UNIVERSIDADE FEDERAL DE PELOTAS

Centro de Desenvolvimento Tecnológico

Programa de Pós-Graduação em Computação

Dissertação

Análise de métodos de mapeamento tecnológico para dispositivos QCA

MELISSA DE SOUZA RABASSA COLVARA

Pelotas, 2013

MELISSA DE SOUZA RABASSA COLVARA

Análise de métodos de mapeamento tecnológico para dispositivos QCA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Leomar Soares da Rosa Jr.
Co-orientador: Prof. Dr. Felipe de Souza Marques

Pelotas, 2013

Banca examinadora:

Prof. Dr. Paulo Francisco Butzen

Profa. Dra. Lisane Brisolara de Brisolara

Prof. Dr. Rafael Iankowski Soares

AGRADECIMENTOS

Agradecer a todos que ajudaram a elaboração desta dissertação não é tarefa fácil. O maior desafio não é decidir quem incluir, mas o receio de não mencionar alguém por mero descuido na hora da escrita. Então, aos colegas do Programa de Mestrado em Computação e do Grupo de Arquiteturas e Circuitos Integrados, que de alguma forma, contribuíram com seu apoio e com críticas construtivas na realização deste trabalho, gostaria de expressar minha sincera gratidão. Aos professores da primeira turma de mestrado em Computação da UFPEL, meu muito obrigada por dividirem seus conhecimentos e pela oportunidade de poder fazer parte dessa turma! Agradeço aos meus pais, em especial minha mãe, por terem sempre me incentivado a continuar os estudos, e terem sempre uma palavra amiga de sustentação para que o objetivo fosse alcançado. Meu maior agradecimento é dirigido ao meu marido, Mateus, pelo apoio e suporte em todos os sentidos. Muito obrigado é pouco para agradecer tamanha compreensão e dedicação. Em especial, agradeço ao meu orientador, Prof. Dr. Leomar Soares da Rosa Jr., e ao meu co-orientador, Prof. Dr. Felipe de Souza Marques, pela dedicação oferecida, ajuda e pelo constante incentivo nos momentos mais difíceis desta trajetória. Sem o auxílio de vocês na condução do trabalho seria impossível a sua conclusão. Agradeço também, os colegas de laboratório, Stéphano Gonçalves por toda ajuda com os testes e ao Júlio Saraçol, pelas contribuições sempre bem pontuais, sem a ajuda de vocês certamente não teria conseguido finalizar esta fase. E por fim mas também, não menos importante, aos colegas, Adriano Maron, Lidiane Visintin, Milena Marques e Murian Ribeiro, obrigada por me acompanharem durante estes dois anos, formamos um ótimo grupo de trabalho e amizade, que pretendo guardar para o resto da vida.

O futuro tem muitos nomes.
Para os fracos é o inalcançável.
para os temerosos, o desconhecido.
Para os valentes é a oportunidade.
— VITOR HUGO

RESUMO

COLVARA, Melissa de Souza Rabassa. **Análise de métodos de mapeamento tecnológico para dispositivos QCA**. 2013. 60 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal de Pelotas, Pelotas.

As técnicas e metodologias atuais tem avançado cada vez mais com relação à otimização de circuitos digitais, proporcionando a construção de circuitos integrados mais eficientes, com uma capacidade de integração jamais vista anteriormente. Parte deste avanço se deve a área de síntese lógica aplicada na tecnologia CMOS (*Complementary Metal-Oxide Semiconductor*). Através da síntese lógica, os projetos digitais podem ser construídos e otimizados de forma mais eficiente, atendendo os requisitos necessários para serem empregados na concepção dos mais diversos circuitos integrados. A tecnologia CMOS é a tecnologia atual de implementação dos circuitos integrados largamente fabricados nos dias de hoje, tendo como componente elementar o transistor, o qual atua como uma chave e é empregado na construção das diferentes portas lógicas que compõem o circuito. Entretanto, diversos trabalhos encontrados na literatura têm apontado para a necessidade de uma nova tecnologia substituta, visto que limites físicos para a construção dos transistores na tecnologia CMOS inviabilizarão a construção de circuitos integrados futuros. Neste sentido, diversos autores têm estudado e apostado na tecnologia QCA (*Quantum Cellular Automata*) como uma potencial substituta da tecnologia CMOS. Este trabalho explora a utilização da síntese lógica, especialmente na etapa de mapeamento tecnológico, aplicada a esta nova tecnologia. O objetivo principal do trabalho consiste na realização de uma análise da aplicação dos métodos de mapeamento tecnológico quando aplicados a tecnologia QCA. Resultados obtidos com a ferramenta ABC e com bibliotecas descritas para a tecnologia QCA são apresentados e comparados com trabalhos da literatura.

Palavras-chave: QCA, síntese lógica, biblioteca de células, mapeamento tecnológico.

ABSTRACT

COLVARA, Melissa de Souza Rabassa. **Analysis of technology mapping applied to QCA devices**. 2013. 60 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal de Pelotas, Pelotas.

Nowadays, the advanced methodology and techniques to optimize digital circuits are able to deliver more efficient integrated circuits in terms of area, power and delay. Logic synthesis is one of the main responsible that have collaborated for the success of the CMOS (Complementary Metal-Oxide Semiconductor) technology. Using several algorithms and methods dedicated to optimize logic cells and digital circuits, the logic synthesis cans delivery solutions able to meet design requirements. Examples are the technology mapping algorithms, methods and its related EDA (Electronic Design Automation) tools. Currently, the CMOS technology is widely used to implement integrated circuits. The main element behind this technology is the transistor, which is able to act as a switch inside a logic cell. However, several and recent works have pointed to the need for a future and substitute technology, since the transistor is reaching its physical limits. In this sense, some authors have investigated the QCA (Quantum Cellular Automata) technology as a potential alternative to the CMOS. This work explores the use of logic synthesis, especially in the technology mapping step, to investigate if traditional solutions applied to CMOS are able to release good results for QCA circuits.

Keywords: QCA, logic synthesis, cell library, technology mapping..

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABC	*A System Sequential Synthesis and Verification*
AIG	*And-Inverter Graph*
AND	Operador booleano ”E”
AOI	*And-Or-Inverter*
BDD	*Binary Decision Diagram*
BLIF	*Berkeley Logic Interchange Format*
CAD	*Computer-Aided-Design*
CMOS	*Complementary Metal-Oxide-Semiconductor*
CONST	Constante
DAG	*Direct Acyclic Graph*
EDA	*Electronic Design Automation*
FPGA	*Field Programmable Gate Array*
FC	*Functional Composition*
LUT	*Look-up table*
MALS	*Majority Logic Synthesizer*
MCNC	*Microeletronics Center of North Carolina*
MV	*Majority Voter*
NOT	Operador booleano Inversor
OR	Operador booleano ”OU”
PGA	*Programmable gate array*
POS	Produto da soma
PI	Entrada primária
QCA	*Quantum-dot Cellular Automata*
SIS	*Sequential Interactive Syntesis*
SOP	Soma de produtos
VLSI	*Very-large-scale integration*

SUMÁRIO

1 INTRODUÇÃO

Durante as últimas décadas, a tecnologia CMOS tem sido amplamente utilizada, com grande sucesso na concepção de circuitos digitais. Mas a cada dia, a necessidade de desenvolvimento de circuitos mais compactos, com maior desempenho e menor consumo de energia, tem aumentado, tornando a concepção de circuitos VLSI (*Very-large-scale integration*) cada vez mais exigente.

Desde 1965, a Lei de Moore tem norteado o desenvolvimento dos microprocessadores (NETO, 2006b). Moore prevê a duplicação do número de transistores em um chip, aproximadamente, a cada dois anos. Nas últimas décadas têm-se visto o desenvolvimento da indústria de semicondutores, que tem conseguido acompanhar o previsto por Moore, ou seja, a cada dois anos em média tem dobrado o número de transistores nos processadores, consequentemente, aumentando o poder de processamento dos computadores. Conforme podemos verificar na figura 1 que ilustra o crescimento exponencial do número de transistores nos processadores da Intel, mostrando um comparativo entre a Lei de Moore e os processadores fabricados até o ano de 2006.

Após grandes evoluções tecnológicas, existem processos de fabricação que permitem o desenvolvimento de circuitos com bilhões de transistores. Estudos feitos na área de síntese lógica ajudaram na otimização desses circuitos. Para tanto, os projetistas adotaram uma forma alternativa para a execução e concepção dos projetos e circuitos, passando a utilizar ferramentas que automatizam e reduzem o tempo de desenvolvimento dos mesmos.

Na tabela 1 é possível observar o número de transistores com a respectiva tecnologia utilizada nos processadores da Intel desde 1971. Graças às pesquisas a partir da nanotecnologia, será possível construir os transistores dos chips cada vez menores, fazendo com que os circuitos atinjam maior desempenho, sem que isto aumente a área física utilizada. O processador Intel 486, por exemplo, utilizava mais de 1 milhão de transistores, e, se comparado aos processadores mais atuais da linha Intel 2° Geração que contam com cerca de 1,4 bilhões de transistores, nota-se o considerável aumento no número de transistores. Os últimos chips fabricados utilizam tecnologia de fabricação de 0,022 *micron* (ou 22 nanômetros), ou seja, maior capacidade de processamento em um espaço compacto de área física.

Apesar do sucesso obtido até aqui, alguns trabalhos tem apontado os limites físicos para a tecnologia CMOS. Entre esses limites podemos citar as consequências elétricas da redução do canal do transistor, dificuldades no processo de litografia e o alto custo de fabricação (YANO et al., 2006). A (SEMICONDUCTORS ITRS, 2009) destaca o limite da litografia.

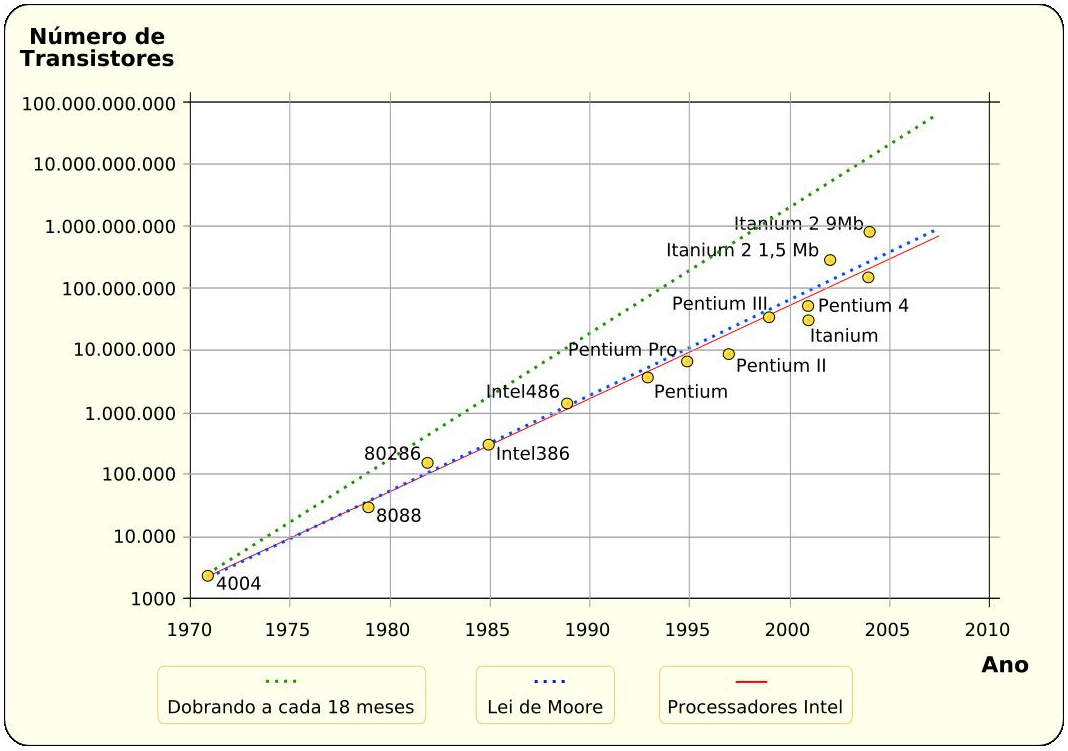


Figura 1: Comparativo entre a Lei de Moore e os processadores atuais (NETO, 2006a)

Na confecção de um transistor a técnica de litografia é utilizada com uma luz ultravioleta que atravessa um vão determinado por uma máscara, atingindo o silício e marcando a área desejada pelo projeto do componente. Conforme podemos observar na figura 2. Com a diminuição do componente a máscara, e por consequência o vão, também diminuem fazendo com que a luz ultravioleta se **difrate** e atinja áreas não determinadas no projeto.

Sendo assim, especula-se que seja possível estender o processo de litografia por mais uma geração depois dos 10 nm, mas a partir daí esbarraremos no limite fundamental do processo de litografia, ou seja, o limite físico do material (LEDESMA, 2010).

Baseado nisso, algumas pesquisas tem apontado novas tecnologias como substitutas para CMOS. Podemos citar algumas, tais como: computação quântica, autômatos celulares com pontos quânticos, nanotubos de carbono, dentre outros (NETO, 2006b). A partir de 1994, alguns autores passaram a ver na tecnologia QCA (*Quantum Cellular Automata*) uma possibilidade para substituir a tecnologia CMOS (TOUGAW; LENT, 1994). QCA tem sido sugerida, pois pode ser utilizada com o propósito geral de projetar circuitos com o consumo de potência extremamente baixo e velocidade muito elevada, se comparada com a tecnologia CMOS (KONG; SHANG; LU, 2010).

A QCA é uma tecnologia de computação e transformação da informação utilizando autômatos celulares. É baseada em codificação binária, definida pela configuração das cargas nas células de pontos quânticos, onde a computação é realizada pela interação de Coulombic entre células vizinhas, e não por corrente elétrica como nos computadores tradicionais. Nenhum fluxo de corrente ocorre entre estas células. Para construção de circuitos digitais, estruturas básicas da

Tabela 1: Crescimento do número de transistores nos processadores Intel (INTEL, 2012)

Ano	Processador	Clock	Transistores	Tecnologia
1971	4004	$108Khz$	2.300	$10m$
1972	8008	$800Khz$	3.500	$10m$
1974	8080	$2Mhz$	4.500	$6m$
1978	8086	$5Mhz$	29.000	$3m$
1982	286	$6Mhz$	134.000	$1,5m$
1985	386	$16Mhz$	275.000	$1,5m$
1989	486	$25Mhz$	$1,2$ $milhões$	$1m$
1993	$Pentium$	$66Mhz$	$3,1$ $milhões$	$0,8m$
1995	$PentiumPro$	$200Mhz$	$5,5$ $milhões$	$0,35m$
1997	$PentiumII$	$300Mhz$	$7,5$ $milhões$	$0,25m$
1998	$Celeron$	$266Mhz$	$7,5$ $milhões$	$0,25m$
1999	$PentiumIII$	$600Mhz$	$9,5$ $milhões$	$0,25m$
2000	$PentiumIV$	$1.5Ghz$	42 $milhões$	$0,18m$
2001	$Xeon$	$1.7Ghz$	42 $milhões$	$0,18m$
2003	$PentiumM$	$1.7Ghz$	55 $milhões$	$90n$
2006	$Core2Duo$	$2.66Ghz$	291 $milhões$	$65n$
2008	$Core2Duo$	$2.4Ghz$	410 $milhões$	$45n$
2008	$Atom$	$1.86Ghz$	47 $milhões$	$45n$
2010	$2°$ $Geração$ $Core$	$3.8Ghz$	$1,16$ $bilhões$	$32n$
2012	$3°$ $Geração$ $Core$	$2.9Ghz$	$1,4$ $bilhões$	$22n$

tecnologia são utilizadas, são elas: inversor QCA, fio QCA e porta majoritária QCA.

Recentes artigos na literatura que focam o uso de dispositivos QCA tem verificado a viabilidade de usá-la como tecnologia substituta à CMOS (ZHANG et al., 2004), (MOMENZADEH et al., 2005), (ZHANG; GUPTA; JHA, 2007), (KONG; SHANG; LU, 2010). Por ser, ainda, relativamente nova, poucos trabalhos existem na área sobre síntese lógica de circuitos desenvolvidos com esta tecnologia.

No projeto tradicional de circuitos CMOS, a aplicação de métodos de síntese consiste em converter funções booleanas em expressões descritas como soma de produtos (SOP) ou produtos de soma (POS), e a implementação dos circuitos lógicos é feita com portas AND e OR, baseadas nessas expressões SOP ou POS (KONG; SHANG; LU, 2010). No projeto QCA, os circuitos lógicos são baseados em portas majoritárias. Devido à complexibilidade multi-nível dos circuitos majoritários, não existe uma forma eficiente de converter as funções booleanas SOP (POS) em expressões majoritárias. É preciso então o desenvolvimento de um método eficiente que consiga mapear circuitos CMOS para tecnologia QCA, com uma combinação mínima de portas majoritárias e sem alterar a função lógica inicial. Algumas ferramentas de síntese lógica, utilizadas em CMOS, conseguem ser utilizadas para produzir circuitos QCA com portas majoritárias, mas não de forma eficiente.

Mapas de Karnaugh e Decomposição de Shannon são alguns dos métodos

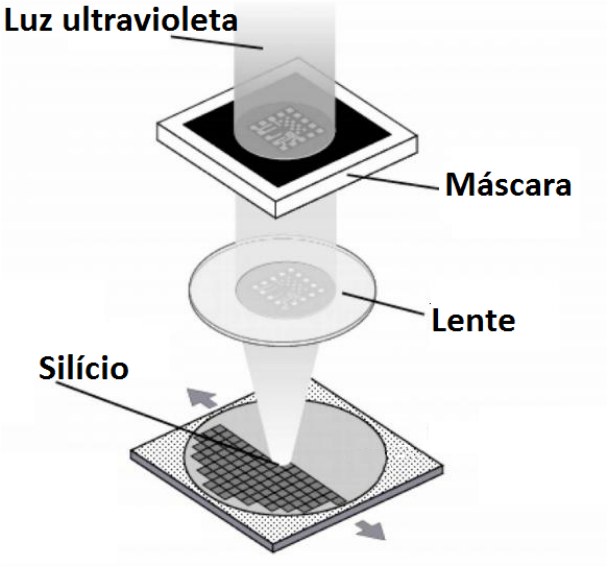


Figura 2: Processo de litografia

de síntese mais utilizados, mas estes têm a desvantagem de serem úteis somente para síntese de pequenas redes, pois são métodos resolvidos manualmente. A pesquisa em síntese lógica majoritária tem sido feita desde a década de 1960. Um método de síntese com base em tabela verdade reduzida e unificada foi dado em (AKERS, 1962), enquanto outra metodologia baseada em Mapas de Karnaugh (K-map) foi apresentada em (MILLER; WINDER, 1962). Em (MUROGA, 1971), um método de síntese baseado em decomposição de Shannon foi empregado. No entanto, este método resultou em um número exponencial de portas majoritárias. Uma séria desvantagem destes métodos é que eles só são adequados para a síntese de pequenas redes, pois são manuais. Desta forma, para grandes redes, estes métodos tornam-se muito complexos.

