (ASIC) a ferramenta *MIGhty* conseguiu otimizar em relação a abordagem tradicional até 3.05%, 13.04% e 3.04%, em média, para área, atraso e dissipação de potência. Quanto a tecnologia de FPGA a abordagem utilizando MIG da ferramenta *MIGhty* resultou em graus de otimização, em média, de 9.56% para área, 9.64% para atraso e 4.58% para dissipação de potência.

3.1.1 Considerações Finais

Diversos estudos foram desenvolvidos com o foco em síntese de portas majoritárias, visto o papel de destaque desta estrutura tanto na tecnologia atual CMOS, quanto nas novas tecnologias. Neste sentido, uma das tecnologias em destaque é a tecnologia QCA, onde diversos trabalhos tentam utilizar as ferramentas tradicionais de síntese para efetuar uma decomposição da lógica tradicional em um conjunto de células QCA, por exemplo Zhang (ZHANG et al., 2004), Momenzadeh (MOMENZADEH

et al., 2005) e Shang (SHANG, 2010). Essas abordagens apresentam algumas evidências sobre as limitações de uso destas ferramentas CMOS na síntese QCA, visto que a estrutura de dados considerada não pode mapear diretamente a estrutura da porta majoritária inviabilizando possíveis otimizações. Além disso, todos os trabalhos citados possuem limitação nas avaliações de escalabilidade, visto que só avaliaram os métodos com circuitos pequenos e considerando a decomposição em funções de até 3 entradas. Somado a isso, todos os métodos possuem etapas manuais, o que inviabiliza avaliações mais robustas.

Por outro lado, Amarú propõe algumas estruturas específica para a representação da lógica majoritária diretamente na estrutura de dados. Desse modo, foi preciso especificar a nova álgebra booleana a ser utilizada para minimizar os elementos considerando a nova descrição. Essas estruturas denominadas BDD-Maj e MIG apresentaram ótimos resultados em relação as abordagens de decomposição. Porém, embora o MIG tenha apresentado melhores resultados sobre todas as outras estruturas, ainda não existe um consenso sobre a melhor abordagem a ser utilizada, principalmente, tratando-se da tecnologia QCA. Apesar da estrutura MIG ser considerada a melhor alternativa para tecnologias que utilizam a porta majoritária como elemento básico, como é o caso da tecnologia QCA, as avaliações de Amarú são focadas somente nas tecnologias atuais de concepção de circuitos, ou seja, CMOS e FPGA. Neste sentido, ainda existe oportunidades de exploração da estrutura MIG no que diz respeito a sua aplicabilidade na síntese QCA. A seguir serão discutidos trabalhos sobre uma outra estrutura alternativa, a qual utiliza o conceito de *Threshold Logic Functions*.

3.2 Threshold Logic Gates

Outra alternativa considerada nos últimos anos para síntese de circuitos em novas tecnologias, são abordagens baseadas em *Threshold Logic Gates* - (TLG). Neste sentido, o diferencial desta abordagem se dá pelo fato de que o conjunto de lógica coberta por funções *Threshold Logic Functions* - (TLF) é maior que o conjunto de lógica majoritária, o que torna a lógica majoritária um subconjunto de funções da TLF.

Na abordagem tradicional CMOS, a lógica é baseada nos elementos primitivos, *Inversor*, *Nand*, *Nor*, porém, nesta tecnologia, as funções básicas são intrinsecamente TLF, dessa forma, o custo para a implementação de uma função complexa (em uma única porta) é praticamente o mesmo da implementação de uma porta básica *Nand*, *Nor*. Diversos novos dispositivos, candidatos a substituir o transistor MOS, não se comportam como chaves lógicas e são mais adequados à implementação de TLGs. Exemplos desses dispositivos são os *Memristores*, *Spintronics Devices*, *Resonant Tunneling Diodes* e *Quantum Cellular Automata* (Neutzling, 2014).

Embora a ideia de utilizar TLF não seja nova, foi nos últimos anos que essa abordagem tem sido considerada para representação da lógica em novas abordagens. Um exemplo é o trabalho de Beiu (BEIU; QUINTANA; AVEDILLO, 2003) onde são abordados as principais utilizações de TLF, sendo assim, é apresentada a relação de funções *Threshold Logic* com portas majoritárias assim como várias possíveis aplicações de TLF em diferentes tecnologias.

A abordagem com TLG apresenta uma nova forma de construir os elementos lógicos, onde essa forma se baseia na utilização de TLFs, que são funções que descrevem a lógica considerando um sistema de pesos para cada váriavel de entrada e seu valor limite (*threshold*). Se a soma do ON-set (1) da função for maior ou igual ao valor limite (*threshold*), a saída da porta será 1, caso contrário 0. A Figura 28 apresenta a definição formal para TLF.

$$f = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i \geq T \\ 0, & otherwise \end{cases}$$

Figura 28 – Definição formal de TLF.
Fonte: Neutzling, retirado de (Neutzling, 2014).

As TLGs nada mais são que circuitos eletrônicos criados a partir de funções TLF, o diferencial dessa abordagem é que a utilização de TLGs viabilizam a representação da lógica com menor número de elementos, visto que cada elemento poderá englobar uma lógica mais complexa. Um exemplo prático dessa abordagem é apresentado na Figura 29 onde são apresentados vários exemplos de representação TLG.

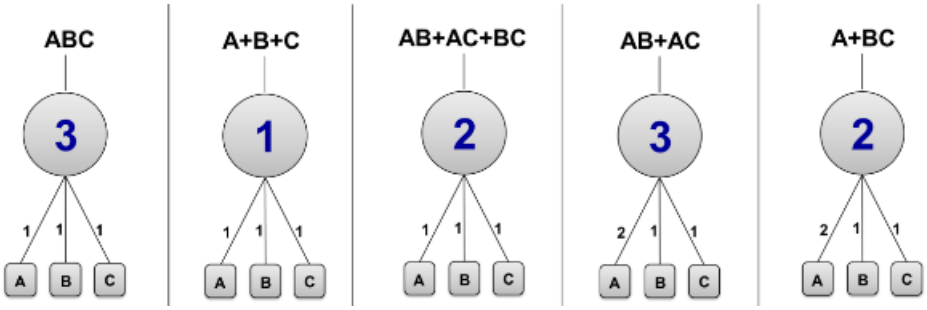


Figura 29 – Exemplos de TLGs.
Fonte: Neutzling, retirado de (Neutzling, 2014).

Esse sistema de pesos pode ser considerado como uma rede neural (BEIU; QUINTANA; AVEDILLO, 2003), onde existem pesos a serem encontrados para cada elemento da rede, os quais em conjunto, produzirão o comportamento lógico da função alvo. Portanto, existe uma complexidade associada as TLGs, que é o processo inicial de identificação se uma dada função é TLF ou não.

Nesse processo, são consideradas diversas características que a TLG precisará respeitar, como por exemplo: se a função é *unate*. Logo após, serão atribuídos pesos as váriaveis de entradas e será computado quais os pesos definitivos para respeitar o valor de *threshold*. Tipicamente, essa etapa é resolvida através de programação linear inteira, com a utilização de alguns pacotes matemáticos disponibilizados de forma *open-source*.

Diversos trabalhos exploram o uso de TLG na síntese de circuitos, principalmente, com o foco em tecnologias de majoritárias. Entretanto, a maioria das abordagens atuais não descreve uma estrutura de dados ou metodologia automática de construção dos elementos em uma tecnologia específica com o uso de TLG. Sendo assim, a seguir serão apresentados alguns trabalhos que envolvem a etapa de síntese de circuitos com a utilização de TLGs.

Um dos trabalhos que obteve destaque nos últimos anos foi o de Neutzling (NEUTZLING et al., 2017), intitulado *A Simple and Effective Heuristic Method for Threshold Logic Identification*. Este trabalho apresenta um método rápido para o cômputo/identificação de funções TLF acima de 6 entradas. Um dos principais diferenciais desta proposta é o baixo tempo de execução (*runtime*) na identificação de TLF com acima de 6 entradas, e sua efetividade de encontrar as funções TLF em comparação com outros métodos estado da arte. Além disso, outro diferencial desse método é que não utiliza programação linear para resolução dos sistemas de desigualdades.

Para verificar a efetividade do método, foram efetuadas avaliações no *benchmark OpenCores*, aplicando o algoritmo de cortes-K (CONG; C.; Y., 1999). Assim, em cada corte (porção do circuito) foi aplicado o algoritmo de reconhecimento de TLF, os resultados demonstraram que a heurística é escalável para funções complexas, e que é mais eficaz que as outras heirísticas estado-da-arte. Portanto, o método proposto obteve soluções com o número mínimo de pesos para os termos em 98$\%$ dos casos demonstrando a aplicabilidade do método. Embora o método seja uma ótima alternativa para a identificação de TLF, ele apresenta uma solução para parte do problema de síntese baseada em TLF, que é a identificação das TLF.

Considerando o exposto anterior, neste trabalho (NEUTZLING et al., 2015) apresenta a aplicação do método criado anteriormente, através de uma modificação do método de mapeamento da ferramenta ABC (BERKELEY, 2014b). Dessa forma, foram efetuados modificações no momento em que o método seleciona possíveis cortes-K para o mapeamento, os cortes selecionados são apenas aqueles que o método de identificação de TLF retornam verdadeiro, o que resulta em apenas funções TLF no mapeamento. A Figura 30 apresenta o fluxograma do método alternativo de mapeamento proposto na ferramenta ABC.

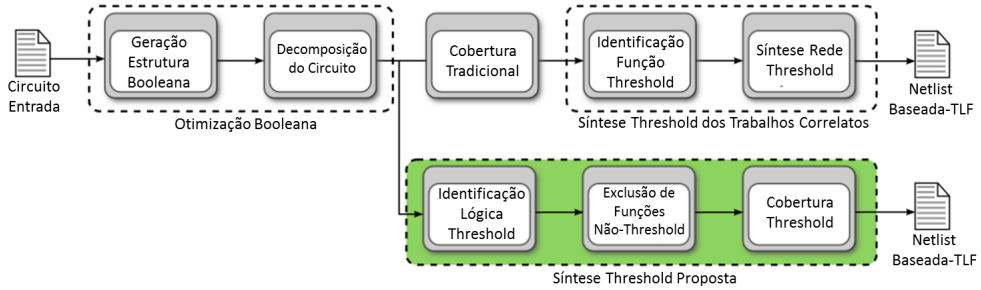


Figura 30 – Fluxograma do fluxo alternativo de mapeamento que considera TLG.
Fonte: Próprio Autor, adaptado de (NEUTZLING et al., 2015).

Os resultados apresentados demonstram que em comparação com o estado da arte a abordagem apresenta em média ganhos de 8% na contagem de TLGs, e uma redução de 46% em média para profundidade lógica do circuito. Contudo, essas avaliações foram efetuadas considerando apenas a etapa de síntese lógica, ou seja, a contabilização ocorreu sobre a *Netlist* de saída da ferramenta ABC, sem vincular nenhuma tecnologia alvo. Em outras palavras, ainda não são consideradas as especificidades da tecnologia QCA, por exemplo, o que deixa pontos em aberto na investigação de síntese QCA com TLF.

O trabalho proposto por Silva, intitulado *A Novel Five-input Multiple-function QCA Threshold Gate* (SILVA et al., 2018), utiliza as ideias de Neutzling (NEUTZLING et al., 2015) com a utilização de TLF para propor uma porta majoritária com cinco entradas e duas saídas, denominada MAJ_5 na tecnologia QCA. Dessa forma, são propostas quatro configurações diferentes de saídas para a MAJ_5, onde é possível representar as funções Majoritárias, e as funções Minoritárias da porta (basicamente, a função majoritária com aplicação da regra de De Morgan (DE MICHELI, 1994)). A Figura 31 apresenta as quatro configurações possíveis da porta MAJ_5.

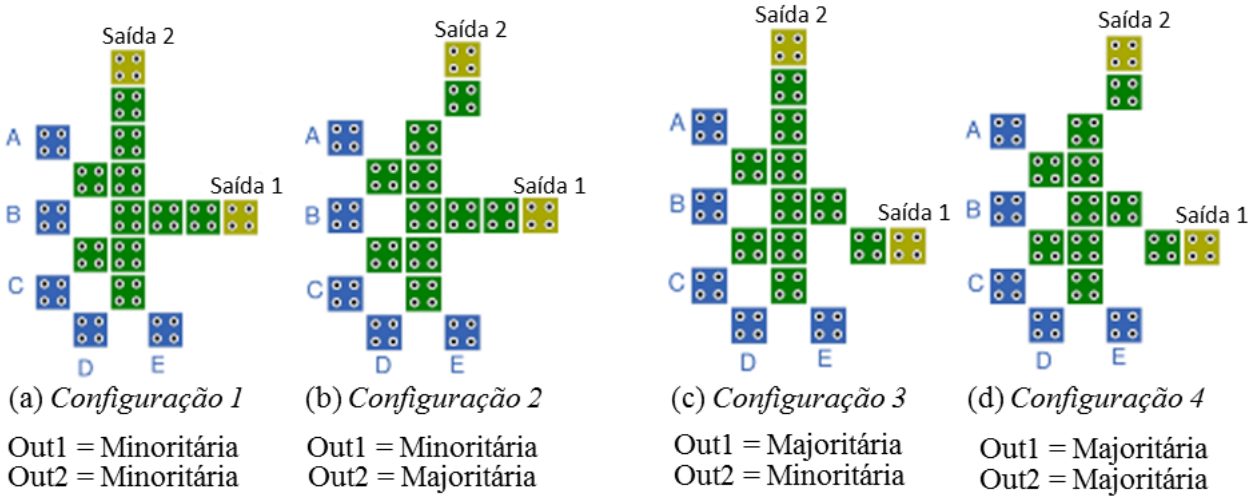


Figura 31 – Quatro configurações possíveis da porta MAJ_5 proposta.
Fonte: Próprio Autor, adaptado de (SILVA et al., 2018).

A porta proposta é criada segundo o método proposto por Neutzling (NEUTZLING et al., 2013), onde é possível configurar pesos para as entradas de uma função *Threshold*, a fim de implementar um comportamento lógico específico. Dessa forma, é apresentada uma tabela de funções na Figura 32 que descreve as configurações computadas de entrada da função *Threshold* e, suas respectivas funções Majoritárias e Minoritárias.

Majoritárias	Minoritárias	Entradas				
AB	!A+!B	A	B	-1	-1	1
A+B	!A!B	A	B	-1	1	1
ABC	!A+!B+!C	A	B	C	-1	-1
A+B+C	!A!B!C	A	B	C	1	1
A(B+C)	!A+!B!C	A	A	B	C	-1
A+BC	!A(!B+!C)	A	A	B	C	1
A(B+C+D)+BCD	(!A+(!B!C!D))(!B+!C+!D)	A	A	B	C	D
A(BC+BD+CD)+BCD	(!A+((!B+!C)(!B+!D)(!C!D)))(!B!C!D)	A	B	C	D	-1
A(B+C+D)+BC+BD+CD	(!A+(!B!C!D))(!B+!C)(!B+!D)(!C!D)	A	B	C	D	1
Maj_5	Min_5	A	B	C	D	E
MAJ_3	Min_3	A	B	C	-1	1

Figura 32 – Tabela de configurações da função *Threshold* MAJ_5 proposta.
Fonte: Próprio Autor, adaptado de (SILVA et al., 2018).

São propostos dois experimentos de síntese para a tecnologia QCA, o primeiro utilizando a porta MAJ_5 para a implementação de um somador *Full Adder*, e o segundo uma célula de memória RAM, considerando dois sistemas de *clock*. A primeira versão considerando o sistema de *clock* sem restrições, e o segundo considerando a abordagem de zonas de *clock* USE (CAMPOS et al., 2016).

Os resultados demonstraram que a abordagem proposta para as implementações dos leiautes dos circuitos em QCA foram melhores que as cinco principais implementações da literatura. No caso do circuito *Full Adder* os resultados quanto ao número de células QCA e atraso (em zonas de *clock*) foram semelhantes a melhor solução da literatura. Porém, a solução proposta não utiliza multicamadas para o roteamento dos elementos, o que simplifica o projeto. Em relação as outras soluções propostas previamente pelo próprio grupo de pesquisa, como Reis (REIS et al., 2016) e Campos (CAMPOS et al., 2016), a solução proposta obtém ganhos de até 62% em área e atraso nas configurações sem restrição de zonas de *clock*. No caso da implementação da célula de memória RAM as comparações foram com trabalhos da literatura onde foi possível, da mesma forma, atingir graus de otimizações de até 10% em relação ao demais.

Essa subseção discutiu alguns trabalhos que apresentam uma outra solução que não os MIGs para a representação e otimização de descrições de funções majoritárias com o enfoque na tecnologia QCA. É importante salientar que a TLG tem se apresentado como uma ótima alternativa para síntese FCN. Entretanto, ainda é preciso o avanço de diversos pontos para que essa abordagem se estabeleça como referência

em síntese dessas novas tecnologias. Uma das restrições é a questão da definição da estrutura de dados para representar as informações, assim como seu conjunto de axiomas, o que permitirá a concepção de uma nova álgebra booleana baseada nas características de TLG, como aconteceu na estrutura MIG.

3.3 Outros Trabalhos Importantes

Além dos trabalhos com o enfoque em estruturas para melhor representação da lógica majoritária, diversos trabalhos merecem destaque sobre as diferentes etapas da síntese de circuitos na tecnologia QCA, como por exemplo, a proposição de novos esquemas de *clock*, ou algoritmos de posicionamento e roteamento. A seguir, serão abordados alguns dos trabalhos estado-da-arte utilizados na proposição desta tese.

Um dos primeiros trabalhos a propor uma mudança de paradigma do sistema de *clock* para a tecnologia QCA foi o de Vankamamidi, intitulado *Two-Dimensional Schemes for Clocking/Timing of QCA Circuits* (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008). Este trabalho propõe um novo sistema de *clock*, denominado 2DDWave, pois a mudança de *clock* acontece como uma onda no circuito sendo da parte superior e esquerda do circuito para a parte inferior e direita. Diferentemente da proposta original de Lent (Lent; Tougaw, 1997), onde a informação flui da esquerda para a direita unicamente.

Essa nova proposta foi considerada como um sistema $2D$, isso se deve ao fato que a atividade de comutação é realizada em paralelo, em todas as zonas localizadas ao longo das diagonais, onde a informação flui simultaneamente. Portanto, o esquema 2DDWave tem um desempenho melhor em termos de velocidade de processamento do que o esquema $1D$ proposto anteriormente por Lent (Lent; Tougaw, 1997).

Nesta abordagem, o fluxo de dados é executado em um arranjo bidimensional semelhante a uma grade com interconexões ortogonais. Os dados computados se movem do noroeste para o sudeste. O grid do fluxo de dados deste esquema de *clock* 2DDWave é apresentado na Figura 33.

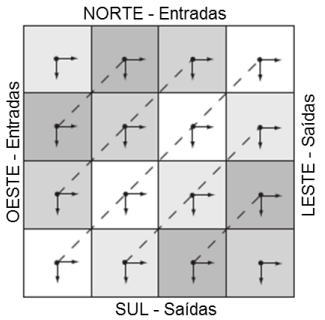


Figura 33 – Esquema de *Clock* 2DDwave com caminhos de *feedback*.
Fonte: Próprio Autor, adaptado de (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008).

O grande diferencial dessa abordagem é permitir maior liberdade na concepção de circuitos, visto que existem mais possibilidades de direções para as entradas e saídas de informações. Entretanto, essa proposta ainda não resolve um dos principais problemas que surgem nos esquemas de *clock* para o QCA, a capacidade de lidar com os caminhos de *feedback*. No esquema de *clock* unidimensional tradicional e no esquema de *clock* bidimensional proposto, a propagação do sinal é estritamente unidirecional (oeste para leste no caso $1D$ (Lent; Tougaw, 1997) e noroeste para sudeste no caso $2D$ (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008)). Portanto, embora os esquemas de *clock* sejam facilmente aplicáveis aos circuitos combinacionais, o caminho no caso dos circuitos sequenciais podem exigir uma técnica diferente.

Considerando essa limitação, Vankamamidi ainda propõe uma adaptação do esquema trapezoidal proposto por Niemer (NIEMIER; KOGGE, 2001) para o esquema unidimensional permitir caminhos de *feedback* em QCA. Assim, com essa adaptação é possível utilizar melhor a área de leiaute, além de viabilizar esquemas de *clock* com caminhos de *feedback*.

A Figura 34 apresenta o *loop* de zonas de *clock* para implementação sob um esquema bidimensional. Para permitir caminhos de *feedback*, o esquema 2DDWave é aplicado em cada região, de modo que as propagações de sinal sejam as seguintes: do noroeste para o sudeste na região 1 e na região 2, nordeste a sudoeste na região 3 e região 4, sudeste a noroeste na região 5 e sudoeste a nordeste na região 6. Assim, os circuitos nas seis regiões podem gerar suas saídas, assim como um das suas regiões seguintes podem utiliza-las em suas entradas, através dos caminhos de *feedback*. Os circuitos também podem receber novas entradas e propagar suas saídas, por exemplo, enquanto a região 2 recebe uma entrada de *feedback* do oeste e propaga o caminho de *feedback* através do sul, pode receber novas entradas do norte e enviar os resultados pelo leste.

Embora a solução demonstre melhores resultados em relação ao sistema $1D$ ainda enfrenta sérias restrições quando se trata de circuitos que necessitam de caminhos de *feedback*, como por exemplo os circuitos sequenciais. Além disso, mesmo com a solução alternativa para implementar caminhos de *feedback* a abordagem apresenta limitações quanto a escalabilidade em um circuito com milhares de portas.

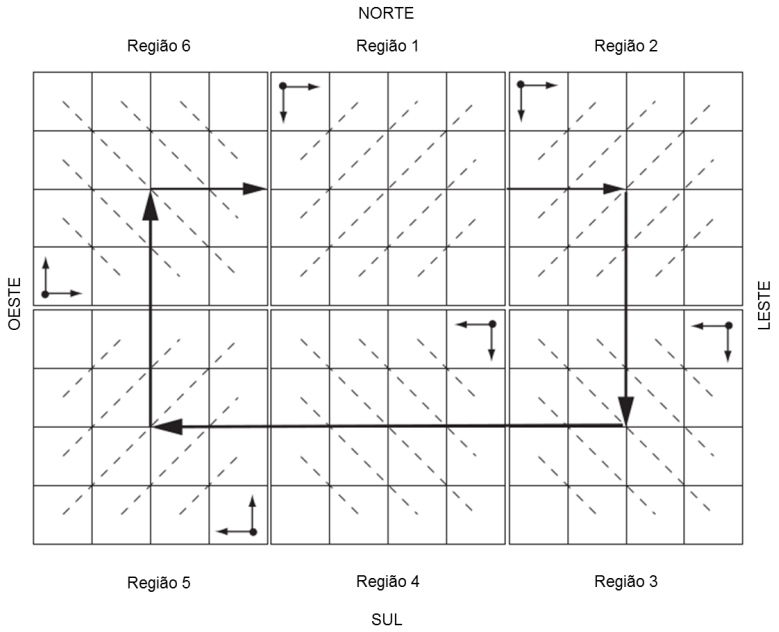


Figura 34 – Esquema de *Clock* 2DDwave com caminhos de *feedback*.
Fonte: Próprio Autor, adaptado de (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008).

Mais recentemente Campos com o trabalho *USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA*, propõe um novo modelo de esquema de *clock*, denominado *Universal, Scalable and Efficient Clocking* (USE) (CAMPOS et al., 2016). A proposta se basea no princípio que cada zona de *clock* deve ser disposta sequencialmente estabelecendo um fluxo de informação.

Um *grid* mínimo 4x4 proposto garante a organização correta das zonas de *clock*, associando as células QCA localizadas dentro de cada *subgrid* a uma das quatro zonas de relógio (CAMPOS et al., 2016). Dentro de cada uma desta zonas o programador pode escolher o tamanho do *subgrid*, que tradicionalmente é composto por um conjunto de células QCA 5x5. Duas linhas ou colunas adjacentes da grade USE sempre propagam informações em direções opostas, o que permite várias possibilidades de roteamento, alem de permitir de forma fácil caminhos de *feedback*.

Assim, a proposta do sistema de *clock* USE se torna atrativa, visto que é mais escalável, permitindo sintetizar circuitos maiores, e por já permitir a implementação de caminhos de *feedback* nativamente. Além disso, o USE permite a implementação de fios (*wire*) coplanares e multicamadas, o que resolve diversas limitações no roteamento e posicionamento apontados por Niemer (CHAUDHARY et al., 2005), os quais são problemas que devem ser abordados por um esquema de relógio QCA (como a diferença no comprimento entre fios, zonas de *clock* com larguras não uniformes, grande diferença no número de células entre as zonas, caminhos de *feedback* e área não utilizada). Adicionalmente, o esquema USE poder ser estendido, posicionando vários *grids* lado a lado em ambas as direções (horizontal e vertical). A Figura 35 apresenta o *grid* USE com o arranjo de zonas mínimo (4x4).

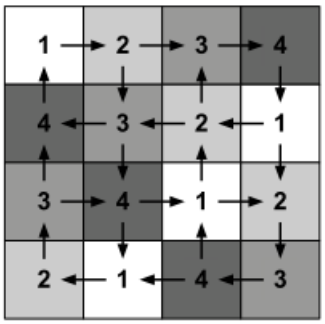


Figura 35 – *Grid* do esquema de *clock* USE.
Fonte: Campos, retirado de (CAMPOS et al., 2016).

Para validar a proposta do *clock* USE uma biblioteca de células (implementada manualmente) baseada no estilo de células padrão é proposta, denominada de QCA-ONE. Essa biblioteca possui algumas funções elementares para a implementação de circuitos, como por exemplo as células: INVERTER, MAJORITY, AND, OR, NOR, NAND, XOR, MUX2-1, LATCH-D e LATCH-SR. A Figura 36 apresenta as células que compõem a biblioteca QCA-ONE.

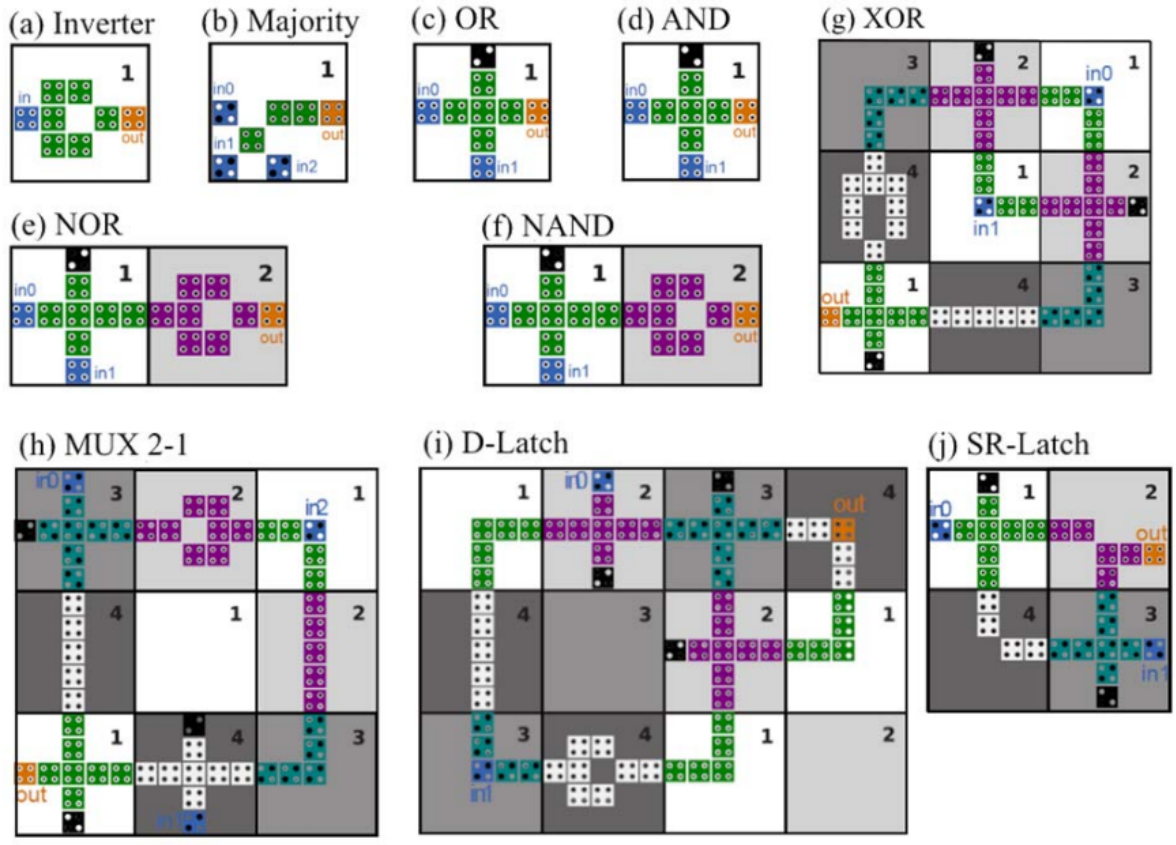


Figura 36 – Biblioteca proposta QCA-ONE.
Fonte: Campos, retirado de (CAMPOS et al., 2016).

Alguns experimentos foram efetuados a partir do USE e da biblioteca (QCA-ONE) proposta. Esses experimentos consideraram a implementação de alguns circuitos manualmente, os resultados apresentaram uma redução de área de até 5 vezes e diminuição do atraso de até 3 vezes em comparação com a esquemas de *clock* existentes. As características do esquema de *clock* proposto, especialmente a possibilidade de criar fios retos e pequenos *loops*, permitiram que diversas células combinacionais ou sequenciais fossem criadas, além de favorecer o posicionamento e o roteamento dessas células. Entretanto, neste trabalho ainda todo o fluxo é efetuado manualmente. Por último, vale destacar dentre os elementos propostos na implementação de Campos, a porta majoritária no esquema de *clock* USE. É importante destacar esse elemento, visto que foi a base para o elemento de *design* da majoritária proposto nesta tese para a ferramenta Fiction, a Figura 37 apresenta a representação gráfica da porta majoritária em questão no *subgrid* 5x5 .

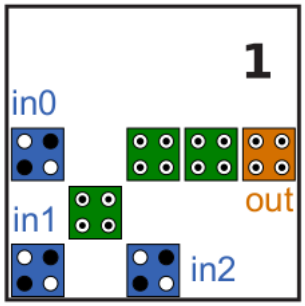


Figura 37 – Porta majoritária proposta por Campos no esquema USE.
Fonte: Campos, retirado de (CAMPOS et al., 2016).

Se utilizando dos conceitos propostos por Campos, Reis (REIS et al., 2016) propõe experimentos no sentido de especificar uma metodologia de síntese *standard cell* para QCA, na qual utiliza-se da biblioteca de células QCA-ONE, desenvolvida manualmente, proposta por Campos (TEIXEIRA, 2016). Esta síntese utilizou o ambiente da ferramenta QCADesigner (WALUS et al., 2004) como base, onde foram inseridos algumas extensões para aplicar o *clock* USE nos elementos automaticamente, além de permitir o carregamento das células da bilioteca QCA-ONE. Assim, o objetivo central foi evitar erros humanos na distribuição de *clock*, os quais são mencionados no tabalho como um dos principais problemas na síntese de circuitos para a tecnologia QCA. Diversos trabalhos discutem a limitação intrínseca da tecnologia, e os possíveis problemas na fabricação, como por exemplo Neimer (CHAUDHARY et al., 2005) e Reis (REIS, 2016). Entretanto, vale ressaltar que algumas etapas como as etapas de posicionamento e roteamento ainda são efetuadas manualmente e de forma específica para os circuitos apresentados como exemplos.

Como experimentos foram implementados dois *designs* com a proposta de metodologia, e utilizando os elementos da biblioteca QCA-ONE, juntamente com o esquema de *clock* USE, são eles: o somador completo de 1 bit (*Full Adder*) e um somador *ripple carry* de 8 bits. A Figura 38 apresenta o leiaute final do circuito *Full Adder*, assim como a Figura 39 apresenta o leiaute final do circuito *ripple Carry*.

É importante salientar que neste trabalho é possível verificar a utilização da porta majoritária proposta por Campos em uso, visto que nos experimentos do trabalho do próprio Campos, não é possível verificar a utilização da porta majoritária em seus leiautes finais. Embora os somadores apresentados neste trabalho não sejam os mais compactos observados na literatura, as implementações demostraram a viabilidade do desenvolvimento de circuitos baseados nas características do fluxo *standard cell* para tecnologias como o QCA.

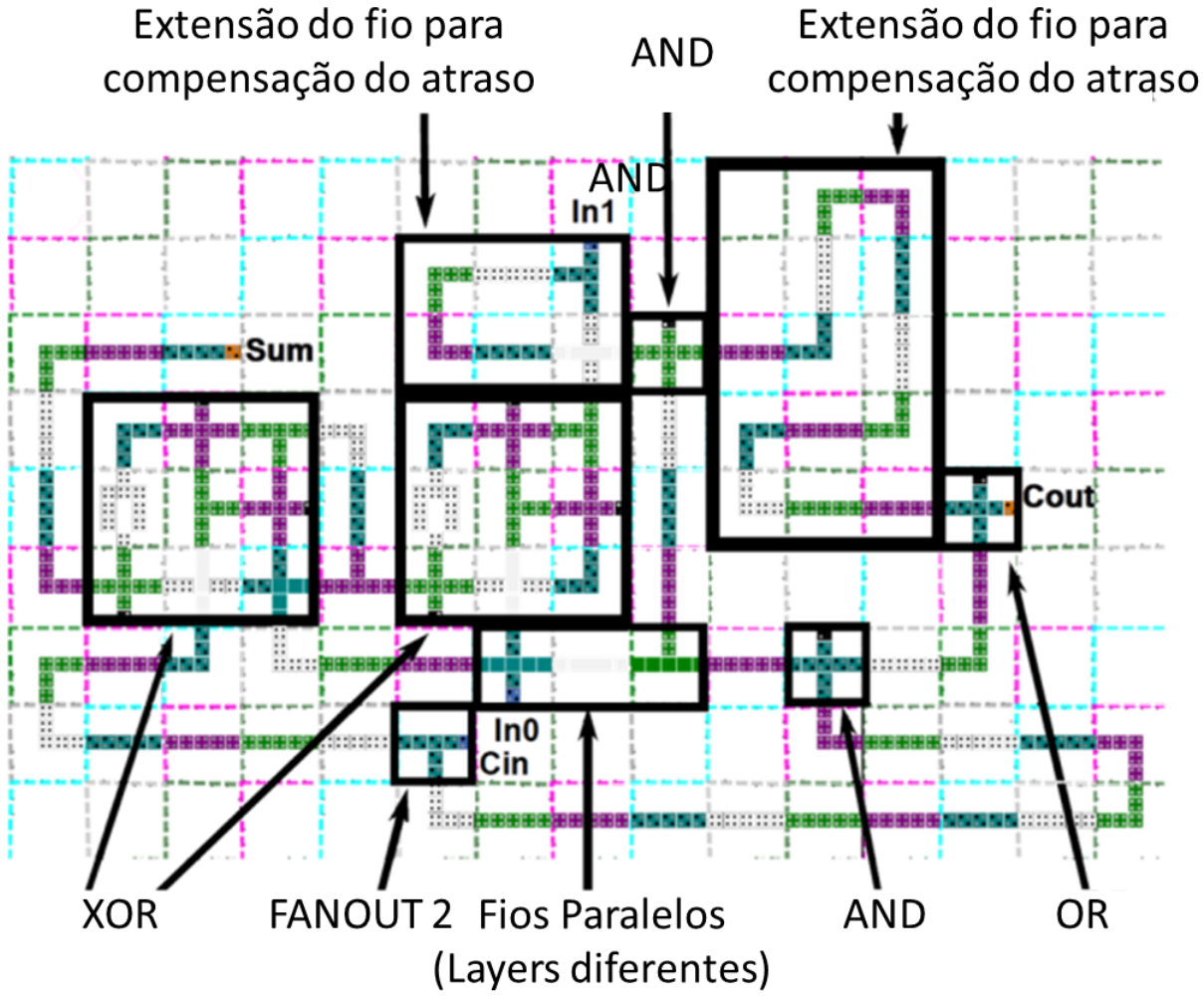


Figura 38 – Full Adder de 1 bit proposto por Dayane com *clock* USE.
Fonte: Próprio Autor, adaptado de (REIS et al., 2016).

