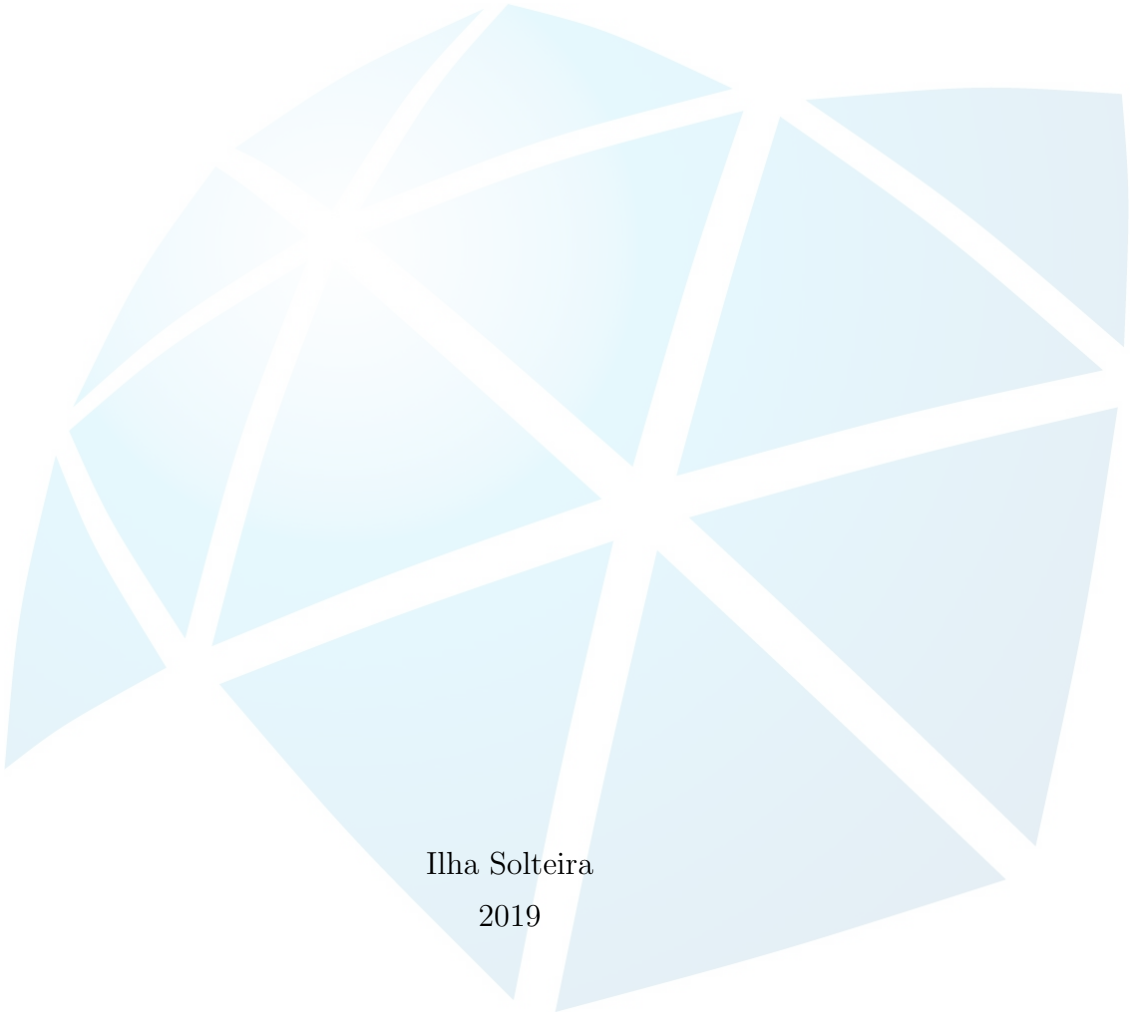
UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

FACULDADE DE ENGENHARIA

CAMPUS DE ILHA SOLTEIRA

JEFERSON DE LIMA MUNIZ

MK4: Programa para síntese de funções majoritárias com até 4 variáveis de entrada.