Outros métodos de síntese majoritária para circuitos QCA foram descritos mais recentemente. Uma das principais contribuições na área foi dada por (ZHANG et al., 2004), onde é demonstrada uma técnica para simplificar funções booleanas, de três entradas, tendo por base portas majoritárias. A partir disto são apresentadas 13 funções padrão que podem mapear circuitos de até 3 entradas. Em 2005, (MOMENZADEH et al., 2005) apresentou a porta AOI, que pode implementar funções de 3, 4 ou 5 entradas, no intuito de substituir a porta majoritária, visto que esta alcançava resultados melhores nas ferramentas de síntese existentes. Entretanto, a porta AOI não teve sucesso, pois seu custo em área foi mais elevado quando comparado a porta majoritária. (ZHANG; GUPTA; JHA, 2007) propôs o primeiro método automatizado para síntese QCA, chamado MALS. O trabalho estado da arte em síntese QCA é o de (KONG; SHANG; LU, 2010), onde são apresentados resultados melhores que os apresentados pelo método MALS, propondo *scripts* para executar de forma automatizada a síntese para dispositivos QCA utilizando a ferramenta SIS (SENTOVICH et al., 1992).

Nos capítulos seguintes estes trabalhos serão detalhados, assim como outros conceitos importantes que servem à base de estudo desta dissertação.

1.1 Objetivo

Este trabalho tem por objetivo analisar os métodos existentes para síntese lógica de projetos de circuitos QCA. O estudo é realizado visando verificar se é possível alcançar resultados satisfatórios para síntese lógica de circuitos QCA quando métodos de mapeamento tecnológico da tecnologia CMOS são empregados. Na atualidade não há uma ferramenta própria para a tecnologia QCA que sintetize os circuitos com eficiência. Assim, esse trabalho propõe bibliotecas baseadas em células QCA, com objetivo de verificar a qualidade dos métodos de mapeamento tecnológico CMOS existentes. Bibliotecas com tamanhos diferentes são utilizadas na ferramenta ABC e experimentos são realizados com o conjunto de *benchmarks* ISCAS'85.

1.2 Organização do trabalho

Este trabalho está organizado da seguinte forma: o primeiro capítulo contém a introdução do trabalho, a motivação e o seu objetivo. O segundo capítulo apresenta um apanhado de conceitos e definições para permitir uma maior compreensão do que será abordado ao longo do texto. Os trabalhos mais recentes em síntese lógica são a essência da análise proposta neste trabalho. Por este motivo, o terceiro capítulo discute cada um dos trabalhos analisados, apresentando suas principais características. Após introduzir os conceitos relevantes para o entendimento, o desenvolvimento do trabalho e uma série de experimentos são propostos no capítulo 4. Os resultados dos experimentos são apresentados no quinto capítulo, onde também inclui-se uma comparação entre o resultante dos experimentos propostos e os resultados obtidos pelo estado da arte. Por fim são apresentadas as conclusões.

2 CONCEITOS E DEFINIÇÕES

Este capítulo apresenta conceitos utilizados no desenvolvimento deste trabalho. Iniciando com a tecnologia QCA, tem-se uma breve descrição e forma de funcionamento de seus dispositivos. O próximo item é o mapeamento tecnológico, onde serão explanados seus conceitos e definições. Por fim, o conceito de biblioteca de células é apresentado.

2.1 Tecnologia QCA

Nesta seção serão apresentados alguns conceitos que ajudam a entender a tecnologia QCA, ou seja, o funcionamento e a interação das células entre si.

2.1.1 Base QCA

Uma célula QCA consiste de 4 pontos (poços) quânticos localizados em um quadrado e 2 elétrons livres, que se movimentam através de um túnel entre os pontos adjacentes. Pela repulsão de Coulombic, os elétrons procuram manter-se em um estado de repouso. Sua fórmula afirma que quanto menor a distância entre os elétrons, maior será a força de repulsão entre eles. Devido a isso, os elétrons tendem a ocupar pontos quânticos de extremos opostos aos da célula, ou seja, a repulsão eletrostática faz com que os elétrons extras ocupem locais diagonalmente opostos. Portanto, uma célula isolada tem dois estados equivalentes de energia.

Estes estados são denominados de polarização de célula, P = +1 e P = -1. O termo polarização de célula refere-se ao arranjo das cargas no interior da célula de QCA. As configurações desses dois elétrons denotam células polarizadas P=+1 e P=-1. Na codificação para informação binária usa-se P=+1 representando o $"1"$ lógico e P=-1 representando o $"0"$ lógico, que é representado na figura 3.

2.1.2 Dispositivos Lógicos QCA

Para construir os dispositivos lógicos QCA, as células devem ser posicionadas de maneira a aproveitar a interação existente entre elas, chegando assim à lógica desejada. Os principais elementos da tecnologia são o Fio QCA, o Inversor QCA e o principal dispositivo que é a porta majoritária QCA.

O fio QCA é o dispositivo mais simples, pois trata-se de uma linha de células QCA. No fio QCA a propagação é dirigida de uma célula de entrada com polarização fixa até a última célula, ou seja, esta estrutura leva o estado lógico de

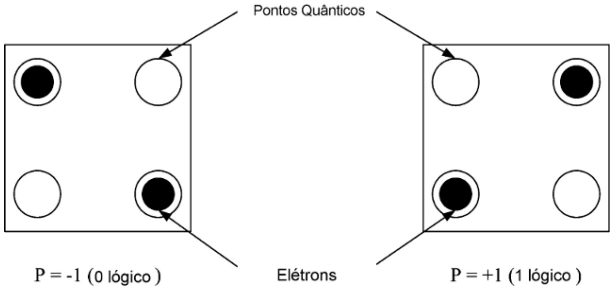


Figura 3: Célula QCA básica. Apresenta informação binária e é codificada em duas células polarizadas diagonalmente, em seus dois possíveis estados de polarização. (KONG; SHANG; LU, 2010)

um ponto a outro do circuito. Quando está no estado de menor energia, todas as células posicionadas em linha terão a mesma polarização da célula de entrada.

Assim, a polarização da célula de entrada se propaga ao longo do fio sem a necessidade de fluxo de corrente elétrica. Então o sinal binário é propagado da entrada para a saída devido a interações eletrostáticas entre as células. A propagação da informação pode ser de 90º e 45º conforme a figura 4 apresenta:

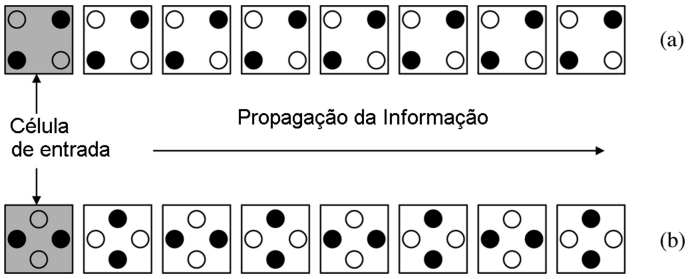


Figura 4: Fio QCA com propagação de (a) 90° e (b) 45° (KONG; SHANG; LU, 2010)

Células colocadas em diagonal uma para outra tem polarização inversa, então organizando o fio de forma que as células fiquem dispostas em um ângulo de 45º forma-se a estrutura do Inversor QCA. As células são orientadas a 45º para outra tomar a polarização oposta, esta inversão de polarização é devida à repulsão entre os elétrons.

Na figura 5 é apresentado o leiaute de um inversor QCA. A lógica deste componente é a mesma de um inversor na tecnologia CMOS, ou seja, o valor de entrada é alterado para o valor inverso na saída.

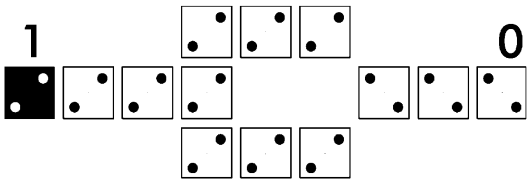


Figura 5: Inversor QCA - (KONG; SHANG; LU, 2010)

O dispositivo lógico QCA básico é a estrutura *majority gate*, ou porta majoritária, de 3 entradas, que têm como função lógica:

M(a,b,c)= ab+ac+bc

onde a, b e c são entradas arbitrárias. Esta estrutura é composta por 5 células QCA e tem capacidade de mapear funções lógicas de portas AND e OR na tecnologia CMOS. Na figura 6 é apresentado o leiaute de uma porta majoritária QCA.

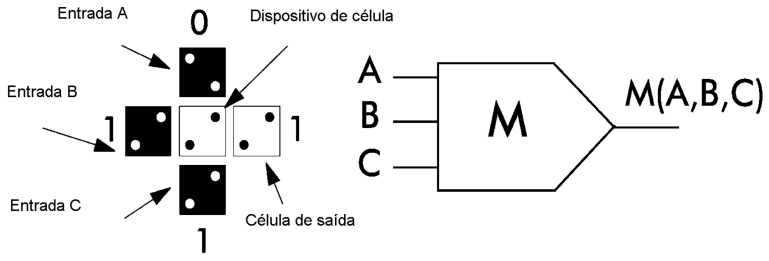


Figura 6: Porta Majoritária QCA (ZHANG et al., 2004)

Na literatura, o estudo de portas majoritárias incide principalmente sobre as implementações de uma função majoritária de "n" entradas, usando a lógica de limiar. Uma porta majoritária de n-entrada produz uma saída lógica "1" se a maioria de suas entradas é lógica "1" (ZHANG et al., 2004).

A tendência dos dispositivos de célula na porta majoritária de passar para um estado fundamental garante que ele assume a polarização da maioria das células vizinhas. O dispositivo de célula tende a seguir a polarização majoritária porque esta representa o estado de menor energia, ou seja, quando a célula central tem sua energia mais baixa, ela assume a polarização da maioria das três células de entrada. Isto ocorre devido à repulsão entre os elétrons das três células de entrada e da célula central ser menor.

Este comportamento pode ser observado na figura 6, onde a célula de entrada "A" está polarizada representando o "0" lógico, a entrada "B" e "C" estão polarizadas com o "1" lógico, mas a célula central (dispositivo de célula) tem a mesma polarização das células de entrada "B" e "C", enviando para a célula de saída o valor determinado pela maioria das células de entrada.

Uma porta majoritária possui 3 entradas, e uma dessas entradas é o que seleciona o comportamento que a porta majoritária assumirá, que pode ser de uma função OR representado com o valor "1" (1), ou AND representado com o valor "0" (2), conforme representado abaixo:

M(a,b,1)=a+b (1)
M(a,b,0)=a.b (2)

Por exemplo, ainda na figura 6, é possível observar este comportamento quando a célula de entrada "A" é fixada na polarização que representa o binário "0", tem-se uma porta lógica AND de duas entradas, "B" e "C". E quando a célula "A" for fixada representando o binário "1", tem-se uma porta lógica OR. Com isso consegue-se mapear todos os circuitos lógicos QCA em uma porta majoritária QCA de 3 entradas, o que pode ser visto e comparado na tabela 2 que representa essas duas funções AND e OR. Na coluna A têm-se a representação do sinal que determinará qual a função a porta majoritária irá representar, e nas colunas B, C e saída consegue-se verificar a tabela verdade de determinada

função. Aqui cita-se apenas as portas AND e OR como exemplo, mas é possível representar todas as funções lógicas através de portas majoritárias.

Tabela 2: Tabela verdade de Funções lógicas AND e OR

Funções	**A**	**B**	**C**	**Saída**
AND	0	0	0	0
AND	0	0	1	0
AND	0	1	0	0
AND	0	1	1	1
OR	1	0	0	0
OR	1	0	1	1
OR	1	1	0	1
OR	1	1	1	1

2.1.3 QCA *Clock*

Na tecnologia CMOS, o *clock* é um sinal elétrico que fornece uma referência de tempo para todas as atividades e permite o sincronismo das operações internas de um computador. O *clock* é um pulso alternado de sinais de tensão que oscila entre os estados alto e baixo (habilitado ou desabilitado).

Na tecnologia QCA, o conceito de *clock* também é inserido, mas não como um sinal elétrico, e sim como um campo de energia que controla as barreiras de tunelamento dentro de uma célula, controlando quando a célula pode ou não ser/estar polarizada. Na figura 7, é possível verificar estas barreiras de tunelamento dentro de uma célula.

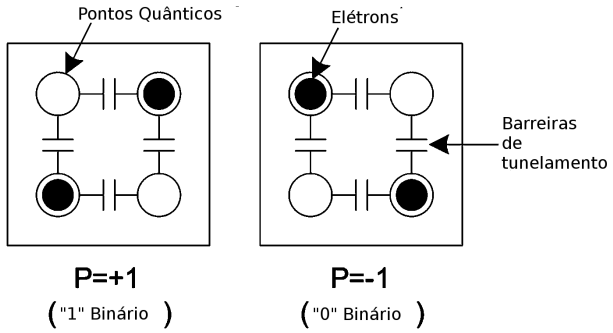


Figura 7: Barreiras de tunelamento em uma célula básica (CHO H.; SWARTZLANDER, 2007)

O *clock* de um circuito QCA é dividido em quatro fases: *Switch*, *Hold*, *Release* e *Relax*. Na figura 8, consegue-se observar estas fases. O circuito QCA é particionado em zonas (fases). Estas podem ser de tamanho irregular, mas devem obedecer aos limites de fabricação e dissipação. As células que estão dentro dessa zona são controladas por um mesmo sinal de *clock*.

A primeira fase é a *Switch*. Nesta, as células de QCA começam despolarizadas e com o potencial das barreiras de tunelamento baixo. Durante esta fase as barreiras entre os pontos adjacentes são levemente elevadas e as células QCA

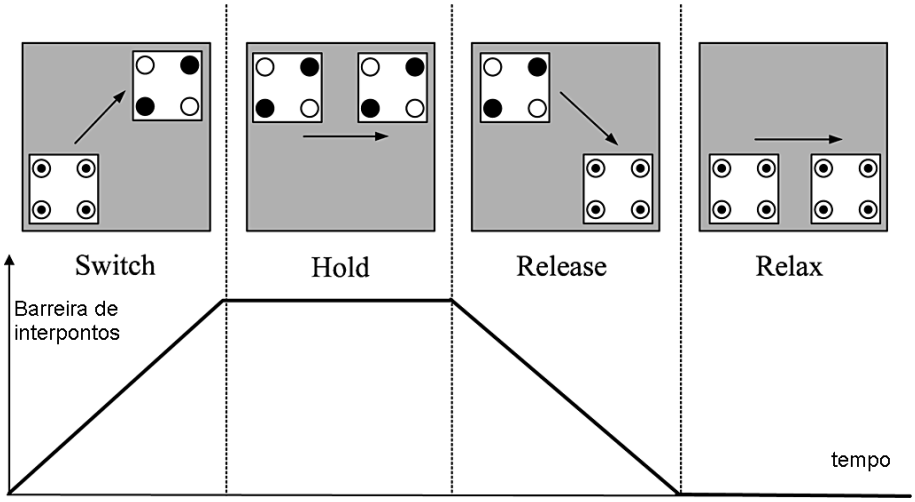


Figura 8: Fases de *Clock* QCA (KONG; SHANG; LU, 2010)

tornam-se polarizadas de acordo com o estado das suas células de entrada. Neste ponto é que ocorre a computação, ao final desta fase as barreiras estão elevadas o suficiente para evitar o tunelamento de qualquer elétron, e os estados das células estão fixos.

A segunda fase é o *Hold*. Pode-se dizer que é o estado estável da célula, pois a computação iniciada na fase *switch* mantém o valor durante o *Hold*. Com as células em estado fixo, elas podem ser usadas como entrada para o próximo estágio. Na fase *Release*, as barreiras são reduzidas, permitindo as células relaxar e passar para um estado não polarizado.

Na ultima fase, o *Relax*, a célula passa a ser inativa, as barreiras são mantidas baixas e sem polarização. Assim sendo, a célula é sempre atualizada a cada ciclo.

2.2 Mapeamento tecnológico

O processo de síntese lógica tem como uma das bases o mapeamento tecnológico. Nesta fase procura-se uma maior otimização do projeto do circuito. Isto é realizado utilizando um conjunto de elementos denominado células, pertencentes a uma biblioteca, que irá implementar o circuito em determinada tecnologia.

A tecnologia de mapeamento é parte importante na fase de síntese do circuito, pois é nessa fase que é realizada a seleção final de portas contidas em uma determinada biblioteca. Em geral, assume-se que o circuito independente da tecnologia já tenha passado por uma otimização estrutural booleana. Geralmente, os algoritmos de mapeamento não mudam a estrutura inicial do circuito (MARQUES, 2008).

Sendo assim, o mapeamento consiste na escolha de um conjunto de instancias de células disponíveis em uma biblioteca e suas interconexões, de forma a implementar todo o conjunto de funções descritas por um circuito lógico, minimizando uma determinada função objetivo (MARQUES et al., 2009).

O mapeamento tecnológico converte um circuito, descrito como uma *netlist* de portas lógicas, em uma descrição que faz referencia a elementos de uma dada tecnologia. O objetivo desse processo é decidir qual o conjunto de células

será utilizado para implementar o circuito de forma que se atinjam metas como frequência de operação do circuito, área utilizada, consumo de energia, etc... Isto mostra que os algoritmos de mapeamento têm de ser inteligentes para se adaptar as bibliotecas existentes.

Para a tecnologia CMOS, a indústria tem utilizado geralmente a metodologia *standard-cell*, que é baseada no uso de portas lógicas pré-definidas para construção de circuitos mais complexos. A escolha da biblioteca a ser utilizada, não é uma tarefa fácil, pois uma biblioteca com grande número de células pode trazer um alto custo computacional na fase de cobertura. Entretanto, se é utilizada uma biblioteca com número reduzido de células, pode-se restringir o resultado final pois as soluções encontradas serão sub-ótimas.

Visando a qualidade do circuito, além da escolha da biblioteca, a estrutura de dados utilizada para representar o circuito desejado, está fortemente ligada à eficiência do mapeamento. O processo de mapeamento pode ser dividido em três fases: decomposição lógica, identificação de padrões e cobertura. Nas próximas seções serão abordadas algumas estruturas de dados e serão abordados maiores detalhes de cada fase de mapeamento.

2.2.1 Estruturas de dados

A escolha da estrutura de dados pode determinar o sucesso de um algoritmo de mapeamento. Existem várias estruturas de dados, e cada metodologia utiliza a estrutura a que melhor se adapta. Para citar algumas estruturas existentes, se pode salientar estruturas de árvores, BDD (Diagrama de Decisão Binária), DAG (*Direct Acyclic Graph*) e AIG (*And-Inverter Graph*).

Visando a tecnologia QCA, vamos nos deter somente às estruturas baseadas em árvores e em DAG, considerando as ferramentas que utilizaremos para testes basearem-se nestas estruturas. Abaixo segue descrição:

- DAG é um grafo acíclico direcionado, que é formado por nodos (vértices) e arestas dirigidas. Em um DAG, cada nodo representa uma função lógica e cada aresta representa a interconexão. Cada saída é conectada a entrada de outra porta lógica, ou é uma saída primária do circuito.

- A estrutura baseada em árvore (*Tree*) é um caso particular de um DAG, onde nenhum nodo tem mais que uma aresta conectada a sua saída. Sendo assim, cada porta lógica do circuito que tiver sua saída conectada a mais de uma porta é , então, representada por uma árvore. Quando o circuito inteiro passa a ser representado, diz-se, então, ser representado por uma floresta de árvores.