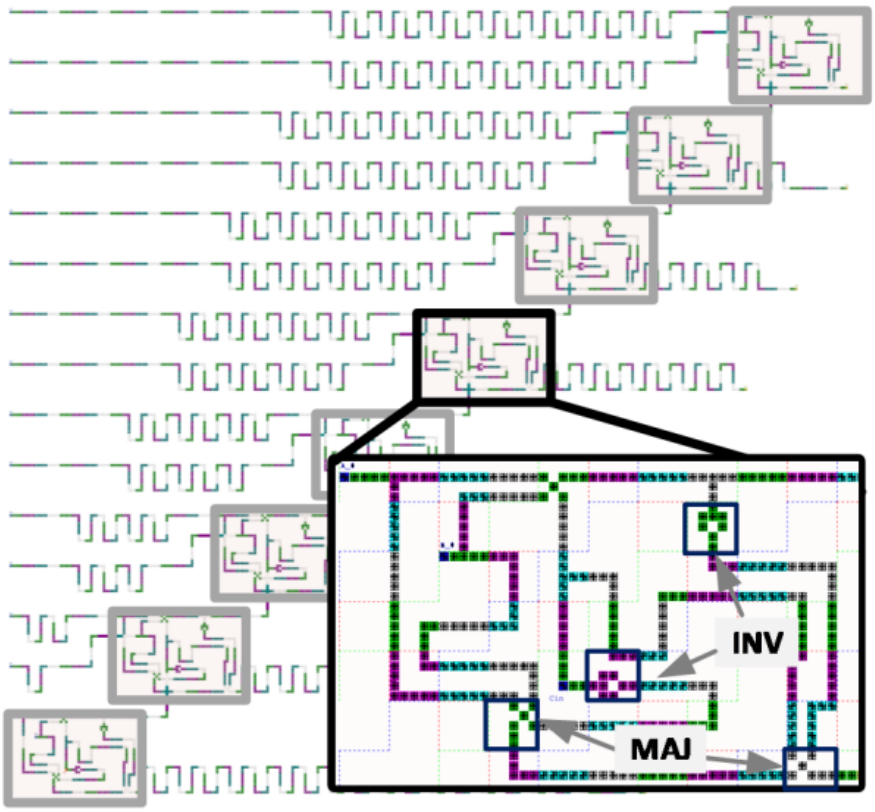


Figura 39 – Ripple Carry proposto por Reis com *clock* USE.
Fonte: Reis, retirado de (REIS et al., 2016).

O trabalho intitulado *Placement and Routing by Overlapping and Merging QCA Gates* proposto por Fontes em (FONTES et al., 2018) (FONTES JUNIOR, 2018), propõe um método de síntese semi-automática para as etapas de posicionamento e roteamento. A partir de trabalho anterior de Trindade (TRINDADE et al., 2016), o qual apresenta uma síntese baseada no algoritmo de Teodósio, onde o circuito é representado através de grafos acíclicos direcionados. Teodósio aplica uma etapa na ferramenta QCA-LG (TEODOSIO; SOUSA, 2007), a qual transforma a representação do circuito em um grafo planar, através da duplicação de nodos. Essa duplicação é efetuada com o objetivo de diminuir o *"wire crossing"* e facilitar o posicionamento, o que pode afetar o resultado final.

Por outro lado, a abordagem de Trindade tenta contornar essa limitação, considerando subgrafos em pontos de reconvergência. Porém, embora essa abordagem resolva o problema de inserção de lógica para a transformação planar, ela resulta em uma alta complexidade computacional, o que limita que os circuitos em questão não tenham mais que 10 portas. Fontes resolve as restrições da proposta de Trindade, com uma abordagem inovadora para o posicionamento e roteamento de circuitos QCA, baseando-se em sobreposição de leiautes criados sobre o esquema de *clock* USE.

É apresentada uma forma eficiente de particionamento criada sobre o paradigma de divisão e conquista a fim de reduzir o esforço na obtenção de soluções. Uma abordagem de *backtracking* similar ao problema das *N*-Rainhas é utilizado para criar a disposição dos elementos levando em consideração o grid USE. A Figura 40 apresenta de forma geral como o algoritmo realiza a busca por uma disposição de uma porção do circuito.

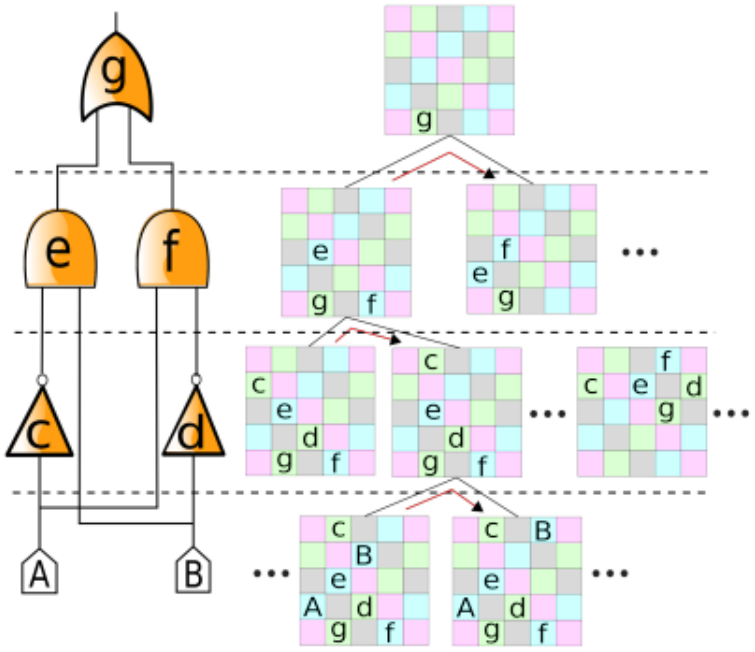


Figura 40 – Algoritmo para posicionamento de subgrafos de Fontes.
Fonte: Fontes, retirado de (FONTES et al., 2018).

Para lidar com a alta complexidade da etapa de busca do posicionamento ideal, é aplicado o particionamento do circuito levando em consideração zonas de reconvergência. Em outras palavras, são identificados subgrafos os quais são tratados separadamente, sendo gerado uma disposição inicial para todas as porções do circuito. As partições são criadas através de análises topológicas sobre o grafo, onde são identificados os impactos de caminhos reconvergentes com base nos conceitos de interações de redes complexas. O uso de portas majoritárias na construção de circuitos apresenta consideráveis benefícios ao projeto de circuitos QCA, reduzindo sua complexidade e melhorando a qualidade das soluções encontradas. Por fim, os subgrafos são analisados para que sejam encontradas áreas com sobreposição (*Overlapping*), e então, é efetuada a união (*merge*) entre as partes sobrepostas. A Figura 41 apresenta uma etapa do algoritmo onde dois subgrafos (partição 1 e 2) são analisados com o objetivo de identificar uma sobreposição, e assim, gerar uma solução parcial.

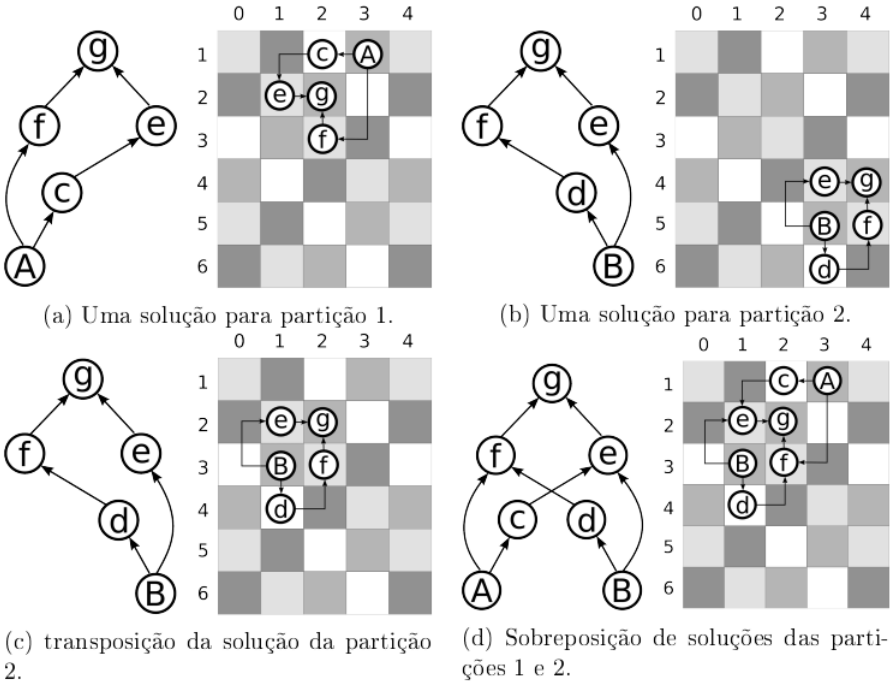
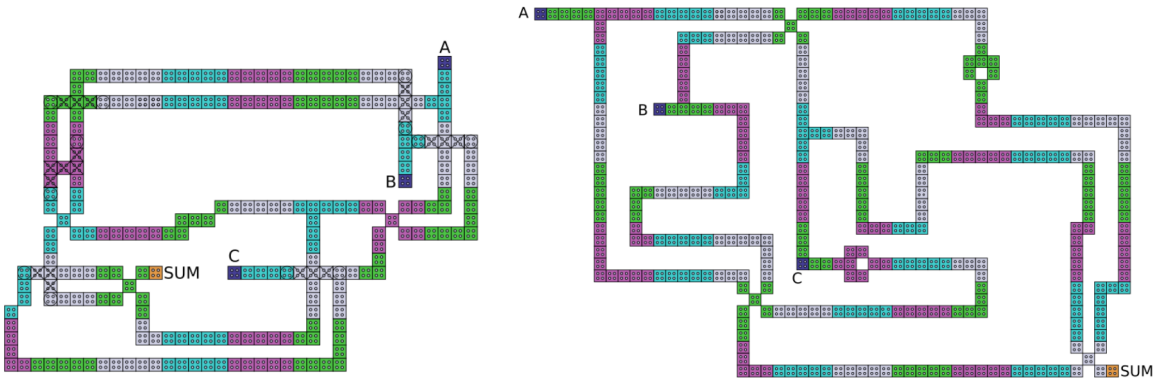


Figura 41 – Etapas de união dos subgrafos sobrepostos.
Fonte: Geraldo, retirado de (FONTES JUNIOR, 2018).

Os resultados obtidos nesse trabalho apresentam redução na área total das soluções de mais de 50% em comparação com abordagens anteriores de Reis(REIS et al., 2016). Por fim, as soluções geradas são validadas na ferramenta de simulação QCADesigner. A Figura 42 apresenta a solução proposta por Fontes para o mesmo circuito *Full Adder* de 1 bit em comparação com a versão apresentada por Reis em (REIS et al., 2016).



(a) (FONTES JUNIOR, 2018) (b) (REIS et al., 2016)

Figura 42 – Full Adder de 1 bit proposto por Fontes em relação a solução de Reis.
Fonte: Fontes, retirado de (FONTES JUNIOR, 2018).

Embora a solução de Fontes tenha apresentado bons resultados frente aos trabalhos da literatura, sua solução apresenta diversos problemas quanto a questão de escalabilidade. Desenvolvida na linguagem JAVA, alguns experimentos foram efetuados, onde foi identificado que conforme aumenta-se o circuito de entrada, a heurística de *backtracking* apresenta sérias limitações quanto ao *runtime*. Outra limitação encontrada é que a heurística necessita de um arquivo de configuração inicialmente, onde são definidos alguns espaçamentos (zonas de *clock*) entre os elementos do circuito (portas), esses espaçamentos precisam ser calculados manualmente por força bruta para cada novo circuito a ser submetido pela heurística, o que torna a abordagem impraticável para grandes *designs*.

Mais recentemente Walter propôs dois métodos de síntese automáticos, os quais se tornaram o estado da arte em síntese automática para o paradigma FCN. O primeiro método proposto por Walter no artigo intitulado *An exact Method for Design Exploration of Quantum-dot Cellular Automata* (Walter et al., 2018), apresenta uma solução alternativa automática para o roteamento e posicionamento das tecnologias QCA e NML baseado em utilização de resolvedores SAT. O resolvedor SAT utilizado é o *SMT Solver Z3* (MOURA; BJØRNER, 2008). A proposta tenta solucionar algumas limitações dos métodos propostos até então, visto que devido as restrições impostas tradicionalmente pelos projetos (sistemas de *clock*, utilização de mais camadas para roteamento, etc.), são necessárias a utilização de heurísticas que não garantem a obtenção da melhor solução. Dessa forma, este trabalho através da modelagem de cláusulas SAT, permite que o método implementado consiga gerar leiautes que atendem as restrições iniciais impostas. O método foi denominado *Exact*, e pode ser invocado no ambiente da ferramenta Fiction (WALTER et al., 2019) através do comando **exact**.

Neste sentido, o trabalho apresenta a formulação simbólica dos elementos e das possíveis restrições a serem consideradas na execução do método SAT. Diversos experimentos foram realizados considerando métodos heurísticos estado-da-arte como o de Fontes (FONTES et al., 2018) (apresentadas na Figura 43 com o identificador [12]). Primeiro, os resultados demonstram que, de fato, determinar resultados com a área mínima, conforme realizado através de métodos SAT é computacionalmente caro e requer uma quantidade significativa de tempo. No entanto, a abordagem proposta permite determinar os resultados desejados em tempo razoável para circuitos relativamente pequenos. Além disso, mesmo que a abordagem heurística tenha executado em uma máquina com desempenho inferior, o tempo de execução é notavelmente alto em relação a abordagem baseada em SAT.

O diferencial desta solução é permitir pela primeira vez avaliar com precisão a qualidade das soluções heurísticas propostas. Ficando evidente a qualidade da solução gerada, visto que a abordagem de Fontes produziu resultados quase três vezes maiores. Neste sentido, o trabalho evidencia a lacuna ainda aberta para a proposição de

soluções heurísticas melhores. A Figura 43 apresenta os resultados comparativos dos métodos em relação ao número de *subgrids 5x5* denominados por (*área*), essa contagem considera a quantidade de linhas e colunas utilizados com elementos do *subgrid 5x5*, assim como o caminho crítico (CC) e o tempo de execução (*t in s*) em segundos.

	Heurística [12]			Exact		
Circuito	Área	CC	t em (s)	Área	CC	t em (s)
2:1 MUX	20×25	5	9	15×15	5	< 1
XOR	20×35	7	11	15×15	5	< 1
XNOR	30×30	8	13	20×20	8	2
Half adder	35×30	8	55	25×25	10	13
c17	50×30	13	15	25×30	16	56
ParGen	45×50	14	27	35×30	14	791
ParCheck	50×70	14	3014	20×60	16	1140
4:1 MUX	55×40	19	9612	35×35	22	5131

Figura 43 – Resultados comparativos do método Exact vs Método de Fontes.
Fonte: Próprio Autor, adaptado de (Walter et al., 2018)

Embora a solução proposta apresente os melhores resultados da literatura, a solução possui alto custo computacional, tornando-se impraticável para circuitos reais. Neste sentido, são apresentados apenas soluções para pequenos circuitos, o que evidencia que o método não se torna escalável.

Considerando as limitações dos métodos heurísticos identificados no trabalho anterior, Walter (Walter et al., 2019) propõe uma nova abordagem heurística para síntese automática de circuitos para a tecnologia QCA e NML. O método apresentado no artigo intitulado *Scalable Design for Field-coupled Nanocomputing Circuits*, apresenta um método escalável baseado em OGDs, denominado *Ortho*, o qual pode ser invocado no ambiente da ferramenta Fiction (WALTER et al., 2019) pelo comando **ortho**. Essa abordagem apresenta uma vantagem significativa de tempo de execução em comparação com a abordagem *exact* conforme a entrada aumenta de tamanho. Embora os leiautes gerados por esse método não sejam considerados ótimos em termos de área, essa técnica é aplicável mesmo para circuitos maiores, e fornece resultados em tempo de execução razoável (Walter et al., 2019).

Os resultados estão resumidos na tabela da Figura 44, onde a contagem de portas (*gates*) refere-se ao número de portas 2-AND, 2-OR e INVERTER. A coluna E/S fornece informações sobre o número de entradas e saídas principais do circuito. As colunas a seguir listam, para cada técnica de projeto considerada, a dimensão (linhas e colunas) resultante em termos de zonas de relógio, que têm o tamanho do *subgrid 5x5*, caminho crítico (CC) e o tempo de execução necessário, respectivamente (em segundos da CPU). Observe que os processos de *design* que não puderam ser concluídos dentro de um tempo limite de 10 horas são apresentados com um traço ($-$).

Os resultados confirmam claramente que o método proposto é aplicável a grandes *designs*. Mais precisamente, em relação ao método *exact* (referenciado na Figura por [35]) e o método proposto por Fontes (identificado pela referência [20]) são capazes de realizar a execução apenas com circuitos pequenos com até quatro entradas primárias e não mais do que duas dezenas de portas, ao contrário da solução proposta que pode lidar com funções com milhares de elementos.

Benchmark			Heurística Fontes[20]			Heurística Exact [35]			Heurística Ortho		
Nome	Número Portas	E / S	Dimensão	CC	t em (s)	Dimensão	CC	t em (s)	Dimensão	CC	t em (s)
2:1 MUX	5	3 / 1	4 × 5	5	9	3 × 3	5	< 1	4 × 2	5	< 1
XOR	6	2 / 1	4 × 7	7	11	3 × 3	5	< 1	5 × 3	7	< 1
XNOR	8	2 / 1	6 × 6	8	13	4 × 4	8	2	6 × 4	9	< 1
Half adder	10	2 / 2	7 × 6	8	55	5 × 5	10	13	7 × 5	11	< 1
ParGen	14	3 / 1	9 × 10	14	27	7 × 6	14	791	10 × 7	16	< 1
ParCheck	21	4 / 1	10 × 14	14	3014	4 × 12	16	1140	15 × 10	24	< 1
4:1 MUX	18	4 / 1	11 × 8	19	9612	7 × 7	22	5130	12 × 7	18	< 1
c17	15	5 / 2	10 × 6	13	15	4 × 6	16	56	12 × 4	14	< 1
c432	551	36 / 7	—	—	*TO*	—	—	*TO*	426 × 161	584	< 1
c499	963	41 / 32	—	—	*TO*	—	—	*TO*	690 × 306	995	< 1
c1355	1515	41 / 32	—	—	*TO*	—	—	*TO*	1243 × 369	1611	< 1
c1908a	2043	33 / 25	—	—	*TO*	—	—	*TO*	1540 × 536	2077	< 1
c2670a	2455	155 / 64	—	—	*TO*	—	—	*TO*	1756 × 760	2511	< 1
c3540a	3588	50 / 22	—	—	*TO*	—	—	*TO*	2523 × 1111	3639	1
c5315a	5478	177 / 123	—	—	*TO*	—	—	*TO*	3857 × 1751	5577	2
c6288	6928	32 / 32	—	—	*TO*	—	—	*TO*	5714 × 1246	6957	2
ctrl	498	7 / 25	—	—	*TO*	—	—	*TO*	356 × 149	495	< 1
router	658	60 / 3	—	—	*TO*	—	—	*TO*	488 × 231	717	< 1
int2float	699	11 / 7	—	—	*TO*	—	—	*TO*	514 × 196	708	< 1
i2c	3508	133 / 127	—	—	*TO*	—	—	*TO*	2515 × 1123	3632	1
bar	8592	135 / 128	—	—	*TO*	—	—	*TO*	6547 × 2180	8724	6
sin	14314	24 / 25	—	—	*TO*	—	—	*TO*	10549 × 3828	14374	14
voter	39476	1001 / 1	—	—	*TO*	—	—	*TO*	30542 × 9935	40476	10

Número de Portas : Contagem de portas 2-AND/2-OR/NOT + E/S CC : Caminho Crítico E/S : Número de Entradas e Saídas Primárias
Dimensão : área ocupada pelas zonas de *clock (ou seja, blocos células 5x5)* t em (s) : Tempo em segundos TO : Timeout depois de 10 horas

Figura 44 – Resultados comparativos do método Ortho vs Exact vs Método de Fontes.
Fonte: Próprio Autor, adaptado de (Walter et al., 2019)

Contudo, apesar do método ser escalável para grandes circuitos com milhares de portas, ainda apresenta resultados não tão otimizados quanto a abordagem baseada em SAT (Walter et al., 2018). Provavelmente, uma das razões pode ser que a heurística tenha atingido algum mínimo local na busca por soluções. Além disso, a estrutura utilizada para a representação da lógica no método, denominada *3-graph*, foi a da lógica AOI. Em outras palavras, a lógica é decomposta em portas AND, OR de 2 entradas juntamente com os inversores. Assim, como apontado pelos experimentos anteriores, essas estruturas podem inviabilizar a configuração de uma solução mais otimizada diretamente em majoritárias, como é o caso da utilização da estrutura MIG.

3.4 Conclusões

Esta seção apresenta a Tabela 1, onde é efetuada a comparação entre os trabalhos correlatos discutidos na seção anterior. Na coluna mais a esquerda é possível identificar o nome do trabalho, assim como nas colunas subsequentes (mais a direita) são apresentadas as comparações sobre as seguintes características: esquema de *clock*, estrutura de dados, capacidade de geração de leiaute automático, biblioteca de células e tecnologia foco do trabalho.

Tabela 1 – Tabela comparativa dos trabalhos correlatos

Trabalho	Esquema *Clock*	Estrutura de dados	Leiaute Automático	Biblioteca de Células	Tecnologia
(ZHANG et al., 2004)	1DLent	SOP	-	13 Funções	QCA
(MOMENZADEH et al., 2005)	1DLent	AOI	-	13 Funções AOI	QCA
(SHANG, 2010)	1DLent	SOP	-	40 Funções	QCA
(AMARÚ; GAILLARDON; MICHELI, 2013a)	-	BDD-Maj	-	-	CMOS
(AMARÚ; GAILLARDON; MICHELI, 2014a)	-	MIG	-	-	CMOS
(AMARÚ; GAILLARDON; MICHELI, 2015)	-	MIG	-	-	CMOS
(AMARÚ et al., 2015)	-	MIG	-	-	FPGA
(AMARÚ; GAILLARDON; MICHELI, 2016)	-	MIG	-	-	CMOS/FPGA
(NEUTZLING et al., 2017)	-	TLF	-	-	-
(NEUTZLING et al., 2015)	-	TLF	-	Funções TL	CMOS
(SILVA et al., 2018)	USE	TLG	-	-	QCA
(VANKAMAMIDI; OTTAVI; LOMBARDI, 2008)	2DDwave	-	-	-	QCA
(CAMPOS et al., 2016)	USE	-	-	QCA-ONE	QCA
(REIS et al., 2016)	USE	-	-	QCA-ONE	QCA
(FONTES et al., 2018)	USE	AOI/TL	SIM	QCA-ONE	QCA
(Walter et al., 2018)	USE	SMT-Solver	SIM	QCA-ONE	QCA
(Walter et al., 2019)	2DDwave	OGD/AOI	SIM	QCA-ONE	QCA

Considerando as diferentes tecnologias testadas e prototipadas nos útimos anos, ainda não existe uma específica a qual venha, de fato, a substituir a tecnologia atual de construção de circuitos denominada CMOS. Dessa forma, existem diversas propostas, as quais em sua maioria utiliza a sua lógica baseada em portas majoritárias para construção dos seus elementos. Neste sentido, a tecnologia QCA tem ganhado destaque em trabalhos como (ZHANG et al., 2004), (MOMENZADEH et al., 2005) e (SHANG, 2010), os quais apresentam propostas de métodos de síntese utilizando as ferramentas e estruturas de dados tradicionais da tecnologia CMOS. Em grande parte, esses métodos apresentam algumas limitações, justamente em função da estrutura de dados utilizada para a representação da lógica majoritária ou dos métodos e ferramentas utilizados que consideram as características da tecnologia CMOS para realização da síntese.

Por outro lado, trabalhos mais recentes como de Amarú (AMARÚ; GAILLARDON; MICHELI, 2013a,b, 2014b,a, 2015, 2016), apresentam uma nova abordagem sobre a síntese com foco em majoritária. O grande diferencial é que são propostas estruturas de dados que representam diretamente a lógica da porta majoritária, o que viabiliza que os métodos considerem otimizações sem precisar traduzir a representação para elementos da tecnologia. Embora já altamente estudados, ainda existem discussões em aberto para síntese automática considerando essas tecnologias.

Outra abordagem considerada nos últimos anos é a utilização de TLF na síntese de elementos baseados em majoritárias, embora ainda não investigada profundamente, os trabalhos como (NEUTZLING et al., 2013, 2015, 2017) e (SILVA et al., 2018) apresentam indícios que TLGs se tornam uma ótima alternativa para esses problemas. Entretanto, ainda é preciso o avanço de diversos pontos para que essa abordagem se estabeleça como referência em síntese dessas tecnologias. Uma das restrições é a questão da definição da estrutura de dados para representar as informações, assim como seu conjunto de axiomas, o que permitirá a concepção de uma nova álgebra booleana baseada nas características de TLG, como aconteceu na estrutura MIG.

Contudo, nos últimos anos diversos trabalhos referentes a questão específicas da tecnonologia QCA vem sendo discutidas, como por exemplo: o desenvolvimento de bibliotecas de células como no trabalho de Campos (TEIXEIRA, 2016), assim como o desenvolvimento de novas propostas de esquemas de *clock*, como em (CAMPOS et al., 2016) e (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008), ou até mesmo propondo métodos de síntese semi-automáticos, como os trabalhos de Reis (REIS et al., 2016) e Fontes (FONTES et al., 2018). Esses trabalhos corroboram para o desenvolvimento e amadurecimento de metodologias de síntese automática para as tecnologias FCN, em especial, a tecnologia QCA. Neste sentido, as abordagens propostas por Walter nos últimos anos (Walter et al., 2018) e (Walter et al., 2019), evidenciam o amadurecimento da tecnologia QCA, sendo a mais próxima da substituição da tecnologia CMOS. Estes trabalhos são a referência em métodos automáticos para a etapa de síntese física QCA.

Este capítulo apresentou uma discussão acerca dos principais trabalhos correlatos sobre síntese para tecnologias no paradigma FCN, baseadas em portas majoritárias assim como os principais métodos de síntese. Existem diversos pontos em aberto para pesquisa no que tange a síntese automática de tecnologias do paradigma FCN, em destaque cita-se os métodos baseados em MIG e métodos que utilizam TLG, em conjunto com esses tópicos, as meta-heurísticas apresentam uma ótima alternativa para varrer o conjunto de soluções no processo de otimização. No próximo capítulo será apresentada a proposta deste trabalho sobre uma concepção de metodologia de síntese automática para a tecnologia QCA.

4 UMA METODOLOGIA DE SÍNTESE AUTOMÁTICA COM O FOCO NA TECNOLOGIA QUANTUM-CELLULAR-AUTOMATA

Neste capítulo será apresentada a proposta deste trabalho, a qual visa a proposição de uma nova metodologia de síntese para a tecnologia QCA. Neste sentido, inicialmente será apresentada a discussão a qual culminou na formulação da proposta de metodologia. Logo após, com o objetivo de validar a metodologia proposta, será apresentada a prototipação efetuada a partir do ambiente da feramenta Fiction (WALTER et al., 2019). Além disso, serão apresentadas as modificações nos métodos presentes no ambiente da ferramenta, os quais são o estado-da-arte em síntese automática para as tecnologias QCA e NML. Por fim, serão apresentadas as conclusões do capítulo.

4.1 Metodologia para Síntese QCA

Conforme discutido anteriormente no capítulo 2.1, existem diferentes metodologias para execução da síntese de circuitos. Dentre as metodologias existentes, a abordagem *semicustom* é a mais estabelecida na indústria, principalmente, quando se trata da abordagem denominada células padrão a qual possui processos mais maduros e bem confiáveis utilizados em larga escala. Neste sentido, este trabalho tenta utilizar os conceitos já bem estabelecidos do fluxo de células padrão como base, para propor uma metodologia automática para a tecnologia QCA. Este fluxo tem alto potêncial de adaptação a nova tecnologia, visto que uma de suas principais características é a modularidade, desde que as interfaces entre os módulos sejam respeitados.

É importante salientar que a proposta de abordagem de síntese automática QCA baseada em células padrão não é nova. Dayane em (REIS et al., 2016) propõe uma abordagem baseada em células padrão. Contudo, sua abordagem utiliza uma pequena biblioteca de células QCA desenvolvida manualmente, proposta em (TEIXEIRA, 2016), além de executar a maior parte de suas etapas manualmente, e sem considerar a etapa de síntese lógica.

De forma geral essa metodologia se divide em três grandes etapas, síntese de alto-nível ou estrutural, síntese lógica e síntese física. Tradicionalmente, a primeira etapa possui como objetivo descrever o circuito de forma geral, no nível de registradores, sem muitas preocupações com especificidades da tecnologia de implementação.

Logo após é aplicada a etapa de síntese lógica, a qual possui como objetivo geral transformar a descrição de mais alto nível em uma descrição já mais próxima da tecnologia a ser implementada, considerando suas especificidades. Contudo, nesta etapa, tipicamente existe a tradução da descrição de mais alto nível para uma estrutura de dados, onde são aplicados métodos que visam otimizar a descrição inicial em função de alguns objetivos e características da tecnologia alvo. Posteriormente, ainda durante a síntese lógica, a fase de mapeamento tecnológico tenta encontrar e vincular porções do circuito a células de uma biblioteca de células, considerando uma função custo.

Na última etapa a síntese física é aplicada, o mapeamento tecnológico resulta em uma *netlist* de portas, as quais são instâncias de uma bibliotecas de células previamente caracterizadas e testadas. Na síntese física existe a preocupação com aspectos inerentes a tecnologia alvo, como o posicionamento dos elementos, roteamento e distribuição de sinal de relógio (*clock*). Por fim, como resultado desta etapa é gerada a especificação do leiaute final do circuito para a manufatura. A Figura 45 apresenta o fluxo da metodologia de células padrão com as fases de cada etapa do fluxo.

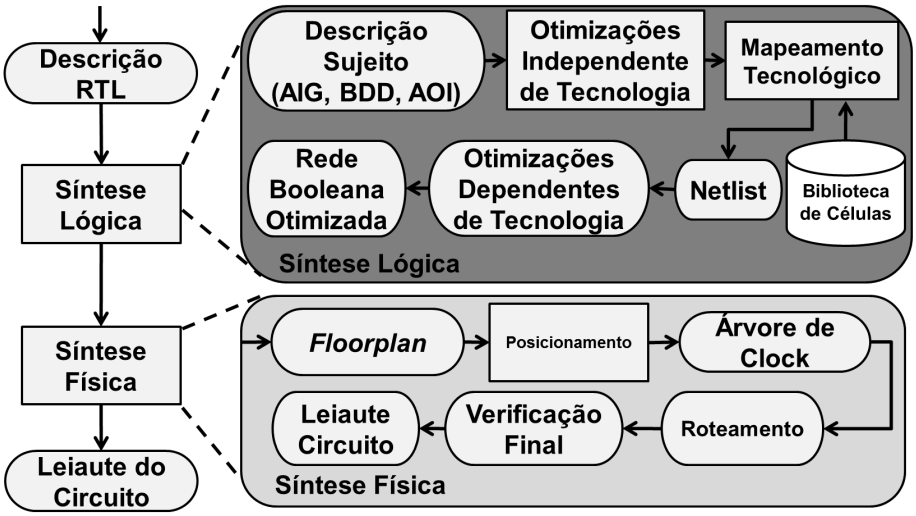


Figura 45 – Metodologia *Standard Cell* para a tecnologia CMOS.
Fonte: Próprio Autor.

Esse fluxo se estabeleceu na indústria devido a sua modularidade e grande confiabilidade no processo, em função da grande confiabilidade e previsão do comportamento das células quando conectadas em um circuito real. Considerando o exposto anterior, este trabalho tenta utilizar as características já bem estabelecidas da metodologia *standard cell* para propor uma metodologia de síntese automática para a tecnologia QCA. Dessa forma, considerando os trabalhos da literatura e as oportunidades ainda em aberto sobre síntese QCA, é possível vislumbrar uma proposta de metodologia de síntese QCA que se utilize das principais soluções propostas de forma integrada e automática.

Por outro lado, embora existam abordagens estado da arte, quando se refere a síntese para circuitos FCN, elas apenas tratam das etapas específicas do projeto de circuitos nessas tecnologias, ou seja, não existe nenhuma preocupação com a etapa de síntese lógica por exemplo. De forma geral, cabe ressaltar ainda que a composição desses ambientes de síntese por si só juntamente com as bibliotecas também não são capazes de implementar diretamente a metodologia proposta. Parte das contribuições desse trabalho também podem ser observadas na modificação dos ambientes e bibliotecas para permitir a integração e funcionamento adequado da metodologia.

Neste sentido, para a etapa de síntese de alto nível e síntese lógica pode-se conceber a utilização de formatos bem estabelecidos pela comunidade para descrição de circuitos, como por exemplo, a utilização do formato de descrição Verilog.

A estrutura de dados é um dos elementos mais importantes tratando-se da etapa de síntese lógica. Dessa forma, segundo o estado-da-arte, a estrutura MIG (AMARÚ; GAILLARDON; MICHELI, 2014a) é a estrutura de dados mais indicada para síntese de tecnologias baseadas em majoritárias, como é o caso da tecnologia QCA. Desse modo, essa estrutura foi a escolha natural dos autores para a representação das descrições.

Diretamente atrelados a forma de representação, são os seus métodos de otimização, sendo capazes de a partir de uma descrição inicial encontrar uma solução com menor número de elementos (no caso de otimização por área), ou com o menor caminho crítico no circuito (no caso de atraso), sem modificar a funcionalidade lógica. Assim sendo, o conjunto de algoritmos de otimizações MIG propostos por Amarú (AMARÚ; GAILLARDON; MICHELI, 2016), os quais são o estado-da-arte em otimização MIG se tornam a melhor alternativa para a implementação dessa fase da síntese lógica. Na última fase da síntese lógica é aplicado o mapeamento tecnológico, o qual nesta abordagem necessita da utilização de uma biblioteca de células. Nesse sentido, tendo em vista a literatura e os ambientes *open-source*, a biblioteca QCA-ONE foi selecionada por ser a mais utilizada.

Por último, considerando a etapa de síntese física, entende-se que as fases de roteamento, posicionamento e distribuição de sinais de *clock* são as mais importantes. Buscando uma reprodução mais análoga a etapa de síntese física, foram selecionados os métodos propostos mais recentemente por Walter, em especial o método heurístico *Ortho* (Walter et al., 2019) por possuir claramente as fases citadas anteriormente. Considerando este cenário, a concepção conceitual da metodologia proposta pode ser observada na Figura 46. Com o objetivo de validar e verificar a viabilidade da metodologia proposta, foi efetuada uma prototipação. A próxima seção apresentará os experimentos de prototipação da metodologia.

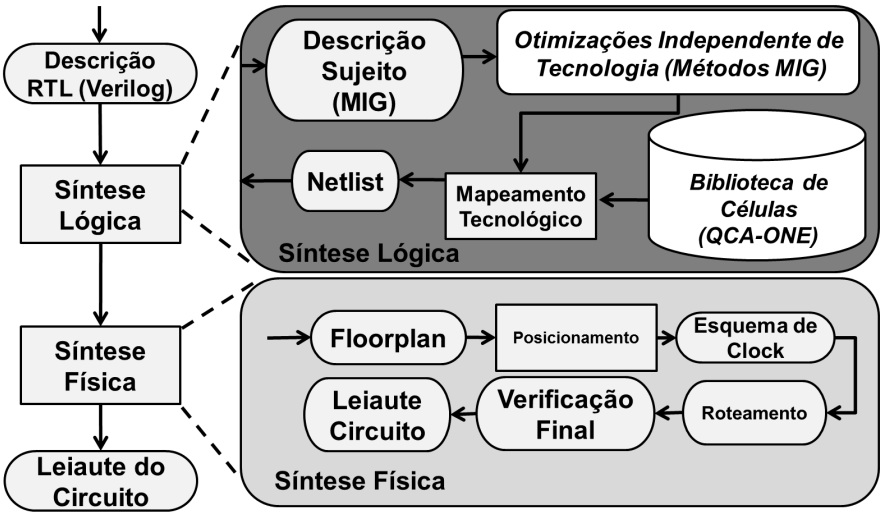


Figura 46 – Fluxo conceitual da metodologia de síntese QCA proposta.
Fonte: Próprio Autor.