Ilha Solteira

2019

JEFERSON DE LIMA MUNIZ

MK4: Programa para síntese de funções majoritárias com até 4 variáveis de entrada.

Dissertação apresentada à Faculdade de Engenharia – UNESP – Campus de Ilha Solteira como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica. Área de Conhecimento: Automação.

Prof. Dr. Alexandre César Rodrigues da Silva
Orientador

Ilha Solteira
2019

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Muniz, Jeferson de Lima.
M966m Mk4: programa para síntese de funções majoritárias com até 4 variáveis de entrada. / Jeferson de Lima Muniz. -- Ilha Solteira: [s.n.], 2019
98 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia. Área de conhecimento: Automação, 2019

Orientador: Alexandre César Rodrigues da Silva
Inclui bibliografia

1. Lógica majoritária. 2. Minimização de funções majoritárias. 3. Síntese de circuitos majoritários.

Sandra Maria Clemente de Souza
DSTBD/STRAUD
Bibliotecária
CRB 8-4740

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: MK4: Programa para síntese de funções majoritárias com até 4 variáveis de entrada

AUTOR: JEFERSON DE LIMA MUNIZ
ORIENTADOR: ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovado como parte das exigências para obtenção do Título de Mestre em ENGENHARIA ELÉTRICA, área: Automação pela Comissão Examinadora:

Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. CARLOS ANTONIO ALVES
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. MARCOS NAPOLEÃO RABELO
Matemática Industrial / Universidade Federal de Goiás

Ilha Solteira, 16 de janeiro de 2019

AGRADECIMENTOS

Agradeço aos meus pais, Valdir e Geni pela força e incentivo para continuar meus estudos.

Agradeço ao meu orientador, Prof. Dr. Alexandre César Rodrigues da Silva, por ter me aceitado como orientando e possibilitado a conclusão deste trabalho.

Agradeço à CAPES pelo apoio financeiro permitindo que eu dedicasse meu tempo exclusivamente a minha pesquisa.

E agradeço aos meus colegas de laboratório Evandro, Alexandre, Willian e Douglas pelo apoio e pelos momentos divertidos que tivemos ao longo desses últimos anos.

Agradeço à minha namorada Gabriela pelos momentos de alegria e incentivo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

Com a evolução da tecnologia, os CIs (Circuitos Integrados) com tecnologia CMOS (*Complementary Metal-Oxide Semicondutor*) têm se tornado cada vez menores e mais eficientes, entretanto, esta tecnologia está atingindo os limites físicos. Para minimizar ainda mais os circuitos digitais, novas tecnologias foram desenvolvidas como, por exemplo, a tecnologia QCA (*Quantum-Dot Cellular Automata*) que em conjunto com a lógica majoritária tem despertado o interesse da comunidade acadêmica no que se refere ao desenvolvimento de ferramentas de síntese e de otimização. Neste trabalho implementou-se o programa denominado MK4 que tem como proposta realizar a minimização de funções majoritárias com até quatro variáveis, utilizando as ideias contidas no mapa de Karnaugh. Os resultados obtidos pelo MK4 foram comparados com os do programa *exact_mig*. De 65.536 funções comparadas, 92,60% das funções geradas pelo programa MK4 tiveram custos iguais ou inferiores em relação as funções geradas pelo *exact_mig*.

Palavras-chave: Lógica majoritária. Minimização de funções majoritárias. Síntese de circuitos majoritários.

ABSTRACT

With the evolution of technology, the ICs (Integrated Circuits) with CMOS (Complementary Metal-Oxide Semiconductor) technology has become smaller and more efficient. However, this technology is reaching its physical limits. To further minimize digital circuits, new technologies are presented such as, QCA (*Quantum-Dot Cellular Automata*) technology that together with majority logic has aroused the interest of the academic community in the development of synthesis and optimization tools. In this work the program denominated MK4 was implemented, with the purpose of minimizing majority functions with up to four variables, using the Karnaugh map. The results obtained by MK4 were compared with those of the exact_mig program. From 65,536 functions compared, 92.60% of the functions generated by the MK4 had equal or lower costs in relation to the functions generated by the exact_mig.