As árvores são estruturas de dados simples, as quais são capazes de representar partes do circuito. Isso faz com que essa estrutura de dados possa não ser tão eficiente, pois podem não alcançar as melhores soluções para o circuito como um todo. Com o DAG, é possível ter uma visão geral do circuito, e isto pode fazer com que resultados melhores sejam obtidos pelos algoritmos de mapeamento tecnológico. Portanto, a escolha da estrutura pode ser decisiva para um melhor resultado do algoritmo de mapeamento. A figura 9 mostra a representação de um circuito (a), por uma floresta de árvores (b) e por um DAG (c).

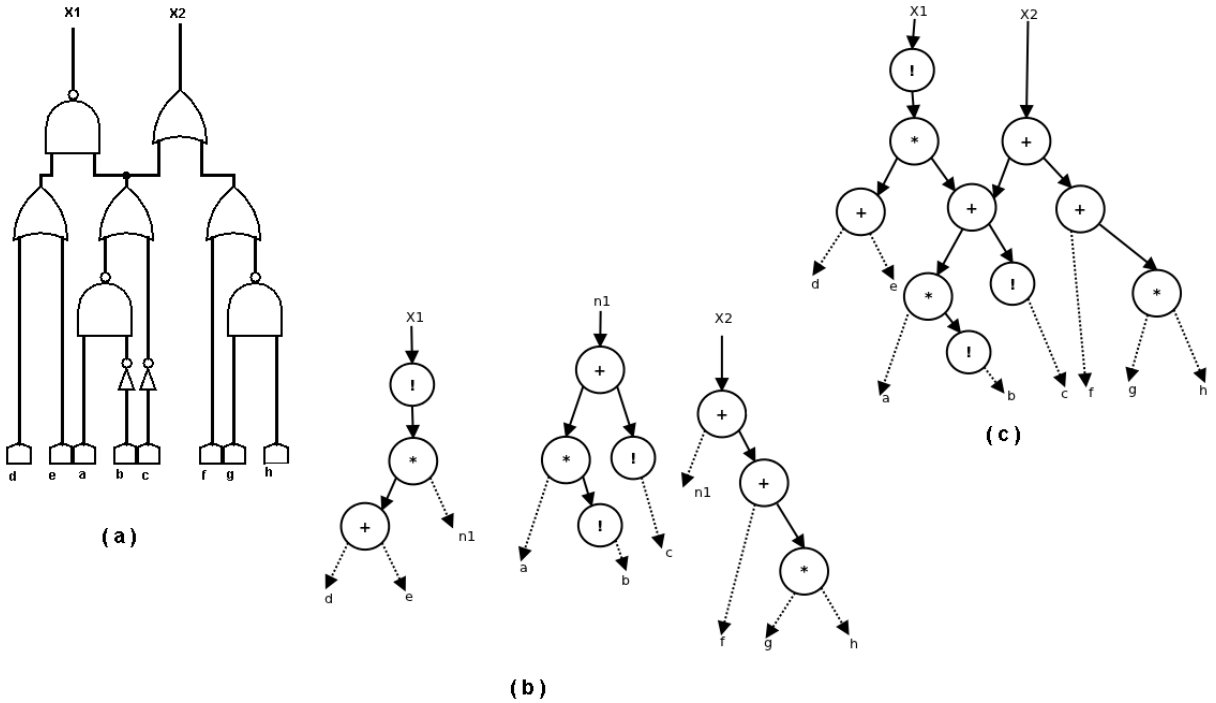


Figura 9: Representações de um circuito lógico(a): representação por floresta de árvores(b) e por DAG (c).

2.2.2 Decomposição lógica

A decomposição lógica é a primeira etapa do mapeamento tecnológico. É onde há a transformação da descrição inicial do circuito, que pode conter portas lógicas complexas, em uma descrição simples usando apenas lógica primitiva. Ou seja, qualquer porta complexa que está contida nesta parte selecionada do circuito é substituída por um conjunto de portas primitivas lógicas: AND, OR e INV, ou apenas AND e INV. Assim, permite-se que o grafo do circuito seja padronizado em um número mínimo de portas.

Esta etapa tem o objetivo de decompor usando as mesmas primitivas em que os padrões da célula na biblioteca estão representados. Quanto maior a decomposição, granularidade, do circuito em questão, com mais facilidade a próxima etapa será executada.

2.2.3 Identificação de padrões

A identificação de padrões, também conhecida como *matching*, toma porções do circuito decomposto e compara com as células existentes na biblioteca, verificando se existe equivalência lógica entre os elementos. O casamento dos padrões se dá quando a equivalência lógica é confirmada. Desta forma, a célula da biblioteca que foi identificada poderá compor a solução final do circuito mapeado.

2.2.4 Cobertura lógica

A cobertura lógica é a última etapa do mapeamento tecnológico. Esta fase tem como objetivo escolher o melhor conjunto de padrões (ou células) dentre os identificados na etapa anterior, de forma que o grafo que representa o circuito

seja completamente coberto, e que as restrições do projeto sejam atendidas. Este processo pode ser executado através de programação dinâmica, dependendo da estrutura de dados utilizada e do critério que se deseja minimizar.

Nesta etapa é onde há o cálculo dos custos, a cobertura é feita de forma a encontrar o menor custo, a menor quantidade e tipos de portas contidas na biblioteca.

A cobertura de um circuito descrito como uma floresta de árvores visando minimizar a área é resolvida em tempo linear com relação ao número de nodos. A cobertura de um DAG é feita com tempo exponencial quanto ao número de nodos. Como tem complexibilidade exponencial, o espaço de soluções de um DAG é maior, então o resultado final tende a ser melhor (MARQUES et al., 2009).

2.3 Biblioteca de células

Uma biblioteca de células pode ser definida por um conjunto finito de células que estão disponíveis para construção de um circuito. Estas células são escolhidas com diferentes topologias e tamanhos, para que consigam representar o maior número de funções possível.

Normalmente, os métodos de mapeamento tecnológico trabalham com bibliotecas estáticas pré-caracterizadas, visando otimizações de área, de atraso, e de potência. A caracterização é feita através de muitas simulações utilizando métodos numéricos complexos. O resultado deste processo é um conjunto de informações precisas sobre o comportamento da célula, sobre o atraso, o consumo de energia e sua área física.

Contudo, o custo de caracterização de uma biblioteca é alto (SECHEN; AL;, 2003). Assim, bibliotecas comerciais são compostas por conjunto com algumas poucas centenas de células. Projetistas de circuitos lógicos ficam, então, restritos a utilizar somente essas células em seus projetos. A figura 10 mostra o fluxo de projeto de circuito usual considerando metodologias de tecnologia de mapeamento com base em bibliotecas com um tamanho fixo.

Bibliotecas de células podem ser divididas em dois grupos: bibliotecas pré-projetadas e bibliotecas geradas sob demanda. As bibliotecas pré-projetadas, ou estáticas, são as mais utilizadas nos métodos de mapeamento. Pois são bem caracterizadas e testadas, visando ter o conhecimento do comportamento esperado de cada célula, o que resulta em maior precisão e eficiência na construção de um circuito.

As bibliotecas sob demanda, chamadas também de virtuais, visam aumentar a diversidade de células existentes na biblioteca, sem ter os custos de pré-caracterização. A disponibilidade das células na biblioteca é definida por um conjunto de restrições, como por exemplo, o número máximo de entradas. Todas as células que se encaixam neste conjunto de restrições podem ser encomendadas quando necessárias. Para utilização deste tipo de biblioteca é indispensável o uso de geradores de portas lógicas e, de estimadores que devem ser rápidos o suficiente para disponibilizar as características da célula gerada, e precisos, pois a qualidade do circuito que está sendo gerado depende diretamente da qualidade das estimativas (MARQUES et al., 2009).

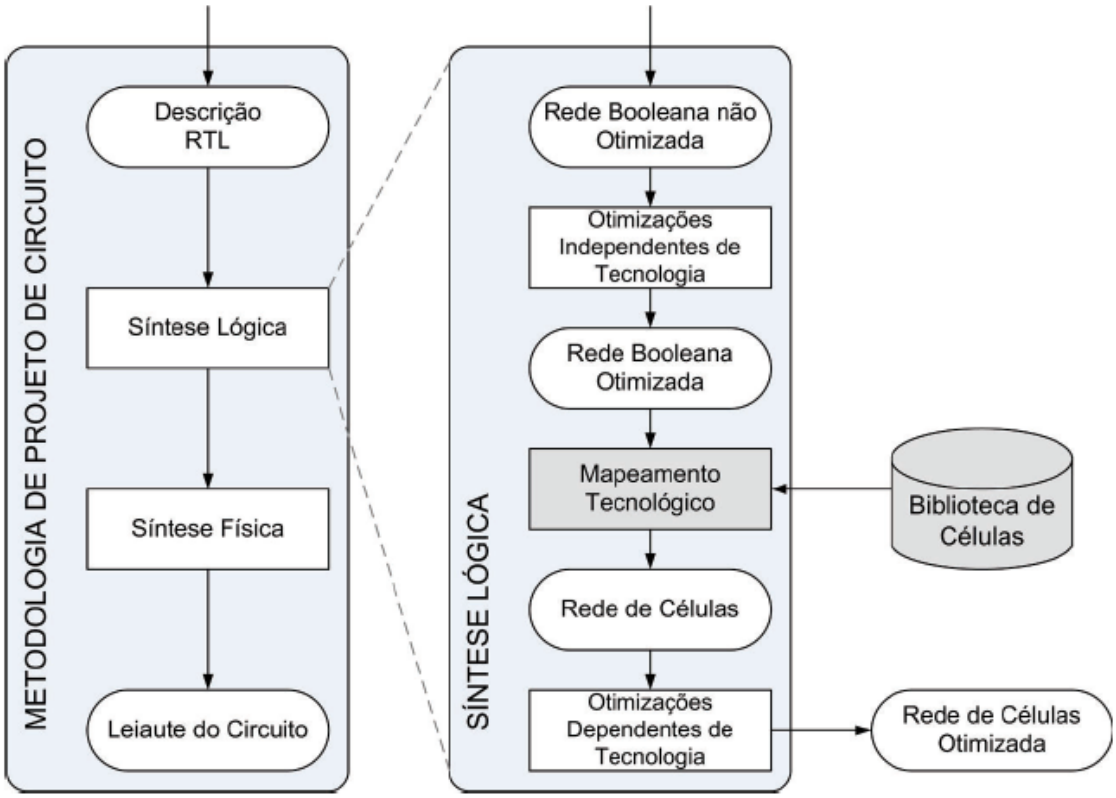


Figura 10: Metodologia de projetos de circuitos digitais (MARQUES, 2008)

3 SÍNTESE LÓGICA QCA

Para o desenvolvimento deste trabalho, alguns trabalhos recentes relacionados à área de síntese lógica aplicada a QCA foram estudados. As seções seguintes apresentam uma breve discussão sobre eles.

3.1 Método manual e 13 funções (ZHANG et al., 2004)

A referência inicial na área de síntese QCA é a abordagem de (ZHANG et al., 2004), que apresenta um método para redução do número de portas majoritárias necessárias para a computação de funções booleanas de 3 entradas, e é desenvolvido para facilitar a conversão de uma expressão SOP em uma célula majoritária QCA.

Em seu trabalho, Zhang desenvolve uma álgebra booleana baseada na interpretação geométrica de funções de 3 variáveis, visando facilitar a conversão de expressões SOP em lógica majoritária reduzida. Primeiramente, a síntese lógica proposta por Zhang, faz as representações gráficas de funções booleanas de 3 variáveis através de cubos.

Para funções booleanas de 3 variáveis binárias, por exemplo, "a", "b" e "c", obtém-se 23 mintermos distintos, onde cada um corresponde a um vértice do 3-cubo. Para 3 variáveis existem 256 funções booleanas. Através da **simplificação exaustiva e manual**, 13 funções são encontradas. Para cada uma dessas funções é apresentada a expressão majoritária reduzida correspondente. A tabela 3 apresenta estas 13 funções com as suas funções majoritárias correspondentes e representação em forma de diagrama.

Para demonstrar o método de redução majoritária proposto, uma aplicação deste é realizada sobre um projeto de somador QCA, demonstrado na figura 11.

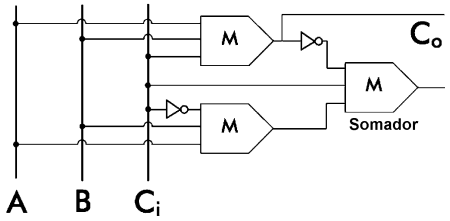
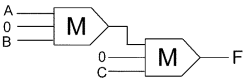
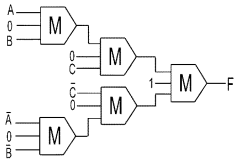
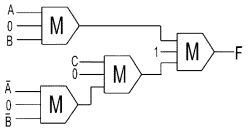
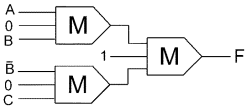
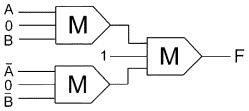
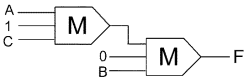
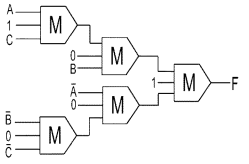
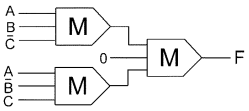
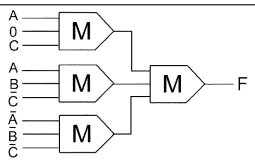
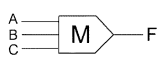
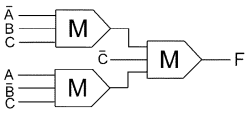


Figura 11: Somador QCA proposto por Zhang (ZHANG et al., 2004)

Este somador QCA de 1 bit consiste de 3 portas majoritárias e 2 inversores. Se comparado com o projeto original proposto por (TOUGAW; LENT, 1994),

Tabela 3: Expressões Majoritárias das 13 funções - (ZHANG et al., 2004)

N° da função	Função Padrão	Expressão Majoritária	Diagrama
1	$F = ABC$	$M(M(A, B, 0), C, 0)$	
2	$F = AB$	$M(A, B, 0)$	
4	$F = ABC + \bar{A}\bar{B}\bar{C}$	$M(M(M(A, B, 0), C, 0),$ $M(M(\bar{A}, \bar{B}, 0), \bar{C}, 0), 1)$	
6	$F = AB + \bar{A}\bar{B}C$	$M(M(A, B, 0),$ $M(M(\bar{A}, \bar{B}, 0), C, 0), 1)$	
8	$F = A$	$M(A, 0, 1)$	
10	$F = AB + \bar{B}C$	$M(M(A, B, 0),$ $M(\bar{B}, C, 0), 1)$	
12	$F = AB + \bar{A}\bar{B}$	$M(M(A, B, 0),$ $M(\bar{A}, \bar{B}, 0), 1)$	
5	$F = AB + BC$	$M(B, M(A, C, 1), 0)$	
11	$F = AB + BC + \bar{A}\bar{B}\bar{C}$	$M(M(B, M(A, C, 1), 0),$ $M(\bar{A}, M(\bar{B}, \bar{C}, 0), 0), 1)$	
3	$F = ABC + A\bar{B}\bar{C}$	$M(M(A, B, \bar{C}),$ $M(A, \bar{B}, C), 0)$	
7	$F = ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$	$M(M(A, C, 0),$ $M(A, B, \bar{C}), M(\bar{A}, \bar{B}, \bar{C}))$	
9	$F = AB + BC + AC$	$M(A, B, C)$	
13	$F = ABC + \bar{A}\bar{B}C + A\bar{B}\bar{C} + \bar{A}B\bar{C}$	$M(M(\bar{A}, B, C),$ $M(A, \bar{B}, C), \bar{C})$	

que utiliza 5 portas majoritárias e 3 inversores, o resultado obtido com relação ao número de portas utilizadas é menor.

O leiaute do somador proposto por Zhang pode ser observado na figura 12, o qual foi projetado e simulado usando o QCA Designer (WALUS et al., 2004). Para o projeto desse somador com células QCA, assumiu-se as mesmas medidas utilizadas por Tougaw, onde as células são separadas por 10 nm e pontos quânticos de 2nm (ZHANG et al., 2004).

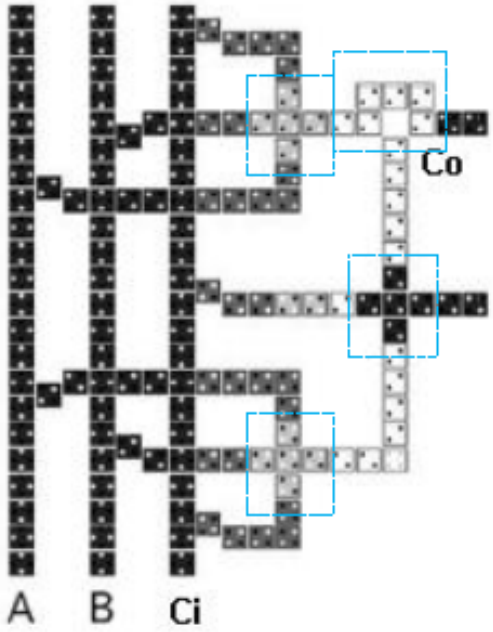


Figura 12: Leiaute de um Somador QCA de 1 bit (ZHANG et al., 2004)

3.2 Porta AOI (MOMENZADEH et al., 2005)

Outro trabalho que apresentou contribuições científicas significativas na área de síntese foi o de (MOMENZADEH et al., 2005). Este trabalho demonstra a necessidade de ferramentas de CAD (*Computer-Aided-Design*) eficientes que utilizem portas majoritárias, MV (*Majority Voter*), durante o mapeamento tecnológico. Para suprir tal necessidade, é proposto um novo método de representação das células QCA, uma porta complexa chamada AOI (*AND-OR-INVERTER*).

A porta AOI é composta de sete células, cinco são entradas, um dispositivo de célula e uma saída. A figura 13 apresenta o projeto de uma porta AOI, por células (a) e o projeto representado por diagrama (b).

Todas as funções lógicas podem ser implementadas através de arranjos AOI (MOMENZADEH et al., 2005). Testes em nível lógico foram abordados e conjuntos de teste apropriados foram desenvolvidos. A AOI forma uma porta lógica universal, pois com ela todas as portas elementares podem ser implementadas.

Segundo Momenzadeh, funções lógicas de dois níveis podem ser diretamente representadas por uma única porta AOI. Diferente de uma porta majoritária convencional, a porta AOI se mostra bastante favorável em termos de síntese lógica digital, e pode ser eficientemente usada pelas ferramentas de síntese existentes.