4.2 Prototipação da Metodologia Proposta

Inicialmente para a etapa de síntese de alto nível, pode-se citar o *framework* EPFL Logic Synthesis Libraries (SOEKEN et al., 2018), o qual possui uma base sólida para a utilização dos mais diferentes formatos de descrição, dentre eles o mais utilizado, o formato Verilog. Esta estrutura de representação e *parsers* de descrições é oferecida através da biblioteca *Lorina*, assim como a interface com usuários através de comandos *Shell* denominada CLI (*Command-Line Interface*) pode ser disponibilizada com o acréscimo da biblioteca *Alice*.

Considerando a estrutura de dados selecionada, o MIG, existem algumas possibilidades de implementações *open-source*. Desse modo, é possível utilizar a modelagem da interface da estrutura de dados MIG disponibilizada pela ferramenta de síntese lógica *Cirkit* proposta também pela EPFL (SOEKEN et al., 2019c), assim como a própria representação da estrutura MIG da biblioteca *Mockturtle*. Neste sentido, a biblioteca *Mockturtle* do *framework* EPFL Logic Synthesis Libraries (SOEKEN et al., 2019b) provê um conjunto de algoritmos de otimização MIG baseados em Amarú (AMARÚ; GAILLARDON; MICHELI, 2016). A luz da abordagem de células padrão, a qual vislumbra-se uma fase de otimizações livres de tecnologia durante a síntese lógica, essa biblioteca se tornou a melhor escolha com seus métodos de otimização baseados em MIG.

Para a etapa de síntese física a ferramenta Fiction foi selecionada, isso se deve ao fato que essa ferramenta Fiction é a ferramenta estado-da-arte para a etapa de síntese física em tecnologias FCN, provendo duas abordagens distintas para síntese automática. Portanto, a abordagem baseada no método escalável denominado *ortho* apresenta as melhores características, visto que possui um método escalável aplicável a circuitos com alta complexidade. Além disso, o método heurístico utiliza como biblioteca de células padrão a biblioteca QCA-ONE, além de utilizar o esquema *clock* 2DDDWAVE e permitir uma capacidade maior de modificações possíveis.

Ademais, a ferramenta é desenvolvida utilizando como base os módulos do *framework* EPFL, o que viabiliza a rápida integração de novas soluções propostas com esse conjunto de bibliotecas. Assim, almejando uma prototipação rápida, esse ambiente foi escolhido para realizar uma integração dos diferentes trabalhos afim de obter um fluxo automático de síntese com o enfoque na tecnologia QCA. A Figura 47 ilustra o fluxo implementado de prototipação da metodologia proposta, é importante observar que algumas etapas da estão sombreadas em preto, essas etapas possuem as principais modificações efetuadas por esse trabalho para viabilizar a prototipação da metodologia.

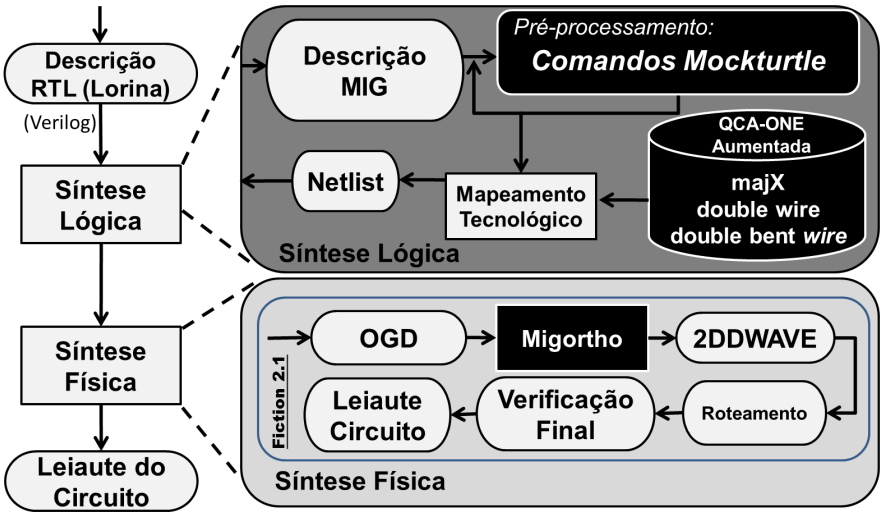


Figura 47 – Metodologia de síntese QCA proposta.
Fonte: Próprio Autor.

Como citado anteriormente, a etapa de síntese de alto nível a partir de uma descrição HDL pôde ser facilmente implementada através do conjunto de bibliotecas *Lorina* e *Alice*, já previamente incorporadas nativamente na ferramenta *Fiction*. Contudo, para permitir o carregamento de descrições de majoritárias diretamente em uma estrutura MIG, foi preciso incorporar a ferramenta um conjunto de classes da ferramenta *Cirkit* e a biblioteca *Mockturtle*. Assim, com pequenas modificações no conjunto de classes foi possível realizar o carregamento de uma descrição Verilog diretamente para uma estrutura MIG. Com a inclusão da biblioteca *Mockturtle*, além da possibilidade de representação da lógica com MIGs, também foram adicionados diversos métodos de otimização baseados nessa estrutura. Dessa maneira foi possível viabilizar a implementação da etapa de síntese lógica explorando otimizações na estrutura MIG, a seguir serão apresentados os conceitos sobre a a biblioteca *Mockturtle*, e as implementações realizadas para explorar as otimizações no fluxo durante a síntese lógica.

4.2.1 Mockturtle Library - Métodos MIG

A filosofia da biblioteca *Mockturtle* é baseada em uma arquitetura de 4 camadas, que dependem uma da outra em uma ordem linear, conforme apresenta a Figura 48. A função da camada inferior, é fornecer uma *API* (*Application Programming Interface*) com as representações de redes booleanas denominadas *Network*. Esta camada define convenções e nomenclaturas para tipos e métodos em classes que implementam as interfaces com diferentes tipos de *Networks*. A API do elemento *Network interface*, não fornece nenhuma implementação para uma *Network* apenas sua modelagem (SOEKEN et al., 2019b).

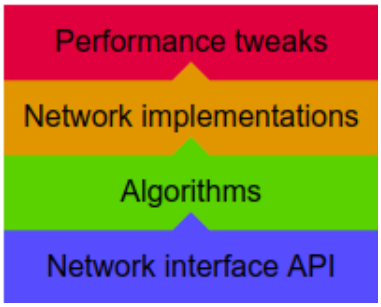


Figura 48 – Arquitetura API *Mockturtle*.
Fonte: Soeken, retirado de (SOEKEN et al., 2019b).

Os algoritmos (*Algorithms*), na segunda camada, são implementações em termos de funções genéricas, as quais recebem como entrada uma instância de algum tipo de *Network* hipotética e exigem que esse tipo implemente todas as interfaces obrigatórias e algumas opcionais. A camada algoritmos, no entanto, não considera a implementação interna da *Network* de entrada. Por exemplo, ela não considera como os elementos lógicos da rede são representados internamente.

A terceira camada (*Network Implementations*) consiste em implementações de fato da *Network*, neste caso, já existem algumas pré-definidas, por exemplo, AIGs (AIGER, 2012), MIGs (AMARÚ; GAILLARDON; MICHELI, 2016) ou redes *k*-LUT (BERKELEY, 2014b). Os algoritmos da segunda camada podem ser invocados em instâncias desses tipos de *Network*, se eles implementarem as interfaces necessárias. Assim, as asserções em tempo de compilação estática garantem que a compilação seja bem-sucedida apenas para as implementações de *Network*, as quais fornecem todos os tipos e métodos necessários.

Finalmente, para melhorar o desempenho, alguns detalhes algorítmicos podem ser especializados para alguns tipos de *Network*, com base em sua implementação interna. Isso pode ser efetuado para cada *Network* individualmente, sem afetar a implementação do algoritmo genérico nem a implementação de outros tipos de rede. Esta camada é representada na Figura 48, pela quarta camada (*Performance Tweaks*) na arquitetura *Mockturtle*.

Logo após o estudo do módulo *Mockturtle*, ele foi adicionado na ferramenta Fiction, para isso, foi preciso uma série de modificações em seus arquivos de construção e sincronização dos arquivos de dependências, visto que a biblioteca *Mockturtle* possui diversos arquivos duplicados em relação aos módulos já utilizados (*Lorina*, *Alice*). Além disso, foi preciso instalar algumas dependências de compilação, pois a biblioteca *Mockturtle* é desenvolvida considerando a versão do C++17, a qual é mais atual em relação a versão original da ferramenta Fiction. Neste momento é importante destacar que esse trabalho foi desenvolvido considerando a versão 2.0 da ferramenta Fiction, a qual não possuía os recursos da biblioteca *Mockturtle* nativamente, o que já acontece na versão 3.0 lançada durante a execução deste trabalho.

Após a adição da interface e representação da estrutura de dados MIG via módulo *Alice*, foi possível carregar descrições (invocando os métodos do módulo *Lorina*) diretamenta para uma estrutura MIG, na qual pode ser aplicada os métodos contidos no módulo *Mockturtle*.

Com a interface funcional, foi preciso estudar os métodos disponibilizados pelo módulo *Mockturtle* para identificar quais aplicam otimizações com o foco na estrutura MIG, além de sua sintaxe e requisitos para execução. Após o estudo, foi implementada uma nova classe de comandos do módulo *Alice*, o qual invoca os métodos do módulo *Mockturtle*, os comandos implementados são apresentados abaixo. O código fonte da classe implementada é apresenta no Apêndice A.

migAlg: O comando **migAlg**, invoca o método *MIG algebraic rewriting* descrito na documentação *Mockturtle* (SOEKEN et al., 2019b). Esse algoritmo tenta reescrever uma rede com portas majoritárias para otimização de profundidade lógica utilizando as regras de associatividade e distributividade da lógica majoritária, propostas por Amarú em (AMARÚ; GAILLARDON; MICHELI, 2016). É importante ressaltar que este método também pode ser aplicado a redes que não sejam MIGs, mas considera apenas conjuntos de nós que implementam a função majoritária.

migCutRewrite: O comando **migCut**, invoca o método *Cut rewriting* descrito na documentação *Mockturtle* (SOEKEN et al., 2019b). Esse método aplica o algoritmo de cortes-K (*K-cuts*) (BERKELEY, 2014b).

Definição 2: Um corte-k de uma rede booleana é um par (n, L), em que n é um nó, chamado raíz, e L é um conjunto de nós, chamados folhas, de modo que: (i) cada caminho de qualquer entrada primária para n passe pelo menos uma folha e (ii) para cada folha $l \in L$, há pelo menos um caminho de uma entrada primária para n passando por l e não por qualquer outra folha (SOEKEN et al., 2019b).

A função implementada no *Mockturtle* denominada $cut\_enumeration$ implementa a enumeração de cortes, que é um algoritmo que calcula todos os possíveis

cortes de todos os nós em uma rede booleana. Como o número de cortes é muito grande, o número de cortes enumerados é limitado por um parâmetro l (cut_size) para o tamanho do corte e um parâmetro p (cut_limit) para o número máximo de cortes para cada nó. Essa técnica é denominada de cortes prioritários (*Priority Cuts*), pois seleciona o subconjunto de todos os cortes com relação a alguma função de custo.

Após enumerar o cortes que são subgrafos de uma rede booleana, o algoritmo tenta reescrever a lógica contida neste subgrafo em termos de portas lógicas da mesma rede booleana. As estruturas reescritas são adicionadas à rede e, se levarem à melhoria da área, serão utilizadas como novas partes da lógica. Portanto, a rede resultante possui muitos nós pendentes de candidatos sem êxito, que podem ser removidos invocando o método $cleanup\_dangling$, após o algoritmo de reescrita.

Diferentemente de outros métodos como *Node Resyntesis*, onde a reescrita gera novas estruturas, esta abordagem utiliza a mesma rede de entrada e saída. Consequentemente, o algoritmo não retorna uma nova rede, mas aplica mudanças locais na rede de entrada.

migResub: O comando **migResub**, invoca o método *Resubstitution* descrito na documentação *Mockturtle* (SOEKEN et al., 2019b). Este método aplica otimizações baseadas na proposta de Amarú (AMARÚ; GAILLARDON; MICHELI, 2014a) com as regras Ψ, onde existe a própria regra denominada *Substitution*. Essa regra é a base para as otimizações tanto em profundidade lógica, quanto em área propostas por Amarú. O algoritmo *Resubstution* utiliza um algoritmo de geração para cortes orientados à reconvergência. O corte cresce em direção às entradas primárias a partir de um nó dinâmico.

migNode: O comando **migNode**, invoca o método *Node Resyntesis* descrito na documentação *Mockturtle* (SOEKEN et al., 2019b). Neste caso foi efetuada a ressíntese através de uma rede *k-LUT* derivada de um MIG utilizado para o mapeamento LUT, com o auxílio de redes MIG ótimas pré-computadas e o algoritmo de cortes-K baseado no comando *resyn* da ferramenta ABC (BERKELEY, 2014b).

O mapeamento LUT ($lut\_mapping$) visa encontrar um bom mapeamento em relação a uma função de custo, por exemplo, um caminho crítico curto no mapeamento ou um pequeno número de nós (área). O algoritmo de mapeamento LUT pode atribuir um peso a um corte para funções de custo alternativas. A função de ressíntese é baseada em MIGs ótimos com tamanho pré-calculados. Tanto os métodos $node\_resynthesis$, $cut\_rewriting$ e $refactoring$ podem utilizar a função de ressíntese. Neste caso, ele produzirá um MIG baseado em MIGs de tamanho

ótimo pré-computados com até no máximo 4 variáveis. Consequentemente, os tamanhos de *fanin* (entradas) dos nós na rede de entrada não devem exceder 4.

migRefac: O comando **migRefac**, invoca o método *Boolean Refactoring* descrito na documentação *Mockturtle* (SOEKEN et al., 2019b). Esse método executa métodos de refatoração através do algoritmo *Collapsing Maximal Fanout-Free Cones* - (MFFCs). Através da criação de tabelas verdade e, recriando novas redes booleanas a partir dessas estruturas. O algoritmo executa aplicando modificações diretamente a rede booleana de entrada, o que necessita a remoção das estruturas antigas com a execução do comando $cleanup\_dangling$ após a reescrita.

O algoritmo implementa o método de síntese lógica baseada em majoritárias de 3 entradas proposto em Akers (AKERS, 1962). Este método é implementado na função $akers\_synthesis$, que recebe como entrada uma função representada como tabela de verdade com possíveis *Don't cares*.

Diante do exposto anteriormente, a adição do módulo *Mockturtle* viabiliza a exploração de uma série de otimizações baseadas na estrutura MIG, a qual a literatura aponta como a mais indicada para representação e síntese de circuitos para a tecnologia baseadas em majoritárias. Neste sentido, foram implementados diversos comandos dentro do ambiente da ferramenta Fiction 2.1 com o objetivo de explorar possíveis otimizações das descrições de entrada, antes do processo de geração do leiaute final. Assim, foi possível explorar otimizações na etapa de síntese lógica, aliadas as novas modificações do ambiente que serão descritas na próxima seção.

4.2.2 Método Ortho

Como mencionado anteriormente, o método *Ortho* apresenta um método escalável baseado em OGDs, o qual pode ser invocado no ambiente da ferramenta pelo comando **ortho**. Essa abordagem apresenta uma vantagem significativa de tempo de execução em comparação com a abordagem *exact* conforme a entrada aumenta de tamanho, o que a torna um bom objeto de estudo e base para o desenvolvimento desse trabalho (Walter et al., 2019).

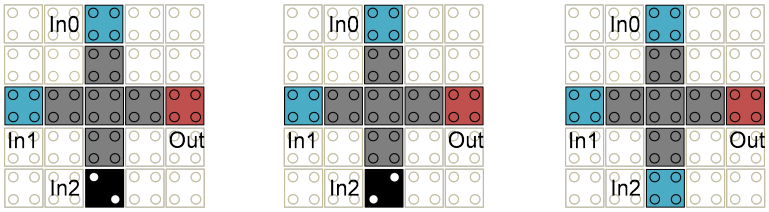
Considerando a restrição necessária dos esquemas de *clock* inerente a tecnologia QCA, o que acarreta naturalmente uma disposição em *grid* dos elementos, a escolha da estrutura OGD torna-se natural para a proposição de soluções. Neste sentido, existem alguns algoritmos propostos anteriormente que apresentam boas soluções para a etapa de Posicionamento e Roteamento. Dentre esses, destaca-se a proposta de Biedl (BIEDL, 1998), onde foi proposto para grafos com grau máximo 3 (denominados *3-graphs*), sendo capazes de gerar uma solução de disposição planar dos nodos.

O método *Ortho* utiliza o algoritmo de Biedl para realizar a síntese do circuito em questão. Sendo assim, o algoritmo proposto já cobre uma parte significativa dos problemas de *design* em FCN, satisfazendo as restrições de *Exclusividade* e *Adjacência* (restrições 1 e 2, apresentadas na subseção 2.2). Entretanto, essa solução é estendida pelos autores da ferramenta Fiction, para que adicionalmente as zonas de *clock* sejam atribuídas corretamente e os comprimentos dos caminhos sejam iguais, isto é, tal que a restrição 3 (*Zonas do Clock*) e a opcional (Comprimento do caminho) sejam satisfeitas.

Os *3-graphs* são grafos que possuem vértices com grau máximo 3. Dessa forma, considerando entradas e saídas é necessário descrever a lógica através de elementos que satisfaçam essa restrição para que o algoritmo se torne aplicável.

Em outras palavras, a descrição de entrada (neste caso uma descrição Verilog) é decomposta em elementos de no máximo duas entradas e uma saída, ou seja, operações $AND2$, $OR2$ e com a ajuda do inversor NOT ($INVERTER$). Da mesma forma, além de ser necessário reescrever algumas funções que não respeitam a restrição grau 3, ainda existem algumas manipulações na estrutura de dados base para atender nodos que possuem *fanout* (número de elementos conectados a uma saída) maior que 1. Neste caso, quando existe um grau maior que 1 na saída de um nodo, um elemento intermediário denominado $F1O2$ é adicionado. Este elemento tem como objetivo reduzir o grau de saída de um nodo, realizando o papel de distribuidor no circuito. Assim, é possível reduzir o grau do nodo adicionando em série quantos $F1O2$ forem necessários para atender as restrições do algoritmo.

Neste sentido, como o elemento básico nesta tecnologia é a porta majoritária, é preciso utilizá-la fixando a polaridade de uma das suas entradas para representar tanto a função lógica $AND2$, quanto a função lógica $OR2$. É importante salientar que essas portas já estão pré-definidas na biblioteca QCA-ONE (CAMPOS et al., 2016), utilizada no método *Ortho*. A Figura 49 apresenta os tipos de configurações utilizadas na ferramenta Fiction para representações de funções.



(a) And (b) Or (c) Maj

Figura 49 – Representação das portas And, Or, Majoritária na ferramenta Fiction.
Fonte: Próprio Autor.

Para guiar o método e, ao mesmo tempo, abordar as restrições de *design* da FCN (incluindo a opcional), é empregada uma ordem topológica no grafo. Uma ordenação linear é denominada topológica se, e somente se, para cada aresta direcionada $(u, v) \in E$ do vértice u ao vértice v, u vem antes de v na ordenação (CORMEN et al., 2002). Esta etapa é necessária, uma vez que o algoritmo OGD existente é definido para grafos não direcionados, enquanto que a síntese de funções lógicas necessita da definição do fluxo de dados de suas operações. Portanto, o método aplica a ordenação para guiar o fluxo em uma direção específica de cada aresta no grafo. A ordem resultante também define a sequência de processamento dos vértices, ou seja, a ordem que as operações relacionadas a cada vértice serão localizadas no *grid* pelo algoritmo. Dessa forma, pode-se garantir que todos os antecessores do vértice atualmente considerado já tenham sido processados e, portanto, os locais das respectivas operações podem ser considerados ao colocar a operação do vértice atual.

Quanto ao esquema de *clock*, o método *Ortho* basea-se no esquema 2DDWave (Vankamamidi; Ottavi; Lombardi, 2006), entretanto, ainda é possível configurar a quantidade de fases (3 ou 4), sendo a configuração *default* igual a 4. Sendo assim, ao restringir o esquema de *clock* ao método 2DDWave assume-se restrições importantes, como a questão das faces do *grid*. Em outras palavras, segundo o esquema 2DDWave entradas serão consideradas apenas nas faces Oeste (*West*) e Norte (*North*), assim como suas saídas apenas nas faces Leste (*East*) e Sul (*South*), conforme apresenta a Figura 33.

Ao restringir ainda mais as operações de novos vértices a serem localizados apenas ao Sul ou Leste dos já posicionados, obtém-se várias vantagens: (1) a restrição 1 (Exclusividade) é satisfeita, ou seja, cada operação (*gate*) está localizada de forma única, (2) as conexões se tornam facilmente realizáveis, satisfazendo a restrição 2 (Adjacência) e, (3) o fluxo de dados e, portanto, a atribuição posterior da numeração da zona do *clock* é bem definida (atendendo a restrição 3, Zonas de *Clock*). Além disso, a restrição opcional (comprimento do caminho) é satisfeita, explorando com a distância de *Manhattan* entre uma operação já localizada e seus predecessores. Isso significa que, quando novas operações só podem ser localizadas na direção Sudeste, todos

os caminhos diretos da saída da operação para qualquer operação recém-localizada têm a mesma distância de *Manhattan*, ou seja, passam pelo mesmo número de zonas de *clock*. O Algoritmo do método Ortho descrito em (Walter et al., 2019), pode ser observado em Algoritmo 1.

Algoritmo 1: Pseudo-Algoritmo Método Ortho, adaptado de Walter.

```
   Entrada: função booleana f
   Saída: layout FCN L
 1 Sintetizar netlist N = (O, W) de f ;
 2 L ← layout FCN vazio de tamanho (x = 0) x (y = 0);
 3 Gerar atribuição de direção d: W → leste, sul, e subdividir as bordas se necessário;
 4 Calcule a ordenação topológica o_1,. . . ,o_n ∈ O ;
 5 para o ∈ o_1,. . . ,o_n faça
 6     se indeg(o) = 0 então
 7         //sem arestas de chegada;
 8         Adicione uma linha e uma coluna a L;
 9         Localize o na posição (x, y);
10     senão
11         //duas arestas de chegada e_1,e_2 de o;
12         se d(e_1) = d(e_2) = leste então
13             //posicionamento para leste;
14             Adicione uma coluna a L;
15             y_p ← max. posicao y dos predecessores de o;
16             Localize o na posição (x, y_p);
17         senão
18             //posicionamento para sul;
19             Adicione uma linha a L;
20             x_p ← máx. posição x dos predecessores de o;
21             Localize o na posição (x_p, y);
22         fim
23         Desenhe os fios para conectar o com seu(s) predecessor(es) adequadamente;
24     fim
25 fim
26 retorno L;
```

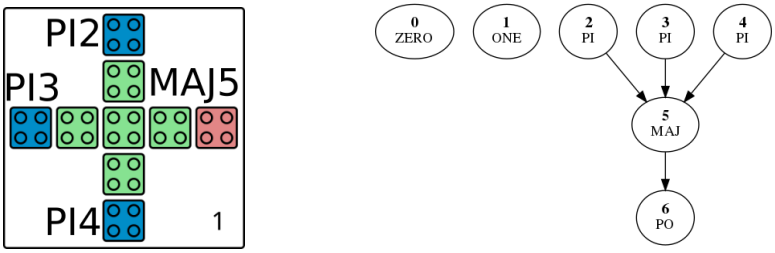
4.2.3 Limitações do Método Ortho

Embora o método *Ortho* apresente bons resultados em tempo de computação e qualidade dos leiautes resultantes, existem algumas limitações que ainda podem ser exploradas. Neste sentido, este trabalho realizou um estudo a cerca das limitações do método em questão.

Inicialmente, uma das limitações identificadas seria a questão da estrutura de dados envolvida na representação do circuito. Neste caso, considerando a restrição do algoritmo de Biedl, toda a representação utilizada como entrada é descrita em uma estrutura de AOIG, obrigando assim, a toda lógica ser descrita através das portas lógicas $AND2$, $OR2$ e $INVERTER$. Dessa forma, a utilização de portas majoritárias em sua representação da função original (com 3 entradas) é impedida, o que inviabiliza a utilização de uma estrutura de MIG por exemplo. A majoritária existente no método é utilizada apenas com a configuração para AND ou OR, conforme apresentam as Figuras 49(a) e 49(b).

Essa restrição na representação das funções inviabiliza a utilização da estrutura e dos métodos estado-da-arte de minimização baseada em majoritárias, por exemplo, os métodos propostos por Amarú (apresentados no Capítulo 3 de trabalhos correlatos), os quais apresentam alto grau de otimização da descrição. Por outro lado, da mesma forma, a utilização da estrutura de AOIG inviabiliza a melhor representação da lógica possível, a qual é baseada diretamente na função majoritária.

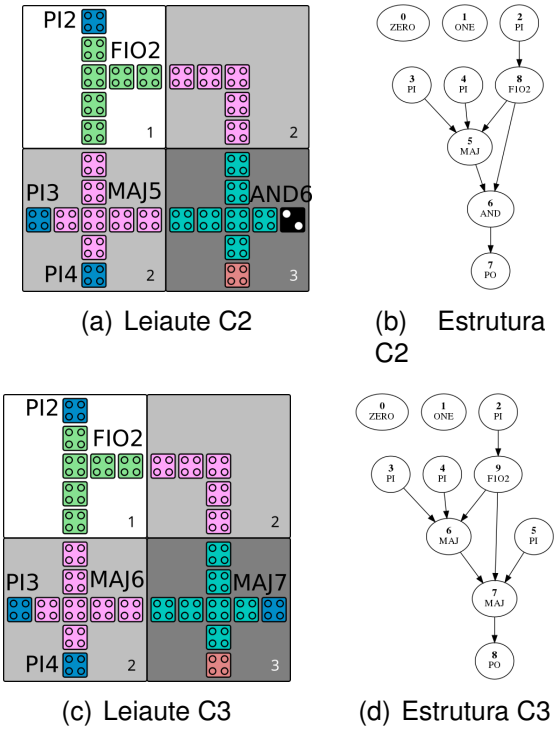
Considerando essa restrição, alguns experimentos foram efetuados para evitar o uso de AOIGs. O experimento visou verificar como o método *Ortho* se comportaria para a realização da síntese utilizando portas majoritárias. Sendo assim, inicialmente aplicou-se o método em uma descrição de um circuito (C1) com apenas um elemento, a porta majoritária. Ao final foi verificado a capacidade de método em lidar com o circuito de entrada, visto que não existe roteamento a ser efetuado. A Figura 50(a) apresenta o leiaute do circuito C1 resultante com uma porta majoritária, assim como a Figura 50(b) apresenta a estrutura de dados da ferramenta utilizada para representar a descrição.



(a) Leiaute C1 (b) Estrutura de dados C1

Figura 50 – Experimentos método Ortho circuito C1.
Fonte: Próprio Autor.

Em um segundo momento, dois novos circuitos foram descritos (C2 e C3), primeiramente adicionando outro elemento de lógica com duas entradas (uma porta $AND2$), e posteriormente, substituindo esse elemento por outra porta majoritária. Este experimento tentou avaliar se o método ainda seria capaz de lidar com o roteamento e posicionamento da majoritária, o que foi verificado com êxito. As Figuras 51(a) e 51(c) apresentam os leiautes resultantes, assim como as Figuras 51(b) e 51(d) apresentam as suas estruturas de dados.



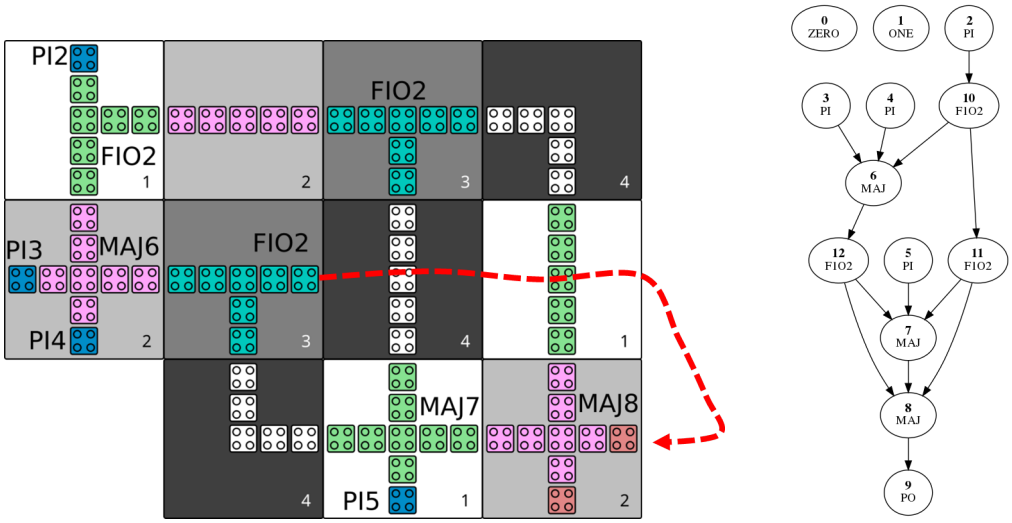
(a) Leiaute C2 (b) Estrutura C2

(c) Leiaute C3 (d) Estrutura C3

Figura 51 – Experimentos método Ortho nos circuitos C2(a,b) e C3(c,d).
Fonte: Próprio Autor.

Assim, foi possível verificar que o método é capaz de lidar com a descrição da porta majoritária diretamente. Deste modo um novo experimento foi efetuado, o qual descreveu um circuito (C4) com mais elementos (4) e, possivelmente, com uma maior complexidade de roteamento e posicionamento. A Figura 52(a) e 52(b) apresentam o leiaute e a estrutura de dados do circuito C4.

Neste experimento foi possível verificar alguns problemas gerados pela inserção de majoritárias na descrição do circuito. É possível identificar um erro no roteamento entre a majoritária $MAJ8$ e a saída da majoritária $MAJ6$. Embora a ferramenta Fiction permita a utilização de *multi-layers* e *wire-crossing* (que é o cruzamento de fios em diferentes camadas) para aplicar o roteamento mais complexos, a dificuldade do algoritmo nesse momento foi a incapacidade de posicionar uma entrada pela face Leste (na porta $MAJ8$), visto que infringe as propriedades do esquema de *Clock* 22DDWave por ser sempre uma face de saída. Vários outros experimentos foram efetuados na



(a) Leiaute C4 (b) Estrutura C4

Figura 52 – Experimentos método Ortho circuito C4.
Fonte: Próprio Autor.

sequência, e todos confirmaram essa limitação.

Após efetuar uma análise das inconsistências encontradas, algumas características foram identificadas, as quais podem viabilizar o uso de portas majoritárias em alguns casos. O método é capaz de lidar com majoritárias as quais possuem em suas entradas, pelo menos uma entrada primária, descrita diretamente no *subgrid* 5x5. Em outras palavras, quando a majoritária em questão necessita de até duas entradas externas (via fios de roteamento) não existe problema, pois não infringe o número de faces do esquema 2DDWave. Assim, similarmente ao que acontece com a representação da porta $AND2$ e $OR2$, onde uma de suas entradas é fixada sem necessitar de roteamento.

Em suma, os diferentes experimentos demonstraram que o método é capaz manipular majoritárias diretamente, o que é desejável, visto que diversos trabalhos demonstram que essa representação é a melhor para o paradigma FCN. Entretanto, é preciso realizar algumas modificações no algoritmo de roteamento e posicionamento para que as restrições do esquema de *clock* 2DDWave sejam respeitadas. Na próxima seção serão apresentadas as soluções propostas para contornar essas restrições no método *Ortho*, a qual culminou na proposta de uma nova versão do método denominada *Migortho*.

4.3 Migortho

Considerando as restrições identificadas, algumas modificações foram propostas no método *Ortho* original. Neste contexto, uma nova versão da ferramenta Fiction é apresentada, denominada aqui por Fiction 2.1, a qual possui métodos que viabilizam a etapa de síntese lógica, assim como uma nova versão do método *Ortho* capaz de utilizar a estrutura MIG denominada *Migortho*.

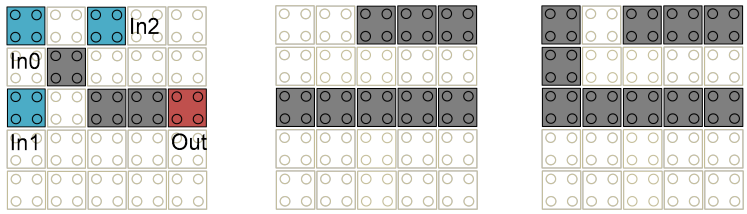
4.3.1 Novos Elementos de Design: Majx, Double Wire, Double Bent Wire

Com o objetivo de resolver a limitação identificada com a utilização de descrições da porta majoritária como entrada, foi proposta a inserção de alguns novos elementos de *design*, a fim de possibilitar que o método *Migortho* fosse capaz de lidar com esse tipo de descrição. Neste sentido, foi proposta uma nova porta majoritária baseada na proposta de Campos em (TEIXEIRA, 2016) (apresentada na Figura 37 na seção de trabalhos relacionados). Sendo assim, essa majoritária (denominada neste trabalho por $MajX$) resolve a limitação de entradas referentes ao esquema de *clock* 2DDWave, encontradas nos experimentos anteriores. Isso porque essa nova composição permite que as faces Oeste e Norte sejam utilizadas por até 3 entradas, o que viabiliza que o algoritmo de roteamento utilize as duas faces para rotear as 3 possíveis entradas da porta majoritária, o que é impossível na configuração tradicional.

Além disso, outros elementos foram inseridos para permitir o correto roteamento, isso se deve ao fato que na $MajX$ é necessário que dois fios cheguem por uma das duas faces possíveis. Desta forma, o método *Ortho* não considerava essa abordagem na sua lógica, não prevendo elementos para esse tipo de roteamento.

Primeiramente, foi analisada a estrutura de dados referente a representação dos circuitos, a fim de avaliar qual solução poderia ser proposta. Assim, foi identificada a necessidade de inserção de um novo elemento na estrutura de dados, o distribuidor $F2O2$. Desta forma, similarmente ao distribuidor $F1O2$ mencionado anteriormente, esse elemento é reponsável por mapear duas possíveis entradas em um mesmo *sub-grid*.

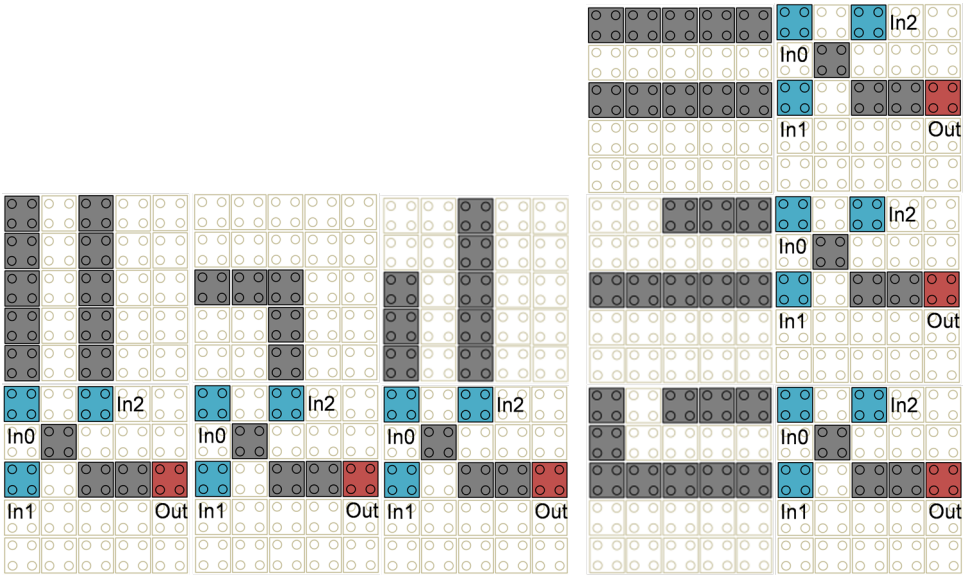
Por consequência, este elemento permite o mapeamento ideal dos fios dos elementos antecessores com as entradas do elemento $MajX$. Ademais, esse elemento $F2O2$ pode ser identificado nos leiautes, sendo mapeado em dois possíveis elementos de *design* dependendo dos roteamentos necessários, são eles: $Double\,Wire$ e $Double\,Bent\,Wire$. A Figura 53 apresenta os novos elementos de *design* inseridos no método *Migortho*.