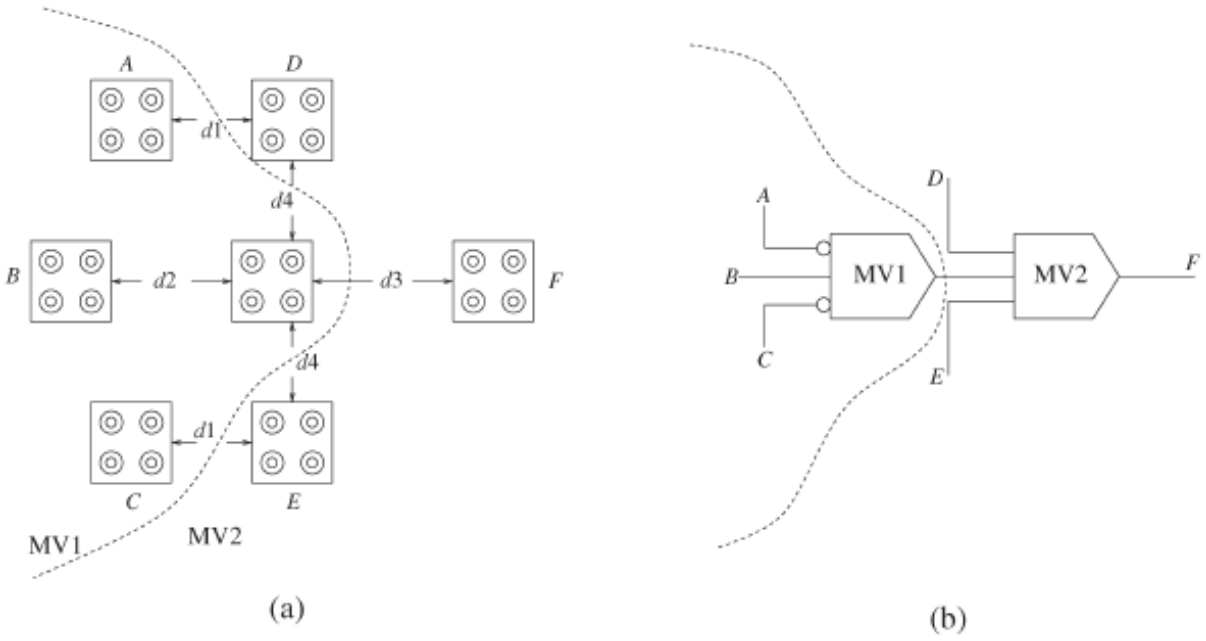


Figura 13: Porta AOI (a)projeto representado por conjunto de células e (b)representação em diagrama (MOMENZADEH et al., 2005)

Na comparação apresentada por Momenzadeh, as propostas com as portas AOI apresentaram bons resultados quando aplicadas a ferramentas de síntese existentes para tecnologia CMOS, ao contrario da MV. No entanto, além de ocupar maior espaço em área do que a porta majoritária, esta porta realiza a lógica irreversível e, portanto, não pode ser a escolha certa para um projeto de energia eficiente (SEN et al., 2011).

3.3 Primeira ferramenta de síntese lógica para QCA

Até este ponto, foi visto que, tratando-se de síntese QCA, nenhuma ferramenta específica foi utilizada nos trabalhos apresentados. Foram utilizados métodos manuais de síntese, que tem como base a tecnologia CMOS, adaptados a QCA.

Em (ZHANG; GUPTA; JHA, 2007) foi apresentada a primeira ferramenta de síntese de rede majoritária, chamada MALS (*Majority Logic Synthesizer*) (GROUPS; PRINCETON UNIVERSITY., 2006).

A metodologia de Zhang, inicialmente define um conjunto de 38 funções majoritárias de três entradas. Um padrão de uma célula é chamado de padrão admissível se este pode ser realizado por uma porta majoritária. Para funções de três entradas é considerado o padrão unate positivo (KOHAVI, 1987). Para chegar ao máximo de funções que podem ser representadas por células majoritárias, foi removida esta restrição da função ser unate positivo, chegando então há um total de 38 funções de três entradas. Com essa informação, chega-se a um número de funções maior do que foi apresentado em (ZHANG et al., 2004).

Uma função é unate positivo se apresentar uma transição positiva (0?1) em sua saída quando a transição positiva acontece em uma das entradas.E uma função unate negativa, se dá quando houver uma transição negativa (1?0) na

saída quando uma transição positiva ocorre em uma das entradas. Uma função é dita ser simplesmente unate se cada uma de suas variáveis aparece, apenas uma vez na equação na forma complementada ou não complementada, senão, é dito ser binate. Quando a função apresenta ambos comportamentos com transições positivas e negativas nas suas saídas sempre que uma transição positiva acontece na entrada, então esta função é dita binate (**?**)

Com as funções definidas, a metodologia de síntese de rede majoritária é demostrada no diagrama da figura 14. Como entrada tem-se uma rede G combinacional minimizada e algebricamente fatorada, e como saída obtêm-se uma rede majoritária Gm funcionalmente equivalente. Um circuito minimizado e algebricamente fatorado é usado como entrada, pois foi a melhor opção encontrada para proporcionar um bom ponto de partida para a síntese de rede majoritária.

Conforme pode-se acompanhar na figura 14, o procedimento inicia com o pré-processamento da rede G. A rede é decomposta de forma que nenhum nodo (n) na rede tenha mais do que três variáveis de entrada. Então, cada nodo na rede decomposta é verificado para determinar se ele é uma função majoritária. Se for, passa-se a sintetizar o próximo nodo. Caso contrário, verifica-se a existência de um literal comum em todos os termos do produto da função do nodo.

Se existe um literal comum, um mapeamento de AND ou OR é então executado no nodo fatorado. Se não existe literal comum, verifica-se se este nodo pode ser implementado com menos de quatro portas AND ou OR. Se este for o caso, é executado o mapeamento com AND ou OR neste nodo. Caso contrário, mapeia-se o nodo em, no máximo, quatro portas majoritárias usando como base o método de K-map. O procedimento termina quando todos os nodos da rede decomposta foram sintetizados.

Os autores utilizaram o método K-map-base para encontrar três funções de um nível majoritário, f1, f2, f3 para cada nodo, de modo que a função do nodo pode ser representada como M (f1, f2, f3).

Nessa metodologia, tenta-se reduzir ”n”, fazendo uso do método de decomposição em síntese interativa sequencial presente no SIS (SENTOVICH et al., 1992), que é dirigido a matriz de portas programáveis (PGA). Existem duas arquiteturas principais PGA: tabelas de pesquisa (LUTs) e multiplexador. O bloco de construção de uma arquitetura LUT implementa qualquer função que tem até ”r” entradas.

Neste trabalho Zhang, restringe ”r” a 3 entradas. Com a ferramenta SIS pode-se fazer qualquer rede de três entradas viável e também minimizar o número total de nodos na rede. Além disso, a síntese majoritária pode ser feita com base nessa rede otimizada decomposta.

Para testes da metodologia e da ferramenta MALS foram utilizados 40 circuitos do *benchmark* MCNC. Todas as redes sintetizadas foram simuladas para verificar a equivalência funcional com a rede original. Resultados experimentais mostraram que a qualidade das redes obtidas foram melhores quando comparadas com às observadas em trabalhos prévios. Um ponto comum entre todos os métodos citados até aqui é o fato que apenas uma de todas as expressões majoritárias admissíveis para uma função booleana pode ser obtida. Portanto, tais métodos não podem garantir que seus resultados são expressões majoritárias mínimas. Expressões mínimas são vitais para circuitos de QCA, porque elas

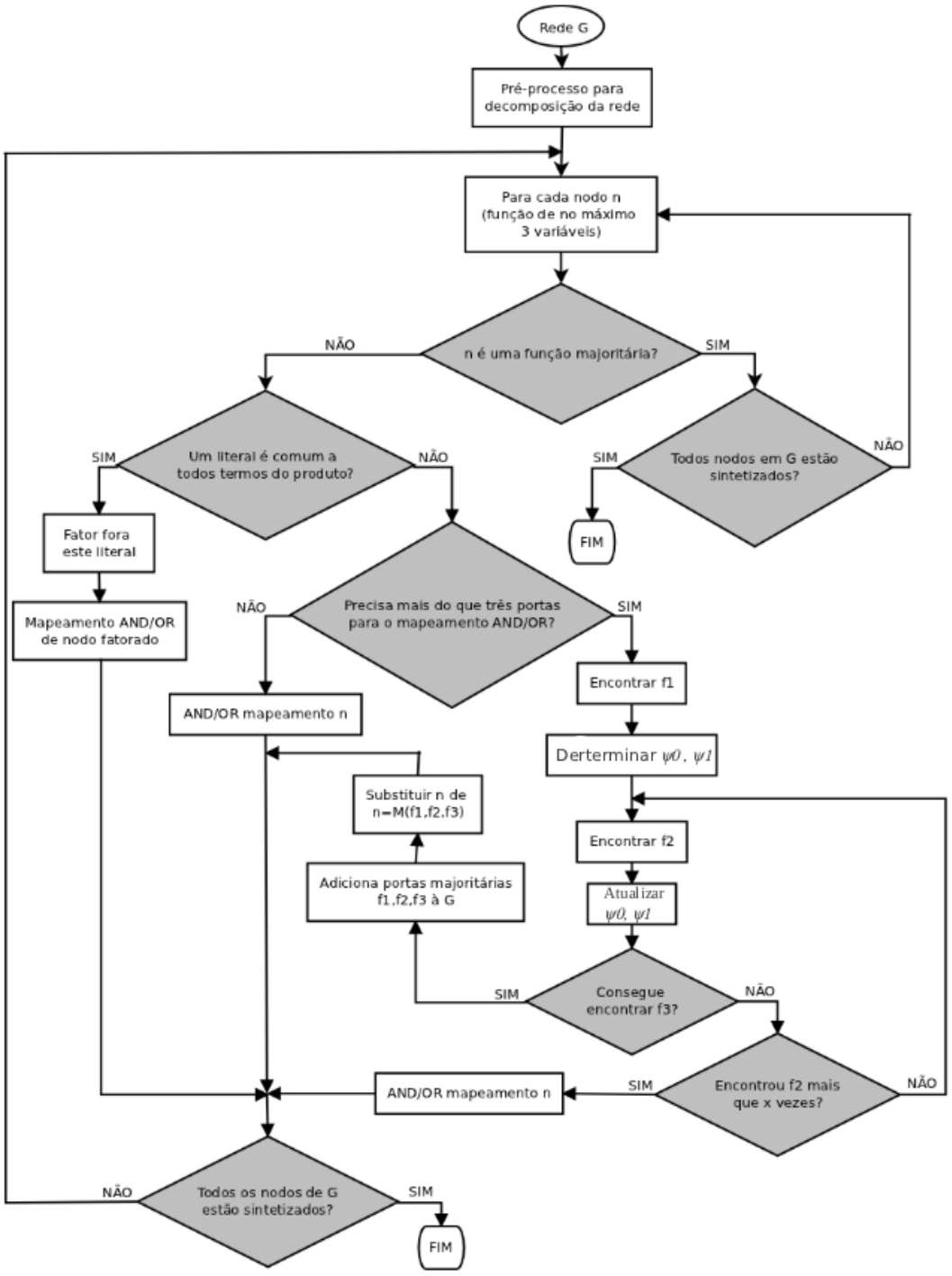


Figura 14: Diagrama com passos da metodologia proposta por Zhang - (ZHANG; GUPTA; JHA, 2007)

podem possibilitar implementações com menor custo.

A metodologia proposta na ferramenta MALS, foi integrada a ferramenta SIS (ZHANG; GUPTA; JHA, 2007).

3.4 Estado da Arte (KONG; SHANG; LU, 2010)

O trabalho mais recente na literatura que trata de síntese lógica para funções de 3 entradas e que também motiva este estudo é o de (KONG; SHANG; LU, 2010). Neste trabalho Kong propõe uma metodologia para síntese lógica majoritária, baseada no que foi proposto por (ZHANG; GUPTA; JHA, 2007).

Kong verificou alguns aspectos que não eram levados em consideração pelos métodos existentes da síntese de lógica majoritária. Com isso ele apontou as seguintes lacunas:

- Não há qualquer trabalho que trate sistematicamente o custo de uma implementação QCA, correspondente à expressão majoritária, que seja capaz de desempenhar a troca de implementação de circuitos QCA;
- Nas metodologias existentes de síntese lógica majoritária, algumas anteriormente já citadas, todas as redes booleanas são decompostas por somente um método de decomposição, o que pode resultar em muitas redes decompostas complexas;
- Quando da conversão de uma rede decomposta para uma majoritária, surgem nodos redundantes, o que pode gerar exigências desnecessárias de hardware.

O diagrama exposto na figura 15 apresenta os passos da metodologia proposta por Kong, onde verifica-se as adaptações feitas no método anteriormente proposto por Zhang.

Como entrada, é inserida uma função booleana F arbitrária de múltiplas saídas. Estas saídas resultam uma rede majoritária G funcionalmente equivalente.

O autor segue 4 passos para obter a solução ótima, que seriam: pré-processamento, decomposição, conversão e remoção. Verificamos no diagrama estes passos e a seguir a descrição de cada passo é apresentada:

- Pré-processamento - Durante o pré-processamento, F é verificada para determinar se está correta. Também verifica-se se esta F é simplificada e fatorada algebricamente, caso contrário, apresenta mensagem de erro e finaliza o processo. Neste passo é onde se dá a detecção de erro. Para fatorar e simplificar é utilizada a ferramenta SIS.
- Decomposição - Neste passo a forma algébrica fatorada é decomposta em redes booleanas. A função F é decomposta em uma rede em que cada nodo tem no máximo três variáveis de entrada, neste caso a rede G é criada.
- Conversão - A rede decomposta é convertida em mínimas expressões majoritárias, este item refere-se à síntese da rede. Para isso, cada nodo da rede G é verificado para determinar se ele representa uma função majoritária. Se for positivo, faz-se a conversão para uma expressão majoritária correspondente diretamente. Caso contrário, faz-se a conversão em uma expressão majoritária mínima utilizando outro método.

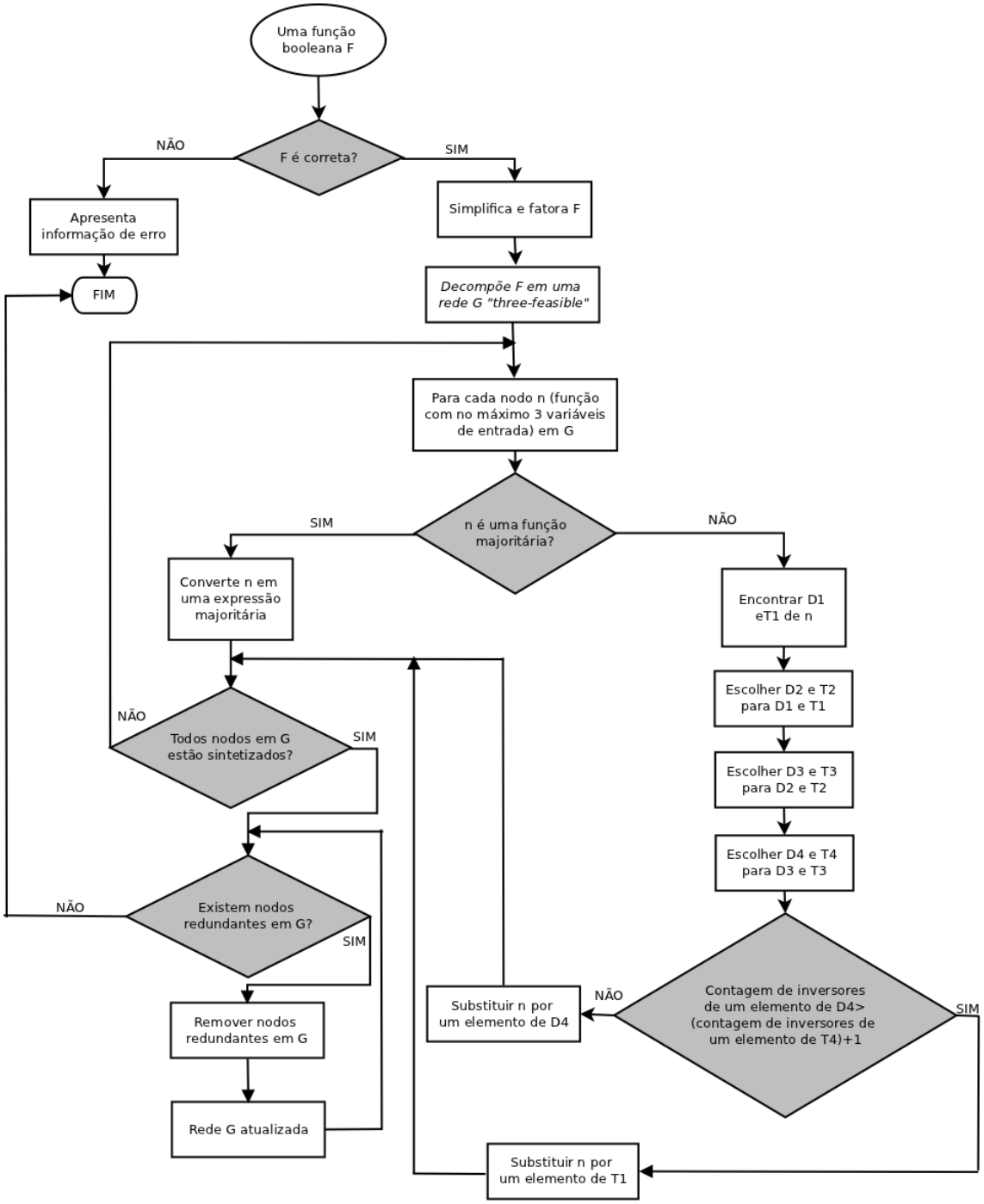


Figura 15: Diagrama com passos da metodologia proposta por Kong (KONG; SHANG; LU, 2010)

- Remoção - Como último passo, depois de todos os nodos da rede G terem sido sintetizados, faz-se a verificação das redundâncias de nodos nesta rede. Se não houver nodos redundantes o processo chega ao final, caso contrário, todas as possíveis redundâncias produzidas na conversão serão removidas e a rede é atualizada, passando novamente pelo processo de verificação de redundância. Esta análise é feita enquanto são encontrados nodos redundantes. No momento em que não se encontra mais nenhum, o processo é finalizado.

Esta metodologia foi desenvolvida a partir de ferramentas de síntese já existentes, como o SIS (SENTOVICH et al., 1998) e o MALS (GROUPS; PRINCETON UNIVERSITY., 2006), no intuito de produzir uma ferramenta de síntese de rede majoritária otimizada. Para testes, os mesmos 40 *benchmarks* do MCNC (YANG, 1991) testados por Zhang foram utilizados. Foi possível verificar a redução na contagem: de nível, de portas majoritárias, de portas de saída e de inversores na rede majoritária, em atraso, em área e no número de células utilizadas na construção do leiaute no circuito.

As adaptações realizadas por Kong no método de Zhang, são principalmente na parte de decomposição.Kong em seu trabalho apresenta 3 diferentes formas de decomposição, algumas ainda manuais, mas a grande diferença para síntese otimizada é quando inserido o método de remoção dos nodos redundantes. Aplicando este passo é notável a redução de área e atraso, conseguindo, assim, atingir redes de células majoritárias mínimas.

3.5 Ferramentas e Algoritmos

Nos trabalhos estudados, descritos nas seções anteriores, percebe-se o uso da ferramenta SIS na maioria das abordagens citadas, visando à síntese lógica dos circuitos QCA. Para uma síntese lógica de qualidade, precisa-se utilizar ferramentas que tornem o trabalho de síntese mais acessível e que retorne resultados satisfatórios.

Conforme descrito nas seções anteriores, para uma síntese lógica otimizada, é importante o mapeamento tecnológico que depende em grande parte da estrutura de dados utilizada para ter sucesso.

Com isso para este trabalho, enfatizam-se duas ferramentas de síntese lógica que são costumeiramente utilizadas para tecnologia CMOS, e que podem ser utilizadas para tecnologia QCA. São elas, a ferramenta SIS e a ferramenta ABC (*A System for Sequential Synthesis and Verification*).