(a) MajX (b) Double Wire (c) Double Bent Wire

Figura 53 – Representação dos novos elementos de *design* propostos: $MajX$, $Double\ Wire$, $Double\ Bent\ Wire$.
Fonte: Próprio Autor.

A inserção destes novos elementos permite a composição de novas soluções de roteamento para a lógica da majoritária. As Figuras 54(a) e 54(b) apresentam as possíveis soluções viabilizadas com o elemento $MajX$ e os elementos de fios duplos ($Double\ Wire$ e $Double\ Bent\ Wire$). Assim, diversas modificações foram implementadas a fim de viabilizar a representação desses novos elementos. Essas modificações foram tanto na biblioteca de células QCA-ONE com a descrição do novo elemento, quanto na estrutura de dados utilizadas para representar o circuito no método *Migortho*, denominada *Logic Network*. Essas modificações viabilizaram a identificação das subestruturas na descrição lógica de entrada onde seria necessário utilizar a instância da porta $MajX$ no lugar da porta majoritária tradicional.



(a) MajX com fios verticiais (b) MajX com fios horizontais

Figura 54 – Soluções viabilizadas no método Migortho no Fiction 2.1 com os novos elementos de *Design*.
Fonte: Próprio Autor.

4.3.2 Algoritmo Migortho

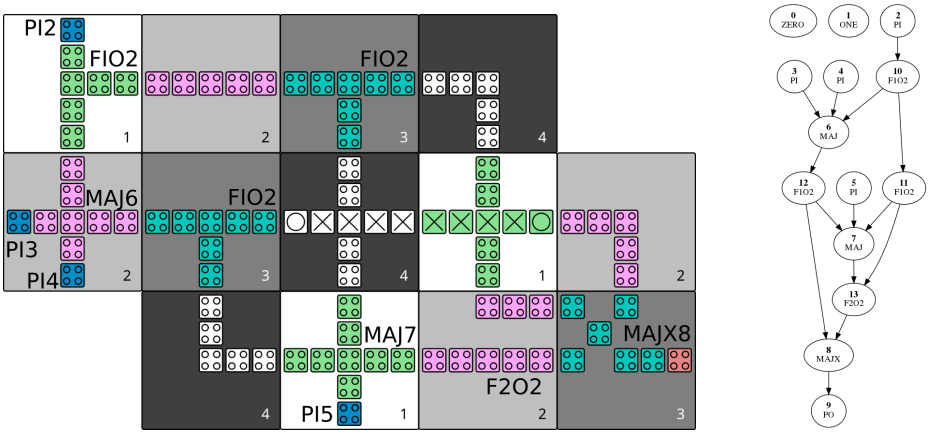
Logo após a adição dos novos elementos de *design*, e as modificações implementadas na estrutura *Logic Network* para a representação do lógica MIG com o uso do elemento $MajX$, foi necessário realizar modificações no algoritmo de roteamento para viabilizar a geração de leaiutes com a nova proposta. Neste sentido, o pseudo-algoritmo do método *Migortho* é apresentado no Algoritmo 2, esse pseudo-algoritmo foi adaptado a partir da versão original proposta por Walter em (Walter et al., 2019).

Algoritmo 2: Pseudo-Algoritmo Método Migortho.

```
   Entrada: função booleana f descrita em MIG com Maj e MajX
   Saída: layout FCN L
1  Sintetizar netlist N = (O, W) de f ;
2  L ← layout FCN vazio de tamanho (x = 0) x (y = 0);
3  Gerar atribuição de direção d: W → leste, sul, e subdividir as bordas se necessário;
4  Calcule a ordenação topológica o1,. . . ,on ∈ O ;
5  para o ∈ o1,. . . ,on faça
6      se indeg(o) = 0 então
7          //sem arestas de chegada;
8          Adicione uma linha e uma coluna a L;
9          Localize o na posição (x, y);
10     senão
11         //duas arestas de chegada e1,e2 de o;
12         se d(e1) = d(e2) = leste então
13             //posicionamento para leste;
14             Adicione uma coluna a L;
15             yp ← max. posicao y dos predecessores de o;
16             Localize o na posição (x, yp);
17         senão
18             //posicionamento para sul;
19             Adicione uma linha a L;
20             xp ← máx. posição x dos predecessores de o;
21             Localize o na posição (xp, y);
22         fim
23         se predecessor(es) é igual a F202 então
24             Desenhe os fios duplos para conectar o com seu(s) predecessor(es)
                adequadamente;
25         senão
26             Desenhe os fios para conectar o com seu(s) predecessor(es) adequadamente;
27         fim
28     fim
29 fim
30 retorno L;
```

As principais modificações são quanto aos parametros de entrada, onde nesta versão a descrição da estrutura de dados *Logic Network* já considera a descrição da lógica MIG, e os elementos Maj e $MajX$. Além disso, a lógica implementada pelas linhas 16 e 21, onde é efetuado o posicionamento, foram modificadas para lidar com o caso do novo elemento de design $MajX$. Por último, entre as linhas 23 e 27 também foram efetuadas modificações para viabilizar o roteamento dos elementos que necessitam a utilização de fios duplos.

Por fim, ao efetuar todas modificações necessárias na heurística do método *Migortho*, foram realizados novos experimentos a fim de identificar se a nova versão do método com a adição dos novos elementos estavam sendo roteados corretamente. Assim, o primeiro experimento reproduziu a síntese para o circuito C4, o qual havia apresentado erros no roteamento, o que não ocorreu nesta versão. A Figura 55(a) apresenta o leiaute final do circuito com os novos elementos inseridos, assim como a Figura 55(b) apresenta a estrutura de dados do circuito C4.

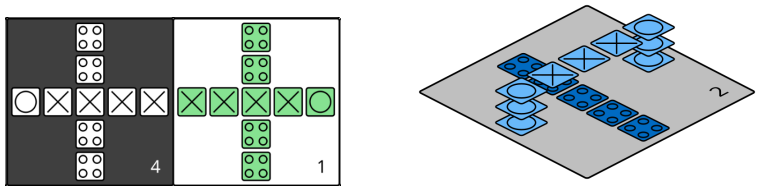


(a) Leiaute C4

(b) Estrutura C4

Figura 55 – Experimentos método Ortho no Fiction 2.1 com circuito C4.
Fonte: Próprio Autor.

É importante observar que, ao viabilizar uma alternativa de roteamento com a inserção dos novos elementos propostos (a porta majoritária *MAJX8* e o distribuidor *F2O2*), a solução do circuito C4 fez uso de uma nova camada de roteamento (*Layer*), a qual pode ser observada na Figura 56(a). Essa nova camada permite o cruzamento de fios, denominado *Wire-crossing* como apresenta a Figura 56(b).



(a) *Wire-Crossing* no Circuito C4

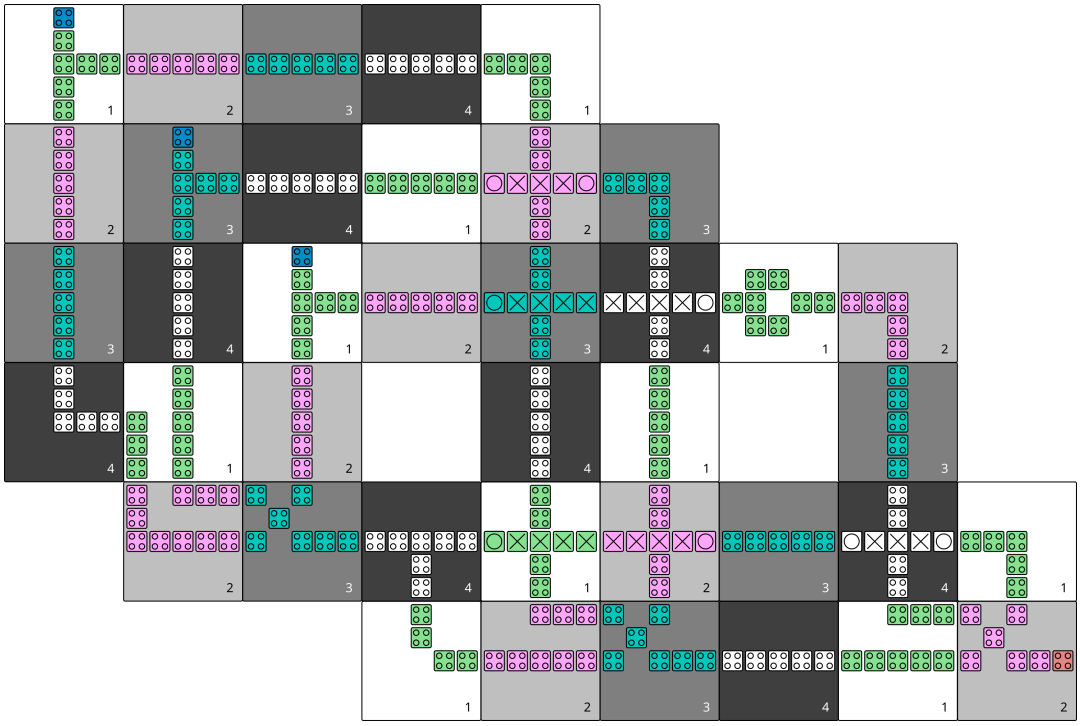
(b) *Wire-Crossing*

Figura 56 – *Wire-crossing* utilizado no circuito C4 (a) e ilustração da implementação de *Wire-Crossing* na ferramenta Fiction (b).
Fonte: Próprio Autor.

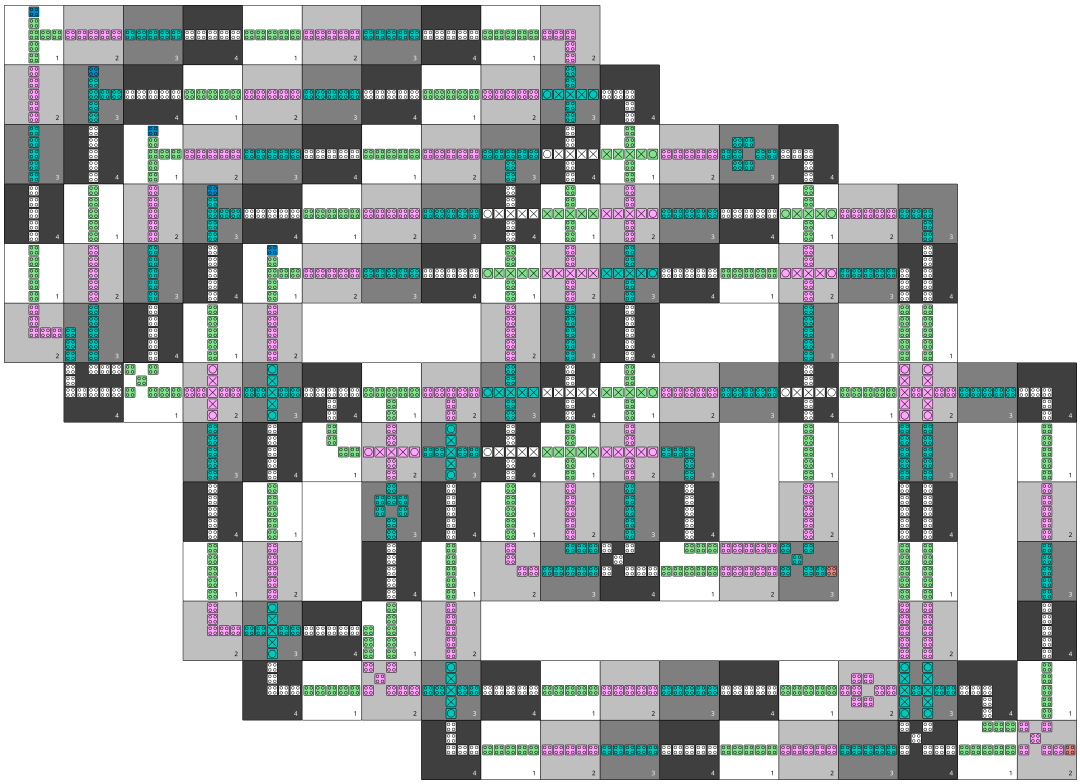
Embora as tecnologias FCN sejam consideradas planas, os cruzamentos de fios devem ser minimizados. No entanto, existem alguns casos em *QCA* onde é possível utilizar uma segunda camada para realizar cruzamentos de fios em distâncias curtas, ou até mesmo rotacionar algumas células para criar fios co-planares. Como ambos são apenas implementações técnicas do mesmo conceito, a ferramenta Fiction2.1 implementa os cruzamentos de fios através da inserção da segunda camada (WALTER et al., 2019). O uso dessa alternativa viabiliza roteamentos mais otimizados, visto que é possível criar caminhos menores entre os elementos, porém, existem várias desvantagens em sua utilização, como por exemplo: o aumento da área com a utilização de um segundo *Layer*, ou até mesmo a maior possibilidade de interferência ("ruído") entre células QCA sobrepostas (CHAUDHARY et al., 2005).

Considerando os resultados preliminares, mais dois experimentos com *designs* maiores foram efetuados a fim de tentar identificar alguma inconsistência no método proposto. Desta forma, foram efetuadas sínteses com os circuitos somador de 1 bit descrito com majoritárias (1bitAdderMaj), assim como o circuito somador de dois bits com majoritárias (2bitAdderMaj), os quais são circuitos do benchmark TOY (WALTER et al., 2019), e que já haviam apresentado problemas anteriormente. As Figuras 57(a) e 57(b) apresentam os leiautes resultantes com os novos elementos. Através de simulações na ferramenta QCADesigner os circuitos foram validados, confirmando assim, a corretude da nova versão do método.

Por conseguinte, a versão do método *Migortho* apresentou uma ótima solução para síntese QCA, visto que neste momento pode explorar o poder de representação da porta majoritária diretamente. Assim, não é preciso dividir a lógica da majoritária em portas $AND2$ e $OR2$, diminuindo drasticamente em alguns casos o número de elementos no circuito. Em suma, considera-se que a metodologia de síntese proposta é factível, principalmente considerando a prototipação efetuada. O capítulo de resultados e discussões apresentará uma avaliação mais detalhada do impacto desta nova versão.



(a) 1bitAdderMaj



(b) 2bitAdderMaj

Figura 57 – Experimentos método Migortho no Fiction 2.1 com circuitos 1bitAdderMaj e 2bitAdderMaj.
Fonte: Próprio Autor.

4.4 Conclusões

Este capítulo apresentou uma proposta de metodologia de síntese automática para a tecnologia QCA baseada nos conceitos do fluxo de células padrão. Nesse sentido, uma prototipação da metodologia foi efetuada com o objetivo de verificar a viabilidade da proposta para o processo de síntese automática para a tecnologia QCA, utilizando o ambiente da ferramenta Fiction. Após identificar diversas limitações no método *Ortho*, foi proposta uma nova versão denominada *Migortho*, além de novos elementos de *design*. Além disso, foram incluídas novas estruturas de dados para representação da lógica diretamente em majoritárias. Adicionalmente, foi importado a ferramenta um novo módulo da biblioteca *EPFL Logic Libraries* denominado *Mockturtle*. Com isso, foi possível implementar a metodologia proposta no ambiente da ferramenta Fiction, a qual foi denominada Fiction 2.1. Ademais, também foram implementados uma série de comandos de otimização baseados em MIG, os quais podem viabilizar otimizações na etapa de síntese lógica para a tecnologia QCA. Uma série de *scripts* foram propostos para avaliar e identificar quais são os métodos mais efetivos.

Por fim, embora as abordagens da ferramenta Fiction sejam o estado da arte, quando se refere a síntese para circuitos FCN, ela apenas trata das etapas finais do projeto de circuitos FCN, ou seja, não existe nenhuma preocupação com a etapa de síntese lógica. De forma geral, cabe ressaltar ainda que o ambiente da ferramenta Fiction por si só juntamente com as bibliotecas importadas e os outros algoritmos também não são capazes de implementar diretamente a metodologia proposta. Parte das contribuições desse trabalho são observadas na modificação dos ambientes e bibliotecas para permitir a integração e funcionamento adequado da metodologia. O próximo capítulo apresentará os resultados e as discussões a cerca das avaliações efetuadas, tanto do impacto do uso dos comandos implementados, quanto do impacto no leiaute final QCA com os novos elementos de *design* no método *Migortho* proposto.

5 RESULTADOS E DISCUSSÕES

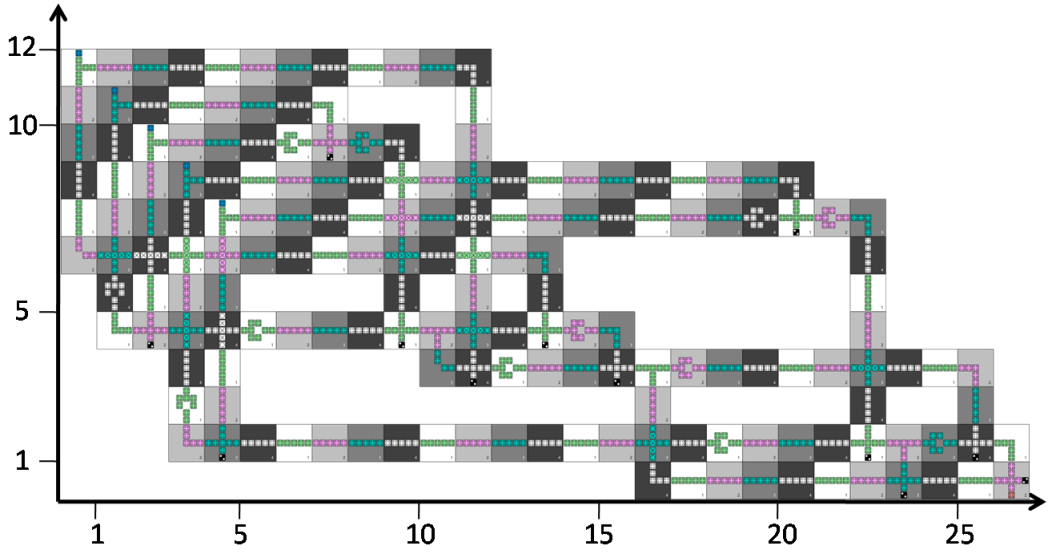
Neste capítulo serão apresentados os resultados das avaliações efetuadas da metodologia proposta através da prototipação no ambiente da ferramenta Fiction 2.1, onde foram efetuadas cinco avaliações considerando diferentes aspectos. Inicialmente, foram efetuadas avaliações com o objetivo de mensurar o impacto dos novos elementos de *design* e a nova estrutura de dados nos resultados do método *Migortho*. Em um segundo momento, foram investigados e analisados, os impactos de otimização alcançados com a fase de síntese lógica implementada através dos comandos da biblioteca *Mockturtle*, o que possibilitou a proposição de uma nova versão mais otimizada dos circuitos dos *benchmarks* utilizados para a síntese QCA. Assim, com base nos resultados dos *scripts* de comandos, foram propostas novas avaliações do método *Migortho*, com o melhor método de otimização na etapa de síntese lógica identificado. Sendo assim, o método *Migortho* foi executado com a adição dos comandos *Mockturtle*, a fim de avaliar o impacto da etapa de síntese lógica no leiaute final. Adicionalmente, é proposta uma avaliação comparativa do método *Migortho* com o método *Ortho* e o método *Exact*, considerando um subconjunto de circuitos. Por fim, também uma avaliação foi efetuada da metodologia proposta com o *benchmark* específico de MIG definido por Amarú, resultante da ferramenta MIGthy. Essa avaliação comparou o impacto de otimização MIG em relação aos resultados do MIGthy, além de gerar estimativas de leiautes QCA para as duas versões dos circuitos.

5.1 Análise Fiction 2.1 - Migortho: MAJ vs AOI

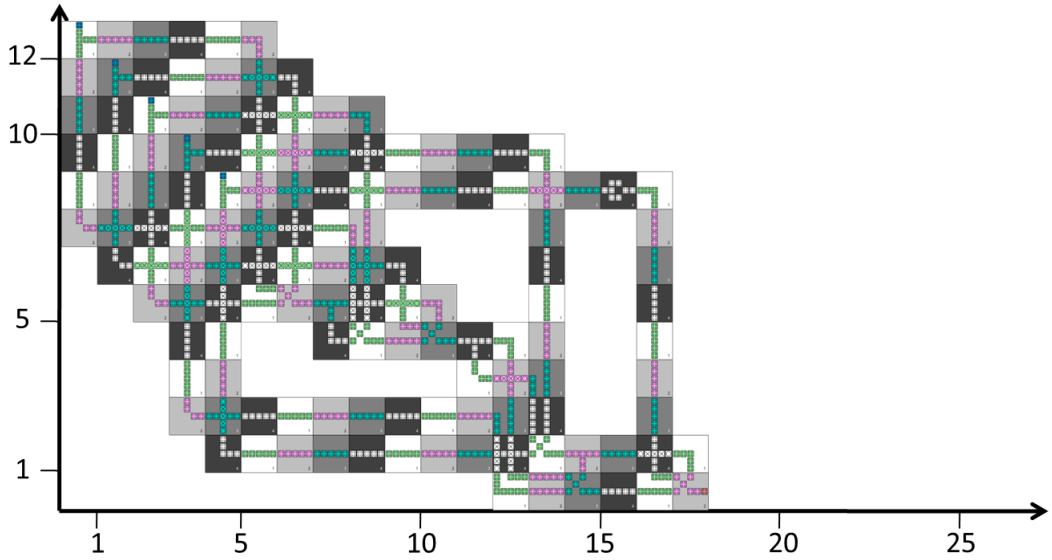
O experimento inicial considerou a contabilização e análise mais detalhada do circuito *Xor5-r1* em duas formas de descrição distintas disponibilizadas nativamente pela ferramenta Fiction 2.1 (no formato Verilog), foi utilizada tanto a descrição do *benchmark* TOY (WALTER et al., 2019) (utilizando a descrição AOI, a qual é a estrutura de descrição original do método *Ortho*), quanto a descrição do *benchmark* MAJ (WALTER et al., 2019) (utilizando a estrutura de descrição com majoritárias viabilizada através das modificações propostas nesse trabalho). As duas descrições dos circuitos *Xor5-*

r1, assim como as diferenças nas representações da lógica com a estrutura AOI e MAJ, podem ser observadas no Anexo A.

A Figura 58(a) apresenta um leiaute QCA gerado pelo método *Migortho*, utilizando a descrição AOI como entrada, a qual possui um área de *grid* de 27 x 12 (linhas e colunas de *subgrids* 5x5) e 38 portas. Por outro lado, a Figura 58(b) apresenta um leiaute QCA resultante baseada na descrição com majoritárias, a qual possui uma área de *grid* de 18 x 13, e 26 portas. É possível verificar que a abordagem proposta alcançou uma redução em área de *grid* de 28 % e 24 % na contagem de portas. A Figura 58 apresenta uma comparação dos dois leiautes *Xor5-r1*, onde é possível verificar um melhor aproveitamento do *grid* com a metodologia proposta neste trabalho.



(a) *Xor5-r1* descrito em AOI



(b) *Xor5-r1* descrito com MAJ

Figura 58 – Leiaute QCA do circuito *Xor5-r1*.

Da mesma forma, foi considerado todo o conjunto de circuitos do *benchmark* TOY, MAJ e o *benchmark* ISCAS'85 (BRGLEZ; FUJIWARA, 1985). O *benchmark* TOY possui um conjunto de dezenove circuitos, os quais são descritos através de operações AND, OR e INVERTER, desta forma, utiliza uma estrutura AOI. Por outro lado, o *benchmark* MAJ, possui catorze dos dezenove circuitos do *benchmark* TOY, porém, descritos através de portas majoritárias interconectadas. Para uma melhor comparação, foram descritos através de portas majoritárias os 5 circuitos faltantes. A versão anterior do método *Ortho* não é capaz de fornecer uma solução de leiaute QCA para os circuitos do *benchmark* MAJ, contudo, a versão proposta neste trabalho (*Migortho*) é capaz de gerar. Sendo assim, foi efetuada uma comparação entre todos os circuitos do *benchmark* MAJ em relação ao *benchmark* TOY com a versão do método *Migortho*. Essa avaliação viabilizou mensurar o impacto da utilização da estrutura da majoritária nos leiautes, a Tabela 2 apresenta os resultados obtidos.

Tanto a Tabela 2 quanto a Tabela 3 apresentam os resultados sobre as seguintes características: em termos da área do *grid* (A), ou seja, estimativa de colunas e linhas de *subgrids* 5x5 utilizados, também número de portas (G), número de elementos de fio (W), número de cruzamento entre fios (CROSS), o caminho crítico do circuito (C.P.) e o número total de *subgrids* utilizados (Total Subgrid). Cada linha da tabela apresenta os dois resultados de cada circuito, o identificador $\clubsuit$ representa os dados da solução baseada em MAJ, assim como o identificador $\triangle$ representa os dados da solução baseada em AOI, por último, o identificador $\star$ indica qual das duas soluções é a mais otimizada em relação a área.

Tabela 2 – Comparação do método *Migortho benchmark TOY* vs *benchmark* MAJ

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
1bitAdderAOIG♣	13 x 8	18	56	5	20	104	
1bitAdderAOIG△ ⋆	11 x 7	15	40	3	16	77	-35,1%
1bitAdderMaj♣ ⋆	2 x 2	2	0	0	2	4	
1bitAdderMaj△	22 x 11	30	121	25	32	242	98,3%
b1-r2♣ ⋆	12 x 8	17	48	7	17	96	
b1-r2△	15 x 7	19	59	8	19	105	8,6%
clpl♣	18 x 9	22	108	18	26	162	
clpl△ ⋆	14 x 2	14	0	0	14	28	-478,6%
FA♣ ⋆	7 x 4	9	13	1	10	28	
FA△	15 x 7	19	69	9	19	105	73,3%
FS♣ ⋆	2 x 2	2	0	0	2	4	
FS△	17 x 6	21	75	13	20	102	96,1%
HA♣	6 x 6	10	24	5	10	36	
HA△ ⋆	7 x 5	10	21	4	11	35	-2,9%
HS♣	9 x 4	11	23	3	11	36	
HS△ ⋆	7 x 5	10	19	1	11	35	-2,9%
mux21♣	5 x 3	6	5	0	7	15	
mux21△ ⋆	4 x 2	5	3	0	5	8	-87,5%
mux41♣	15 x 10	22	77	15	24	150	
mux41△ ⋆	12 x 6	16	48	8	17	72	-108,3%
newtag♣ ⋆	12 x 4	13	20	0	15	48	
newtag△	17 x 7	19	69	4	23	119	59,7%
par-check♣ ⋆	13 x 8	17	73	18	20	104	
par-check△	15 x 9	21	71	15	23	135	23,0%
par-gen♣ ⋆	9 x 7	13	38	4	15	63	
par-gen△	10 x 7	14	39	5	16	70	10,0%
RCA2♣ ⋆	25 x 15	36	256	39	37	375	
RCA2△	29 x 14	38	248	43	40	406	7,6%
t♣ ⋆	7 x 4	10	12	1	9	28	
t△	11 x 6	14	35	6	14	66	57,6%
xnor2♣	8 x 4	10	15	3	11	32	
xnor2△ ⋆	6 x 4	8	12	3	9	24	-33,3%
xor2♣	7 x 4	9	15	3	10	28	
xor2△ ⋆	5 x 3	6	8	0	7	15	-86,7%
xor5-r1♣ ⋆	18 x 13	26	126	16	30	234	
xor5-r1△	27 x 12	34	154	15	38	324	27,8%
xor5R♣ ⋆	18 x 13	26	126	16	30	234	
xor5R△	35 x 15	45	276	38	49	525	55,4%
Otimização △vs♣		**0,78**	**0,76**	**0,77**	**0,80**	**0,71**	
Desvio Padrão △		**10,90**	**77,83**	**12,37**	**11,83**	**142,63**	
Desvio Padrão ♣		**8,78**	**63,69**	**10,00**	**9,75**	**98,68**	

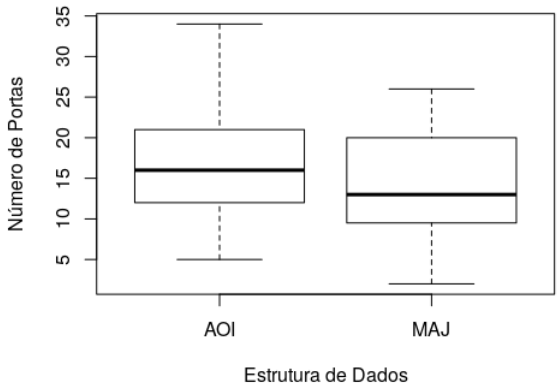
△-AOI ♣-MAJ ⋆-Melhor

Os resultados demonstram que em 58$\%$ dos casos, a nova proposta baseada em majoritárias gerou a melhor solução em comparação com a versão baseada em AOI. No entanto, é importante enfatizar que, nos casos os quais a estrutura AOI foi melhor, o circuito mínimo é representado através da própria lógica *And-Or-Inverter*. Além disso, é possível observar que quanto maior o circuito, mais significativa é a redução no leiaute QCA. Em média, a relação de otimização na representação do número de portas utilizando majoritárias, se manteve linear em relação as outras características do leiaute final, número total de *subgrids* utilizados e caminho crítico.

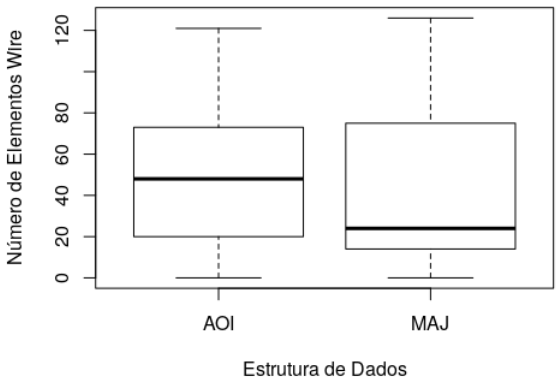
A Figura 59 apresenta análise gráfica de todo o experimento através de gráficos do tipo caixa (*boxplot*), onde algumas características sobre os dados foram identificadas. Inicialmente, é possível verificar através do gráfico na Figura 59(a) que a utilização da estrutura de majoritárias na representação dos circuitos, acarretou em uma otimização em relação ao número de portas, sendo verificada pela posição da mediana no gráfico *boxplot* (maior concentração dos valores). Da mesma forma, esse comportamento pode ser analisado para os outros gráficos, como por exemplo, cruzamento de fios (Figura 59(c)), caminho crítico (Figura 59(d)) e total de *subgrids* 5x5 utilizados (Figura 59(e)). É importante salientar, que esse último gráfico está diretamente relacionado com a área final do leiaute QCA. Outra informação importante é que os *outliers* foram retirados da *plotagem* (através da função $outline = false$ do ambiente da linguagem *R*) pois seus valores não influenciam significativamente no comportamento gráfico final. Além disso, a exclusão dos *outliers* possibilitam uma melhor visualização do gráfico em função da escala e sua amplitude interquatílica.

Além disso, é possível verificar que os valores dos limites superiores também foram otimizados, com exceção do gráfico na Figura 59(b), onde o limite superior foi incrementado, o que se deve ao fato de que a utilização de elementos do tipo *MajX* aumenta a complexidade de roteamento do circuito em alguns casos. Neste sentido, é esperado um acréscimo na amplitude do gráfico caixa para a utilização do elemento fio (*Wire*), visto que o roteamento se torna mais complexo em alguns casos. Assim, de forma geral é possível verificar que a proposta de otimização através da estrutura de representação utilizada foi efetiva.

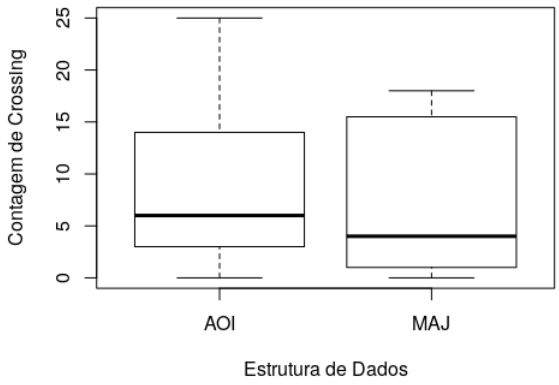
Por último, na Figura 59(f) é apresentado o gráfico histograma com a contagem das diferenças entre cada representação. É possível verificar que enquanto alguns casos (8 ocorrências) houve um aumento do número de elementos necessários para a representação da lógica com portas majoritárias (valores negativos), na maioria dos casos ou não foi necessário incremento de elementos na representação, ou houve diminuição do número de elementos para representar a mesma lógica.



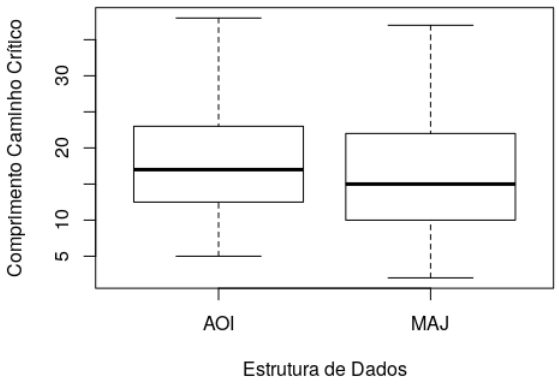
(a) Portas



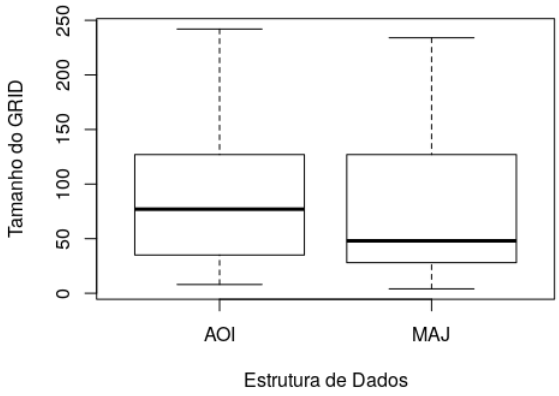
(b) Wire



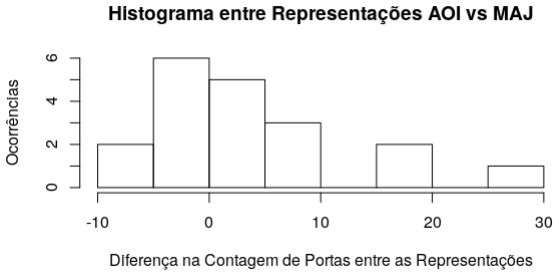
(c) Crossing



(d) Caminho Crítico



(e) Subgrids



(f)

Figura 59 – Análise gráfica *benchmark TOY vs MAJ*.

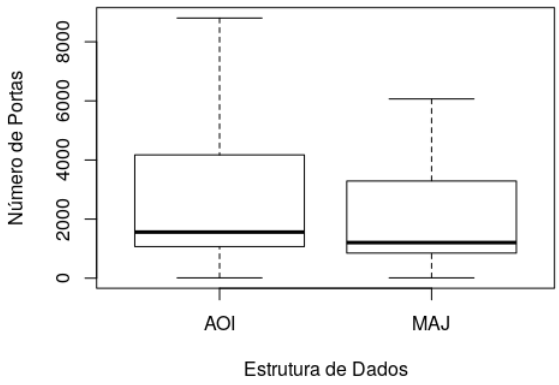
Em um segundo momento, os circuitos do *benchmark* ISCAS'85, em uma estrutura AOI, foram transcritos para uma descrição baseada em majoritárias. Logo após, foram efetuadas as mesmas avaliações sobre o impacto dessas estruturas e dos novos elementos no método *Migortho*. Essa avaliação se torna interessante, visto que o *benchmark* ISCAS'85 é um dos mais tradicionais *benchmarks* utilizados pela comunidade de síntese lógica, além de possuir circuitos com diferentes tamanhos e complexidades. A Tabela 3 apresenta os resultados dos leiautes QCA referentes aos dois tipos de estruturas.