A diferença entre elas é a estrutura utilizada, a ferramenta SIS é baseada em árvores, enquanto a ferramenta ABC em DAG. Segue a descrição de cada uma:

- **SIS** (*System for Sequential Circuits Synthesis*) - é uma ferramenta para síntese e otimização de circuitos combinacionais e sequenciais. Esta ferramenta foi desenvolvida na Universidade de Berkeley na Califórnia e inclui diversos algoritmos de assinalamento de estados, minimização de estados, *retiming*, otimização e decomposição de funções, mapeamento tecnológico, verificação e análise de tempo. A entrada de um circuito no SIS pode ser feita principalmente de duas maneiras: uma descrição *netlist* do circuito e uma descrição de uma máquina de estados finitos. A descrição *netlist* é descrita no formato BLIF (*Berkeley Logic Interchange Format*), que consiste em uma descrição de funções combinacionais com uma saída apenas e latches (SENTOVICH et al., 1992)

- **ABC** (*System for Sequential Synthesis and Verification*) - é um sistema para síntese e verificação de circuitos lógicos binários sequenciais. A ferramenta ABC reúne otimização lógica baseada em AIGs, tecnologia de mapeamento DAG e algoritmos dedicados à síntese e verificações sequenciais. Além de fornecer uma implementação experimental destes algoritmos

> e um ambiente de programação para a construção de aplicações similares. A entrada de um circuito lógico no ABC é uma descrição *netlist* do circuito. A descrição *netlist* é apresentada no formato BLIF, da mesma forma que para o SIS. Além disso, o ABC foca na melhoria dos algoritmos previamente descritos no SIS e possibilita a utilização de seus recursos de uma maneira mais independente. Isso permite que o usuário personalize o ABC para as suas necessidades como se fosse uma caixa de ferramentas ao invés de uma ferramenta completa (SYNTHESIS; GROUP, 2011).

A ferramenta ABC permite a realização de *matching* booleano. A decomposição (*matching*) é feita através de algoritmos de *K-cuts*. Estes algoritmos permitem o particionamento do grafo em subgrafos a partir das entradas, possibilitando, assim, uma maior cobertura dos nodos, avaliando o circuito como um todo, o que pode resultar em uma melhor otimização.

Por fim, é importante mencionar a ferramenta QCADesigner (WALUS et al., 2004), que é o estado da arte em edição e simulação de leiautes de circuitos. Praticamente todas as contribuições de autores listadas nesse trabalho fizeram uso dessa ferramenta. Porém, esta não é capaz de realizar mapeamento tecnológico. Com esta ferramenta é possível apenas desenhar, obter estimativas de área e atraso, e verificar o comportamento lógico das redes QCA. Além desta ferramenta, em 2007 foi proposta por (TEODOSIO; SOUSA, 2007) uma ferramenta que gera automaticamente um leiaute de pontos quânticos para um determinado circuito combinacional, em um formato compatível com a ferramenta QCADesigner. Mas, esta ferramenta auxiliar também não faz a síntese lógica dos circuitos.

4 METODOLOGIA PARA O DESENVOLVIMENTO DO TRABALHO

Com o estudo da tecnologia QCA ainda em desenvolvimento, os resultados de síntese lógica QCA apresentados, até então, são obtidos com mapeadores voltados a tecnologia CMOS (MOMENZADEH et al., 2005). Embora seja possível mapear circuitos QCA com essas ferramentas, pois as duas tecnologias consideram e manipulam elementos que representam funções booleanas, os resultados obtidos podem não ser soluções eficientes. Em muito casos, os resultados alcançados com estas ferramentas, não chegam a ser satisfatórios, pois uma única célula QCA pode representar mais de uma função lógica. Como na tecnologia CMOS isso não acontece, as ferramentas não são adaptadas com algoritmos de mapeamento que considerem outras possíveis soluções (combinações) alternativas ao fazer a associação de células durante o processo.

A etapa de decomposição funcional, nos trabalhos anteriormente citados, é realizada pela ferramenta SIS (SENTOVICH et al., 1992) e MALS (GROUPS; PRINCETON UNIVERSITY., 2006). Entretanto, segundo (MOMENZADEH et al., 2005), estas ferramentas não fazem bom uso das células majoritárias durante o mapeamento e trabalham com bibliotecas bastante restritas.

No trabalho proposto por (KONG; SHANG; LU, 2010), é apresentado uma técnica de síntese partindo da geração de células otimizadas e remoção de nodos redundantes. Contudo, mesmo propondo uma nova metodologia, o autor ainda se utiliza da ferramenta SIS, para realizar algumas etapas do processo de otimização.

Como a tecnologia QCA ainda é relativamente nova, não contando com um fluxo de mapeamento próprio para a construção de seus circuitos, este trabalho apresenta resultados de testes utilizando a ferramenta ABC, a qual possui um método de verificação diferente da SIS, por ter como base a estrutura DAG, visando possibilitar uma cobertura mais ampla no mapeamento do circuito. Esta é a principal diferença e contribuição deste trabalho, visto que nos trabalhos relacionados (e estado da arte) somente a ferramenta SIS tem sido utilizada.

Para este trabalho, além da utilização de uma ferramenta com outra metodologia de mapeamento, foram descritas 4 bibliotecas de células, que receberam os seguintes nomes: (i) Básica, (ii) Zhang 2004, (iii) Kong 2010 e (iv) P-class. Estas bibliotecas foram descritas no formato genlib, devido este ser o formato aceito pela ferramenta ABC, e são apresentadas no Anexo A deste trabalho.

A biblioteca Básica é composta apenas de CONST0, CONST1, AND, inversor e porta majoritária. A segunda biblioteca é composta pelas 13 funções da classe

NPN propostas por (ZHANG et al., 2004), inversor, CONST0 e CONST1. A terceira biblioteca é baseada nas 40 funções propostas por (KONG; SHANG; LU, 2010), sendo, estas, funções da classe PN. A última biblioteca gerada foi a P-class que contém todas as funções de 3 entradas da classe P, contabilizando um total de 80 funções. As três primeiras bibliotecas foram apenas descritas para o formato genlib, e eram baseadas no apresentado na literatura, para o caso da última bibliteca, P-class, esta foi gerada e desenvolvida para utilização nos testes.

As funções de "n" entradas podem ser classificadas em conjuntos para reduzir o espaço de busca. Esta classificação é feita em classes conhecidas como P, PN, NP e NPN. Passando a identificar as funções das bibliotecas por suas classes de equivalência, então passamos a denominar estas por suas classes: a primeira biblioteca continua recebendo o nome (i) Básica, por conter o mínimo de células exigido pela ferramenta para que um mapeamento possa ser realizado; a segunda passou a ser chamada (ii) NPN-class, a terceira (iii) PN-class e a última (iv) P-class. O grupo de funções da classe P são funções, que permanecem equivalentes após realizar-se a permutação das entradas. O conjunto PN é composto de funções as quais permanecem equivalentes com a permutação de entradas e negação das saídas. O menor conjunto é o NPN, ou seja, a menor biblioteca, devido a permutação e/ou negação das entradas e/ou negação da saída (CORREIA; REIS, 2001).

A medida que as bibliotecas foram sendo definidas, testes foram efetuados, e os resultados foram comparados.

4.1 Experimentos Realizados

No primeiro momento do trabalho, realizou-se uma pesquisa bibliográfica na literatura sobre síntese lógica para circuitos com células QCA, no intuito de verificar as lacunas existentes. Os métodos que foram estudados apresentam diferentes técnicas, mas todos visam a síntese de circuitos.

Após o estudo bibliográfico, foram feitos testes de ambientação com a ferramenta ABC. Diferentes algoritmos de mapeamento para FPGA foram testados com circuitos *benchmark* MCNC ISCAS'85. Estes algoritmos para mapeamento tecnológico são disponibilizados no ABC. São eles: *if, fpga, ffpga, lut,* e *imfs*.

Um FPGA é um circuito pré -fabricado, que pode ser programado para implementar a funcionalidade desejada. Nesses circuitos, as funções são implementadas por LUTs que, em sua essência, são tabelas verdade programáveis. Elas podem computar qualquer função booleana de "n" variáveis, sendo que estas correspondem às entradas da LUT. Assim, a restrição que define a biblioteca é o número de entradas de cada LUT (ou célula lógica). Como o circuito já existe fisicamente, dados como área, atraso e consumo já são previamente conhecidos, restando ao mapeador a tarefa de respeitar restrições de desempenho, minimizando o número necessário de LUTs.

Os algoritmos de mapeamento FPGA são limitados pelo mapeamento de LUT de tamanho variável, onde a LUT de cada tamanho é caracterizada por área e atraso. Tipicamente, o maior tamanho permitido de uma LUT é de 6 entradas, e o mínimo de 3 entradas. A ferramenta mapeia o circuito para atingir atraso ótimo utilizando algoritmos clássicos para o mapeamento, seguido por

uma recuperação de área heurística.

Os testes foram realizados primeiramente com foco em FPGA para funções de 3 entradas. Como a célula majoritária é definida por 3 entradas, com o estudo feito na literatura percebeu-se que os autores tentaram aplicar decomposição funcional considerando funções de 3 entradas e baseado nisso era feita a síntese. Kong em seu método utiliza-se de decomposição funcional, buscando ”quebrar”a descrição em várias células de 3 entradas, de forma que estas possam ser casadas com a lista de 40 funções que podem ser implementadas por uma célula majoritária. Os métodos de FPGA tendem a fazer uma síntese voltados a isso, porque definimos um $"K"$(número de variáveis de uma LUT) e o método tenta quebrar o circuito de forma que use o menor número de LUTs possível.

Pensou-se aplicar este método com foco em FPGA, buscando relacionar a técnica de quebrar em várias LUT's de 3 variáveis na tecnologia QCA, pensando em construir células numa estrutura de 3 variáveis.

Para realizar os experimentos com os *benchmarks* MCNC ISCAS'85 na ferramenta ABC, foi criado um *script* para o carregamento, execução de mapeamento e impressão de resultados dos circuitos mapeados. A figura 16 apresenta os *scripts* utilizados durante a execução do mapeamento.

Para obtermos resultados os seguintes algoritmos de tecnologia de mapeamento foram utilizados:

- ***fpga*** (I) - Realiza mapeamento FPGA usando a biblioteca. Se o circuito a ser mapeado consiste em uma descrição AIG, o mapeamento é realizado diretamente. Caso o circuito esteja descrito como uma rede lógica, o algoritmo transforma a descrição para o formato AIG rede lógica, antes de mapear (MISHCHENKO et al., 2005).

- ***ffpga*** (II) - Consiste em uma implementação simples de mapeamento onde são introduzidos um número maior de cortes, mas com tamanhos menores. Aumentando o número de cortes é possível obter uma melhoria no tempo de execução do algoritmo e oferecer melhorias na utilização de memória (CHO H.; SWARTZLANDER, 2007). Em termos de mapeamento, o uso de cortes menores faz com que o resultado do mapeamento possa ser alterado em virtude das equivalências possíveis entre as porções do circuito e as células disponíveis na biblioteca.

- ***if*** (III) - É um algoritmo de mapeamento para a tecnologia FPGA com LUTs de K-entradas, aplicável a grandes circuitos combinacionais e sequenciais. O algoritmo evita a enumeração de excessivos cortes para cada nodo da rede (MISHCHENKO et al., 2005).

- ***imfs*** (IV) - É um mecanismo de ressíntese em área orientada para a rede mapeada em K-LUTs, com base na alteração de uma função lógica em um nodo por extração de *”don't cares”* a partir de uma janela e usando re-substituição booleana para reescrever a função do nodo usando possivelmente novas entradas (MISHCHENKO et al., 2005).

- ***lutpack*** (V) - Também é um mecanismo de ressíntese em área orientada para a rede mapeada em K-LUTs. É um algoritmo rápido para a

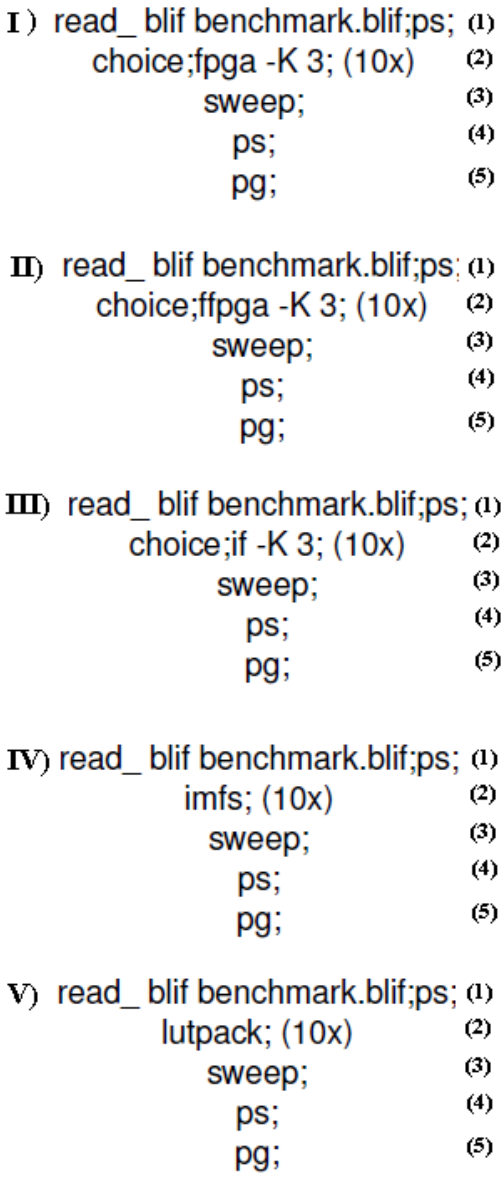


Figura 16: Scripts de comandos para mapeamento em FPGA na ferramenta ABC

decomposição de funções lógicas. O cálculo é efetuado de forma mais rápida do que nos algoritmos anteriores, contando com BDDs para isto (MISHCHENKO et al., 2005).

- ***map*** (fig. 17) - Realiza mapeamento de células padrão (*standard cell*) da rede usando a biblioteca carregada. Se a rede é um AIG então ela é utilizada diretamente para a realização do mapeamento. Se a rede a ser mapeada é apresentada como uma rede lógica, então ela é convertida para uma descrição AIG antes da realização do mapeamento tecnológico (CHATTERJEE et al., 2005).

Alguns comandos básicos dentro da ferramenta ABC, descritos na figura 16, foram utilizados:

O comando *read_ blif*(linha 1) é responsável por carregar o circuito a ser mapeado;

O comando *choice*(linha 2) combina escolhas estruturais derivadas de três versões diferentes da rede atual (a rede original e duas da rede funcionalmente equivalentes). Através dele, pode-se melhorar a área e atraso da rede resultante. Com o comando de mapeamento, seguido pelo *flag* -K, determina-se o número de entradas a serem consideradas na função. Este comando é executado 10x, devido a termos variações de valores no mapeamento. Estas 10x são o suficiente para o comando ser executado, após isso não há mais variações. Nos exemplos da figura 15 limitou-se a funções de 3 entradas;

O comando *sweep*(linha 3) executa uma varredura com as seguintes tarefas: remover nodos redundantes, recolher inversores e *buffers* em suas saídas, e remover entradas duplicadas.

Por fim, os comandos *ps* e *pg* (linhas 4 e 5, respectivamente) demonstram os números que representam o circuito antes e depois do mapeamento.

Ao finalizar os testes verificou-se que os resultados alcançados não foram nada bons, pois quando é aplicada uma inversão para FPGA o custo é zero, e em QCA isto não acontece. Ao contrário, do FPGA, uma inversão tem o custo mais alto se comparado a porta majoritária.

É importante salientar que a porta majoritária e o inversor QCA receberam, como custo atribuído, os valores 3 e 5, respectivamente. Estes custos estão relacionados com a área de tais dispositivos, tendo como base o número de células quânticas utilizadas para sua construção. A porta majoritária é formada por 5 células quânticas, ocupando uma área de 3x3 unidades. Desta forma, o custo 3 em área foi atribuído à porta majoritária. O custo do inversor foi definido como 5 para respeitar a relação de 1,6 vezes, conforme especificado por (MOMENZADEH et al., 2005). Estes custos não representam a área efetiva, é somente um dado quantitativo. Para os resultados apresentados nas tabelas a seguir, são considerados apenas custos de porta majoritárias e inversores, ou seja, não são levados em consideração demais conexões (fios) no circuito.

Realizado o estudo preliminar da ferramenta e a avaliação do comportamento com os algoritmos FPGA, a próxima etapa do trabalho consistiu na geração da biblioteca básica. Esta biblioteca foi desenvolvida manualmente, pois até então não havia a disponibilidade de uma ferramenta que fizesse esta etapa de forma automatizada.

A biblioteca básica, foi descrita em formato genlib e composta de constantes ”0” e ”1”, porta AND2 e INV. Na sequência, o mapeamento foi realizado na ferramenta ABC. Como pode ser observado no *script* apresentado pela Figura 17, o mapeamento foi realizado através do comando *map*.

1) $read\_library \quad nomebib.genlib$
$read\_blif \quad benchmark.blif$
$map; \quad (10x)$
$ps;$
$pg;$

Figura 17: Scripts de comandos para mapeamento com biblioteca básica

Na sequência do trabalho, a biblioteca básica foi expandida com as 13

funções propostas por (ZHANG et al., 2004). Neste trabalho, essa biblioteca recebeu o nome de *NPN-class*. De posse desta biblioteca o algoritmo *map* foi rodado na ferramenta ABC.

Após realização dos testes e coleta de resultados com a biblioteca NPN, uma nova biblioteca foi gerada de forma manual, contendo constantes $"0"$ e $"1"$, inversor e as 40 funções apresentadas por (KONG; SHANG; LU, 2010). A esta biblioteca foi atribuído o nome de *PN-class*, por conter funções de classe PN. Com a biblioteca carregada na ferramenta, os testes foram realizados, buscando-se obter resultados melhores daqueles encontrados por Kong. Esses resultados também são apresentados na tabela 5. O comando utilizado para o mapeamento com as bibliotecas NPN e PN é descrito no *script* da figura 18.

$$\begin{array}{l} 1) \ read\_library \quad nomebib.genlib \\ read\_blif \quad benchmark.blif \\ choice; map; \quad (10x) \\ ps; \\ pg; \end{array}$$

Figura 18: Scripts de comandos para mapeamento com bibliotecas PN e NPN

Considerando as bibliotecas QCA utilizadas, assim como acontece na tecnologia CMOS, os resultados demonstraram que quanto maior for o número de funções em uma biblioteca de células melhor é o resultado do mapeamento, obtendo-se circuitos com uso de menor número de células. Isso resulta na diminuição da área do circuito mapeado.

Buscando uma redução mais expressiva no número de células utilizadas no projeto de um circuito QCA, procurou-se gerar uma biblioteca composta por um número maior de funções. Assim, o passo seguinte consistiu na geração de uma biblioteca QCA de classe P. Entretanto, a geração de uma biblioteca com maior número de células não é uma tarefa trivial. Na tabela 4 é possível verificar a complexidade de construção de bibliotecas com três e mais entradas. O número de funções cresce exponencialmente de acordo com o número de entradas. Nota-se este comportamento tanto na biblioteca mais simples, a NPN, onde para funções de 4 entradas precisaríamos de 222 funções, quanto na biblioteca P, onde precisaríamos de 3984 funções, dificultando sua construção manual.

Tabela 4: Número de funções com 3, 4 e *n* variáveis

Número de entradas	**Número de funções**	**Classe P**	**Classe NPN**
2	16	12	4
3	256	80	14
4	65536	3984	222
n	2^{2^n}		

Assim, ficou evidente a necessidade de uma ferramenta capaz de gerar as bibliotecas QCA de forma automatizada. A geração automatizada de bibliotecas

necessita ser realizada por técnicas capazes de otimizar ao máximo as funções a serem utilizadas. Algoritmos eficientes necessitam ser empregados, de forma a possibilitar que a biblioteca alcance o menor custo possível, permitindo, assim, que o circuito seja minimizado durante a realização do mapeamento tecnológico.

Nesse sentido, passamos a utilizar um gerador de biblioteca automática, chamado Composição Funcional (FC) (MARTINS; RIBAS; REIS, 2012), desenvolvido pelo grupo de pesquisa LOGICS da UFRGS (LOGIC CIRCUITS, 2013).

A partir do uso dessa ferramenta foi possível gerar bibliotecas maiores, compostas por um conjunto maior de funções.

Este gerador de bibliotecas foi de grande importância para a execução deste trabalho, pois a geração manual de 80 funções, como no caso da classe P, é uma tarefa bastante custosa. A fim de verificar o impacto de uma biblioteca maior, foram realizados novos testes na ferramenta ABC, utilizando o *script* descrito na figura 19.

$$
\begin{aligned}
&\text{1)}\ read\_blif \quad benchmark.blif \\
&choice; map \quad -a; (10x) \\
&short\_names; \\
&write\_eqn \quad benchmark\_map.eqn
\end{aligned}
$$

Figura 19: Scripts de comandos para mapeamento com biblioteca classe P

A partir da execução deste *script*, além do mapeamento realizado, obtivemos a descrição dos circuitos no formato *eqn*. Através dessa descrição foi possível analisar visualmente o resultado do mapeamento. Para isso foi utilizada uma ferramenta, ainda em desenvolvimento, pelo grupo de pequisa GACI da UFPEL.

No capítulo a seguir os resultados obtidos são apresentados e discutidos.

5 ANÁLISE DE RESULTADOS

Este capítulo apresenta uma discussão em relação aos resultados obtidos com os experimentos realizados.

A tabela 5 apresenta os resultados obtidos para as bibliotecas QCA de 3 entradas. A primeira coluna da tabela 5 descreve os circuitos pertencentes ao *benchmark* MCNC ISCAS'85. As colunas seguintes apresentam os resultados obtidos ao utilizar as bibliotecas P, PN, NP e NPN. Por fim, a última coluna apresenta os resultados por Kong (KONG; SHANG; LU, 2010) em sua metodologia de síntese.

Analisando os resultados dispostos na tabela 5, inicialmente é interessante notar, o somatório total dos custos obtidos pelos mapeamentos. É possível perceber que a maior biblioteca, na maioria das vezes, possibilita um melhor resultado no mapeamento, com custo total menor para os circuitos mapeados.

O principal motivo para a redução dos custos dos circuitos mapeados se deve a diminuição do uso de inversores no circuito. Foi possível observar que através do uso de bibliotecas maiores, o número de inversores necessários para compor os circuitos foi diminuído. Como o custo de inversores na tecnologia QCA é considerável (custo 5), essa redução é refletida nos custos dos circuitos mapeados.

Os circuitos mapeados com a biblioteca NPN apresentaram custos mais elevados. Isto era esperado por se tratar de uma biblioteca com poucas funções. Em alguns poucos casos aumentando o número de funções presentes nas bibliotecas, percebe-se o aumento do custo do circuito mapeado. Pode-se sentir falta dos resultados da biblioteca básica, comentada no inicio do trabalho desenvolvido, estes não são apresentados nesta tabela devido a serem piores dos que o da biblioteca NPN (GONçALVES et al., 2012).

Na comparação entre a coluna NPN e a coluna NP é possível verificar que os circuitos *cht, cm152a, cm82a* e o *z4ml* apresentam custos iguais para as duas diferentes bibliotecas. Este comportamento também é observado na comparação das colunas NP e PN para os circuitos *cm82a, unreg* e o *z4ml*. Isto acontece em virtude dos critérios utilizados pela ferramenta para a realização do mapeamento:

- Profundidade
- Número de portas majoritárias
- Número de interconexões e inversores

Um exemplo que comprova o descrito acima é o apresentado na figura 20, onde temos representado parte de um circuito, em que cada caixa representa

Tabela 5: Tabela com resultados dos testes feitos com diferentes bibliotecas na ferramenta ABC

Bench. ISCAS'85	***NPN-CLASS***	***NP-CLASS***	***PN-CLASS***	***P-CLASS***	**Método de Kong**
9symml	1379	1297	1289	907	256
alu2	2356	2306	2244	1727	1555
apex6	5109	4429	4266	2576	2776
cht	1004	1004	999	814	405
cm150a	338	303	298	218	238
cm151a	151	161	141	111	119
cm152a	138	138	133	98	98
cm162a	267	250	216	167	193
cm163a	276	229	218	186	199
cm42a	230	158	145	99	84
cm82a	79	79	79	54	51
cm85a	267	237	194	179	78
cmb	331	263	228	153	104
cu	358	314	303	229	225
decod	267	227	231	144	114
frg1	389	326	318	168	620
i2	1728	1510	1410	640	627
k2	10453	8182	7790	3733	4998
ldd	620	534	488	274	321
majority	63	53	43	18	18
mux	358	308	283	213	198
pcler8	422	363	355	189	315
pcle	618	523	463	234	271
pm1	324	249	239	181	185
term1	1122	1022	904	543	568
ttt2	1032	991	906	778	695
unreg	826	551	551	531	417
vda	5261	4022	3773	1877	2810
x2	328	276	243	226	186
z4ml	116	116	116	105	57
TOTAL	36390	30421	28866	17372	18781

uma porta majoritária, de valor 3, e o inversor representado com valor 5. Se analisarmos cada célula individualmente, na figura são identificadas pelos quadrados pontilhados, calcula-se o custo para cada uma delas. As células que tem o inversor possuem custo 8 e as demais custo 3. Se o *matching* é feito individualmente, têm-se o custo de 28 (2x8+4x3). Considerando esta parte do circuito como uma única célula complexa, demarcado pelo pontilhado externo, chegamos ao mesmo custo de valor 28. Então, independentemente do circuito ser mapeado com 6 células simples ou com uma única célula o custo de mapeamento alcançado é o mesmo.

Constata-se que o custo final do circuito não se altera, independentemente do *matching* realizado pela ferramenta de mapeamento ter considerado porções

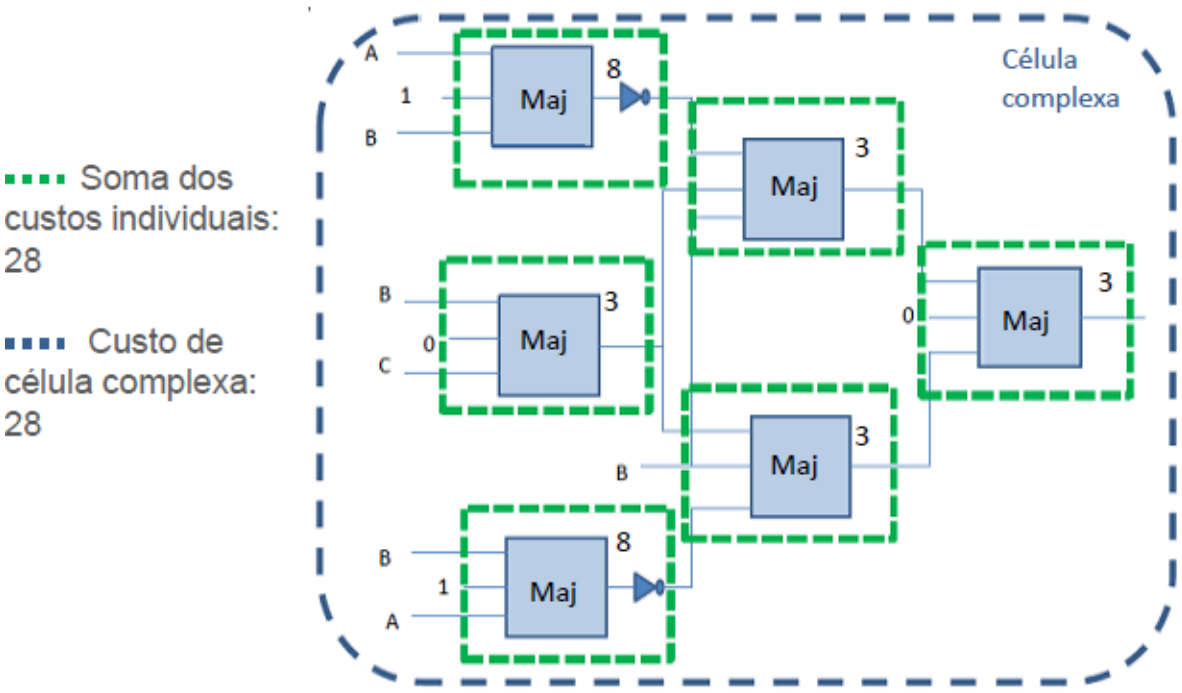


Figura 20: Exemplo de critérios básicos de síntese

individuais ou uma célula maior e mais complexa. Se atribuíssemos valores a cada célula que compõe um fio, cairíamos na mesma situação de que dados quantitativos não diferenciam células simples de complexas.

Outro comportamento interessante pode ser verificado para os circuitos *cm151a*, na comparação entre as bibliotecas NPN e NP, e o circuito *decod*, na comparação entre as bibliotecas NP e PN. Para estes casos, os custos de mapeamento obtidos são menores quando a biblioteca menor é utilizada, divergindo do que havia sido observado e do resultado esperado.

No caso desses circuitos uma análise cuidadosa foi realizada para verificar o motivo desse comportamento. Para tanto, dividimos um desses circuitos a partir das saídas e, na sequência, mapeamos cada saída individualmente a fim de verificar aonde estava ocorrendo a discrepância.

Analisando as saídas individualmente, os resultados do mapeamento apresentaram-se na forma esperada, ou seja com a biblioteca maior fornecendo custos menores para os mapeamentos. Isso não acontece se considerarmos o circuito completo, devido a ferramenta trabalhar com algoritmos heurísticos que tem por base outro foco de otimização que não é a área.

Esses algoritmos heurísticos fazem com que as escolhas realizadas possam fazer, ou não, o compartilhamento de inversores, por exemplo. Se um inversor é compartilhado, reduz-se o custo de um inversor no circuito. Por outro lado, se ele é utilizado individual em cada corte menor, aumenta-se o custo no mapeamento. Isso é ilustrado na figura 21.

Por exemplo, observando a figura 21, o inversor que está ligado à entrada A poderia ser compartilhado pelas duas células que a utilizam. Entretanto, no exemplo ilustrado na figura, isso não ocorre. Assim, o custo total desta parte do circuito totaliza em 24. Se o algoritmo conseguisse identificar esse comparti-

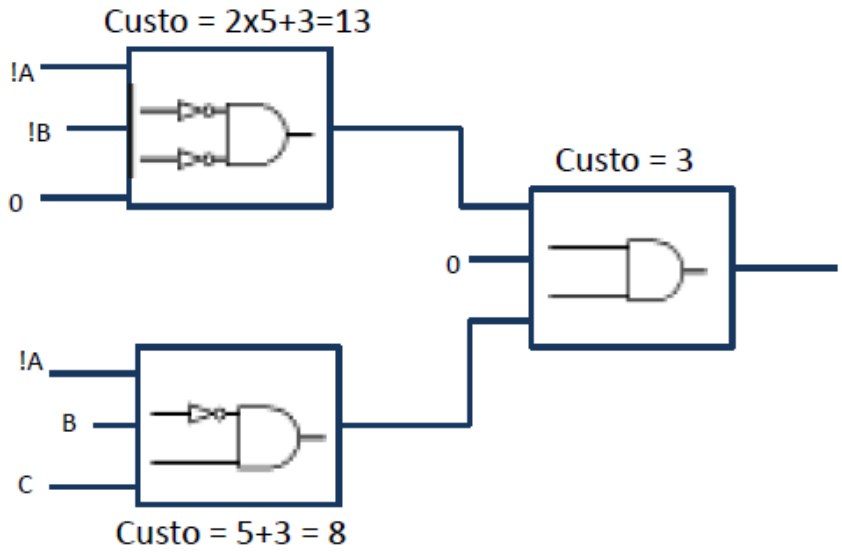


Figura 21: Compartilhamento de inversores

lhamento, seria possível reduzir o custo de um inversor, passando então para o custo total de 19.

Outra possibilidade que permite esse comportamento inesperado no mapeamento é ilustrado na figura 22.

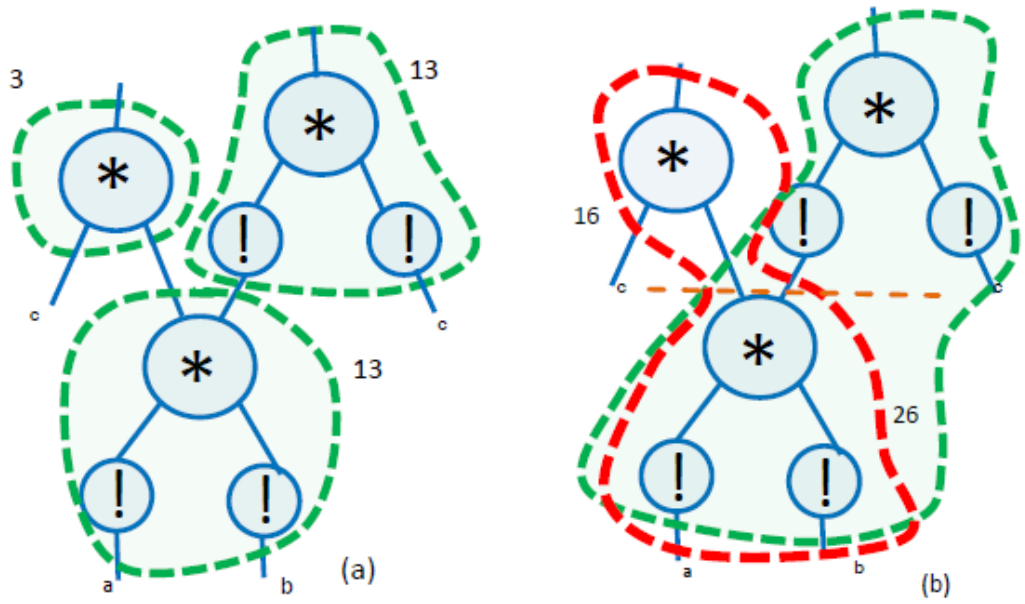


Figura 22: Cortes de células simples e complexas

Na figura 22.a , temos a representação de possíveis cortes, considerando uma biblioteca menor com células simples Baseados nestes cortes o custo obtido é de 29. Na figura 22.b, temos outros possíveis cortes, onde o custo total de cobertura em 42. Como pode ser visto, há aumento de custo com a biblioteca

maior, pois há compartilhamento de células. Isso é representado no traço pontilhado da figura 22.b, o que faz com que o custo dessa célula seja duplicado, pois dois cortes utilizam a mesma lógica. Isso justifica os resultados obtidos para os circuitos mapeados.

Uma outra análise importante nos resultados consiste na comparação dos resultados obtidos com a biblioteca P e os resultados de Kong, o qual é considerado o estado da arte ma síntese lógica para QCA. Embora parte dos circuitos mapeados com a ferramenta ABC tenham alcançado bons resultados, em alguns casos o método de Kong atingiu resultados melhores. Isto ocorre devido a estes circuitos terem uma estrutura que facilita a síntese pelo método de Kong.

O circuito onde isto ocorre de forma mais expressiva é o **9symml**, onde o método de Kong é capaz de entregar um circuito com custo 70% menor que o obtido pelo mapeamento com a biblioteca P. Embora não tenha sido possível apontar os motivos que levaram a este resultado, os resultados demonstram que existe uma ineficiência por parte dos algoritmos de mapeamento presentes no ABC. Isso aponta para a possibilidade de melhorias nos métodos de mapeamento dedicados para a tecnologia QCA.

Experiências similares realizadas pelo grupo LOGICS da UFRGS (MARTINS et al., 2013), utilizando-se ferramentas comerciais, da mesma forma que a ferramenta acadêmica, também evidenciaram que melhoras ainda podem ser feitas para mapear circuitos QCA como maior eficiência. No sentido de avançar na análise do mapeamento de circuitos com portas lógicas QCA, investigou-se, também, o comportamento ao se utilizar bibliotecas com funções de até 4 entradas. Da mesma forma que a biblioteca P de 3 entradas, as bibliotecas de 4 entradas foram geradas automaticamente através do método de Composição Funcional. Os resultados de mapeamento obtidos com a ferramenta ABC são apresentados na tabela 6. Esta tabela é composta pelo mesmo subconjunto de circuitos do *benchmark* ISCAS'85 utilizado nos experimentos anteriores. Da mesma forma, as colunas representam os resultados de mapeamentos utilizando as bibliotecas P, PN, NP e NPN.

Comparando os resultados obtidos com a biblioteca de funções de 4 entradas(tabela 6) com os resultados da biblioteca de 3 entradas (tabela 5), é possível verificar a melhoria dos circuitos mapeados,conforme podemos visualizar no gráfico da figura 23, onde grande maioria dos circuitos tem resultados de mapeamento mais reduzido quando feito com a biblioteca P-class de 4 entradas.

O fato de poder gerar bibliotecas para funções de mais entradas, é bastante interessante. A princípio como a porta majoritária tem apenas 3 entradas, ter funções de 4 entradas em uma biblioteca pode não parecer útil. Entretanto, levando em consideração arranjos que podem ser realizados, o fato de termos funções de mais entradas pode minimizar o custo do circuito.

Por fim, também é importante reforçar que em circuitos QCA, diferentemente dos circuitos CMOS, o custo dos fios é relevante para o custo total do circuito. Em tecnologia QCA, os fios são implementados utilizando células QCA, as quais ocupam considerável área física do circuito. A figura 24 ilustra essa afirmação considerando a implementação de um somador de 1 bit realizado com a ferramenta QCA Designer.

Esses custos de fios não são considerados por nenhuma ferramenta de mapeamento tecnológico quando aplicadas na tecnologia QCA. Assim, na etapa

Tabela 6: Tabela com resultados dos testes feitos com diferentes bibliotecas com funções de 4 entradas

Bench. ISCAS'85	***NPN-CLASS***	***NP-CLASS***	***PN-CLASS***	***P-CLASS***
9symml	1244	1162	1142	631
alu2	2224	2107	1957	1266
apex6	4225	3890	3735	2161
cht	913	913	711	474
cm150a	249	244	239	163
cm151a	151	134	122	91
cm152a	116	114	104	78
cm162a	272	242	221	150
cm163a	277	209	195	152
cm42a	213	143	145	74
cm82a	79	79	79	54
cm85a	223	200	192	132
cmb	294	248	218	133
cu	340	273	271	178
decod	267	227	231	104
frg1	338	269	269	168
i2	1312	1087	1030	632
k2	9066	7634	7114	3552
ldd	550	508	453	237
majority	50	33	30	15
mux	241	247	236	157
pcler8	428	338	328	234
pcle	189	445	520	553
pm1	299	244	224	150
term1	855	628	838	454
ttt2	935	862	812	550
unreg	551	541	533	386
vda	4536	3796	3631	1662
x2	309	237	231	153
z4ml	116	116	116	100
Total	31226	27245	25852	14480

de síntese lógica para QCA deve-se observar paralelamente questões físicas do projeto, respeitando o espaço mínimo que deve haver entre dispositivos de células para que não haja interferência na propagação da informação, devido a interação de Coulombic nas células QCA. Segundo (CHO H.; SWARTZLANDER, 2007), o espaço entre dispositivos de células pode variar, mas o mínimo permitido seria a distância padrão é de 20nm. (TEODOSIO; SOUSA, 2007) complementa que este padrão de espaçamento (20 nm) é igual ao tamanho de célula QCA (18 nm), mais o espaçamento entre células (2 nm).