Tabela 3 – Comparação do método *Migortho benchmark ISCAS'85*

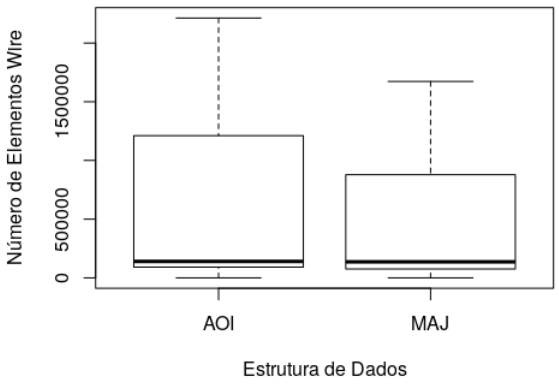
Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
c1355♣⋆	781 x 457	1205	136.608	16.069	1.237	356.917	
c1355△	1.140 x 452	1.559	140.019	9.557	1.591	515.280	30,7 %
c17♣	8 x 4	11	12	0	10	32	
c17△	8 x 4	11	12	0	10	32	0,0 %
c1908♣⋆	662 x 404	1.033	106.885	12.517	1.065	267.448	
c1908△	852 x 400	1.219	113.097	11.021	1.251	340.800	21,5 %
c2670♣⋆	1.117 x 568	1.628	257.451	28.160	1.671	634.456	
c2670△	1.515 x 630	2.086	31.2930	29.314	2.126	954.450	33,5 %
c3540♣⋆	1.598 x 812	2.366	528.707	66.870	2.409	1.297.576	
c3540△	2.200 x 842	2.997	659.521	69.930	3.041	1.852.400	30,0 %
c432♣⋆	350 x 171	485	26.667	2.563	518	59.850	
c432△	458 x 172	594	32.459	3.236	629	78.776	24,0 %
c499♣⋆	633 x 397	997	97.279	10.162	1.029	251.301	
c499△	828 x 412	1.207	112.084	9.629	1.239	341.136	26,3 %
c5315♣⋆	2.772 x 1.560	4.210	1.672.771	137.085	4.296	4.324.320	
c5315△	3.796 x 1.675	5.346	2.212.779	184.034	5.439	6.358.300	32,0 %
c6288♣⋆	4.404 x 1.694	6.066	1.228.198	54.270	6.097	7.460.376	
c6288△	6.694 x 2.142	8.804	1.762.602	69.907	8.835	14.338.548	48,0 %
c7552♣⋆	3.245 x 1.691	4.855	1.504.955	102.768	4.905	5.487.295	
c7552△	4.644 x 1.891	6.476	1.954.408	114.602	6.509	8.781.804	37,5 %
c880♣⋆	484 x 265	701	54.872	7.553	740	128.260	
c880△	693 x 288	932	70.563	7.650	966	199.584	35,7 %
Otimização △vs♣		**0,75**	**0,76**	**0,86**	**0,76**	**0,60**	
Desvio Padrão △		**2.815,79**	**864.188,47**	**58.712,38**	**2.824,00**	**4.737.184,06**	
Desvio Padrão ♣		**2.003,28**	**639.922,76**	**45.501,02**	**2.013,94**	**2.636.126,18**	

△-AOI ♣-MAJ ⋆-Melhor

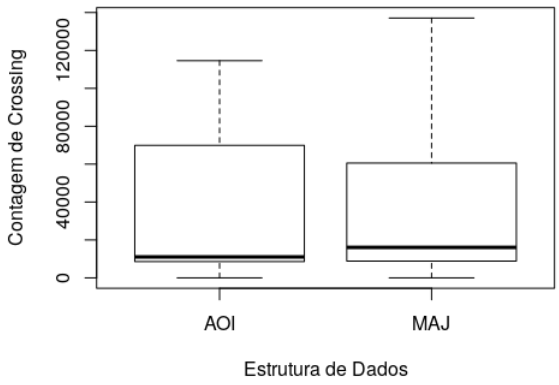
No segundo experimento, os resultados demonstram que em 90% dos casos, a nova proposta baseada em majoritárias gerou a melhor solução em comparação com a versão baseada em AOI. Além disso, é possível observar que a relação de quanto maior o circuito, mais significativa é sua redução no leiaute QCA se mantém. Em média, a relação de otimização na representação do número de portas utilizando majoritárias se manteve linear em relação as outras características do leiaute final e caminho crítico. Quanto ao número total de *subgrids* utilizados, os ganhos foram mais expressivos que na primeira avaliação. É interessante notar que embora a diminuição no número de portas na representação com majoritária tenha sido de 25 %, na relação de área total utilizada do leiaute o método obteve uma redução de 40 % em relação a solução original. A Figura 60 apresenta análise gráfica através de gráficos do tipo caixa (*boxplot*).



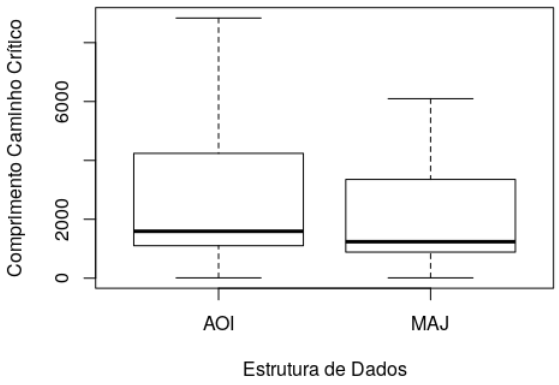
(a) Portas



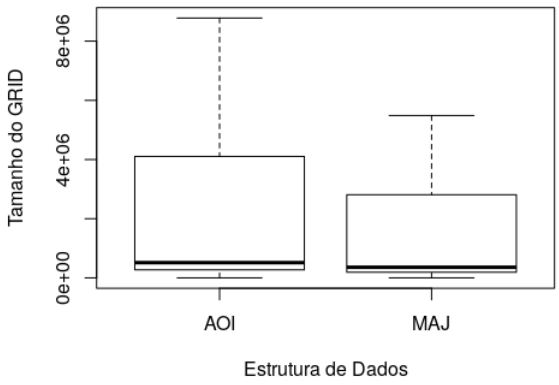
(b) Wire



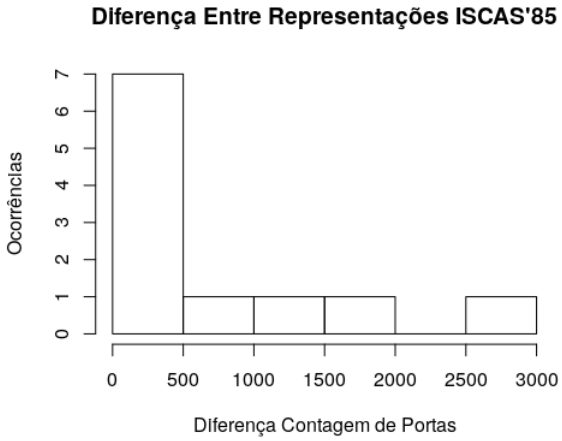
(c) Crossing



(d) Caminho Crítico



(e) Subgrids



(f)

Figura 60 – Análise gráfica *benchmark ISCAS'85*.

De forma análoga ao experimento anterior, a Figura 60 apresenta gráficos do tipo caixa (*boxplot*), onde é possível realizar algumas observações. É importante salientar que da mesma forma do gráfico anterior, os *outliers* foram retirados da *plotagem* (através da função $outline = false$ do ambiente da linguagem *R*) pois seus valores não influenciam significativamente no comportamento gráfico final. Além disso, a exclusão dos *outliers* possibilitam uma melhor visualização do gráfico em função da escala e sua amplitude interquatílica. Inicialmente, é possível verificar através do gráfico na Figura 60(a) que a utilização da estrutura de majoritárias na representação dos circuitos também acarretou uma otimização em relação ao número de portas, sendo verificada pela posição da mediana no gráfico *boxplot* (maior concentração dos valores) e amplitude quartílica da caixa. Da mesma forma, esse comportamento pode ser verificado para os outros gráficos, como por exemplo, caminho crítico (Figura 60(d)) e total de *subgrids* 5x5 utilizados (Figura 60(e)). Este último responsável por representar as informações de área total do leiaute.

Além disso, é possível verificar que os valores dos limites superiores também foram otimizados, com exceção do gráfico *crossing* (Figura 60(c)), onde o limite superior foi incrementado assim como a mediana, o que se deve ao fato que a utilização de elementos do tipo *MajX*, o qual aumenta a complexidade de roteamento do circuito em alguns casos. Neste sentido, é esperado um acréscimo neste quesito, visto que existe uma maior probabilidade de colisões de fios dado o tamanho dos circuitos. Assim, de forma geral é possível verificar que a proposta de otimização através da estrutura de representação utilizada foi efetiva, principalmente, porque os circuitos utilizados são de um *benchmark* conhecido da comunidade, e possuem milhares de portas. Por último, a Figura 60(f) apresenta o gráfico histograma com a contagem das diferenças entre cada representação. É possível observar que houve otimização em todos os casos no número de elementos necessários para a representação da lógica utilizando majoritária.

5.1.1 Considerações Finais

O primeiro experimento apresentou resultados que corroboram com a literatura, demonstrando que a utilização de lógica majoritária diretamente na estrutura de dados viabiliza melhores resultados para síntese QCA. Diversos experimentos foram efetuados com o método *Migortho* que permite a utilização de portas majoritárias, os resultados de otimização chegaram a 40$\%$ menos área de leiaute para o conjunto de circuitos do *benchmark* ISCAS'85. As otimizações alcançadas são resultados apenas na mudança da estrutura de dados, nenhuma técnica de manipulação de síntese lógica para tentar minimizar a rede booleana foi utilizada. O primeiro experimento permitiu analisar e validar a síntese utilizando portas majoritárias, além de vislumbrar possíveis melhorias, como por exemplo a exploração dos métodos de otimização MIG.

5.2 Análise do Impacto dos Scripts de Pré-processamento

O segundo experimento foi efetuado com o objetivo de identificar a efetividade dos comandos implementados através dos métodos da biblioteca *Mockturtle*, nas descrições MIG de entrada. Assim, seria possível propor otimizações na descrição de entrada, reduzindo o tamanho da descrição MIG (número de portas), o que reflete diretamente no leiaute. Neste sentido, uma série de comandos foram definidos em *scripts* de execução descritos na linguagem *SHELL*. Além disso, foram selecionados sete *benchmarks* distintos, os *benchmarks* disponibilizados pelo ambiente da ferramenta Fiction, são eles: TOY (WALTER et al., 2019), MAJ (WALTER et al., 2019), ISCAS'85 (BRGLEZ; FUJIWARA, 1985), EPFL(AMARÚ; GAILLARDON; DE MICHELI, 2015). Adicionalmente, foram considerados os *benchmarks* propostos por Amarú, resultados de sínteses no ambiente *Mighty* denominados MIG, MIGoriginal (AMARÚ, 2020), além dos circuitos utilizados por Fontes em (FONTES JUNIOR, 2018) denominados aqui como *benchmark* UFV, ao total foram considerados 144 circuitos de entrada. A seção a seguir apresenta a construção desses *scripts*, assim como a seção posterior apresenta as análises dos resultados obtidos.

5.2.1 *Scripts* de Comandos Mockturtle de Pré-processamento

Com o objetivo de propor uma etapa de síntese lógica efetiva na metodologia proposta, foi necessário avaliar o impacto das possíveis otimizações implementadas pelos comandos sobre os métodos *Mockturtle*. Assim, alguns *scripts* de execução foram desenvolvidos para realizar uma avaliação inicial entre os métodos implementados. É importante ressaltar que a ferramenta Fiction 2.1 permite um modo de execução através de *scripts*, onde é possível definir uma série de comandos a serem utilizados. Neste sentido, diferentes configurações foram definidas a fim de avaliar diferentes aspectos, como por exemplo:

- A capacidade de cada método de otimização, identificando qual método é o mais efetivo;
- O impacto de execuções repetidas de um mesmo método;
- O impacto da modificação de alguns parâmetros dos métodos, neste caso, o tamanho do K (onde o K representa a quantidade de entradas) para métodos que utilizam o algoritmo de cortes-K (Cortes-K são definidos na **Definição 2** na subseção 4.2.1);
- O impacto de execuções de diferentes comandos em sequência, criando assim, conjunto de comandos;

Desta forma, inicialmente o arquivo do circuito descrito na linguagem Verilog é carregado pela ferramenta para a estrutura de dados MIG. Logo após, os comandos são aplicados a rede em questão. Ao final, um novo arquivo Verilog é gerado como saída, com a descrição do circuito após a aplicação dos comandos. Juntamente com o circuito de saída, também são gerados arquivos de *log* para serem utilizados nas avaliações posteriores.

Para a avaliação do impacto de execuções repetidas do mesmo comando, foram propostas configurações com uma execução única até quatro vezes em série de todos os comandos implementados. Já para a questão da avaliação do K do algoritmo de cortes-K foram propostas configurações com K=3 até K=8 (onde o K representa o número máximo de entradas), é importante salientar que o aumento do K afeta diretamente o desempenho da heurística, visto que expande consideravelmente e espaço de soluções a serem geradas.

Além disso, ao final, foram propostos alguns conjuntos de comandos executados em alternância, denominados *mix*, onde através de quatro configurações diferentes $mix1$, $mix2$, $mix3$ e $mix4$, juntamente com a variação do K de 3 até 8 foi possível avaliar o impacto e a efetividade da alternância entre os comandos. Ao fim, foram propostas 68 configurações diferentes de comandos, os resultados serão discutidos a seguir na seção de análises. A configuração dos comandos é apresentada no Apêndice B.

5.2.2 Resultados Mockturtle (MIG) vs Circuito Original

Para realizar a análise do comportamento dos comandos implementados, todos os *benchmarks* em questão foram utilizados como entradas para todas as configurações de comandos propostas. Foram 68 configurações aplicadas aos 144 circuitos, o que gerou 9.792 diferentes soluções de circuitos de saída, contabilizando $3,9$ GB de conteúdo. Esses experimentos foram realizados em uma máquina Intel(R) Xeon(R) CPU $E5620$ com 2.4GHz, 8 núcleos e 8 GB de memória RAM, rodando uma versão linux server com os serviços básicos. É importante ressaltar que os experimentos foram efetuados sem a pretensão de análise por tempo de execução, visto que os experimentos originais foram efetuados em uma máquina Intel(R) Xeon $E52630\ v3$ com 3.2 GHz e 64 GB de memória RAM. A partir dos dados obtidos diversas avaliações foram realizadas, os resultados serão apresentados a seguir. Contudo, é importante salientar que todos os arquivos para as análises foram gerados em aproximadamente 120 horas.

5.2.3 Análise do K

Inicialmente, foram consideradas diferentes avaliações para encontrar o valor de K para o algoritmo de *cortes-K* com uma taxa de otimização razoável, aliado a uma baixa complexidade e *runtime*. Para isso, foram definidas algumas execuções considerando diferentes valores para *K*. Assim, foram considerados 3 diferentes experimentos, são eles:

1 Utilizando o comando **migNode**, o qual é possível variar o *K* com valores 3 e 4, como citado na subseção 4.2.1 para identificar o impacto na variação do *K*;

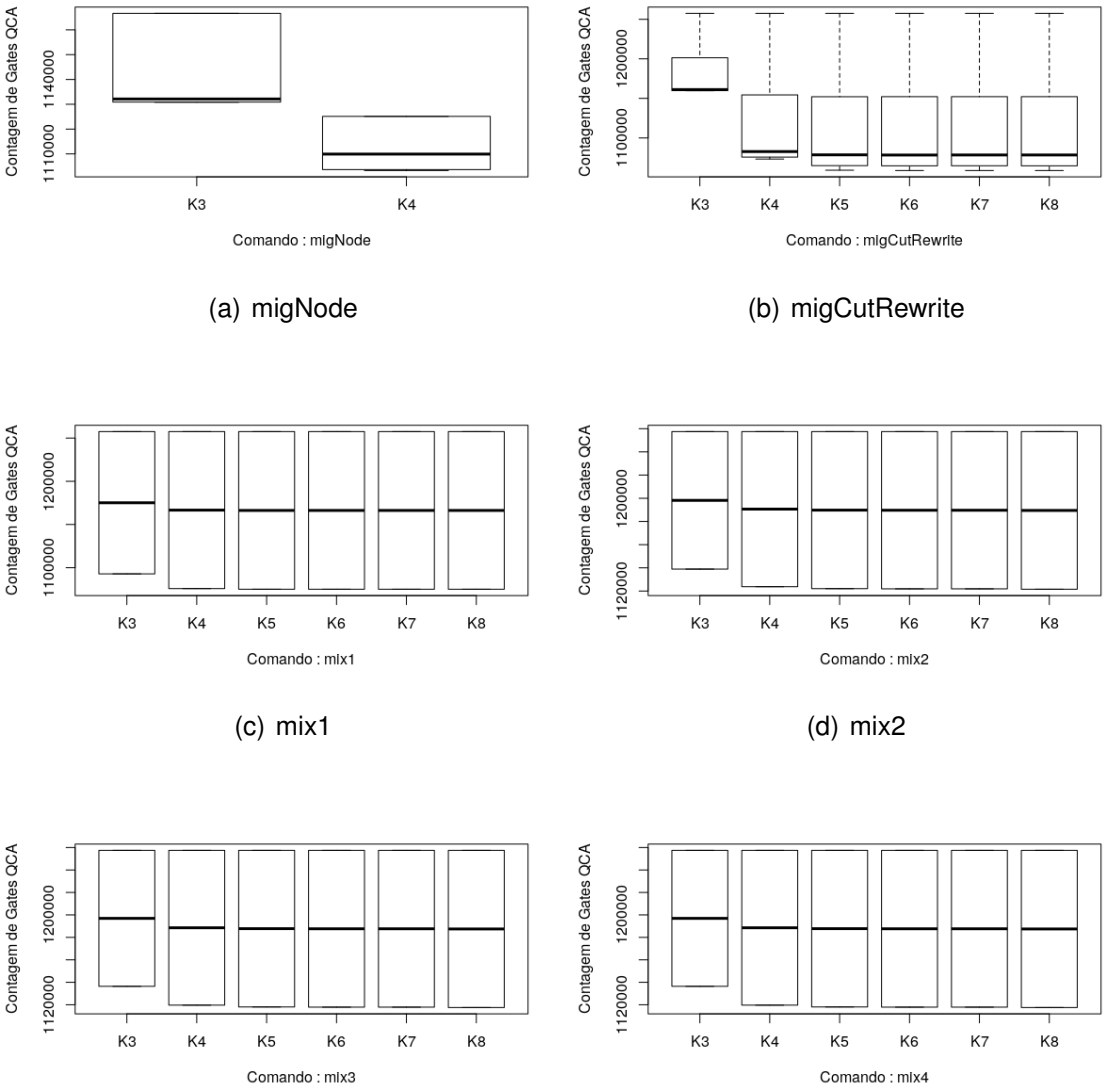
2 Utilizando o comando **migCutRewrite**, onde é possível configurar o algoritmo de *cortes-K*, sendo este utilizado com execuções de K=3 até K=8. Desta forma seria possível verificar se existe algum ponto de estabilização sobre o valor de *K*;

3 Utilizando o *mix* de comandos propostos (*mix1, mix2, mix3, mix4*), onde são invocados diferentes comandos (inclusive o comando migCutRewrite) em alternância com os demais implementados, variando o *K* com valores de 3 até 8;

Assim, todos os circuitos dos *benchmarks* avaliados foram executados nessas configurações, os resultados foram sumarizados e classificados. A Figura 61 apresenta os gráficos referentes as 3 diferentes avaliações.

Após a análise gráfica e numérica dos valores encontrados, percebe-se que ao incrementar o valor de *K*, obtém-se ganhos de otimização até um ponto de equilíbrio, o qual dependerá da heurística utilizada (comando). Entretanto, é possível observar que de modo geral para o comando *migNode* a melhor configuração sugerida seria a utilização do *K=4*.

Por outro lado, na análise geral tanto nos comandos *mix*, quanto no comandos *migCutRewrite* o valor de *K* deve ser fixado entre *K=5* e *K=6*, visto que são os pontos onde a porcentagem de otimização é equivalente a *K=8*, porém, com um esforço computacional consideravelmente menor e diferença de otimização em menos 0.01% na contagem total de elementos, conforme Tabela 4. É importante salientar que o valor máximo de K no experimento foi fixado em 8, em função de execuções com valores superiores apresentarem *runtime* impraticáveis para os experimentos. Considerando a média de otimização, a diferença para a melhor configuração encontrada é de 2% em média com desvio padrão menor, o Apêndice C apresenta a tabela das relações médias de otimizações encontradas para os comandos executados de forma sumarizada. Entretanto, a média de otimização pode não ser totalmente fidedigna, visto que existe um alto desvio padrão. Por outro lado, a Tabela 4 apresenta as relações de otimização total em relação a contagem original de elementos, o que resulta em dados mais consistentes.



(a) migNode

(b) migCutRewrite

(c) mix1

(d) mix2

(e) mix3

(f) mix4

Figura 61 – Análise gráfica incremento do *K*.

5.2.4 Análise do Número de Execuções

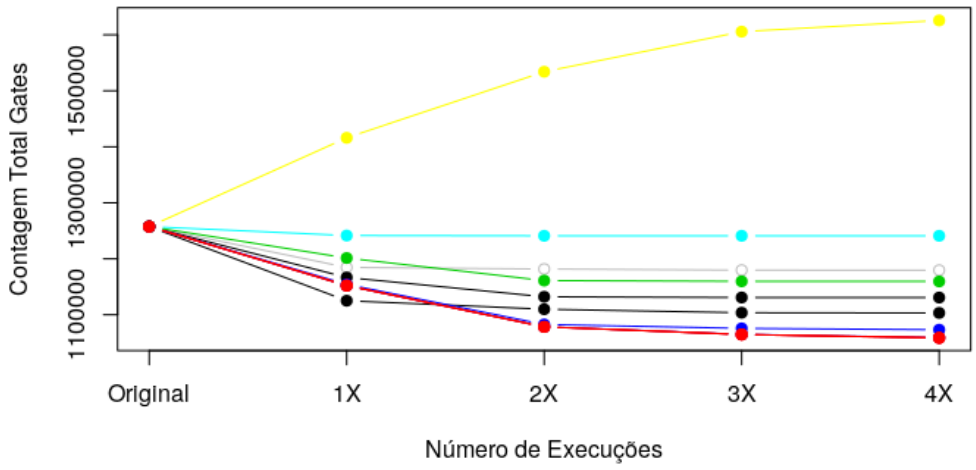
Outra análise necessária, foi quanto ao número de execuções de um mesmo comando. Assim, é conhecido que a maioria das heurísticas retornam bons resultados através de métodos iterativos. Essas metodologias são capazes de convergir para resultados melhores através de diversas execuções. Desta forma, foram efetuados alguns estudos a cerca da capacidade de otimização dos diferentes comandos implementados, avaliando o impacto da repetição de comandos, assim como qual a quantidade ideal de repetições para a conversão de um resultado razoável.

A Figura 62 apresenta os resultados das avaliações através de uma análise gráfica. Para isso, foram efetuadas de uma execução única, até 4 execuções dos comandos selecionados nas seguintes configurações, são eles:

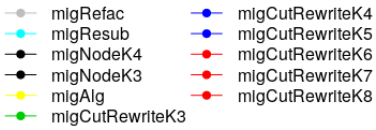
- migCutRewrite com variação do *K=3* até *K=8*;
- migNode com variação do *K=3* e *K=4*;
- migAlg;
- migResub;
- migRefac;

Após a análise gráfica e dos valores encontrados, foi possível observar que ao incrementar o número de execuções de um mesmo comando, obtém-se ganhos de otimização até um ponto de equilíbrio, que da mesma forma do experimento anterior, dependerá da heurística utilizada (comando). Entretanto, é possível observar que de modo geral, o ponto de equilíbrio tende a encontrar a melhor configuração entre duas ou três execuções em sequência.

Por outro lado, em alguns casos, dependendo do método é possível ainda obter um acréscimo de otimização realizando uma execução extra, como demonstrado no gráfico pelas curvas do método *migCutRewrite*, em especial, a partir de $K = 5$ onde as curvas todas estão sobrepostas a curva da configuração *migCutRewriteK6* com a cor vermelha no gráfico da Figura 62.



(a) Gráfico

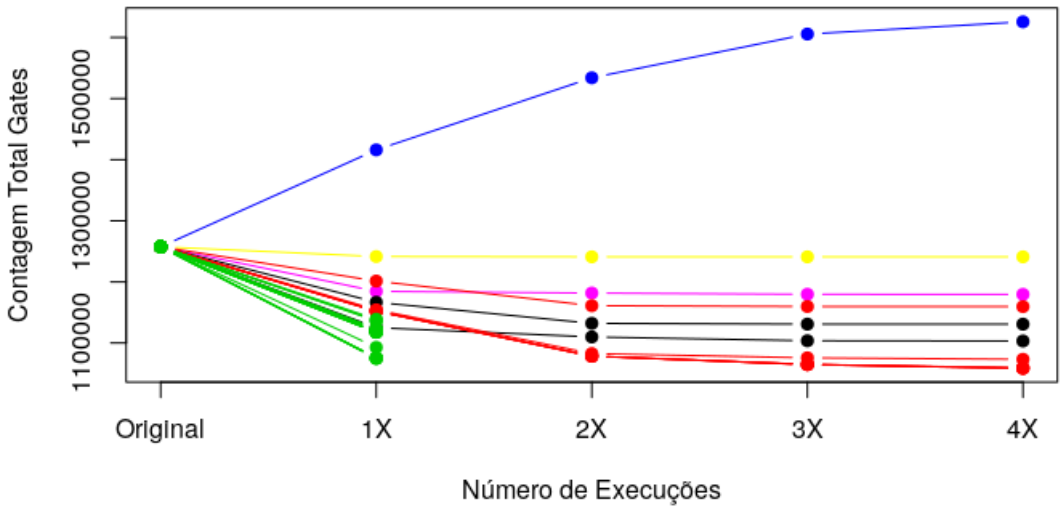


(b) Legenda

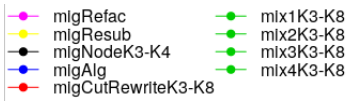
Figura 62 – Análise gráfica número de execuções.

5.2.5 Análise dos Comandos

Ao final, foi efetuada uma análise geral da efetividade dos comandos implementados em suas diferentes configurações de execução. Desta forma, foi possível identificar qual conjunto de comandos foi o mais efetivo na questão de otimização de elementos para todos os *benchmarks* analisados. A Figura 63 apresenta um gráfico da contagem de portas totais dos circuitos em função dos diferentes métodos. É importante ressaltar que para facilitar a visualização, as curvas estão agrupadas por cores pelos grupos de comandos, em outras palavras, todas as curvas das diferentes configurações de um mesmo comando aparecerão em uma mesma cor.



(a) Gráfico



(b) Legenda

Figura 63 – Análise gráfica dos comandos agrupados.

A Figura 64 apresenta um gráfico de pontos com a contagem total de elementos separados por cada comando, no eixo x são representados cada comando pelo seu ID o qual pode ser verificado na Tabela 4. Assim, é possível observar que os pontos mínimos (com o preenchimento preto) são os definidos pelos comandos citados anteriormente (ID = 37,41,45), corroborando com a análise.

A Tabela 4 apresenta a contagem final de elementos em função das diferentes configurações propostas, onde é possível verificar que os melhores comandos observados para todo o conjunto de *benchmarks* são os comandos **migCutRewrite**, como aponta o gráfico na Figura 63, em especial, considerando 4 execuções em série e com as configurações $K = 6$, $K = 7$ e $K = 8$.

Tabela 4 – Comparação da contagem de elementos comandos Mockturtle vs Original

ID	Comando	Total Elementos	% Otimização
1	Original	1.257.460	0,00
2	1XMigNodeK3	1.166.627	7,22
3	2XMigNodeK3	1.132.127	9,97
4	3XMigNodeK3	1.130.878	10,07
5	4XMigNodeK3	1.130.802	10,07
6	1XMigNodeK4	1.125.111	10,53
7	2XMigNodeK4	1.109.919	11,73
8	3XMigNodeK4	1.103.686	12,23
9	4XMigNodeK4	1.103.288	12,26
10	1XMigAlg	1.416.045	-12,61
11	2XMigAlg	1.534.004	-21,99
12	3XMigAlg	1.605.606	-27,69
13	4XMigAlg	1.625.618	-29,28
14	1XMigRefac	1.184.546	5,80
15	2XMigRefac	1.181.743	6,02
16	3XMigRefac	1.179.755	6,18
17	4XMigRefac	1.179.607	6,19
18	1XMigResub	1.241.523	1,27
19	2XMigResub	1.241.052	1,30
20	3XMigResub	1.241.037	1,31
21	4XMigResub	1.241.037	1,31
22	1XMigCutRewritek3	1.201.359	4,46
23	2XMigCutRewritek3	1.161.086	7,66
24	3XMigCutRewritek3	1.159.776	7,77
25	4XMigCutRewritek3	1.159.585	7,78
26	1XMigCutRewritek4	1.154.411	8,20
27	2XMigCutRewritek4	1.082.720	13,90
28	3XMigCutRewritek4	1.075.746	14,45
29	4XMigCutRewritek4	1.073.322	14,64
30	1XMigCutRewritek5	1.152.114	8,38
31	2XMigCutRewritek5	1.078.600	14,22
32	3XMigCutRewritek5	1.065.011	15,30
33	4XMigCutRewritek5	1.059.177	15,77
34	1XMigCutRewritek6	1.152.145	8,38
35	2XMigCutRewritek6	1.078.377	14,24
36	3XMigCutRewritek6	1.064.773	15,32
37	**4XMigCutRewritek6**	**1.058.828**	**15,80**
38	1XMigCutRewritek7	1.152.194	8,37
39	2XMigCutRewritek7	1.078.442	14,24
40	3XMigCutRewritek7	1.064.813	15,32
41	**4XMigCutRewritek7**	**1.058.830**	**15,80**
42	1XMigCutRewritek8	1.152.197	8,37
43	2XMigCutRewritek8	1.078.453	14,24
44	3XMigCutRewritek8	1.064.759	15,32
45	**4XMigCutRewritek8**	**1.058.763**	**15,80**
46	Mix1K3	1.092.854	13,09
47	Mix1K4	1.075.864	14,44
48	Mix1K5	1.075.090	14,50
49	Mix1K6	1.075.075	14,50
50	Mix1K7	1.075.092	14,50
51	Mix1K8	1.075.100	14,50
52	Mix2K3	1.138.921	9,43
53	Mix2K4	1.123.770	10,63
54	Mix2K5	1.122.073	10,77
55	Mix2K6	1.121.895	10,78
56	Mix2K7	1.121.886	10,78
57	Mix2K8	1.121.545	10,81
58	Mix3K3	1.136.422	9,63
59	Mix3K4	1.119.678	10,96
60	Mix3K5	1.118.046	11,09
61	Mix3K6	1.117.870	11,10
62	Mix3K7	1.117.860	11,10
63	Mix3K8	1.117.522	11,13
64	Mix4K3	1.136.309	9,63
65	Mix4K4	1.119.772	10,95
66	Mix4K5	1.118.277	11,07
67	Mix4K6	1.118.138	11,08
68	Mix4K7	1.118.133	11,08
69	Mix4K8	1.117.792	11,11

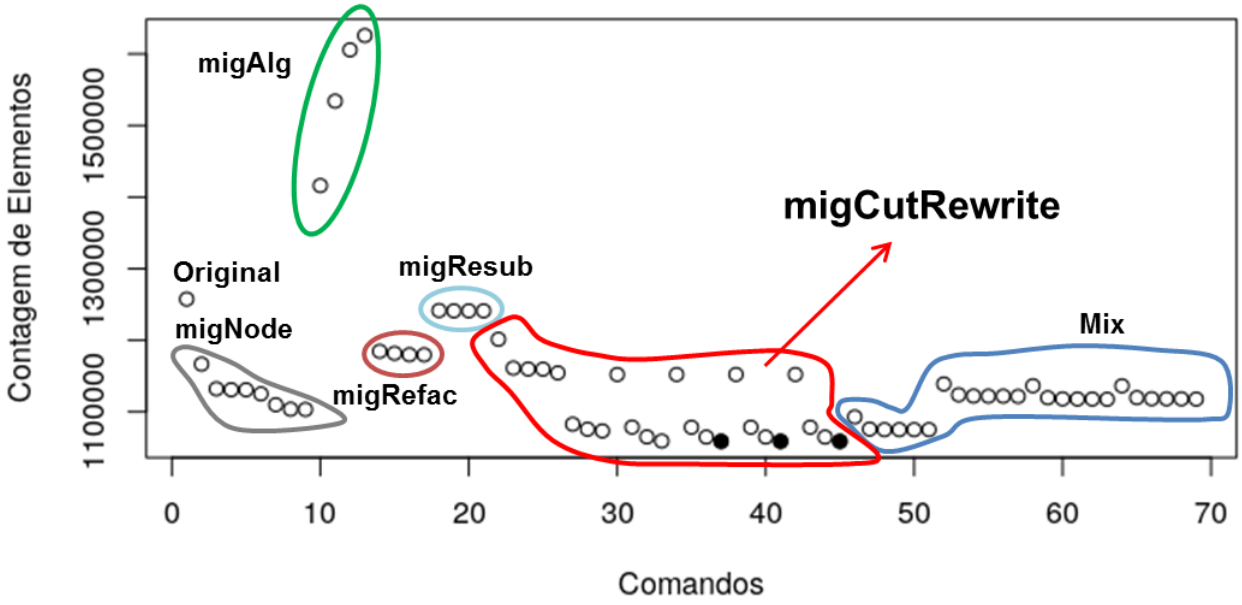


Figura 64 – Análise gráfica geral dos comandos em relação ao número total de elementos.

5.2.6 Benchmark Mockturtle Identificado

Considerando as informações identificadas sobre as avaliações dos melhores comandos implementados, foi possível propor um novo ponto de síntese para a tecnologia QCA. Em outras palavras, foi possível identificar duas possiveis configurações de circuitos iniciais otimizados, ou seja, novas versões otimizadas dos circuitos dos *benchmarks* selecionados. A primeira configuração sugerida é utilizando a melhor configuração (a que atingiu maior grau de otimização) identificada de forma geral para todos os circuitos, denominada **4migCutRewriteK6**, onde é possível atingir $15,8\%$ de otimização em média sobre o circuito inicial na contagem de elementos para a construção de leiautes. Essa configuração utiliza 4 execuções em série do comando **migCutRewrite**, com o valor de $K = 6$. Embora essa taxa de otimização seja possível alcançar com outras configurações de comandos, essa configuração é a que exige menor esforço computacional, esta nova versão dos circuitos foi denominada *middle*. A tabela 5 abaixo apresenta a taxa de otimização com a configuração *middle* para cada *benchmark*.

A segunda configuração, é referente a melhor configuração dentro os comandos para cada conjunto de circuitos (*benchmark*), ou seja, o comando que é possível obter o maior grau de otimização, visto que é possível selecionar dentre todas as configurações qual atingiu maior otimização para cada *benchmark* específico. A tabela 6 abaixo apresenta as configurações encontradas para cada *benchmark*, assim como sua taxa de otimização, este nova versão do conjunto de circuitos foi denominado *best*.