Deste modo, para o futuro da síntese lógica em tecnologia QCA, será necessário mudar o paradigma atual, misturando técnicas de síntese lógica e física em conjunto para a obtenção de circuitos com baixo custo em área. Os experi-

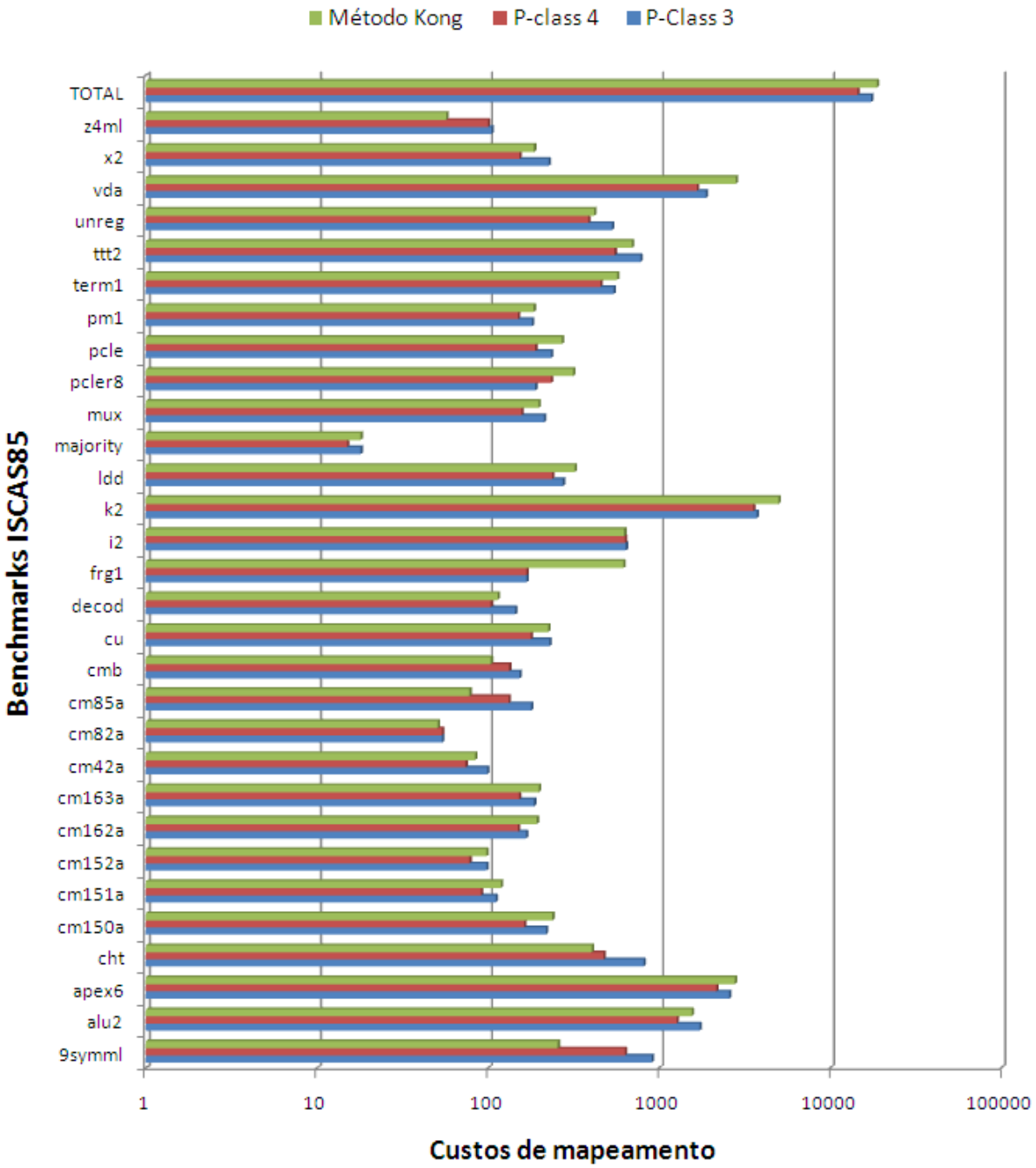


Figura 23: Gráfico comparando resultados de mapeamento com a biblioteca P-class, com 3 e 4 entradas, e o método de Kong

mentos realizados, somente considerando portas majoritárias e inversores QCA, serviram para demonstrar que as ferramentas de síntese existentes não são suficientes para mapear eficientemente circuitos QCA.

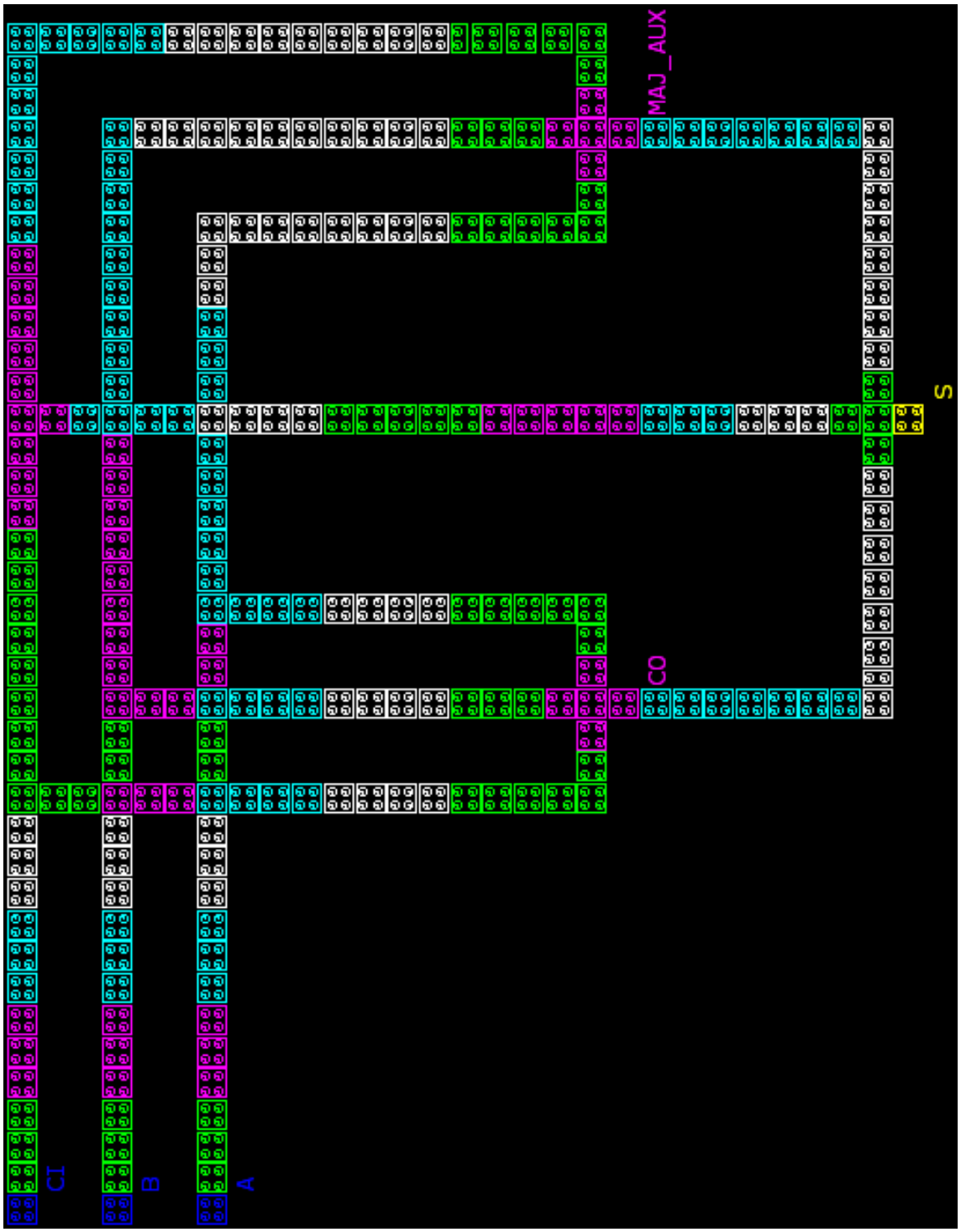


Figura 24: Somador Completo de 1 bit projetado com células QCA (TEODOSIO; SOUSA, 2007)

6 CONCLUSÃO

Com a tecnologia CMOS chegando a um limite físico para o desenvolvimento de circuitos integrados, novas tecnologias passam a ser estudadas. Dentre estas, a tecnologia QCA têm-se mostrado uma forte concorrente para substituir (ou complementar) a tecnologia CMOS. Tendo isto em vista, estudos relacionados à síntese lógica QCA foram realizados buscando investigar a possibilidade da utilização das técnicas atuais de mapeamento tecnológico. Este trabalho apresentou uma análise dos métodos de síntese existentes para a tecnologia QCA. Experimentos foram realizados com a ferramenta ABC, considerada o estado da arte em síntese lógica para tecnologia CMOS. Comparações foram realizadas com os resultados obtidos pelo trabalho estado da arte em síntese QCA (KONG; SHANG; LU, 2010).

Para a execução destes experimentos foram utilizados circuitos pertencentes ao *benchmark* ISCAS'85 e quatro diferentes bibliotecas, para funções de 3 e 4 entradas. Com os resultados obtidos foi possível observar a redução dos custos dos circuitos mapeados. Essa redução se deve, em grande parte, a diminuição do número de inversores necessários para compor o circuito. Em alguns casos, os resultados demonstraram que os algoritmos de síntese CMOS não conseguem convergir para bons resultados em virtude do mau compartilhamento de lógica.

Células mais complexas em QCA nem sempre são a melhor escolha, pois elas encapsulam inversores que não podem ser compartilhados pelas heurísticas de mapeamento. Diversos resultados de mapeamento obtidos foram melhores daqueles apresentados por Kong, demonstrando que técnicas melhores podem ser propostas para obter circuitos menores. Os resultados apresentados consideraram apenas os custos referentes as células que compõem o circuito, assim como os resultados apresentados por Kong e trabalhos relacionados. Entretanto, fica evidente a necessidade da utilização de uma nova abordagem que considere os custos dos fios. Somente assim poderemos obter uma visão mais precisa do custo em área do circuito mapeado. Como trabalhos futuros, pretende-se estudar mais detalhadamente a questão lógica e física de síntese com o auxílio da ferramenta de leiaute QCADesigner, investigar métodos de remoção de redundância e realizar a publicação de um artigo em congresso internacional com os resultados e comparações obtidas neste trabalho.

REFERÊNCIAS

AKERS, S. Synthesis of Combinational Logic Using Three-Input Majority Gates. **Proc. Third Annual Symposium on Switching Circuit Theory and Logical Design**, [S.l.], p.149–158, Oct. 1962.

CHATTERJEE, S.; MISHCHENKO, A.; BRAYTON, R.; WANG, X.; KAM, T. Reducing structural bias in technology mapping. In: COMPUTER-AIDED DESIGN, 2005. ICCAD-2005. IEEE/ACM INTERNATIONAL CONFERENCE ON, 2005. **Anais...** [S.l.: s.n.], 2005. p.519 – 526.

CHO H.; SWARTZLANDER, E. Adder Designs and Analyses for Quantum-Dot Cellular Automata. **Nanotechnology, IEEE Transactions on**, [S.l.], v.6, n.3, p.374–383, may. 2007.

CORREIA, V.; REIS, A. Classifying n-Input Boolean Functions. In: PROC. IWS, 2001. **Anais...** [S.l.: s.n.], 2001. p.58.

GONçALVES, S.; DOMINGUES JR, J.; COLVARA, M.; ROSA JR, L.; MARQUES, F. Evaluating Technology Mapping Methods for QCA Devices. In: XII MICROELECTRONICS STUDENTS FORUM (SFORUM), 2012. **Anais...** [S.l.: s.n.], 2012.

GROUPS, C.; PRINCETON UNIVERSITY. activities at. **TELS/MALS**. http://http://www.princeton.edu/ cad/projects.html.

INTEL, I. C. **Transistor to Transformation - From Sand to Circuits - How Intel Makes Circuits**. http://www.intel.com/content/www/us/en/history/history-intel-chips-timeline-poster.html.

KOHAVI, Z. **Switching and finite automata theory**. [S.l.]: Tata McGraw-Hill, 1987. 658p. (McGraw-Hill Computer Science Series). http://books.google.com.br/books?id=pQy_yQD-hT4C.

KONG, K.; SHANG, Y.; LU, R. An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata. **Nanotechnology, IEEE Transactions on**, [S.l.], v.9, n.2, p.170 –183, march 2010.

LEDESMA, A. R. G. **Metrologia, normalização e regulação de nanomateriais no Brasil**: proposição de um modelo analítico-prospectivo. 2010. Dissertação de Mestrado em Metrologia, Qualidade e Inovação — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro/RJ.

LOGIC CIRCUITS, S. L. **Logic Circuits Synthesis Labs - Instituto de Informática UFRGS**. http://www.inf.ufrgs.br/logics.

MARQUES, F. **Technology Mapping for Virtual Libraries Based on Cells with Minimal Transistor Stacks**. 2008. Tese doutorado — Universidade Federal do Rio Grande do Sul - UFRGS, Porto Alegre/RS.

MARQUES, F.; MARTINELLO JR, O.; RIBAS, R.; DA ROSA JR, L.; REIS, A. **Mapeamento Tecnológico no Projeto de Circuitos Integrados Digitais**. Pelotas, Brasil: Editora e Gráfica Universitária - PREC UFPel, 2009. 177-195p.

MARTINS, M.; CALLEGARO, V.; GONçALVES, S.; COLVARA, M.; MARQUES, F.; ROSA JR, L.; RIBAS, R.; REIS, A. Majority-based Library Generation for Nanometric Technologies. In: XXVIII SIMPóSIO SUL DE MICROELETRôNICA, 2013. **Anais...** [S.l.: s.n.], 2013.

MARTINS, M.; RIBAS, R.; REIS, A. Functional composition: A new paradigm for performing logic synthesis. In: QUALITY ELECTRONIC DESIGN (ISQED), 2012 13TH INTERNATIONAL SYMPOSIUM ON, 2012. **Anais...** [S.l.: s.n.], 2012. p.236 –242.

MILLER, H.; WINDER, R. Majority-Logic Synthesis by Geometric Methods. **IRE Transactions on Electronic Computers**, [S.l.], v.EC-11, n.1, p.89–90, Feb. 1962.

MISHCHENKO, A.; CHATTERJEE, S.; BRAYTON, R.; WANG, X.; KAM, T. **Technology Mapping with Boolean Matching, Supergates and Choices**. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.152.5130.

MOMENZADEH, M.; HUANG, J.; TAHOORI, M.; LOMBARDI, F. Characterization, test, and logic synthesis of and-or-inverter (AOI) gate design for QCA implementation. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, [S.l.], v.24, n.12, p.1881 – 1893, dec. 2005.

MUROGA, S. **Threshold Logic and its Applications**. [S.l.]: Wiley-Interscience, 1971. http://books.google.com.br/books?id=wvtQAAAAMAAJ.

NETO, J. M. S. **Graphic of Moore Law, the graphic show the evolution of the number of transitors nos microchips Intel over the time.** http://pt.wikipedia.org/wiki/Lei_de_Moore.

NETO, O. **Simulation and Automatic Syntheses of Quantum Dots Cellular Automata Circuits Thought Intelligent Techniques**. 2006. Dissertação (Mestrado em Ciência da Computação) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro/RJ.

SECHEN, C.; AL; et. LIBRARIES: Lifejacket or straitjacket. **IEEE DESIGN AUTOMATION CONFERENCE**, [S.l.], v.s.n., p.642–643, 2003.

SEMICONDUCTORS ITRS, I. T. R. for. **Lithography**. http://www.itrs.net.

SEN, B.; ADAK, T.; ANAND, A.; SIKDAR, B. Synthesis of reversible universal QCA gate structure for energy efficient digital design. In: TENCON 2011 - 2011 IEEE REGION 10 CONFERENCE, 2011. **Anais...** [S.l.: s.n.], 2011. p.806 –810.

SENTOVICH, E.; SINGH, K.; LAVAGNO, L.; MOON, C.; MURGAI, R.; SALDANHA, A.; SAVOJ, H.; STEPHAN, P.; BRAYTON, R.; SANGIOVANNI-VINCENTELLI, A. SIS: A system of sequential circuit synthesis. **Memorandum No. UCBERL M92/41**, [S.l.], May. 1992.

SENTOVICH, E.; SINGH, K.; LAVAGNO, L.; MOON, C.; MURGAI, R.; SALDANHA, A.; SAVOJ, H.; STEPHAN, P.; BRAYTON, R.; SANGIOVANNI-VINCENTELLI, A. **SIS - Sequential Interactive System**. http://www.cs.columbia.edu/ cs4861/s07-sis/.

SYNTHESIS, B. L.; GROUP, V. **ABC**: A System for Sequential Synthesis. http://www.eecs.berkeley.edu/ alanmi/abc/abc.html.

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.

TOUGAW, P.; LENT, C. Logical devices implemented using quantum cellular automata. **Journal of Applied Physics**, [S.l.], v.75, n.3, p.1818–1825, 1994.

WALUS, K.; DYSART, T.; JULLIEN, G.; BUDIMAN, R. QCADesigner: a rapid design and Simulation tool for quantum-dot cellular automata. **Nanotechnology, IEEE Transactions on**, [S.l.], v.3, n.1, p.26 – 31, march 2004.

YANG, S. **Logic Synthesis and Optimization Benchmarks User Guide Version 3.0**. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.591.

YANO, C.; FURTADO, F.; MENDES, H.; A., P. M.; FERNANDES, T. Caderno 19 SBPC - Tecnologia e Inovação. In: SBPC 58 REUNIãO ANUAL DA SOCIEDADE BRASILEIRA PARA O PROGRESSO DA CIêNCIA, 2006. **Anais...** [S.l.: s.n.], 2006. p.9–22.

ZHANG, R.; GUPTA, P.; JHA, N. Majority and Minority Network Synthesis With Application to QCA-, SET-, and TPL-Based Nanotechnologies. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, [S.l.], v.26, n.7, p.1233 –1245, july 2007.

ZHANG, R.; WALUS, K.; WANG, W.; GRAHAM, A. A Method of Majority Logic Reduction for Quantum Cellular Automata. **IEEE TRANSACTIONS ON NANOTECHNOLOGY**, [S.l.], v.3, n.4, p.443 – 450, 2004.

ANEXO A BIBLIOTECAS DE 3 ENTRADAS UTILIZADAS

Estas foram as bibliotecas utilizadas para os testes realizados na ferramenta ABC, para o comando de mapeamento *map*.