Tabela 5 – Tabela de comandos vs taxa de otimização para benchmark *middle*

Benchmark	Número de Circuitos	Comando	% Otimização
UFV	21	4XmigCutRewriteK6	28,34
TOY	19	4XmigCutRewriteK6	31,3
MAJ	14	4XmigCutRewriteK6	0
ISCAS'85	11	4XmigCutRewriteK6	19,8
EPFL	20	4XmigCutRewriteK6	19,85
MIG	31	4XmigCutRewriteK6	10,39
MIGoriginal	28	4XmigCutRewriteK6	17,47

Tabela 6 – Tabela de comandos vs taxa de otimização para benchmark *best*

Benchmark	Número de Circuitos	Comando	% Otimização
UFV	21	mix1K4-Gates	29,95
TOY	19	1XmigNodeK4	39,46
MAJ	14	1XmigNodeK3	1,32
ISCAS'85	11	4XmigNodeK4	27,32
EPFL	20	4XmigCutRewriteK6	19,86
MIG	31	4XmigCutRewriteK8	10,41
MIGoriginal	28	4XmigCutRewriteK8	17,49

5.2.7 Considerações Finais

Esta seção apresentou a segunda parte das avaliações, as quais foram a cerca da efetividade dos comandos implementados sobre a ferramenta Fiction 2.1, utilizando o módulo *Mockturtle* do conjunto de bibliotecas *EPFL Logic Synthesis Libraries*. De forma geral, foram exploradas 68 configurações de comandos, considerando 7 *benchmarks* diferentes com 144 circuitos ao total, o que resultou em 9.792 circuitos. Todos os resultados foram tabelados e analisados. Ao final, foi possível identificar diversas informações importantes sobre o comportamento das diferentes configurações de comandos. Inicialmente foi identificado que o comando *migCutRewrite* obteve as melhores taxas de otimizações, com o valor de $K = 6$, chegando a $15,8\%$ em média na taxa de otimização. Além disso, foram propostos duas novas configurações de circuitos dos *benchmarks* utilizados, as quais diminuem a contagem do número de elementos inicial dos circuitos originais. A primeira aplicando o comando **4migCutRewriteK6**, otimizando $15,8\%$ em média na contagem de elementos, sendo a configuração denominada *middle*, e a segunda executando comandos específicos para cada *benchmark*, chegando a 39.46% no caso do *benchmark* TOY, sendo a configuração denominada de *best*. Na próxima seção, serão apresentados os resultados referentes aos leiautes finais das novas configurações dos circuitos, as quais viabilizaram a análise em relação aos leiautes do método *Migortho* em relação ao método *Ortho*.

5.3 Análise Fiction 2.1 - Migortho + Mockturtle vs Fiction - Ortho

Logo após identificar as melhores configurações de comandos para aplicar otimizações iniciais nas descrições de entrada, foi possível avaliar a efetividade dessas otimizações na descrição inicial, em relação aos leiautes finais gerados. Assim, foi possível avaliar de forma geral as contribuições deste trabalho, congregando todas as suas propostas, tanto na questão da estrutura de dados a ser explorada, neste caso o MIG e seus elementos de *design*, quanto nos métodos de otimização baseados em MIG implementados através dos comandos da biblioteca *Mockturtle*, além das duas configurações (*middle* e *best*) dos circuitos. Esta seção apresenta a comparação entre o método *Migortho* proposto por esse trabalho com as configurações *middle* e *best* dos *benchmarks*, em relação aos resultados do método *Ortho* original. As Tabelas a seguir, apresentam os resultados dos leiautes para os circuitos analisados.

O primeiro experimento avaliou os resultados referentes ao método *Migortho* considerando os circuitos resultantes do melhor comando geral das avaliações *Mockturtle*, neste caso, os circuitos resultantes do comando **4XmigCutRewriteK6** denominada configuração *middle*. Em um segundo momento, foram avaliados os resultados do método *Migortho*, em relação as melhores configurações por *benchmark*, ou seja, os circuitos resultantes da configuração denominada *best*.

Tanto as Tabelas 7, 8, 9 e 10 quanto as Tabelas 11, 12, 13 e 14, apresentam os resultados sobre as seguintes características: em termos da área do *grid* (A), ou seja, estimativa de colunas e linhas de *subgrids* 5x5 utilizados, também número de portas (G), número de elementos de fio (W), número de cruzamento entre fios (CROSS), o caminho crítico do circuito (C.P.) e o número total de *subgrids* utilizados (Total Subgrid). Cada linha da tabela apresenta os dois resultados de cada circuito, o identificador ♣ representa os dados da solução proposta neste trabalho pelo método *Migortho*, assim como o identificador $\triangle$ representa os dados da solução original do método *Ortho*, por último, o identificador $\star$ indica qual das duas soluções é a mais otimizada em relação a área.

É importante salientar que os *benchmarks* MAJ e MIG não foram utilizados nestes experimentos, além de alguns circuitos do *benchmark* UFV, pois possuem portas majoritárias em suas descrições, o que não é aceito pela versão original do método *Ortho*, inviabilizando a comparação. Além disso, no caso do *benchmark* da EPFL (AMARÚ; GAILLARDON; DE MICHELI, 2015), foi aplicada uma restrição de tamanho na seleção dos circuitos a serem executados, visto que foi identificado limitações como um alto tempo computacional e alto consumo de memória para execuções com circuitos acima de 40 mil elementos, mesmo sendo necessário a execução deste *benchmark* em um ambiente diferenciado, com maior poder de processamento e memória, com um processador AMD Opteron 2.3GHz, 64 núcleos e 128GB de memória RAM rodando um linux server. Ademais, no que diz respeito ao *runtime* dos experimentos, é importante salientar que as execuções foram efetuadas já levando em o arquivo Verilog resultante da aplicação dos comandos, o que descarta do tempo de execução a etapa de síntese lógica. Neste sentido, as estimativas de *runtime* do método *Migortho* aqui descritas são aproximações, visto que não foi o foco dos experimentos, além de não ser possível a execução única da síntese no servidor de experimentação. Quanto ao *runtime* dos *benchmarks* de menor tamanho, com circuitos com apenas dezenas ou centenas de portas (*benchmarks* TOY, MAJ, UFV) a estimativa de execução foi de menos de 1 minuto. Por outro lado, no caso do *benchmark* intermediário ISCAS'85 o *runtime* chegou por volta de 70 minutos da versão *Ortho* contra 40 minutos da versão *Migortho*. No caso dos *benchmarks* com maior número de elementos e complexidade (*benchmarks* EPFL, MIG, MIGoriginal) o tempo de *runtime* para todos *benchmarks* foi de aproximadamente 120 horas.

Tabela 7 – Comparação do método *Ortho* vs *Migortho benchmark TOY - middle*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
1bitAdderAOIG♣	11 x 7	15	40	3	16	77	
1bitAdderAOIG△	11 x 7	15	40	3	16	77	0,0 %
1bitAdderMaj♣⋆	2 x 2	2	0	0	2	4	
1bitAdderMaj△	22 x 11	30	117	23	32	242	98,3 %
b1-r2♣⋆	12 x 8	17	48	7	17	96	
b1-r2△	15 x 7	19	37	5	19	105	8,6 %
clpl♣	14 x 2	14	0	0	14	28	
clpl△	14 x 2	14	0	0	14	28	0,0 %
FA♣⋆	7 x 4	9	13	1	10	28	
FA△	15 x 7	19	51	4	19	105	73,3 %
FS♣⋆	2 x 2	2	0	0	2	4	
FS△	17 x 6	21	74	10	20	102	96,1 %
HA♣	6 x 6	10	24	5	10	36	
HA△⋆	7 x 5	10	19	2	11	35	-2,9 %
HS♣	9 x 4	11	23	3	11	36	
HS△⋆	7 x 5	10	18	1	11	35	-2,9 %
mux21♣	4 x 2	5	3	0	5	8	
mux21△	4 x 2	5	3	0	5	8	0,0 %
mux41♣	13 x 8	18	66	13	20	104	
mux41△⋆	12 x 6	16	26	5	17	72	-44,4 %
newtag♣⋆	11 x 3	11	12	0	13	33	
newtag△	17 x 7	19	45	4	23	119	72,3 %
par-check♣⋆	12 x 9	17	72	21	20	108	
par-check△	15 x 9	21	48	7	23	135	20,0 %
par-gen♣⋆	9 x 7	13	41	13	15	63	
par-gen△	10 x 7	14	35	2	16	70	10,0 %
RCA2♣⋆	24 x 13	32	180	25	34	312	
RCA2△	29 x 14	38	210	27	40	406	23,2 %
t♣⋆	8 x 5	11	24	4	12	40	
t△	11 x 6	14	27	5	14	66	39,4 %
xnor2♣	6 x 4	8	12	3	9	24	
xnor2△	6 x 4	8	10	2	9	24	0,0 %
xor2♣	5 x 3	6	8	0	7	15	
xor2△	5 x 3	6	8	0	7	15	0,0 %
xor5-r1♣⋆	18 x 13	26	138	30	30	234	
xor5-r1△	27 x 12	34	86	2	38	324	27,8 %
xor5R♣⋆	18 x 13	26	138	30	30	234	
xor5R△	35 x 15	45	158	11	49	525	55,4 %
Otimização △vs♣		**0,71**	**0,83**	**1,4**	**0,72**	**0,60**	
Desvio Padrão △		**10,90**	**55,33**	**7,41**	**11,83**	**142,63**	
Desvio Padrão ♣		**8,08**	**52,97**	**10,56**	**9,07**	**88,28**	

△-Ortho ♣-Migortho ⋆-Melhor

Tabela 8 – Comparação do método *Ortho* vs *Migortho benchmark UFV - middle*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
1bitAdderAOIG♣	11 x 7	15	40	3	16	77	
1bitAdderAOIG△	11 x 7	15	40	3	16	77	0,0 %
b1-r2♣⋆	12 x 8	17	48	7	17	96	
b1-r2△	15 x 7	19	37	5	19	105	8,6 %
clpl♣	14 x 2	14	0	0	14	28	
clpl△	14 x 2	14	0	0	14	28	0,0 %
majority-5-r1♣⋆	6 x 3	7	7	0	8	18	
majority-5-r1△	18 x 7	21	59	3	24	126	85,7 %
mux2x1♣	4 x 2	5	3	0	5	8	
mux2x1△	4 x 2	5	3	0	5	8	0,0 %
parity♣⋆	70 x 46	100	1.805	290	115	3.220	
parity△	103 x 46	133	525	16	148	4.738	32,0 %
parityChecker♣⋆	14 x 10	20	90	22	23	140	
parityChecker△	15 x 10	21	46	6	24	150	6,7 %
parityGenerator♣⋆	9 x 7	13	41	13	15	63	
parityGenerator△	10 x 7	14	20	4	16	70	10,0 %
t-5♣	8 x 5	11	24	6	10	40	
t-5△	8 x 5	11	22	4	12	40	0,0 %
t♣⋆	8 x 5	11	24	4	12	40	
t△	11 x 6	14	27	5	14	66	39,4 %
xnor♣	6 x 4	8	12	3	9	24	
xnor△	6 x 4	8	10	2	9	24	0,0 %
xor♣	5 x 4	7	10	0	8	20	
xor△	5 x 4	7	9	0	8	20	0,0 %
xor5-r1♣⋆	18 x 13	26	138	30	30	234	
xor5-r1△	27 x 12	34	86	2	38	324	27,8 %
xor5R♣⋆	18 x 13	26	138	30	30	234	
xor5R△	35 x 15	45	158	11	49	525	55,4 %
Otimização △vs♣		**0,78**	**2,28**	**6,69**	**0,85**	**0,67**	
Desvio Padrão △		**32,67**	**136,10**	**4,43**	**40,84**	**1.243,33**	
Desvio Padrão ♣		**23,96**	**472,85**	**75,84**	**35,44**	**843,97**	

△-Ortho ♣-Migortho ⋆-Melhor

Tabela 9 – Comparação do método *Ortho* vs *Migortho benchmark ISCAS'85 - middle*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
c1355♣⋆	775 x 465	1.207	137.537	15.482	1.239	360.375	
c1355△	1.140 x 452	1.559	111.445	4.741	1.591	515.280	30,1 %
c17♣	8 x 4	11	12	0	10	32	
c17△	8 x 4	11	12	0	10	32	0,0 %
c1908♣⋆	647 x 396	1.010	98.137	10.885	1.042	256.212	
c1908△	852 x 400	1.219	88.735	6.459	1.251	340.800	24,8 %
c2670♣⋆	999 x 514	1.456	215.507	24.733	1.499	513.486	
c2670△	1.515 x 630	2.086	201.492	12.737	2.038	954.450	46,2 %
c3540♣⋆	1.564 x 825	2.345	544.295	70.427	2.388	1.290.300	
c3540△	2.200 x 842	2.997	534.425	43.114	3.041	1.852.400	30,3 %
c432♣⋆	331 x 167	462	25.840	2.863	497	55.277	
c432△	458 x 172	594	27.604	2.475	629	78.776	29,8 %
c499♣⋆	628 x 396	991	96.925	10.009	1.023	248688	
c499△	828 x 412	1.207	86.348	4.627	1.239	341.136	27,1 %
c5315♣⋆	2.625 x 1.627	4.125	1.363.095	92.324	4.233	4.270.875	
c5315△	3.796 x 1.675	5.346	1.553.753	94.176	5.428	6.358.300	32,8 %
c6288♣⋆	4.399 x 1.692	6.059	1.225.744	54.208	6.090	7.443.108	
c6288△	6.694 x 2.142	8.804	1.418.414	52.871	8.835	14.338.548	48,1 %
c7552♣⋆	3.049 x 1.610	4.578	1.411.991	10.1495	4.628	4.908.890	
c7552△	4.644 x 1.891	6.476	1.545.994	79.435	6.484	8.781.804	44,1 %
c880♣⋆	447 x 242	641	51.015	7.404	681	108.174	
c880△	693 x 288	932	43.359	3.419	960	199.584	45,8 %
Otimização △vs♣		**0,73**	**0,92**	**1,28**	**0,74**	**0,58**	
Desvio Padrão △		**2.815,79**	**656.508,36**	**34.197,09**	**2.822,65**	**4.737.184,00**	
Desvio Padrão ♣		**1.969,17**	**575.162,16**	**37.471,72**	**1.981,71**	**2.559.839,74**	

△-Ortho ♣-Migortho ⋆-Melhor

Tabela 10 – Comparação do método *Ortho* vs *Migortho benchmark EPFL - middle*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
adder♣	1.403 x 1.149	2.296	953.159	115.321	2.549	1.612.047	
adder△	2.541 x 1.149	3.434	479.560	34.547	3.688	2.919.609	44,8 %
arbiter♣	17.980 x 11.587	29.311	114.912.479	11.521.584	29.482	208.334.260	
arbiter△	23.874 x 11.967	35.585	136.887.956	10.994.525	35.716	285.700.158	27,1 %
bar♣	4.952 x 2.400	7.217	3.811.880	562.675	7.351	11.884.800	
bar△	7.058 x 3.078	10.001	4.725.773	563.099	10.135	21.724.524	45,3 %
cavlc♣	1.155 x 616	1.761	421.531	53.942	1.770	711.480	
cavlc△	1.666 x 638	2.294	519.390	49.473	2.303	1.062.908	33,1 %
ctrl♣	253 x 156	402	25.916	3.572	400	39.468	
ctrl△	417 x 191	601	42.236	5.901	607	79.647	50,4 %
dec♣	571 x 344	907	131.972	17.104	913	196.424	
dec△	567 x 313	872	127.803	17.298	878	177.471	-10,7 %
i2c♣	2.293 x 1.337	3.501	1.351.985	167.070	3.620	3.065.741	
i2c△	2.875 x 1.318	4.064	979.311	93.748	4.163	3.789.250	19,1 %
int2float♣	393 x 231	613	48.008	7.678	615	90.783	
int2float△	574 x 238	801	53.939	5.989	807	136.612	33,5 %
max♣	4.122 x 2.275	5.885	4.140.638	527.322	6.393	9.377.550	
max△	6.379 x 2.726	8.593	4.048.671	235.526	9.101	17.389.154	46,1 %
priority♣	1.425 x 641	1.939	578.159	68.191	2.063	913.425	
priority△	2.356 x 979	3.207	848.983	25.644	3.231	2.306.524	60,4 %
router♣	419 x 225	583	44.897	3.941	640	94.275	
router△	523 x 254	716	18.791	636	749	132.842	29,0 %
sin♣	8.402 x 4.223	12.601	10.750.572	718.043	12.623	35.481.646	
sin△	12.014 x 4.864	16.854	8.712.223	729.675	16.875	58.436.096	39,3 %
voter♣	16.954 x 9.817	25.770	34.280.680	15.99.057	26.770	166.437.418	
voter△	30.542 x 9.935	39.476	43.536.463	1.471.132	19.542	303.434.770	45,1 %
Otimização △vs♣		**0,73**	**1,08**	**1,27**	**0,69**	**0,59**	
Desvio Padrão △		**13.221,80**	**38.344.251,82**	**3.005.439,70**	**10340,19**	**108.210.744,06**	
Desvio Padrão ♣		**9.714,98**	**31.975.710,17**	**3.139.822,97**	**9.881,99**	**69.421.866,12**	

△-Ortho ♣-Migortho ⋆-Melhor

Os resultados para a configuração *middle* demonstram que em média as otimizações com a nova abordagem chegaram entre 33% até 42% em relação a área final (Total de *Subgrids*), onde a solução proposta neste trabalho foi capaz de gerar a melhor configuração em comparação com a versão original. Da mesma forma como nos experimentos anteriores, é importante enfatizar que, nos casos os quais a solução original obteve uma solução menor, o circuito mínimo é composto pela lógica *And-Or-Inverter*. Além disso, é possível observar o mesmo comportamento dos experimentos anteriores, onde quanto maior o circuito, mais significativa é sua redução no circuito QCA.

Em média, a relação de otimização na representação do número de portas, também se manteve linear em relação as outras características do leiaute final e caminho crítico. Entretanto, é importante salientar que houve um crescimento acentuado em alguns casos tanto na utilização de elementos fio (W), quanto no cruzamento de fios (CROSS), o que já se espera pelo aumento da complexidade de roteamento.

Logo após, foram efetuadas avaliações comparativas com a configuração denominada *best* dos *benchmarks* aplicados ao método *Migortho*, em relação ao resultados dos circuitos originais aplicados ao método *Ortho*. Desse modo, foi possível verificar se as otimizações realizadas no circuito de entrada podem se refletir em mesmo grau na taxa de otimização do leiaute final. Da mesma forma que o experimento anterior, os *benchmarks* MAJ e MIG não foram utilizados nestes experimentos, além de alguns circuitos do *benchmark* UFV, pois possuem portas majoritárias em suas descrições, o que não é aceito pela versão original do método *Ortho*, inviabilizando a comparação.

Tabela 11 – Comparação do método *Ortho* vs *Migortho benchmark TOY - best*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
1bitAdderAOIG♣	12 x 9	18	75	18	19	108	
1bitAdderAOIG△⋆	11 x 7	15	40	3	16	77	-40,3 %
1bitAdderMaj♣⋆	2 x 2	2	0	0	2	4	
1bitAdderMaj△	22 x 11	30	117	23	32	242	98,3 %
b1-r2♣	17 x 10	24	97	16	24	170	
b1-r2△⋆	15 x 7	19	37	5	19	105	-61,9 %
clpl♣	14 x 2	14	0	0	14	28	
clpl△	14 x 2	14	0	0	14	28	0,0 %
FA♣⋆	2 x 2	1	0	0	1	4	
FA△	15 x 7	19	51	4	19	105	96,2 %
FS♣⋆	2 x 2	2	0	0	2	4	
FS△	17 x 6	21	74	10	20	102	96,1 %
HA♣	8 x 6	12	26	3	12	48	
HA△⋆	7 x 5	10	19	2	11	35	-37,1 %
HS♣⋆	2 x 2	2	0	0	2	4	
HS△	7 x 5	10	18	1	11	35	88,6 %
mux21♣	7 x 4	9	14	1	10	28	
mux21△⋆	4 x 2	5	3	0	5	8	-250,0 %
mux41♣	12 x 6	16	54	12	17	72	
mux41△	12 x 6	16	26	5	17	72	0,0 %
newtag♣⋆	11 x 4	12	19	0	14	44	
newtag△	17 x 7	19	45	4	23	119	63,0 %
par-check♣⋆	13 x 9	19	62	6	21	117	
par-check△	15 x 9	21	48	7	23	135	13,3 %
par-gen♣⋆	9 x 7	13	38	4	15	63	
par-gen△	10 x 7	14	35	2	16	70	10,0 %
RCA2♣	26 x 18	39	332	63	41	468	
RCA2△⋆	29 x 14	38	210	27	40	406	-15,3 %
t♣	13 x 10	19	103	17	22	130	
t△⋆	11 x 6	14	27	5	14	66	-97,0 %
xnor2♣	8 x 4	10	15	3	11	32	
xnor2△⋆	6 x 4	8	10	2	9	24	-33,3 %
xor2♣	7 x 4	9	15	3	10	28	
xor2△⋆	5 x 3	6	8	0	7	15	-86,7 %
xor5-r1♣⋆	18 x 13	26	126	16	30	234	
xor5-r1△	27 x 12	34	86	2	38	324	27,8 %
xor5R♣⋆	18 x 13	26	126	16	30	234	
xor5R△	35 x 15	45	158	11	49	525	55,4 %
Otimização △vs♣		**0,76**	**1,01**	**1,58**	**0,78**	**0,73**	
Desvio Padrão △		**10,90**	**49,74**	**7,41**	**11,83**	**142,63**	
Desvio Padrão ♣		**9,88**	**76,68**	**14,73**	**10,76**	**115,995**	

△-Ortho ♣-Migortho ⋆-Melhor

Os resultados dos experimentos sobre os *benchmarks* na configuração *best* demonstram que nem sempre a maior redução no circuito de entrada pode gerar um leiaute final mais reduzido. De forma geral, a solução proposta neste trabalho foi capaz de gerar a melhor configuração em comparação com a versão original, com redução do leiaute final entre 26% a 33%, porém, ao analisar os resultados em relação a configuração *middle*, é possível perceber que houve uma degeneração nas taxas de otimização do leiaute final.

As Figuras 65 e 66 apresentam análises gráficas dos experimentos através de gráficos do tipo caixa (*boxplot*), onde é possível observar algumas características sobre os dados. As informações foram separadas em dois conjuntos de gráficos devido a grande diferença de escala dos *benchmarks*. Assim, com a separação dos gráficos em função da contagem de elementos dos *benchmarks* a visualização é facilitada. Neste sentido, a Figura 65 apresenta as informações referentes aos *benchmarks* TOY e UFV, por outro lado, a Figura 66 apresenta os gráficos referentes aos *benchmarks* ISCAS'85 e EPFL.

Tabela 12 – Comparação do método *Ortho* vs *Migortho benchmark UFV - best*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
1bitAdderAOIG♣	11 x 7	15	40	3	16	77	
1bitAdderAOIG△	11 x 7	15	40	3	16	77	0,0%
b1-r2♣⋆	12 x 8	17	48	7	17	96	
b1-r2△	15 x 7	19	37	5	19	105	8,6%
clpl♣	14 x 2	14	0	0	14	28	
clpl△	14 x 2	14	0	0	14	28	0,0%
majority-5-r1♣⋆	6 x 3	7	7	0	8	18	
majority-5-r1△	18 x 7	21	59	3	24	126	85,7%
mux2x1♣	4 x 2	5	3	0	5	8	
mux2x1△	4 x 2	5	3	0	5	8	0,0%
parity♣⋆	70 x 46	100	1.805	290	115	3.220	
parity△	103 x 46	133	525	16	148	4.738	32,0%
parityChecker♣⋆	14 x 10	20	90	22	23	140	
parityChecker△	15 x 10	21	46	6	24	150	6,7%
parityGenerator♣⋆	9 x 7	13	41	13	15	63	
parityGenerator△	10 x 7	14	20	4	16	70	10,0%
t-5♣	8 x 5	11	24	6	10	40	
t-5△	8 x 5	11	22	4	12	40	0,0%
t♣⋆	8 x 5	11	24	4	12	40	
t△	11 x 6	14	27	5	14	66	39,4%
xnor♣	6 x 4	8	12	3	9	24	
xnor△	6 x 4	8	10	2	9	24	0,0%
xor♣	5 x 4	7	10	0	8	20	
xor△	5 x 4	7	9	0	8	20	0,0%
xor5-r1♣⋆	18 x 13	26	138	30	30	234	
xor5-r1△	27 x 12	34	86	2	38	324	27,8%
xor5R♣⋆	18 x 13	26	138	30	30	234	
xor5R△	35 x 15	45	158	11	49	525	55,4%
Otimização △vs♣		**0,78**	**2,28**	**6,69**	**0,79**	**0,67**	
Desvio Padrão △		**32,67**	**136,10**	**4,43**	**36,46**	**1.242,25**	
Desvio Padrão ♣		**23,96**	**472,85**	**75,84**	**27,80**	**842,92**	

△-Ortho ♣-Migortho ⋆-Melhor

Tabela 13 – Comparação do método *Ortho* vs *Migortho benchmark ISCAS'85 - best*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
c1355♣⋆	641 x 397	1.005	100.834	12.013	1.037	254.477	
c1355△	1.140 x 452	1.559	111.445	4.741	1.591	515.280	50,6 %
c17♣	13 x 10	19	103	17	22	130	
c17△⋆	8 x 4	11	12	0	10	32	-306,3 %
c1908♣	847 x 511	1.325	160.436	17.140	1.357	432.817	
c1908△⋆	852 x 400	1.219	88.735	6.459	1.251	340.800	-27,0 %
c2670♣	1.264 x 759	1.914	384.094	36.882	2.009	959.376	
c2670△⋆	1.515 x 630	2.086	201.492	12.737	2.038	954.450	-0,5 %
c3540♣	2.014 x 1.069	3.036	836.311	99.908	3.080	2.152.966	
c3540△⋆	2.200 x 842	2.997	534.425	43.114	3.041	1.852.400	-16,2 %
c432♣	399 x 231	594	34.971	4.117	629	92.169	
c432△⋆	458 x 172	594	27.604	2.475	629	78.776	-17,0 %
c499♣⋆	641 x 397	1.005	100.834	12.013	1.037	254.477	
c499△	828 x 412	1.207	86.348	4.627	1.239	341.136	25,4 %
c5315♣⋆	3.244 x 1.936	5.018	2.251.729	165.513	5.163	6.280.384	
c5315△	3.796 x 1.675	5.346	1.553.753	94.176	5.428	6.358.300	1,2 %
c6288♣⋆	2.779 x 1.395	4.142	919.931	60.661	4.173	3.876.705	
c6288△	6.694 x 2.142	8.804	1.418.414	52.871	8.835	14.338.548	73,0 %
c7552♣	4.263 x 2.456	6.546	3.335.923	310.495	6.711	10.469.928	
c7552△⋆	4.644 x 1.891	6.476	1.545.994	79.435	6.484	8.781.804	-19,2 %
c880♣	652 x 422	1.025	105.694	13.380	1.069	275.144	
c880△⋆	693 x 288	9.32	43.359	3.419	960	199.584	-37,9 %
Otimização △vs♣		**0,82**	**1,47**	**2,41**	**0,83**	**0,74**	
Desvio Padrão △		**2.815,79**	**656.508,36**	**34.197,09**	**2.822,65**	**4.737.184,06**	
Desvio Padrão ♣		**2.086,47**	**1.085.888,06**	**95.326,78**	**2.129,39**	**3.361.680,92**	

△-Ortho ♣-Migortho ⋆-Melhor

Inicialmente, é possível verificar através dos gráficos nas Figuras 65(a) e 66(a) que a utilização das configurações propostas na representação dos circuitos, acarretou em uma otimização em relação ao número de gates utilizados, sendo verificada pela posição da mediana no gráfico *boxplot* (maior concentração dos valores), porém, é possível verificar que embora a configuração *middle* seja a solução intermediária, ela foi capaz de obter a maior redução no leiaute final dos circuitos.

Da mesma forma, esse comportamento pode ser verificado para os outros gráficos, como por exemplo, caminho crítico (Figuras 65(d) e 66(d)) e total de *subgrids* 5x5 utilizados (Figuras 65(e) e 66(e)). É importante salientar, que esse último gráfico está diretamente relacionado com a área final do leiaute QCA. Contudo, também outra informação relevante é que no caso do fator de cruzamento de fios (CROSS), houve um aumento considerável em relação as soluções originais, isso é esperado pela maior complexidade de roteamento com a utilização da nova abordagem, porém, destaca-se a solução da configuração *best*, onde além do cruzamento de fios a contagem de elementos do tipo fio (W) também foi incrementada consideravelmente.

Tabela 14 – Comparação do método *Ortho* vs *Migortho benchmark EPFL - best*

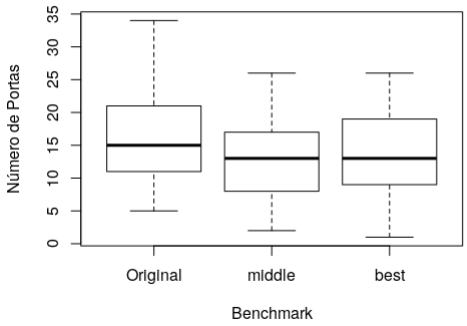
Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
adder♣	1.403 x 1.149	2.296	953.159	115.321	2.549	1.612.047	
adder△	2.541 x 1.149	3.434	479.560	34.547	3.688	2.919.609	44,8%
arbiter♣	17.980 x 11.587	29.311	114.912.479	11.521.584	29.482	208.334.260	
arbiter△	23.874 x 11.967	35.585	136.887.956	10.994.525	35.716	285.700.158	27,1%
bar♣	4.952 x 2.400	7.217	3.811.880	562.675	7.351	11.884.800	
bar△	7.058 x 3.078	10.001	4.725.773	563.099	10.135	21.724.524	45,3%
cavlc♣	1.155 x 616	1.761	421.531	53.942	1.770	711.480	
cavlc△	1.666 x 638	2.294	519.390	49.473	2.303	1.062.908	33,1%
ctrl♣	253 x 156	402	25.916	3.572	400	39.468	
ctrl△	417 x 191	601	42.236	5.901	607	79.647	50,4%
dec♣	571 x 344	907	131.972	17.104	913	196.424	
dec△	567 x 313	872	127.803	17.298	878	177.471	-10,7%
i2c♣	2.293 x 1.337	3.501	1.351.985	167.070	3.620	3.065.741	
i2c△	2.875 x 1.318	4.064	979.311	93.748	4.163	3.789.250	19,1%
int2float♣	393 x 231	613	48.008	7.678	615	90.783	
int2float△	574 x 238	801	53.939	5.989	807	136.612	33,5%
max♣	4.122 x 2.275	5.885	4.140.638	527.322	6.393	9.377.550	
max△	6.379 x 2.726	8.593	4.048.671	235.526	9.101	17.389.154	46,1%
priority♣	1.425 x 641	1.939	578.159	68.191	2.063	913.425	
priority△	2.356 x 979	3.207	848.983	25.644	3.231	2.306.524	60,4%
router♣	419 x 225	583	44.897	3.941	640	94.275	
router△	523 x 254	716	18.791	636	749	132.842	29,0%
sin♣	8.402 x 4.223	12.601	10.750.572	718.043	12.623	35.481.646	
sin△	12.014 x 4.864	16.854	8.712.223	729.675	16.875	58.436.096	39,3%
voter♣	16.954 x 9.817	25.770	34.280.680	1.599.057	26.770	166.437.418	
voter△	30.542 x 9.935	39.476	43.536.463	1.471.132	19.542	303.434.770	45,1%
Otimização △vs♣		**0,73**	**0,85**	**1,08**	**0,88**	**0,63**	
Desvio Padrão △		**13.221,80**	**38.344.251,82**	**3.005.439,70**	**10.340,19**	**108.210.744,06**	
Desvio Padrão ♣		**9.714,98**	**31.975.710,17**	**3.139.822,97**	**9.881,99**	**69.421.866,12**	

△-Ortho ♣-Migortho ⋆-Melhor

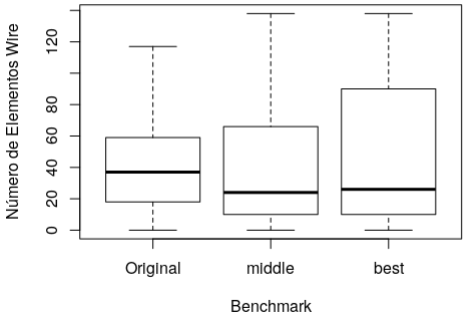
Além disso, no caso dos histogramas apresentados, é possível observar que em alguns casos tanto a configuração *best*, quanto a configuração *middle* apresentaram maior número de elementos na contagem final dos circuitos em alguns casos, o que eleva a mediana da avaliação. Porém, em especial no caso da configuração *best*, fica evidenciado o decréscimo da taxa de otimização, principalmente, porque nesses poucos casos o aumento na contagem do número de elementos adicionados é expressivo em relação a versão original (em torno de 1000 no caso dos *benchmarks* EPFL e IS-CAS'85). Assim como nos gráficos anteriores os *outliers* foram retirados da *plotagem* (através da função $outline = false$ do ambiente da linguagem *R*) pois seus valores não influenciam significativamente no comportamento gráfico final e para facilitar o sistema de visualização. Além disso, a exclusão dos *outliers* possibilitam uma melhor visualização do gráfico em função da escala e sua amplitude interquatílica.

5.3.1 Considerações Finais

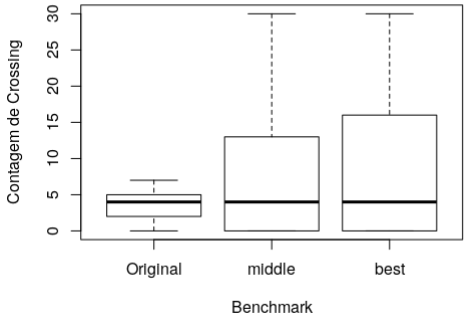
Esta seção apresentou os resultados obtidos nos experimentos de comparação do método proposto denominado *Migortho*, em relação a versão original do método *Ortho*. Foram avaliados o conjunto de 4 *benchmarks* onde os resultados corroboraram para exaltar as contribuições deste trabalho, visto que todos os experimentos apresentaram ganhos com a solução proposta. Neste sentido, diversos experimentos foram efetuados com o método *Migortho*, onde foi possível atingir taxas de otimização entre 33% até 42% referentes as área de leiaute para o conjunto de circuitos com a configuração *middle*. Por outro lado, no caso da configuração *best* os resultados chegaram entre 26% até 33% de ganho em relação a versão do método *Ortho*. Estes experimentos comprovaram a efetividade das modificações propostas neste trabalho, porém, é interessante notar que diferente do que era esperado, a maior otimização dos circuitos de entrada (*best*), não se refletiu nas maiores otimizações dos leiautes finais QCA. Isso pode ser explicado pelo aumento da complexidade do roteamento, o que demandou uma utilização maior de elementos do tipo fio (*wire*) impactando diretamente nas dimensões do leiaute.



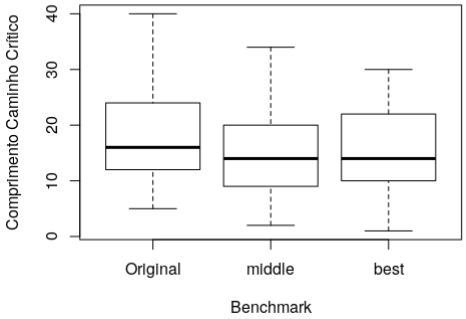
(a) Portas



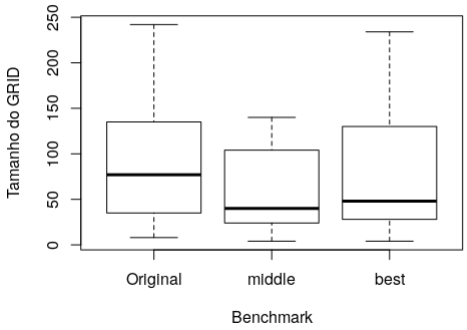
(b) Wire



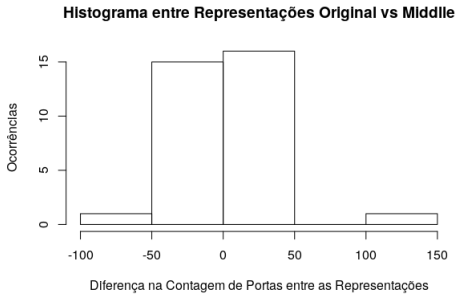
(c) Crossing



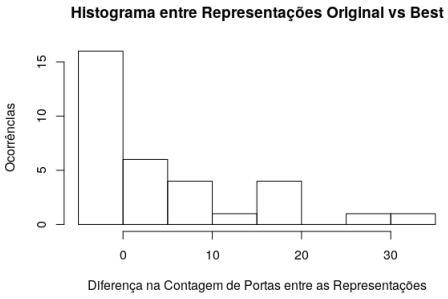
(d) Caminho Crítico



(e) Subgrids

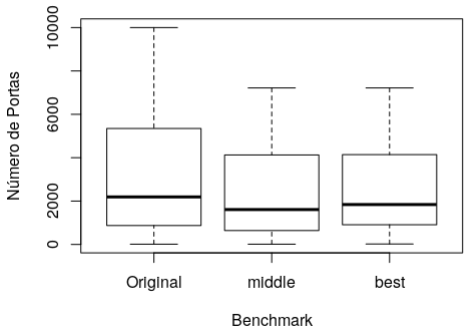


(f)

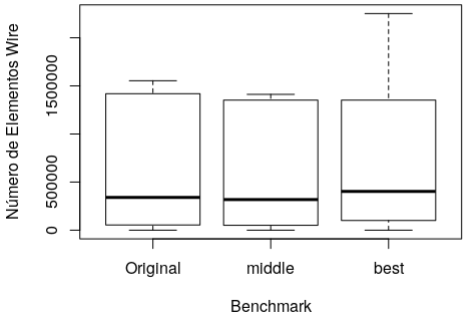


(g)

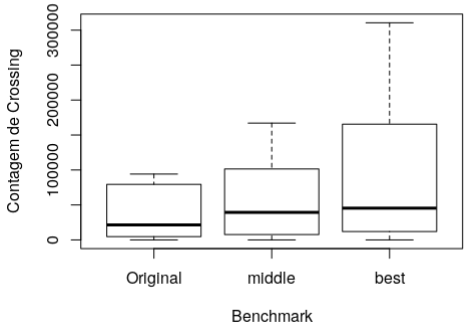
Figura 65 – Análise gráfica *benchmark TOY e UFV* para Fiction 2.1-Ortho.



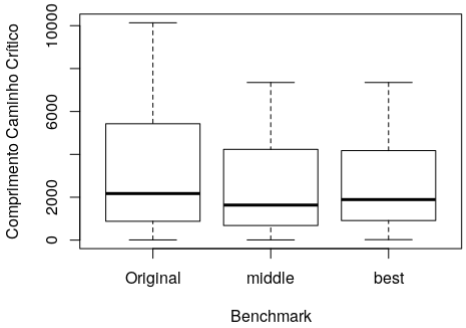
(a) Portas



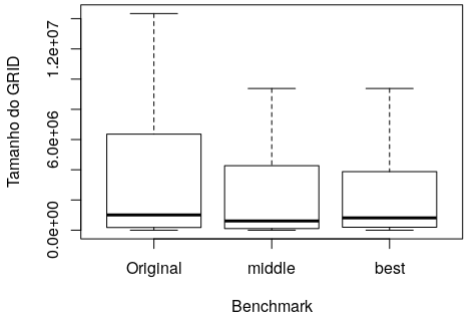
(b) Wire



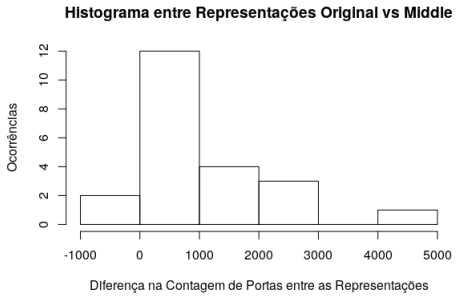
(c) Crossing



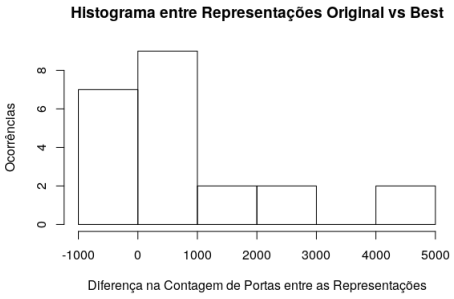
(d) Caminho Crítico



(e) Subgrids



(f)



(g)

Figura 66 – Análise gráfica *benchmark ISCAS e EPFL* para Fiction 2.1-Ortho.

5.4 Análise Fiction 2.1 - Migortho + Mockturtle vs Fiction - Exact

Por último, foi efetuada uma análise de um subconjunto de circuitos publicados por Walter em (Walter et al., 2018), com o objetivo de realizar uma comparação do novo método *Migortho*, em relação aos resultados do método *Exact*.

Desta forma, a Tabela 15 apresenta os resutados da síntese tanto pelo método *Exact* em relação aos resultados do método *Ortho* original e o método *Migortho* proposto por este trabalho (estrutura de dados, elementos de *design* e comandos *mockturtle*). Cada linha da tabela apresenta os três resultados de cada circuito, o identificador ♣ representa os dados da solução proposta neste trabalho (*Migortho*), assim como o identificador △ representa os dados da solução *Ortho* original, por último, o identificador ○ representa os dados da solução *Exact*. É importante salientar que o subconjunto escolhido foi definido pelos próprios autores em (WALTER et al., 2019).

Tabela 15 – Comparação do método *Ortho* vs *Migortho* vs *Exact*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
c17♣	8 x 4	11	12	0	10	32	-33,3 %
c17△	8 x 4	11	12	0	10	32	-33,3 %
c17○	4 x 6	15	-	-	16	24	
HA♣	6 x 6	10	24	5	10	36	-44,0 %
HA△	7 x 5	10	19	2	11	35	-40,0 %
HA○	5 x 5	10	-	-	10	25	
mux21♣	4 x 2	5	3	0	5	8	11,1 %
mux21△	4 x 2	5	3	0	5	8	11,1 %
mux21○	3 x 3	5	3	-	5	9	
mux41♣	13 x 8	18	66	13	20	104	-112,2 %
mux41△	12 x 6	16	26	5	17	72	-46,9 %
mux41○	7 x 7	18	-	-	22	49	
par-check♣	12 x 9	17	72	21	20	108	-125,0 %
par-check△	15 x 9	21	48	7	23	135	-181,3 %
par-check○	4 x 12	21	-	-	16	48	
par-gen♣	9 x 7	13	41	13	15	63	-50,0 %
par-gen△	10 x 7	14	35	2	16	70	-66,7 %
par-gen○	7 x 6	14	-	-	14	42	
xnor2♣	6 x 4	8	12	3	9	24	-50,0 %
xnor2△	6 x 4	8	10	2	9	24	-50,0 %
xnor2○	4 x 4	8	10	-	8	16	
xor2♣	5 x 3	6	8	0	7	15	-66,7 %
xor2△	5 x 3	6	8	0	7	15	-66,7 %
xor2○	3 x 3	6	8	-	5	9	

△-Ortho ♣-Migortho ○-Exact

De forma geral, como esperado a heurística baseada no método *Exact* apresenta melhores resultados em relação aos resultados do método *Migortho*, em torno de 43% menos área para o mesmo conjunto de circuitos. Contudo, é importante observar que em alguns casos a heurística do método *Migortho* é capaz de produzir resultados tão otimizados quanto o método *Exact* (caso do circuito *mux21*. Dessa forma, o método *Migortho* se torna uma ótima alternativa, visto que é computacionalmente menos custoso, além de ser aplicável a grandes circuitos, o que não acontece com o método *Exact*. Infelizmente, não foi possível realizar uma comparação sobre os circuitos

maiores, onde a proposta desse trabalho é de fato mais efetiva e onde os resultados poderiam se aproximar mais dos possíveis resultados do método *Exact*.

5.5 Análise Fiction 2.1 - Migortho + Mockturtle vs MIGthy

Por fim, com o objetivo de realizar uma comparação do impacto deste trabalho em relação aos resultados atingidos por Amarú na ferramenta MIGthy (AMARÚ, 2020), foram efetuados experimentos gerando estimativas de leiautes QCA com o *benchmark* MIG e MIGoriginal disponibilizados por Amarú. Assim, foi aplicado o método *Migortho* aos dois *benchmarks*, comparando com os resultados após a aplicação dos comandos que são necessários para gerar as configurações *middle* e *best*, em outras palavras, com o resultado do comando *4XmigCutRewriteK6* (*middle*) e o comando *4XmigCutRewriteK8* no caso do *best*, conforme a Tabela 6. É importante salientar que até o momento não foram propostas estimativas de leiaute para esses circuitos.

As Tabelas 16 e 17 apresentam os resultados sobre as seguintes características: em termos da área do *grid* (A), ou seja, estimativa de colunas e linhas de *subgrids* 5x5 utilizados, também número de portas (G), número de elementos de fio (W), número de cruzamento entre fios (CROSS), o caminho crítico do circuito (C.P.) e o número total de *subgrids* utilizados (Total Subgrid). Cada linha da tabela apresenta os três resultados de cada circuito, o identificador ♣ representa os dados da solução proposta neste trabalho (*Migortho*) pela configuração *middle* com a adição dos comandos *Mockturtle*, assim como o identificador $\circ$ representa os dados da solução gerada pela configuração *best* com o método *Migortho*, por fim o identificador $\triangle$ apresenta o resultado da síntese a partir do *benchmark* original proposto por Amarú utilizando o método *Migortho*.

Da mesma forma que nos experimentos com o *benchmark* EPFL, foi aplicada uma restrição de tamanho na seleção dos circuitos a serem executados, visto que foi identificado limitações como um alto tempo computacional e alto consumo de memória para execuções com circuitos acima de 40 mil elementos, mesmo sendo necessário a execução deste *benchmark* em um ambiente diferenciado, com maior poder de processamento e memória, com um processador AMD Opteron 2.3GHz, 64 núcleos e 128GB de memória RAM rodando um linux server. A Tabela 16 apresenta os resultados referentes a síntese considerando o *benchmark* MIG para as três configurações de entrada, são elas: *middle*, *best*, Amarú.

Tabela 16 – Comparação do método *Migortho benchmark MIG - Amarú vs middle vs best*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
vMIG-des-area-aig○	6.947 x 3.272	10.145	8.167.680	736.709	10.210	22.730.584	41,7 %
vMIG-des-area-aig♣	6.947 x 3.272	10.145	8.167.680	736.709	10.210	22.730.584	41,7 %
vMIG-des-area-aig△	9.531 x 4.088	13.545	11.495.351	1.436.547	13.610	38.962.728	
vMIG-div16-aig○	7.284 x 4.143	11.395	5.889.109	379.827	11.426	30.177.612	42,9 %
vMIG-div16-aig♣	7.284 x 4.143	11.395	5.889.109	379.827	11.426	30.177.612	42,9 %
vMIG-div16-aig△	10.166 x 5.195	15.328	8.912.709	574.248	15.360	52.812.370	
vMIG-hamming-aig○	3.158 x 1.789	4.747	1.018.437	73.329	4.944	5.649.662	46,1 %
vMIG-hamming-aig♣	3.158 x 1.791	4.749	1.019.136	73.390	4.946	5.655.978	46,0 %
vMIG-hamming-aig△	4.957 x 2.113	6.869	1.499.405	83.472	7.066	10.474.141	
vMIG-i2c-aig○	1.701 x 969	2.542	733.509	92.802	2.607	1.648.269	26,0 %
vMIG-i2c-aig♣	1.701 x 969	2.542	733.509	92.802	2.607	1.648.269	26,0 %
vMIG-i2c-aig△	2.214 x 1.006	3.089	946.485	104.129	3.147	2.227.284	
vMIG-max-aig○	5.734 x 3.392	8.614	7.159.172	822.326	9.121	19.449.728	58,7 %
vMIG-max-aig♣	5.736 x 3.393	8.617	7.162.426	822.322	9.124	19.462.248	58,7 %
vMIG-max-aig△	9.706 x 4.851	14.045	12.548.996	1.219.765	14.555	47.083.806	
vMIG-pci-spoci-ctrl-aig○	1.522 x 919	2.371	730.476	87.104	2.412	1.398.718	34,6 %
vMIG-pci-spoci-ctrl-aig♣	1.522 x 919	2.371	730.476	87.104	2.412	1.398.718	34,6 %
vMIG-pci-spoci-ctrl-aig△	2.060 x 1.039	3.027	978.201	106.032	3.094	2.140.340	
vMIG-sasc-aig○	1.054 x 643	1.571	360.859	41.458	1.551	677.722	27,1 %
vMIG-sasc-aig♣	1.054 x 643	1.571	360.859	41.458	1.551	677.722	27,1 %
vMIG-sasc-aig△	1.461 x 636	1.970	423.657	36.484	1.943	929.196	
vMIG-simple-spi-aig○	1.361 x 832	2.057	585.639	72.226	2.190	1.132.352	29,5 %
vMIG-simple-spi-aig♣	1.361 x 832	2.057	585.639	72.226	2.190	1.132.352	29,5 %
vMIG-simple-spi-aig△	1.925 x 834	2.623	720.254	69.584	2.754	1.605.450	
vMIG-spi-aig○	5.335 x 3.056	8.158	7.581.168	720.857	8.384	16.303.760	31,2 %
vMIG-spi-aig♣	5.303 x 3.064	8.134	7.518.983	711.200	8.360	16.248.392	31,4 %
vMIG-spi-aig△	7.342 x 3.228	10.337	9.950.028	736.868	10.561	23.699.976	
vMIG-sqrt32-aig○	3.140 x 1.804	4.913	1.305.869	92.213	4.943	5.664.560	58,0 %
vMIG-sqrt32-aig♣	3.140 x 1.804	4.913	1.305.869	92.213	4.943	5.664.560	58,0 %
vMIG-sqrt32-aig△	5.266 x 2.560	7.794	2.307.477	133.731	7.825	13.480.960	
vMIG-ss-pcm-aig○	750 x 471	1.120	205.352	24.456	1.063	353.250	21,6 %
vMIG-ss-pcm-aig♣	750 x 471	1.120	205.352	24.456	1.063	353.250	21,6 %
vMIG-ss-pcm-aig△	935 x 482	1.315	243.087	28.482	1.267	450.670	
vMIG-systemcdes-aig○	4.387 x 2.313	6.541	2.979.738	268.997	6.691	10.147.131	32,8 %
vMIG-systemcdes-aig♣	4.387 x 2.313	6.541	2.979.738	268.997	6.691	10.147.131	32,8 %
vMIG-systemcdes-aig△	5.749 x 2.628	8.218	3.734.034	415.107	8.328	15.108.372	
vMIG-usb-phy-aig○	648 x 380	950	137.128	19.183	990	246.240	29,2 %
vMIG-usb-phy-aig♣	648 x 380	950	137.128	19.183	990	246.240	29,2 %
vMIG-usb-phy-aig△	857 x 406	1.178	178.257	23.116	1.245	347.942	
Otimização △vs♣		**0,73**	**0,68**	**0,69**	**0,85**	**0,88**	
Otimização △vs○		**0,73**	**0,68**	**0,69**	**0,85**	**0,88**	
Desvio Padrão △		**5.136,74**	**4.729.560,56**	**478.956,41**	**4.056,77**	**149.738.020,80**	
Desvio Padrão ♣		**3.623,37**	**3.139.897,83**	**299.813,95**	**2.899,18**	**172.981.736,60**	
Desvio Padrão ○		**3.624,86**	**3.147.334,45**	**301.029,75**	**2.901,90**	**172.981.067,47**	

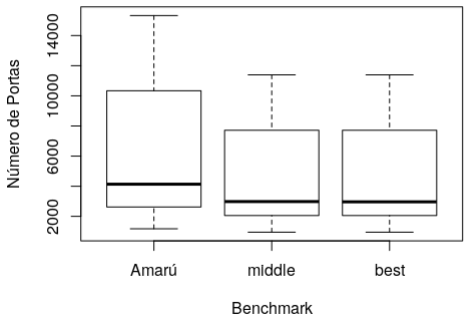
△-Amarú ♣-Middle ○-Best

Em um segundo momento, foram efetuadas as avaliações em relação ao *benchmark* MIGoriginal, os resultados são apresentados na Tabela 17. Por fim, similarmente como os experimentos anteriores, a Figura 67 apresenta as análises gráficas dos experimentos através de gráficos do tipo caixa (*boxplot*). Assim, a Figura 67 apresenta as informações referentes aos *benchmarks* MIG e MIGoriginal.

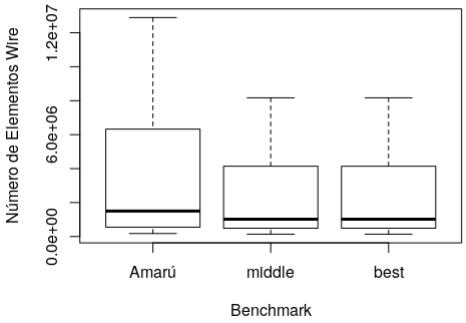
Tabela 17 – Comparação do método *Migortho benchmark MIGoriginal - Amarú vs middle vs best*

Circuito	A	G	W	CROSS	C.P.	Total Subgrid	Otimização
orig-des-area○	7.032 x 3.612	10.520	8.033.846	687.453	10.581	25.399.584	46,5 %
orig-des-area♣	7.032 x 3.612	10.520	8.033.846	687.453	10.581	25.399.584	46,5 %
orig-des-area△	10.946 x 4.337	15.152	12.895.253	1.114.877	15.276	47.472.802	
orig-hamming○	5.188 x 2.730	7.717	1.930.459	123.507	7.915	14.163.240	49,6 %
orig-hamming♣	5.188 x 2.730	7.717	1.930.459	123.507	7.915	14.163.240	49,6 %
orig-hamming△	7.940 x 3.539	11.279	2.925.850	175.203	11.475	28.099.660	
orig-i2c○	1.878 x 1.100	2.850	861.034	105.459	2.964	2.065.800	29,4 %
orig-i2c♣	1.878 x 1.100	2.850	861.034	105.459	2.964	2.065.800	29,4 %
orig-i2c△	2.590 x 1.129	3.590	1.137.521	124.691	3.669	2.924.110	
orig-max○	4.122 x 2.275	5.885	4.140.638	527.322	6.393	9.377.550	46,1 %
orig-max♣	4.122 x 2.275	5.885	4.140.638	527.322	6.393	9.377.550	46,1 %
orig-max△	6.379 x 2.726	8.593	6.326.257	655.529	9.100	17.389.154	
orig-pci-spoci-ctrl○	1.863 x 1.163	2.962	1.238.590	152.992	3.009	2.166.669	40,6 %
orig-pci-spoci-ctrl♣	1.872 x 1.174	2.982	1.258.131	156.520	3.028	2.197.728	39,8 %
orig-pci-spoci-ctrl△	2.969 x 1.229	4.134	1.790.795	183.778	4.170	3.648.901	
orig-sasc○	1.229 x 750	1.853	486.983	64.182	1.830	921.750	24,1 %
orig-sasc♣	1.229 x 750	1.853	486.983	64.182	1.830	921.750	24,1 %
orig-sasc△	1.621 x 749	2.243	550.743	53.593	2.209	1.214.129	
orig-simple-spi○	1.625 x 1.022	2.511	817.515	108.986	2.639	1.660.750	22,9 %
orig-simple-spi♣	1.625 x 1.022	2.511	817.515	108.986	2.639	1.660.750	22,9 %
orig-simple-spi△	2.188 x 985	3.037	903.226	94.433	3.166	2.155.180	
orig-spi○	5.927 x 3.603	9.322	9.879.090	995.335	9.520	21.354.981	26,7 %
orig-spi♣	5.927 x 3.603	9.322	9.879.090	995.335	9.520	21.354.981	26,7 %
orig-spi△	8.380 x 3.475	11.647	11.361.283	861.561	11.841	29.120.500	
orig-sqrt32○	1.897 x 959	2.825	357.634	22.413	2.855	1.819.223	21,6 %
orig-sqrt32♣	1.897 x 959	2.825	357.634	22.413	2.855	1.819.223	21,6 %
orig-sqrt32△	2.631 x 882	3.481	428.681	24.140	3.512	2.320.542	
orig-ss-pcm○	740 x 494	1.135	216.410	25.970	1.080	365.560	11,7 %
orig-ss-pcm♣	740 x 494	1.135	216.410	25.970	1.080	365.560	11,7 %
orig-ss-pcm△	910 x 455	1.265	234.049	22.385	1.209	414.050	
orig-systemcdes○	4.783 x 2.515	7.139	3.090.353	299.703	7.289	12.029.245	35,8 %
orig-systemcdes♣	4.783 x 2.515	7.139	3.090.353	299.703	7.289	12.029.245	35,8 %
orig-systemcdes△	6.770 x 2.766	9.402	3.419.526	372.758	9.516	18.725.820	
orig-usb-phy○	727 x 463	1.109	175.146	24.794	1.186	336.601	26,5 %
orig-usb-phy♣	727 x 463	1.109	175.146	24.794	1.186	336.601	26,5 %
orig-usb-phy△	987 x 464	1.367	216.877	23.572	1.445	457.968	
Otimização △vs♣		**0,74**	**0,74**	**0,85**	**0,75**	**0,60**	
Otimização △vs○		**0,74**	**0,74**	**0,85**	**0,75**	**0,60**	
Desvio Padrão △		**4.703,77**	**4.402.620,77**	**369.581,45**	**4.780,95**	**15.383.534,19**	
Desvio Padrão ♣		**3.303,67**	**3.228.147,18**	**312.954,07**	**3.368,44**	**8.790.778,38**	
Desvio Padrão ○		**3.304,59**	**3.228.892,62**	**313.063,61**	**3.369,33**	**8.792.531,13**	

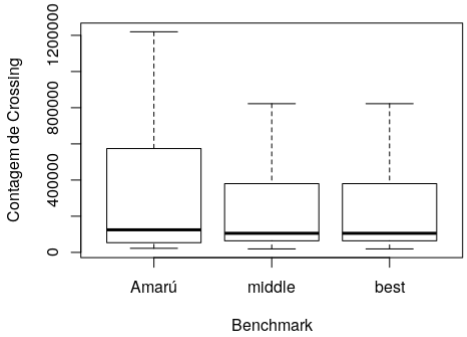
△-Amarú ♣-Middle ○-Best



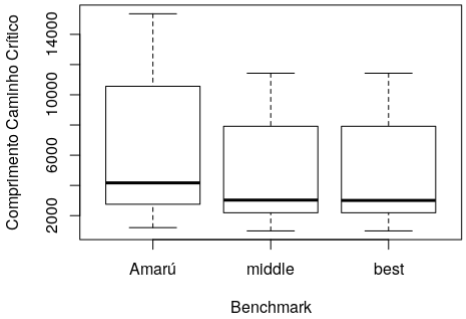
(a) Portas



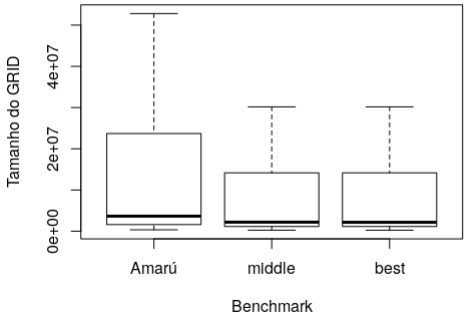
(b) Wire



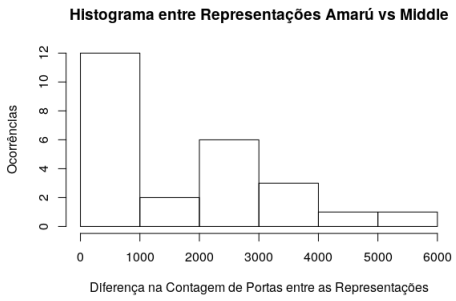
(c) Crossing



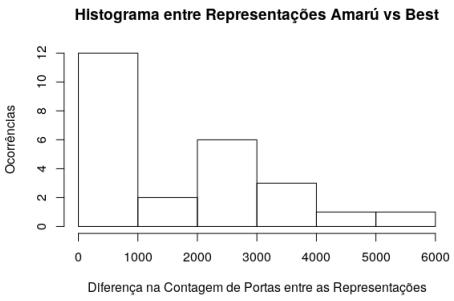
(d) Caminho Crítico



(e) Subgrids



(f)



(g)

Figura 67 – Análise gráfica *benchmarks MIG e MIGoriginal* para Fiction-Ortho vs Fiction 2.1-Ortho-Middle vs Fiction 2.1-Ortho-Best.

Assim como nos gráficos dos experimentos anteriores, é possível verificar através do gráfico na Figura 67(a) que a utilização das configurações propostas na representação dos circuitos, acarretou em uma otimização em relação ao número de portas, sendo verificada pela posição da mediana no gráfico *boxplot* (maior concentração dos valores). É possível verificar que embora a configuração *middle* seja a solução intermediária, ela foi capaz de obter a mesma taxa de otimização que a configuração *best*.

Da mesma forma, esse comportamento pode ser verificado para os outros gráficos, como por exemplo, elementos do tipo fio (*Wire*) (Figura 67(b)), caminho crítico (Figura 67(d)) e total de *subgrids* 5x5 utilizados (Figura 67(e)). É importante salientar que esse último gráfico está diretamente relacionado com a área final do leiaute QCA.

Contudo, também outra informação relevante é que no caso do fator de cruzamento de fios (CROSS), não houve um aumento considerável em relação as soluções originais, diferentemente dos experimentos dos *benchmarks* anteriores. Isso pode ser explicado pela taxa de otimização da entrada a qual foi entre 26% e 27%, diminuindo consideravelmente os roteamentos do circuitos. Além disso, também é importante ressaltar as taxas de otimização do leiaute final em relação ao leiaute da versão original dos *benchmarks*, onde as otimizações foram entre 12% e 40%. Adicionalmente, no caso dos histogramas apresentados, é possível observar que em todos os casos tanto pára a configuração *best*, quanto a configuração *middle* apresentaram menor número de elementos na contagem final dos circuitos.

Esta seção apresentou uma comparação dos resultados do *benchmark* MIG e MIGoriginal (AMARÚ, 2020), em relação ao leiaute QCA utilizando o método proposto neste trabalho denominado *Migortho*. Foi apresentada a estimativa de leiaute QCA para este conjunto de circuitos, o que até então é inédito. Além disso, foi efetuada uma comparação das estimativas de leiautes QCA finais tanto para a solução original de Amarú, quanto para a solução proposta por esse trabalho com a utilização dos comandos da configuração *middle* e *best*. Ao final, foi possível verificar que além de realizar a síntese para o leiaute QCA, também foi possível mensurar o impacto de otimização da proposta deste trabalho em relação a versão original.

5.6 Conclusões

Este capítulo apresentou os experimentos e as análises efetuadas sobre os resultados obtidos. Foram propostos cinco experimentos distintos sobre diferentes aspectos. Inicialmente, foram efetuados experimentos iniciais para prototipação da metodologia de síntese QCA proposta, onde foi possível verificar a efetividade da proposta através do método *Migortho* proposto nesse trabalho, que utiliza a estrutura de dados MIG, em relação a estrutura de dados utiliza originalmente pelo método *Ortho* que é a estrutura AOI. Além disso, também foi possível verificar o impacto da utilização dos novos elementos de *design* propostos, sendo eles: *MajX*, *Double Wire*, *Double Bent Wire*. Essa avaliação permitiu confirmar a efetividade da nova metodologia, além de corroborar com a literatura na questão da utilização da estrutura MIG para síntese QCA. Por fim, esses experimentos permitiram identificar diversos pontos que poderiam ser explorados.

O segundo experimento teve como objetivo avaliar a adição da biblioteca *Mockturtle*, através dos comandos implementados para otimização MIG. Sendo assim, foram efetuados diversos experimentos com 7 diferentes *benchmarks* explorando diferentes configurações de execuções dos comandos implementados através da biblioteca *Mockturtle*. Foram efetuados 68 execuções dos 144 circuitos, os quais resultaram em 9.792 circuitos. Foram avaliadas as diferentes configurações como os valores para o K do algoritmo de cortes-K, o número de execuções em série, além de identificar qual comando seria o mais efetivo para síntese QCA. Esse experimento avaliou os resultados e identificou qual a melhor configuração de comandos para o circuito de entrada, sendo capaz de otimizar a contagem de elementos em $15,8\%$ em média. Por outro lado, também foi identificado qual comando obteve a maior taxa de otimização por *benchmark*. Sendo assim, ao fim dois novos conjuntos de circuitos de entradas (novas versões dos circuitos dos *benchmarks* selecionados) foram propostos. O primeiro denominado *middle*, é resultado da execução do método que obteve a melhor taxa de otimização em média para todos os *benchmarks*. O segundo, denominado *best*, foi proposto a partir da execução de cada comando específico para cada *benchmark* avaliado.

O terceiro experimento, considerou os resultados identificados do experimento anterior, ou seja, após otimizar o circuito de entrada, qual o impacto dessa entrada no leiaute QCA final através do método *Migortho*. Sendo assim, foram elencados as duas configurações de comandos que produziram as versões dos *benchmarks* para serem avaliados. Entretanto, apenas 4 *benchmarks* dos 7 avaliados puderam ser executados por não possuírem descrições com portas majoritárias, visto que a versão do método *Ortho* original não é capaz de lidar com esse tipo de estrutura. Ao fim, os resultados indicaram como esperado, houve uma redução no leiaute final entre 33% e 42% no

caso da configuração *middle*, e entre 26% e 33% no caso da configuração *best*. Estes experimentos comprovaram a efetividade das modificações propostas neste trabalho, porém, é interessante notar que diferentemente do esperado, a configuração *middle* que obtem um grau de otimização intermediário dos circuitos de entrada obteve os melhores resultados no leiaute em relação a configuração *best* que é capaz de atingir o maior grau de otimização do circuito de entrada. Isso pode ser explicado pelo fato que uma maior otimização no circuito de entrada, pode aumentar consideravelmente a complexidade do circuito, o que pode afetar diretamente o roteamento do mesmo, influenciando na contagem de *subgrids*, visto que os fios na tecnologia QCA também são implementados com células QCA.

O quarto experimento foi efetuado com o objetivo de traçar apenas um paralelo em relação a outra abordagem presente na ferramenta Fiction, denominada método *Exact*. Esse método possui ótimos resultados para a síntese de leiautes QCA. Porém, possui apresenta limitações no *runtime* conforme o tamanho da entrada aumenta. Foram efetuadas algumas comparações com o conjunto de circuitos publicados por Walter em (Walter et al., 2018), onde os resultados demonstraram que a heurística baseada no método *Exact* apresenta melhores resultados em relação aos resultados do método *Migortho*, em torno de 43% menos área para o mesmo conjunto de circuitos. Contudo, é importante observar que em alguns casos a heurística do método *Migortho* foi capaz de produzir resultados tão otimizados quanto o método *Exact*. Dessa forma, o método *Migortho* se torna uma ótima alternativa, visto que é computacionalmente menos custoso, além de ser aplicável a grandes circuitos, o que não acontece com o método *Exact*.

Por fim, o último experimento teve como objetivo realizar um análise da aplicação da versão do método *Migortho* proposto nesse trabalho no *benchmark* de circuitos MIG proposto por Amarú (AMARÚ, 2020). Através dos comandos implementados da biblioteca *Mockturtle*, foi possível otimizar os resultados disponibilizados por Amarú em seu *benchmark* MIG e MIGoriginal em 26% e 27%, respectivamente, além de apresentar a estimativa de leiaute QCA para este conjunto de circuitos, o que até então não foi proposto na literatura. Além disso, foi efetuada uma comparação dos leiautes QCA finais tanto para a solução original de Amarú, quanto para a solução proposta por esse trabalho com a utilização das configurações *middle* e *best*. Ao final, foi possível verificar que além de realizar a síntese para o leiaute QCA, também foi possível mensurar o impacto de otimização da proposta deste trabalho em relação a versão original, que atingiu otimizações na contagem de subgrids de 12% e 40% para o *benchmarks* MIG e MIGoriginal, respectivamente, tanto para a versão *middle*, quanto para a versão *best*.

Em resumo, foi possível verificar a viabilidade da metodologia de síntese QCA proposta, além de avaliar o impacto dos novos elementos propostos nesse trabalho. Ademais também foi possível avaliar a nova estrutura de dados para representação da lógica, denominda MIG. Da mesma forma, foi possível identificar qual comando implementado pode impactar na maior redução dos circuitos de entrada, sendo o comando *migCutRewrite*, com uma execução em série de quatro vezes, com o valor de K fixado em seis, o qual pode atingir a taxa média de otimização de $15,8\%$. Além disso, vários circuitos foram avaliados através da síntese pela novo método de síntese proposto denominado *Migortho*, que apresentou ganhos na versão proposta neste trabalho em relação ao resultado da versão do método *Ortho*. Duas versões otimizadas dos circuitos dos *benchmarks* utilizados foram propostas para futuras avaliações de síntese QCA. Por fim, foram efetuadas avaliações com o *benchmark* específico estado-da-arte para a estrutura MIG, o qual foi possível obter tanto a estimativa do leiaute QCA, quanto aplicar otimizações nas descrições de entrada, melhorando ainda mais os circuitos descritos por Amarú em (AMARÚ, 2020).

6 CONCLUSÕES

Este trabalho propôs uma metodologia de síntese automática de circuitos com o foco na tecnologia Quantum Cellular Automata (QCA), utilizando o ambiente da ferramenta Fiction (WALTER et al., 2019) para a prototipação da proposta. Inicialmente foi efetuada uma revisão bibliográfica a cerca dos principais trabalhos e conceitos que envolvem a síntese de circuitos, em especial, a síntese voltada para novas tecnologias, mais especificamente, o paradigma FCN onde este trabalho têm como foco a tecnologia Quantum Cellular Automata (QCA).

Além disso, foram elencados os principais trabalhos da literatura, que propõem desde estrutura de dados até ambientes para síntese. Dessa forma, foi selecionado o ambiente da ferramenta Fiction para ser o ambiente de protitação da metodologia proposta, onde para isso acontecer, modificações foram propostas. A ferramenta Fiction é um *framework* para tecnologias FCN implementado em C++, desenvolvido pela universidade de Bremen na Alemanha, o qual utiliza as bibliotecas do *framework EPFL logic libraries* (SOEKEN et al., 2019a). Esta ferramenta possui o enfoque na etapa de projeto físico de nanotecnologias emergentes, como por exemplo, a tecnologia QCA em suas diferentes formas (ou seja, atômica ou molecular) e dispositivos NML. A ferramenta provê duas abordagens distintas para síntese QCA, sendo um dos métodos denominado *Ortho*, o qual foi selecionado para servir como base para a proposta deste trabalho. Essa abordagem apresenta uma vantagem significativa de tempo de execução em comparação com a outra abordagem disponível, denominada *exact*. Embora os leiautes gerados por esse método não sejam considerados ótimos em termos de área, essa técnica é aplicável mesmo para circuitos maiores, e fornece resultados em tempo de execução razoável (Walter et al., 2019).

Após identificar diversas limitações no método *Ortho*, foi proposta uma nova versão do método denominada *Migortho*, onde foram propostos novos elementos de *design* baseados nas propostas da literatura. Além disso, foram propostas novas estruturas de dados para representação da lógica diretamente em majoritárias.

Adicionalmente, foi inserido um novo módulo da biblioteca *EPFL Logic Libraries* denominado *Mockturtle*. Com isso, foi possível implementar na ferramenta Fiction 2.1 uma série de comandos de otimização baseados em MIG, os quais podem viabilizar otimizações na etapa de síntese lógica na síntese para a tecnologia QCA. Uma série de *scripts* foram propostos para avaliar e identificar quais são os métodos mais efetivos.

Foram propostos 5 experimentos distintos sobre diferentes aspectos. Inicialmente, foram efetuados experimentos iniciais com a proposta de metodologia de síntese QCA, onde foi possível verificar a efetividade da proposta do método denominado *Migortho*, o qual utiliza a estrutura de dados MIG, em relação a estrutura de dados utiliza originalmente pelo método *Ortho* denominada AOI. Além disso, também foi possível verificar o impacto da utilização dos novos elementos de *design* propostos, sendo eles: *MajX*, *Double Wire*, *Double Bent Wire*. Essa avaliação permitiu confirmar a efetividade da nova metodologia, além de corroborar com a literatura na questão da utilização da estrutura MIG para síntese QCA. Por fim, esses experimentos permitiram identificar diversos pontos que poderiam ser explorados.