- **Básica.genlib**

```
GATE    one     1   O=CONST1;
GATE    zero    0   O=CONST0;
GATE    inv     2   O=!a;                   PIN * INV 0 0 0 0 0 0
GATE    maj     1   O=(a*b)+(a*c)+(b*c);    PIN * NONINV 0 0 0 0 0 0
GATE    and2    1   O=a*b;                  PIN * NONINV 0 0 0 0 0 0
```

- **Zhang2004.genlib** (**NPN**)

```
GATE    one     1   O=CONST1;
GATE    zero    0   O=CONST0;
GATE    inv     2   O=!a;                   PIN * INV 0 0 0 0 0 0
GATE    g1      2   O=a*b*c;                PIN * NONINV 0 0 0 0 0 0
GATE    g2      1   O=a*b;                  PIN * NONINV 0 0 0 0 0 0
GATE    g3      3   O=(a*b*c)+(a*!b*!c);    PIN * UNKNOWN 0 0 0 0 0 0
GATE    g4      5   O=(a*b*c)+(!a*!b*!c);   PIN * UNKNOWN 0 0 0 0 0 0
GATE    g5      2   O=(a*b)+(b*c);          PIN * NONINV 0 0 0 0 0 0
GATE    g5      2   O=(a*b)+(b*c);          PIN * NONINV 0 0 0 0 0 0
GATE    g6      4   O=a*b + !a*!b*c;        PIN * UNKNOWN 0 0 0 0 0 0
GATE    g7      4   O=a*b*c + !a*b*!c + a*!b*!c;  PIN * UNKNOWN 0 0 0 0 0 0
GATE    g8      1   O=a;                    PIN * NONINV 0 0 0 0 0 0
GATE    g9      1   O=a*b + b*c + a*c;      PIN * NONINV 0 0 0 0 0 0
GATE    g10     3   O=a*b + !b*c;           PIN * UNKNOWN 0 0 0 0 0 0
GATE    g11     5   O=a*b + b*c + !a*!b*!c; PIN * UNKNOWN 0 0 0 0 0 0
GATE    g12     3   O=a*b + !a*!b;          PIN * UNKNOWN 0 0 0 0 0 0
GATE    g13     3   O=a*b*c + !a*!b*c + a*!b*!c + !a*b*!c;  PIN * NONINV 0 0 0 0 0 0
```

- **Kong2010.genlib** (**PN**)

```
GATE    f1      0   O=CONST0;
GATE    f2      0   O=CONST1;
GATE    f3      0   O=a;                    PIN * NONINV 0 0 0 0 0 0
GATE    f4      5   O=!a;                   PIN * INV 0 0 0 0 0 0
GATE    f5      0   O=b;                    PIN * NONINV 0 0 0 0 0 0
GATE    f6      5   O=!b;                   PIN * INV 0 0 0 0 0 0
```

```
GATE    f7      0    O=c;                          PIN * NONINV 0 0 0 0 0 0
GATE    f8      5    O=!c;                         PIN * INV 0 0 0 0 0 0
GATE    f9      3    O=a*b;                        PIN * NONINV 0 0 0 0 0 0
GATE    f10     8    O=!a + !b;                    PIN * UNKNOWN 0 0 0 0 0 0
GATE    f11     3    O=a*c;                        PIN * NONINV 0 0 0 0 0 0
GATE    f12     8    O=!a + !c;                    PIN * UNKNOWN 0 0 0 0 0 0
GATE    f13     3    O=b*c;                        PIN * NONINV 0 0 0 0 0 0
GATE    f14     8    O=!b + !c;                    PIN * UNKNOWN 0 0 0 0 0 0
GATE    f15     8    O=a*!b;                       PIN * UNKNOWN 0 0 0 0 0 0
GATE    f16     8    O=!a + b;                     PIN * UNKNOWN 0 0 0 0 0 0
GATE    f17     8    O=a*!c;                       PIN * UNKNOWN 0 0 0 0 0 0
GATE    f18     8    O=!a + c;                     PIN * UNKNOWN 0 0 0 0 0 0
GATE    f19     8    O=b*!c;                       PIN * UNKNOWN 0 0 0 0 0 0
GATE    f20     8    O=!b + c;                     PIN * UNKNOWN 0 0 0 0 0 0
GATE    f21     8    O=!a*!b;                      PIN * UNKNOWN 0 0 0 0 0 0
GATE    f22     3    O=a + b;                      PIN * NONINV 0 0 0 0 0 0
GATE    f23     8    O=!a*!c;                      PIN * UNKNOWN 0 0 0 0 0 0
GATE    f24     3    O=a + c;                      PIN * NONINV 0 0 0 0 0 0
GATE    f25     8    O=!b*!c;                      PIN * UNKNOWN 0 0 0 0 0 0
GATE    f26     3    O=b + c;                      PIN * NONINV 0 0 0 0 0 0
GATE    f27     8    O=!a*b;                       PIN * UNKNOWN 0 0 0 0 0 0
GATE    f28     8    O=a + !b;                     PIN * UNKNOWN 0 0 0 0 0 0
GATE    f29     8    O=!a*c;                       PIN * UNKNOWN 0 0 0 0 0 0
GATE    f30     8    O=a + !c;                     PIN * UNKNOWN 0 0 0 0 0 0
GATE    f31     8    O=!b*c;                       PIN * UNKNOWN 0 0 0 0 0 0
GATE    f32     8    O=b + !c;                     PIN * UNKNOWN 0 0 0 0 0 0
GATE    f33     3    O=a*b + a*c + b*c;            PIN * NONINV 0 0 0 0 0 0
GATE    f34     8    O=!a*!b + !a*!c + !b*!c;      PIN * UNKNOWN 0 0 0 0 0 0
GATE    f35     8    O=!a*b + !a*c + b*c;          PIN * UNKNOWN 0 0 0 0 0 0
GATE    f36     13   O=a*!b + a*!c + !b*!c;        PIN * UNKNOWN 0 0 0 0 0 0
GATE    f37     8    O=a*!b + a*c + !b*c;          PIN * UNKNOWN 0 0 0 0 0 0
GATE    f38     13   O=!a*b + !a*!c + b*!c;        PIN * UNKNOWN 0 0 0 0 0 0
GATE    f39     8    O=a*b + a*!c + b*!c;          PIN * UNKNOWN 0 0 0 0 0 0
GATE    f40     13   O=!a*!b + !a*c + !b*c;        PIN * UNKNOWN 0 0 0 0 0 0
```

- **NP-Class.genlib**

```
GATE    "ZERO"  0.0          O=CONST0;
GATE    "ONE"   0.0          O=CONST1;
GATE    "C0_01" 11.0         O=(!c * !b * !a);
PIN     a       INV          999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV          999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       INV          999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_03" 8.0          O=(!b * !a);
PIN     a       INV          999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV          999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_06" 24.0         O=(!a * ((!c * b) + (c * !b)));
PIN     a       INV          999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN      999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN      999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_07" 11.0         O=(!a * (!c + !b));
```

```
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       INV             999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_0F" 5.0             O=!a;
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_16" 27.0            O=((!c + !a) * ((!b * (c + a)) + (!c * b * !a)));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_17" 8.0             O=((!c + !a) * (!b + (!c * !a)));
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       INV             999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_18" 27.0            O=((!c + !a) * (c + !b) * (b + a));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_19" 27.0            O=((!c + b) * (!b + (c * !a)));
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_1B" 19.0 O=((!c + !a) * (c + !b));
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_1E" 27.0            O=((!a + (!c * !b)) * (c + b + a));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_1F" 11.0            O=(!a + (!c * !b));
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       INV 999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_3C" 19.0            O=((!b + !a) * (b + a));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_3D" 27.0            O=((!b + !a) * (!c + b + a));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       INV             999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_3F" 8.0             O=(!b + !a);
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV             999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_69" 24.0 O=((b + (c * a) + (!c * !a)) * (!b + ((!c + !a) * (c + a))));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_6B" 27.0            O=((c * !b) + (!c * b * a) + (!a * (c + !b)));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN 999.0 1.0 1.0 0.2 1.0 0.2
```

```
GATE    "C0_6F" 24.0            O=(!a + (!c * b) + (c * !b));
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_7E" 27.0            O=((!c * b) + ((!b + !a) * (c + a)));
PIN     a       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       UNKNOWN         999.0 1.0 1.0 0.2 1.0 0.2
GATE    "C0_7F" 11.0            O=(!c + !b + !a);
PIN     a       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     b       INV             999.0 1.0 1.0 0.2 1.0 0.2
PIN     c       INV             999.0 1.0 1.0 0.2 1.0 0.2
```

- **Pclass.genlib**

```
GATEf1  0   O=CONST0;
GATEf2  0   O=CONST1;
GATE01  11  O=((!c)*(!b))*(!a);                                 PIN * UNKNOWN 0 0 0 0 0 0
GATE02  16  O=((c)*(!b))*(!a);                                  PIN * UNKNOWN 0 0 0 0 0 0
GATE03  8   O=(!b)*(!a);                                        PIN * UNKNOWN 0 0 0 0 0 0
GATE06  24  O=(!a)*(((c)+(b))*((!c)+(!b)));                     PIN * UNKNOWN 0 0 0 0 0 0
GATE07  11  O=((!c)+(!b))*(!a);                                 PIN * UNKNOWN 0 0 0 0 0 0
GATE08  11  O=((c)*(b))*(!a);                                   PIN * UNKNOWN 0 0 0 0 0 0
GATE09  24  O=(!a)*(((c)+(!b))*((!c)+(b)));                     PIN * UNKNOWN 0 0 0 0 0 0
GATE0A  8   O=(c)*(!a);                                         PIN * UNKNOWN 0 0 0 0 0 0
GATE0B  16  O=((c)+(!b))*(!a);                                  PIN * UNKNOWN 0 0 0 0 0 0
GATE0E  11  O=((c)+(b))*(!a);                                   PIN * UNKNOWN 0 0 0 0 0 0
GATE0F  5   O=!a;                                               PIN * INV 0 0 0 0 0 0
GATE16  27  O=((!c)+(!b))*(((c)+((b)+(a)))*((!a)+((!c)*(!b))));  PIN * UNKNOWN 0 0 0 0 0 0
GATE17  8   O=((!c)+(!b))*((!a)+((!c)*(!b)));                   PIN * UNKNOWN 0 0 0 0 0 0
GATE18  27  O=((c)+(!b))*(((!c)+(!a))*((b)+(a)));               PIN * UNKNOWN 0 0 0 0 0 0
GATE19  27  O=((c)+(!b))*((!c)+((b)*(!a)));                     PIN * UNKNOWN 0 0 0 0 0 0
GATE1A  27  O=((c)+(a))*((!a)+((!c)*(!b)));                     PIN * UNKNOWN 0 0 0 0 0 0
GATE1B  19  O=((c)+(!b))*((!c)+(!a));                           PIN * UNKNOWN 0 0 0 0 0 0
GATE1E  27  O=((c)+((b)+(a)))*((!a)+((!c)*(!b)));               PIN * UNKNOWN 0 0 0 0 0 0
GATE1F  11  O=((!c)*(!b))+(!a);                                 PIN * UNKNOWN 0 0 0 0 0 0
GATE28  19  O=(c)*(((b)+(a))*((!b)+(!a)));                      PIN * UNKNOWN 0 0 0 0 0 0
GATE29  27  O=((c)+(!b))*(((!c)+((b)+(a)))*((!a)+((c)*(!b))));  PIN * UNKNOWN 0 0 0 0 0 0
GATE2A  16  O=((!b)+(!a))*(c);                                  PIN * UNKNOWN 0 0 0 0 0 0
GATE2B  13  O=((c)+(!b))*((!a)+((c)*(!b)));                     PIN * UNKNOWN 0 0 0 0 0 0
GATE2C  22  O=((b)+(a))*((!a)+((c)*(!b)));                      PIN * UNKNOWN 0 0 0 0 0 0
GATE2D  27  O=((!c)+((b)+(a)))*((!a)+((c)*(!b)));               PIN * UNKNOWN 0 0 0 0 0 0
GATE2E  19  O=((c)+(b))*((!b)+(!a));                            PIN * UNKNOWN 0 0 0 0 0 0
GATE2F  16  O=((c)*(!b))+(!a);                                  PIN * UNKNOWN 0 0 0 0 0 0
GATE3C  19  O=((b)+(a))*((!b)+(!a));                            PIN * UNKNOWN 0 0 0 0 0 0
GATE3D  27  O=((!b)+(!a))*((!c)+((b)+(a)));                     PIN * UNKNOWN 0 0 0 0 0 0
GATE3E  22  O=((!b)+(!a))*((c)+((b)+(a)));                      PIN * UNKNOWN 0 0 0 0 0 0
GATE3F  8   O=(!b)+(!a);                                        PIN * UNKNOWN 0 0 0 0 0 0
GATE68  27  O=((c)+(b))*(((!c)+((!b)+(!a)))*((a)+((c)*(b))));   PIN * UNKNOWN 0 0 0 0 0 0
GATE69  24  O=((c)+(((b)+(!a))*((!b)+(a))))*((!c)+(((b)+(a))*((!b)+(!a))));
                                                                PIN * UNKNOWN 0 0 0 0 0 0
```

```
GATE6A  27  O=((c)+((b)*(a)))*((!c)+((!b)+(!a)));                          PIN * UNKNOWN 0 0 0 0 0 0
GATE6B  27  O=((c)*(!b))+(((c)+((!b)+(a)))*((!a)+((!c)*(b))));             PIN * UNKNOWN 0 0 0 0 0 0
GATE6E  27  O=((c)+(b))*((!c)+((!b)+(!a)));                                PIN * UNKNOWN 0 0 0 0 0 0
GATE6F  24  O=(!a)+(((c)+(b))*((!c)+(!b)));                                PIN * UNKNOWN 0 0 0 0 0 0
GATE7E  27  O=((c)*(!b))+(((!c)+(!a))*((b)+(a)));                          PIN * UNKNOWN 0 0 0 0 0 0
GATE7F  11  O=((!c)+(!b))+(!a);                                            PIN * UNKNOWN 0 0 0 0 0 0
GATE80  6   O=((c)*(b))*(a);                                               PIN * NONINV 0 0 0 0 0 0
GATE81  27  O=((c)+(!b))*(((!c)+(a))*((b)+(!a)));                          PIN * UNKNOWN 0 0 0 0 0 0
GATE82  19  O=(c)*(((b)+(!a))*((!b)+(a)));                                 PIN * UNKNOWN 0 0 0 0 0 0
GATE83  22  O=((b)+(!a))*((!b)+((c)*(a)));                                 PIN * UNKNOWN 0 0 0 0 0 0
GATE86  27  O=((c)+(b))*(((!c)+((!b)+(a)))*((!a)+((c)*(b))));              PIN * UNKNOWN 0 0 0 0 0 0
GATE87  27  O=((!c)+((!b)+(a)))*((!a)+((c)*(b)));                          PIN * UNKNOWN 0 0 0 0 0 0
GATE88  3   O=(c)*(b);                                                     PIN * NONINV 0 0 0 0 0 0
GATE89  27  O=((c)+(!b))*((b)+((!c)*(!a)));                                PIN * UNKNOWN 0 0 0 0 0 0
GATE8A  11  O=((b)+(!a))*(c);                                              PIN * UNKNOWN 0 0 0 0 0 0
GATE8B  19  O=((c)+(!b))*((b)+(!a));                                       PIN * UNKNOWN 0 0 0 0 0 0
GATE8E  8   O=((c)+(b))*((!a)+((c)*(b)));                                  PIN * UNKNOWN 0 0 0 0 0 0
GATE8F  11  O=((c)*(b))+(!a);                                              PIN * UNKNOWN 0 0 0 0 0 0
GATE96  24  O=((c)+(((b)+(a))*((!b)+(!a))))*((!c)+(((b)+(!a))*((!b)+(a))));
                                                                           PIN * UNKNOWN 0 0 0 0 0 0
GATE97  27  O=((!c)*(!b))+(((!c)+((!b)+(a)))*((!a)+((c)*(b))));            PIN * UNKNOWN 0 0 0 0 0 0
GATE98  22  O=((c)+(!b))*((b)+((!c)*(a)));                                 PIN * UNKNOWN 0 0 0 0 0 0
GATE99  19  O=((c)+(!b))*((!c)+(b));                                       PIN * UNKNOWN 0 0 0 0 0 0
GATE9A  27  O=((c)+((!b)*(a)))*((!c)+((b)+(!a)));                          PIN * UNKNOWN 0 0 0 0 0 0
GATE9B  27  O=((c)+(!b))*((!c)+((b)+(!a)));                                PIN * UNKNOWN 0 0 0 0 0 0
GATE9E  27  O=((c)*(b))+(((c)+((b)+(a)))*((!a)+((!c)*(!b))));              PIN * UNKNOWN 0 0 0 0 0 0
GATE9F  24  O=(!a)+(((c)+(!b))*((!c)+(b)));                                PIN * UNKNOWN 0 0 0 0 0 0
GATEA8  6   O=((b)+(a))*(c);                                               PIN * NONINV 0 0 0 0 0 0
GATEA9  27  O=((c)+((!b)*(!a)))*((!c)+((b)+(a)));                          PIN * UNKNOWN 0 0 0 0 0 0
GATEAA  0   O=c;                                                           PIN * NONINV 0 0 0 0 0 0
GATEAB  16  O=((!b)*(!a))+(c);                                             PIN * UNKNOWN 0 0 0 0 0 0
GATEAC  14  O=((c)+(!a))*((b)+(a));                                        PIN * UNKNOWN 0 0 0 0 0 0
GATEAD  22  O=((c)+(!a))*((!c)+((b)+(a)));                                 PIN * UNKNOWN 0 0 0 0 0 0
GATEAE  11  O=((b)*(!a))+(c);                                              PIN * UNKNOWN 0 0 0 0 0 0
GATEAF  8   O=(c)+(!a);                                                    PIN * UNKNOWN 0 0 0 0 0 0
GATEBC  22  O=((b)+(a))*((c)+((!b)+(!a)));                                 PIN * UNKNOWN 0 0 0 0 0 0
GATEBD  27  O=((c)*(b))+(((!c)+(a))*((!b)+(!a)));                          PIN * UNKNOWN 0 0 0 0 0 0
GATEBE  19  O=(c)+(((b)+(a))*((!b)+(!a)));                                 PIN * UNKNOWN 0 0 0 0 0 0
GATEBF  16  O=((c)+(!b))+(!a);                                             PIN * UNKNOWN 0 0 0 0 0 0
GATEE8  3   O=((c)+(b))*((a)+((c)*(b)));                                   PIN * NONINV 0 0 0 0 0 0
GATEE9  27  O=((c)*(b))+(((c)+((b)+(!a)))*((a)+((!c)*(!b))));              PIN * UNKNOWN 0 0 0 0 0 0
GATEEA  6   O=((b)*(a))+(c);                                               PIN * NONINV 0 0 0 0 0 0
GATEEB  19  O=(c)+(((b)+(!a))*((!b)+(a)));                                 PIN * UNKNOWN 0 0 0 0 0 0
GATEEE  3   O=(c)+(b);                                                     PIN * NONINV 0 0 0 0 0 0
GATEEF  11  O=((c)+(b))+(!a);                                              PIN * UNKNOWN 0 0 0 0 0 0
GATEFE  6   O=((c)+(b))+(a);                                               PIN * NONINV 0 0 0 0 0 0
```