Foram avaliados os impactos dos novos elementos de *design* propostos nesse trabalho, assim como a utilização da estrutura de dados para representação da lógica, denominada MIG. Da mesma forma foi possível identificar qual comando implementado pode impactar na maior redução dos circuitos de entrada, sendo o comando *migCutRewrite*, com uma execução em série de quatro vezes, com o valor de K fixado em seis, o qual pode atingir a taxa média de otimização de $15,8\%$. Além disso, vários circuitos foram avaliados através da síntese pelo método proposto *Migortho*, que apresentou ganhos em relação ao resultado da versão do método *Ortho*. Duas versões otimizadas dos *benchmarks* utilizados foram propostos para futuras avaliações de síntese QCA. Por fim, foram efetuadas avaliações com o *benchmark* específico estado-da-arte para a estrutura MIG, foi possível obter tanto a estimativa de leiaute QCA final, quanto aplicar otimizações nas descrições de entrada, melhorando ainda mais os circuitos descritos por Amarú em (AMARÚ, 2020).

Portanto, com os dados e as análises efetuadas, é possível concluir que os objetivos específicos e gerais deste trabalho foram atingidos, o que viabiliza a resposta do problema de pesquisa elencado.

6.1 Trabalhos Futuros

Como trabalhos futuros, pretende-se adicionar outras possibilidades de sistemas de *clock*, o que permitiria uma liberdade maior no algoritmo de roteamento, o que pode viabilizar leiautes finais mais enxutos. Além disso, outra alternativa que pode gerar ganhos consideráveis conforme aponta a literatura, seria a utilização de *Threshold Logic Gates* para a síntese, o que poderia melhorar ainda mais as otimizações do circuito de entrada.

REFERÊNCIAS

AIGER, f. **The AIGER And-Inverter Graph (AIG) Format**. Disponível em: http://fmv.jku.at/aiger/. Disponível em: <http://fmv.jku.at/aiger>.

AKERS, S. Binary Decision Diagrams. **IEEE TRANSACTIONS ON COMPUTERS**, [S.l.], v.C-27 , Issue: 6, p.509–516, 1978.

AKERS, S. B. Synthesis of combinational logic using three-input majority gates. In: SWCT, 1962. **Anais...** [S.l.: s.n.], 1962.

AMARÚ, L. **A large set of logic benchmarks optimized via our Majority-Inverter Graph (MIG) synthesis tool (MIGhty).** Disponível em: <https://www.epfl.ch/labs/lsi/page-102566-en-html/mig/>.

Amarú, L. G. **New Data Structures and Algorithms for Logic Synthesis and Verification.** 2015. 165p. Tese (Doutorado em Ciência da Computação) — PROGRAMME DOCTORAL EN INFORMATIQUE, COMUNICATIONS ET INFORMATION. ÉCOLE POLYTECHINIQUE FÉDÉRALE DE LAUSANNE, LAUSANNE, SUÍÇA.

AMARÚ, L.; GAILLARDON, P.-E.; DE MICHELI, G. The EPFL Combinational Benchmark Suite. **Proceedings of the 24th International Workshop on Logic and Synthesis (IWLS)**, [S.l.], 2015.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. BDS-MAJ: A BDD-based logic synthesis tool exploiting majority logic decomposition. In: DESIGN AUTOMATION CONFERENCE (DAC), 2013 50TH ACM/EDAC/IEEE, 2013. **Anais...** [S.l.: s.n.], 2013. p.1–6.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Biconditional BDD: A novel canonical BDD for logic synthesis targeting XOR-rich circuits. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2013, 2013. **Anais...** [S.l.: s.n.], 2013. p.1014–1017.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Majority-Inverter Graph: A novel data-structure and algorithms for efficient logic optimization. In: ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), 2014., 2014. **Anais...** [S.l.: s.n.], 2014. p.1–6.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. An efficient manipulation package for Biconditional Binary Decision Diagrams. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2014., 2014. **Anais...** [S.l.: s.n.], 2014. p.1–6.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Biconditional Binary Decision Diagrams: A Novel Canonical Logic Representation Form. **IEEE Journal on Emerging and Selected Topics in Circuits and Systems**, [S.l.], v.4, n.4, p.487–500, Dec 2014.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Boolean logic optimization in Majority-Inverter Graphs. In: ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.1–6.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Majority-Inverter Graph: A New Paradigm for Logic Optimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.35, n.5, p.806–819, May 2016.

AMARÚ, L.; PETKOVSKA, A.; GAILLARDON, P. E.; MICHELI, G. D. Majority-Inverter Graph for FPGA Synthesis. In: WORKSHOP ON SYNTHESIS AND SYSTEM INTEGRATION OF MIXED INFORMATION TECHNOLOGIES (SASIMI 2015), 2015. **Anais...** [S.l.: s.n.], 2015.

ANDERSON, N. G.; BHANJA, S. (Ed.). **Field-Coupled Nanocomputing - Paradigms, Progress, and Perspectives**. [S.l.]: Springer, 2014. (Lecture Notes in Computer Science, v.8280).

BEIU, V.; QUINTANA, J. M.; AVEDILLO, M. J. VLSI implementations of threshold logic - a comprehensive survey. **IEEE Transactions on Neural Networks**, [S.l.], v.14, n.5, p.1217–1243, Sept 2003.

BERKELEY, U. C. **SIS: A System for Sequential Circuit Synthesis**. Disponível em: http://embedded.eecs.berkeley.edu/pubs/downloads/sis/. Disponível em: <http://embedded.eecs.berkeley.edu/pubs/downloads/sis/>.

BERKELEY, U. C. **ABC: A System for Sequential Synthesis**. Disponível em: http://www.eecs.berkeley.edu/ alanmi/abc/abc.html. Disponível em: <http://www.eecs.berkeley.edu/ alanmi/abc/abc.html2>.

BIEDL, T. C. Three Approaches to 3D-Orthogonal Box-Drawings. In: GRAPH DRAWING, 1998, Berlin, Heidelberg. **Anais...** Springer Berlin Heidelberg, 1998. p.30–43.

BRGLEZ, F.; FUJIWARA, H. A neutral netlist of 10 combinational benchmark circuits and a targeted translator in FORTRAN. In: SPRINGER BERLIN HEIDELBERG, 1985. **Anais...** [S.l.: s.n.], 1985.

CAMPOS, C. A. T.; MARCIANO, A. L.; NETO, O. P. V.; TORRES, F. S. USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.35, n.3, p.513–517, 2016.

CHAUDHARY, A. et al. Eliminating wire crossings for molecular quantum-dot cellular automata implementation. In: ICCAD-2005. IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2005., 2005. **Anais...** [S.l.: s.n.], 2005. p.565–571.

CONG, J.; C., W.; Y., D. Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution. **INT'L SYMP. ON FPGA**, [S.l.], 1999.

CAMPUS (Ed.). **Algoritmos - Teoria e Pratica**. [S.l.: s.n.], 2002.

DE MICHELI, G. **Synthesis and Optimization of Digital Circuits**. [S.l.]: McGraw-Hill Higher Education, 1994. (Electrical and Computer Engineering Series).

FONTES, G. et al. Placement and Routing by Overlapping and Merging QCA Gates. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–5.

FONTES JUNIOR, G. **EXPLORANDO O ESPAÇO DE SOLUÇÕES NO POSICIONAMENTO E ROTEAMENTO DE CÉLULAS QCA NO ESQUEMA DE CLOCK USE**. 2018. Dissertação (Mestrado em Ciência da Computação) — DEPARTAMENTO DE INFORMÁTICA. UNIVERSIDADE FEDERAL DE VIÇOSA, MINAS GERAIS, BRASIL.

Formigoni, R. E.; Vilela Neto, O. P.; Nacif, J. A. M. BANCS: Bidirectional Alternating Nanomagnetic Clocking Scheme. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–6.

Lent, C. S.; Tougaw, P. D. A device architecture for computing with quantum dots. **Proceedings of the IEEE**, [S.l.], v.85, n.4, p.541–557, April 1997.

MARWEDEL, P. **Embedded System Design**: Embedded Systems Foundations of Cyber-Physical Systems. [S.l.]: Springer, 2010.

Mishchenko, A.; Brayton, R. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In: INTERNATIONAL WORKSHOP ON LOGIC AND SYNTHESIS - IWLS, 2006. **Anais...** [S.l.: s.n.], 2006.

Mishchenko, A.; Chatterjee, S.; Brayton, R. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 2006., 2006. **Anais...** [S.l.: s.n.], 2006. p.532–535.

MOMENZADEH, M.; HUANG, J.; TAHOORI, M. B.; LOMBARDI, F. Characterization, Test, and Logic Synthesis of And-Or-Inverter (AOI)Gate Design for QCA Implementation. **IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS**, [S.l.], v.24, n.12, Dec. 2005.

MOURA, L. de; BJØRNER, N. Z3: An Efficient SMT Solver. In: TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEMS, 2008, Berlin, Heidelberg. **Anais...** Springer Berlin Heidelberg, 2008. p.337–340.

Neutzling, A. **Synthesis of Threshold Logic Based Circuits.** 2014. 71p. Dissertação (Mestrado em Ciência da Computação) — PÓS-GRADUAÇÃO EM COMPUTAÇÃO-PPGC. UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL- UFRGS, PORTO ALEGRE, BRASIL.

NEUTZLING, A. et al. Threshold logic synthesis based on cut pruning. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.494–499.

NEUTZLING, A. et al. A Simple and Effective Heuristic Method for Threshold Logic Identification. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.PP, n.99, p.1–1, 2017.

NEUTZLING, A.; MARTINS, M. G. A.; RIBAS, R. P.; REIS, A. I. Synthesis of threshold logic gates to nanoelectronics. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.1–6.

NIEMIER, M.; KOGGE, P. Problems in designing with QCAs: Layout = Timing. **INTERNATIONAL JOURNAL OF CIRCUIT THEORY AND APPLICATIONS Int. J. Circ. Theor. Appl**, [S.l.], v.29, p.49–62, 01 2001.

REIS, D. A. **Robustness Analysis and Enhancement Strategies for Quantum-dot Cellular Automata Structures**. 2016. Dissertação (Mestrado em Ciência da Computação) — DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO. UNIVERSIDADE FEDERAL DE MINAS GERAIS, MINAS GERAIS, BRASIL.

REIS, D. A. et al. A Methodology for Standard Cell Design for QCA. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.2114–2117.

SHANG, K. K. R. L. Y. An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata. **IEEE TRANSACTIONS ON NANOTECHNOLOGY**, [S.l.], v.9, n.2, Mar. 2010.

SILVA, P. A. R. L. et al. A Novel Five-input Multiple-function QCA Threshold Gate. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–5.

SOEKEN, M.; RIENER, H.; HAASWIJK, W.; DE MICHELI, G. **The EPFL logic synthesis libraries**. arXiv:1805.05121.

SOEKEN, M.; RIENER, H.; HAASWIJK, W.; DE MICHELI, G. **The EPFL logic synthesis libraries**. Disponível em: <https://github.com/lsils/lstools-showcase>.

SOEKEN, M.; RIENER, H.; HAASWIJK, W.; DE MICHELI, G. **The Mockturtle Documentation**. Disponível em: <https://mockturtle.readthedocs.io/en/latest/>.

SOEKEN, M.; RIENER, H.; HAASWIJK, W.; DE MICHELI, G. **CirKit is a logic synthesis and optimization framework**. Disponível em: <https://github.com/msoeken/cirkit>.

SYNOPSYS, L. L. F. **Liberty CSS**. Disponível em: http://www.synopsys.com/products/libertyccs/libertyccs.htm.

TEIXEIRA, P. **A FEASIBLE CLOCKING SCHEME (USE) AND A STANDARD CELLS LIBRARY (QCA ONE) FOR FUTURE QUANTUM-DOT CELLULAR AUTOMATA**. 2016. Dissertação (Mestrado em Ciência da Computação) — DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO. UNIVERSIDADE FEDERAL DE MINAS GERAIS, MINAS GERAIS, BRASIL.

TEODOSIO, T.; SOUSA, L. QCA-LG: A tool for the automatic layout generation of QCA combinational circuits. In: NORCHIP 2007, 2007. **Anais...** [S.l.: s.n.], 2007. p.1–5.

TRINDADE, A. et al. A Placement and routing algorithm for Quantum-dot Cellular Automata. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.1–6.

Vankamamidi, V.; Ottavi, M.; Lombardi, F. Clocking and Cell Placement for QCA. In: SIXTH IEEE CONFERENCE ON NANOTECHNOLOGY, 2006., 2006. **Anais...** [S.l.: s.n.], 2006. v.1, p.343–346.

VANKAMAMIDI, V.; OTTAVI, M.; LOMBARDI, F. Two-Dimensional Schemes for Clocking/Timing of QCA Circuits. **Trans. Comp.-Aided Des. Integ. Cir. Sys.**, Piscataway, NJ, USA, v.27, n.1, p.34–44, Jan. 2008.

Walter, M. et al. An exact method for design exploration of quantum-dot cellular automata. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.503–508.

Walter, M. et al. Scalable Design for Field-coupled Nanocomputing Circuits. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 24., 2019, New York, NY, USA. **Proceedings...** ACM, 2019. p.197–202. (ASPDAC 19).

WALTER, M. et al. **fiction: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits**.

WALUS, K.; DYSART, T.; JULLIEN, G.; BUDIMAN, R. QCADesigner: A rapid design and simulation tool for quantum-dot cellular automata. **IEEE TRANSACTIONS NANOTECHNOLOGY**, [S.l.], v.3, p.26–29, Mar. 2004.

WOLF, M. **Computers as Components**: Principles of Embedded Computing System Design. [S.l.]: Morgan Kaufmann, 2012.

YANG, C.; CIESIELSKI, M. BDS: a BDD-based logic optimization system. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.21, n.7, p.866–876, 2002.

ZHANG, R.; WALUS, K.; WANG, W.; GRAHAM, A. J. A Method of Majority Logic Reduction for Quantum Cellular Automata. **IEEE TRANSACTIONS ON NANOTECHNOLOGY**, [S.l.], v.3, n.4, Dec. 2004.

Apêndices

APÊNDICE A – Classe de comandos MIG implementados

Esta seção descreve o código fonte desenvolvido para a implementação dos comandos MIG no ambiente da ferramenta Fiction 2.1, através da biblioteca *Alice* e os métodos de otimização MIG da biblioteca *Mockturtle*.

Código A.1 – Código da Classe C++ que implementa os comandos MIG.

```
#include<iostream>
#include<mockturtle/algorithms/cut_enumeration.hpp>
#include<mockturtle/algorithms/mig_resub.hpp>
#include<mockturtle/algorithms/cleanup.hpp>
#include<mockturtle/algorithms/resubstitution.hpp>
#include<mockturtle/algorithms/akers_synthesis.hpp>
#include<mockturtle/networks/mig.hpp>
#include<alice/command.hpp>
#include"../include/mockturtle/algorithms/node_resynthesis/mig_npn.hpp"
#include"../include/mockturtle/algorithms/lut_mapping.hpp"
using namespace mockturtle;
using namespace std;
namespace alice{
//####################
class migCut_command : public command
{
public:
explicit migCut_command(const environment::ptr& env)
:command(env,"K-cuts Enumeration algorithm"){
opts.add_option("-k,Kcut Size",k,"K size for Kcuts")->required();
}
protected:
void execute(){
auto env_mig {env->store<mig_t>()};
if(env_mig.empty()) {
env->err()<<"\033[1;31m*\033[0mYou must load a mig to execute the K-cuts."<< std::endl;
return;
```

```
28 }
29 mig_t clone = store<mig_t>().current();
30 mig_t mig(clone);
31 cout<< "numero de nodos do mig: "<<mig.get()->size()<<"\n";
32 cut_enumeration_params ps;
33 ps.cut_size = k;
34 cout<<"o tamanho do k: "<<k<<"\n";
35 ps.cut_limit = ps.cut_size;
36 auto cuts = cut_enumeration<mig_network, true>(*mig, ps);
37 cout<<"kcuts ok!\n";
38 cout<<"total de kcuts do mig :"<<cuts.total_cuts()<<"\n";
39 for(int node=0; node < (int) mig.get()->num_gates();node++){
40 cout<<cuts.cuts(mig.get()->index_to_node(node));
41 }
42 }
43 private:
44 int k;
45 };
46 ALICE_ADD_COMMAND(migCut,"FlexMap-Mig");
47 //#####################
48 ALICE_COMMAND(migLut,"FlexMap-Mig", "Mig Lut Mapping algorithm"){
49 auto env_mig {env->store<mig_t>()};
50 if(env_mig.empty()) {
51 env->err() << "\033[1;31m * \033[0mYou must load a mig to execute
      the Mig Lut Mapping." << std::endl;
52 return;
53 }
54 mig_t clone = store<mig_t>().current();
55 mig_t mig(clone);
56 cout<< "numero de nodos do mig: "<<mig.get()->size()<<"\n";
57 mapping_view<mig_network, true> m_view{*mig};
58 cout<<"mapping view ok!\n";
59 cout<< "numero de nodos do mig: "<<mig.get()->size()<<"\n";
60 lut_mapping<mapping_view<mig_network, true>, true>(m_view);
61 cout<<"lut mapping ok!\n";
62 cout<< "numero de nodos do mig: "<<mig.get()->size()<<"\n";
63 }
64 //#####################
65 class migCutRewrite_command : public command{
66 public:
67 explicit migCutRewrite_command( const environment::ptr& env )
```

```
68 : command( env, "Mig Cut-Rewrinting" ){
69 opts.add_option( "-k, Kcut Size", k, "K size for Kcuts" )->required
      ();
70 }
71 protected:
72 void execute(){
73 auto env_mig {env->store<mig_t>()};
74 if(env_mig.empty()) {
75 env->err() << "\033[1;31m * \033[0mYou must load a mig to execute
      the Cut-Rewrinting." << std::endl;
76 return;
77 }
78 mig_t clone = store<mig_t>().current();
79 mig_t mig(clone);
80 cout<< "numero de nodos do mig: "<<mig.get()->size()<<"\n";
81 akers_resynthesis<mig_network> resyn;
82 cut_rewriting_params cutRewPs;
83 cout<<"o tamanho do k: "<<k<<"\n";
84 cutRewPs.cut_enumeration_ps.cut_size = k;
85 cut_rewriting<mig_network>(*mig,resyn,cutRewPs);
86 cout<<"cut rewriting ok"<<"\n";
87 cout<<"total de nodos do mig: "<< mig.get()->size()<<"\n";;
88 mig_network migClean = cleanup_dangling<mig_network>(*mig );
89 cout<<"clean dangling nodes ok"<<"\n";
90 //transformar o migclean em storage
91 cout<<"total de nodos do mig: "<< migClean.size()<<"\n";
92 mig_t migPtr = std::make_shared<mockturtle::mig_network>(migClean);
93 env->store<mig_t>().extend() = migPtr;
94 }
95 private:
96 int k;
97 };
98 ALICE_ADD_COMMAND(migCutRewrite,"FlexMap-Mig");
99 //####################
100 ALICE_COMMAND(migAlg,"FlexMap-Mig", "Mig Algebraic Rewrinting
      algorithm"){
101 auto env_mig {env->store<mig_t>()};
102 if(env_mig.empty()) {
103 env->err() << "\033[1;31m * \033[0mYou must load a mig to execute
      the Mig Algebraic Rewrinting." << std::endl;
104 return;
```

```
105 }
106 mig_t clone = store<mig_t>().current();
107 mig_t mig(clone);
108 cout<< "numero de nodos do mig: "<<mig.get()->size()<<"\n";
109 using view_mig = depth_view<fanout_view<mig_network, true>>;
110 view_mig depthViewMig{*mig};
111 mig_algebraic_depth_rewriting_params psMig;
112 //        psMig.strategy=psMig.aggressive;
113 //        psMig.strategy=psMig.selective;
114 //        psMig.strategy=psMig.dfs;
115 mig_algebraic_depth_rewriting<depth_view<fanout_view<mig_network,
       true>>>(depthViewMig,psMig);
116 cout<<"mig algebraic rewriting ok"<<"\n";
117 cout<<"total de nodos do mig: "<< mig.get()->size()<<"\n";;
118 mig_network migClean = cleanup_dangling<mig_network>(*mig );
119 cout<<"clean dangling nodes ok"<<"\n";
120 //transformar o migclean em storage
121 cout<<"total de nodos do mig: "<< migClean.size()<<"\n";
122 mig_t migPtr = std::make_shared<mig_network>(migClean);
123 env->store<mig_t>().extend() = migPtr;
124 }
125 //####################
126 ALICE_COMMAND(migResub,"FlexMap-Mig", "Mig Resubstitution algorithm
       "){
127 auto env_mig {env->store<mig_t>()};
128 if(env_mig.empty()) {
129 env->err() << "\033[1;31m * \033[0mYou must load a mig to execute
       the Mig Resubstitution." << std::endl;
130 return;
131 }
132 mig_t clone = store<mig_t>().current();
133 mig_t mig(clone);
134 cout<< "numero de nodos do mig: "<<mig.get()->size()<<"\n";
135 using view_mig = mockturtle::depth_view<mockturtle::fanout_view<
       mockturtle::mig_network>>;
136 fanout_view<mockturtle::mig_network> fanout_viewMig{*mig};
137 view_mig resub_view{fanout_viewMig};
138 resubstitution(resub_view);
139 cout<<"mig resubstitution ok"<<"\n";
140 cout<<"total de nodos do mig: "<< mig.get()->size()<<"\n";
141 mig_network migClean = cleanup_dangling<mig_network>(*mig );
```

```
142 cout<<"clean dangling nodes ok"<<"\n";
143 //transformar o migclean em storage
144 cout<<"total de nodos do mig: "<< migClean.size()<<"\n";
145 mig_t migPtr = std::make_shared<mig_network>(migClean);
146 env->store<mig_t>().extend() = migPtr;
147 }
148 //####################
149 class migNode_command : public command{
150 public:
151 explicit migNode_command( const environment::ptr& env )
152 : command( env, "Mig Node-Resynthesis" ){
153 opts.add_option( "-k, Kcut Size", k, "K size for Kcuts" )->required
       ();
154 }
155 protected:
156 void execute(){
157 auto env_mig {env->store<mig_t>()};
158 if(env_mig.empty()) {
159 env->err() << "\033[1;31m * \033[0mYou must load a mig to execute
       the Mig Node-Resynthesis." << std::endl;
160 return;
161 }
162 mig_t clone = store<mig_t>().current();
163 mig_t mig(clone);
164 cout<<"total de nodos do mig: "<< mig.get()->size()<<"\n";
165 cout<<"digite o tamanho do k a ser considerado:\n";
166 mapping_view<mig_network, true> m_view{*mig};
167 lut_mapping_params psLut;
168 psLut.cut_enumeration_ps.cut_size = k;
169 lut_mapping<mapping_view<mig_network, true>, true>( m_view, psLut )
       ;
170 const auto klut = mockturtle::collapse_mapped_network<mockturtle::
       klut_network>( m_view );
171 mig_npn_resynthesis resyn(k);
172 mig_network migClean = node_resynthesis<mig_network>( *klut, resyn
       );
173 cout<<"mig mig npn resynthesis ok"<<"\n";
174 cout<<"total de nodos do mig: "<< migClean.size()<<"\n";
175 mig_t migPtr = std::make_shared<mig_network>(migClean);
176 env->store<mig_t>().extend() = migPtr;
177 }
```

```
178 private:
179 int k;
180 };
181 ALICE_ADD_COMMAND(migNode,"FlexMap-Mig");
182 //####################
183 ALICE_COMMAND(migRefac,"FlexMap-Mig", "Mig Refatoring"){
184 auto env_mig {env->store<mig_t>()};
185 if(env_mig.empty()) {
186 env->err() << "\033[1;31m * \033[0mYou must load a mig to execute
       the Mig Resubstitution." << std::endl;
187 return;
188 }
189 mig_t clone = store<mig_t>().current();
190 mig_t mig(clone);
191 cout<<"total de nodos do mig: "<< mig.get()->size()<<"\n";
192 /* node resynthesis */
193 akers_resynthesis<mig_network> resyn;
194 refactoring<mig_network>(*mig,resyn);
195 cout<<"mig mig refactoring ok"<<"\n";
196 mig_network migClean = cleanup_dangling<mig_network>(*mig );
197 cout<<"clean dangling nodes ok"<<"\n";
198 cout<<"total de nodos do mig: "<< migClean.size()<<"\n";
199 mig_t migPtr = std::make_shared<mig_network>(migClean);
200 env->store<mig_t>().extend() = migPtr;
201 }
202 };
```

APÊNDICE B – Descrição dos comandos *MIX*

O mix de comandos criados para avaliações são apresentados abaixo. É importante salientar que o parâmetro X descrito abaixo refere-se ao valor do K, sendo utilizado em diferentes avaliações de *K=3* até *K=8*.

mix1: migCutRewrite -k X; migRefac; migResub; migResub

mix2: migRefac; migCutRewrite -k X; migResub; migResub

mix3: migRefac; migResub; migResub; migCutRewrite -k X

mix4: migResub; migResub; migRefac; migCutRewrite -k X

APÊNDICE C – Dados da Análise

A Figura 62 apresenta uma tabela referente aos resultados da análise da taxa de otimização média na execução dos comandos implementados com métodos *Mockturtle*, para a análise da efetividade dos scripts de pré-processamento. É possível verificar que o comando **4XmigCutRewriteK6** possui um grau de otimização semelhante a melhor configuração do comando **4XmigCutRewriteK8**, porém, com desvio padrão menor e complexidade do computo do valor de K inferior, o que afeta a complexidade da heurística, e por consequência o tempo de computação. A tabela apresentada possui as seguintes informações: o nome do circuito, a porcentagem máxima (em algum circuito) de otimização em elementos entre todos os *benchmarks*, a porcentagem máxima (em algum circuito) de otimização para níveis do circuito entre todos os *benchmarks*, a porcentagem média de otimização em elementos para todos os circuitos e seu desvio padrão, além da porcentagem média de otimização dos níveis dos circuitos e seu desvio padrão.

Método	%Gates -MAX	%Levels – MAX	% Média Gates	Desv. %GATES	% Média Levels	Desv. %LEVEL	ID
1migAIg-Gates	-3,74	39,40	-10,86	14,68	18,34	13,95	34
2migAIg-Gates	-3,74	40,80	-16,99	21,19	21,03	15,85	1
3migAIg-Gates	-3,74	42,22	-22,05	26,53	21,81	16,69	12
4migAIg-Gates	-3,74	43,07	-23,86	28,45	22,35	17,14	23
1migNodeK3-Gates	38,10	40,32	12,33	10,96	0,28	3,55	41
1migNodeK4-Gates	39,46	40,21	16,04	11,97	6,81	7,14	42
2migNodeK3-Gates	38,10	45,70	13,66	11,20	1,49	4,91	8
2migNodeK4-Gates	39,46	43,03	16,51	11,76	7,13	7,35	9
3migNodeK3-Gates	38,10	45,86	13,65	11,18	1,49	4,90	19
3migNodeK4-Gates	39,46	42,97	16,70	11,76	7,32	7,53	20
4migNodeK3-Gates	38,10	45,86	13,66	11,18	1,49	4,90	30
4migNodeK4-Gates	39,46	42,97	16,71	11,79	7,31	7,53	31
1migCutRewriteK3-Gates	29,25	43,28	10,59	8,10	2,85	2,30	35
1migCutRewriteK4-Gates	31,29	43,24	13,86	8,64	5,29	4,52	36
1migCutRewriteK5-Gates	31,29	43,17	14,28	9,23	5,55	4,26	37
1migCutRewriteK6-Gates	31,29	43,20	14,29	9,22	5,56	4,27	38
1migCutRewriteK7-Gates	31,29	43,20	14,29	9,22	5,56	4,27	39
1migCutRewriteK8-Gates	31,29	43,20	14,29	9,26	5,53	4,27	40
2migCutRewriteK3-Gates	29,25	43,74	11,44	7,61	3,87	3,94	2
2migCutRewriteK4-Gates	31,29	49,59	15,34	8,53	6,61	6,00	3
2migCutRewriteK5-Gates	31,29	49,48	15,87	9,64	6,62	5,58	4
2migCutRewriteK6-Gates	31,29	49,51	15,90	9,63	6,62	5,58	5
2migCutRewriteK7-Gates	31,29	49,51	15,89	9,63	6,62	5,58	6
2migCutRewriteK8-Gates	31,29	49,51	15,89	9,63	6,62	5,58	7
3migCutRewriteK3-Gates	29,25	43,75	11,49	7,61	3,88	3,94	13
3migCutRewriteK4-Gates	31,29	49,88	15,61	8,71	6,59	6,05	14
3migCutRewriteK5-Gates	31,29	49,70	16,51	9,91	6,90	6,04	15
3migCutRewriteK6-Gates	31,29	49,73	16,54	9,89	6,91	6,04	16
3migCutRewriteK7-Gates	31,29	49,73	16,53	9,88	6,91	6,04	17
3migCutRewriteK8-Gates	31,29	49,73	16,52	9,74	7,00	6,04	18
4migCutRewriteK3-Gates	29,25	43,75	11,52	7,63	3,88	3,94	24
4migCutRewriteK4-Gates	31,29	49,88	15,70	8,69	6,62	6,05	25
4migCutRewriteK5-Gates	31,29	49,77	16,72	10,01	7,02	6,15	26
4migCutRewriteK6-Gates	31,29	49,80	16,76	9,99	6,87	6,00	27
4migCutRewriteK7-Gates	31,29	49,80	16,75	9,98	6,87	6,00	28
4migCutRewriteK8-Gates	31,29	49,80	16,75	9,75	7,00	6,00	29
1migRefac-Gates	29,93	19,51	11,82	7,99	3,98	3,88	43
2migRefac-Gates	33,33	24,39	13,24	8,96	4,96	4,34	10
3migRefac-Gates	33,33	24,39	13,29	8,98	4,95	4,34	21
4migRefac-Gates	33,33	24,39	13,29	8,98	4,95	4,34	32
1migResub-Gates	11,56	9,76	4,05	3,26	2,16	2,07	44
2migResub-Gates	11,56	9,76	4,07	3,26	2,16	2,07	11
3migResub-Gates	11,56	9,76	4,07	3,26	2,16	2,07	22
4migResub-Gates	11,56	9,76	4,07	3,26	2,16	2,07	33
mix1K3-Gates	37,41	43,73	17,64	10,78	6,72	3,87	45
mix1K4-Gates	37,41	43,89	18,54	10,24	8,28	4,76	46
mix1K5-Gates	37,41	43,81	18,49	10,13	8,08	5,00	47
mix1K6-Gates	37,41	43,84	18,50	10,12	8,08	5,01	48
mix1K7-Gates	37,41	43,84	18,49	10,12	8,08	5,01	49
mix1K8-Gates	37,41	43,84	18,49	10,12	8,08	5,01	50
mix2K3-Gates	37,41	43,27	17,48	11,67	5,96	3,43	51
mix2K4-Gates	37,41	43,65	18,32	11,41	7,11	4,26	52
mix2K5-Gates	37,41	43,57	18,50	11,78	6,98	4,35	53
mix2K6-Gates	37,41	43,57	18,50	11,81	6,96	4,35	54
mix2K7-Gates	37,41	43,57	18,50	11,81	6,96	4,35	55
mix2K8-Gates	37,41	43,57	18,51	11,81	6,96	4,35	56
mix3K3-Gates	37,41	43,29	17,56	11,70	5,97	3,30	57
mix3K4-Gates	37,41	43,66	18,48	11,33	7,08	4,13	58
mix3K5-Gates	37,41	43,50	18,68	11,74	6,93	4,22	59
mix3K6-Gates	37,41	43,50	18,68	11,74	6,94	4,22	60
mix3K7-Gates	37,41	43,50	18,69	11,80	6,90	4,22	61
mix3K8-Gates	37,41	43,50	18,69	11,80	6,90	4,22	62
mix4K3-Gates	37,41	43,28	16,98	10,75	5,34	2,54	63
mix4K4-Gates	37,41	43,65	17,93	10,48	5,86	2,79	64
mix4K5-Gates	37,41	43,57	18,11	10,80	5,90	3,00	65
mix4K6-Gates	37,41	43,57	18,11	10,80	5,90	3,00	66
mix4K7-Gates	37,41	43,57	18,12	10,80	5,90	3,00	67
mix4K8-Gates	37,41	43,57	18,12	10,80	5,90	3,00	68

Figura 68 – Tabela de análise da taxa de otimização média dos comandos *Mockturtle*

Anexos

ANEXO A – Benchmark Xor5-r1

Descrição do circuito XOR5-r1 do benchmark TOY (WALTER et al., 2019), utilizado nos experimentos.

```
module source (pi0, pi1, pi2, pi3, pi4, po0);
input  pi0, pi1, pi2, pi3, pi4;
output po0;
wire n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17;
assign n7 = ~pi3 & pi4;
assign n8 = pi3 & ~pi4;
assign n9 = ~n7 & ~n8;
assign n10 = ~pi1 & pi2;
assign n11 = pi1 & ~pi2;
assign n12 = ~n10 & ~n11;
assign n13 = pi0 & ~n12;
assign n14 = ~pi0 & n12;
assign n15 = ~n13 & ~n14;
assign n16 = n9 & n15;
assign n17 = ~n9 & ~n15;
assign po0 = n16 | n17;
endmodule
```

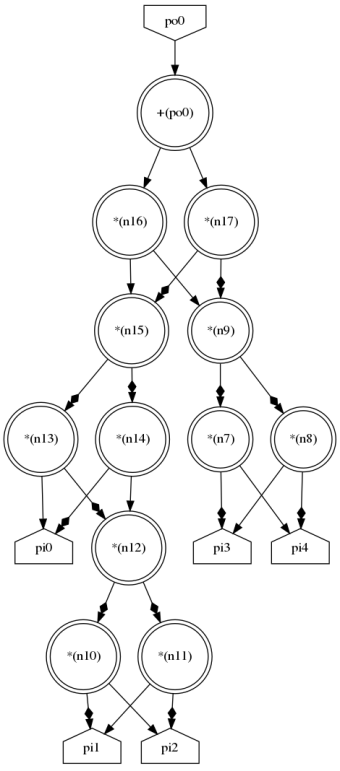


Figura 69 – Representação gráfica circuito *XOR5-r1* utilizando estrutura AOI.

Descrição do circuito XOR5-r1 do benchmark MAJ (WALTER et al., 2019), utilizado nos experimentos.

```
module top(x0, x1, x2, x3, x4, y0);
input x0, x1, x2, x3, x4;
output y0;
wire n6, n7, n8, n9, n10, n11;
assign n6 = (~x0 & x1) | (~x0 & x2) | (x1 & x2);
assign n7 = (x1 & x2) | (x1 & ~n6) | (x2 & ~n6);
assign n8 = (x0 & n6) | (x0 & ~n7) | (n6 & ~n7);
assign n9 = (~x3 & x4) | (~x3 & n8) | (x4 & n8);
assign n10 = (x4 & n8) | (x4 & ~n9) | (n8 & ~n9);
assign n11 = (x3 & n9) | (x3 & ~n10) | (n9 & ~n10);
assign y0 = n11;
endmodule
```

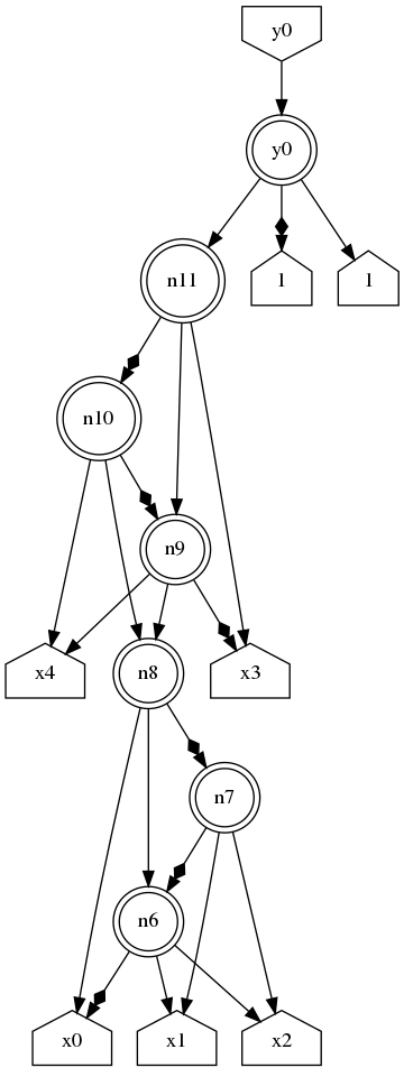


Figura 70 – Representação gráfica circuito *XOR5-r1* utilizando estrutura MAJ.