Keywords: Majority Logic. Minimization of majority functions. Synthesis of majority circuits.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AIG	*And Inverter Graph*
B2M	*Boolean to Majority*
CIs	Circuitos Integrados
CMOS	*Complementary Metal-Oxide Semicondutor*
DSD	*Disjoint-support decomposition*
MALS	*Majority Logic Synthesizer*
MIG	*Majority-Inverter Graph*
MLUT	*Majority expressions Look-up Table*
MPC	*Majority Primitives Combination*
QCA	*Quantum-dot Cellular Automata*
SET	*Single Electron Tunneling*
SMT	Solucionador de módulo e teoria
TPL	*Tunneling Phase Logic*
XMG	*XOR-Majority Graph*

SUMÁRIO

1 INTRODUÇÃO

Com a evolução da tecnologia CMOS (*Complementary Metal-Oxide Semicondutor*) os CIs (Circuitos Integrados) têm se tornado cada vez menores e mais eficientes. Porém, o processo de fabricação destes CIs está atingindo seu limite físico. Assim, a procura por novas tecnologias e algoritmos de minimização de funções booleanas torna-se necessária (WANG *et al.*, 2015).

Geralmente, circuitos que utilizam a tecnologia CMOS fazem o uso da lógica booleana e possuem a porta lógica NAND (*NÃO E* ou *E* negada) como célula básica. Em contrapartida, existem tecnologias que possuem enfoque na computação quântica, como por exemplo os *quantum-dot cellular automata* (QCA), *single electron tunneling* (SET) e *tunneling phase logic* (TPL) que fazem o uso da lógica majoritária e têm como unidade básica a porta majoritária, que possui saída verdadeira caso a maioria de suas entradas tenham valor lógico verdadeiro (WANG *et al.*, 2015).

Um dos primeiros estudos focados na lógica majoritária e na minimização de circuitos majoritários foi apresentado no trabalho de Lindaman (1960), que inseriu o operador majoritário na álgebra booleana.

Em Cohn e Lindaman (1961) foi proposto um conjunto de axiomas que definiu a lógica majoritária independentemente da álgebra booleana.

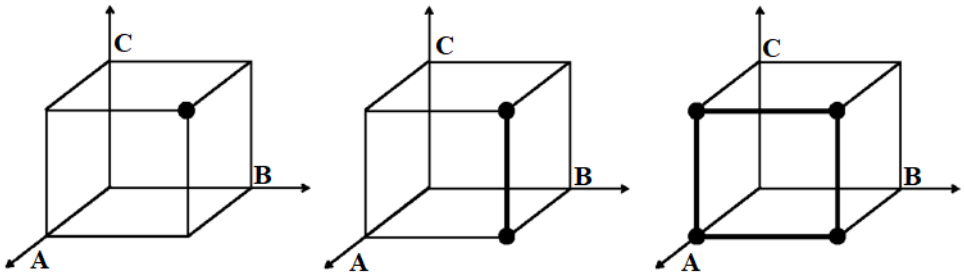
Há diversos métodos e programas de otimização de funções booleanas visando minimizar um circuito, tais como o método do Mapa de Karnaugh (KARNAUGH, 1953), o método de Quine-McCluskey (MCCLUSKEY, 1956), o programa ESPRESSO (BRAYTON *et al.*, 1984), o programa McBOOLE (DAGENAIS; AGARWAL; RUMIN, 1986) e o programa BOOM (HLAVICKA; FISER, 2001). Porém, os métodos mais comuns não levam em consideração o uso da porta majoritária. Desta forma, métodos de minimização que possuem enfoque na lógica majoritária e na porta majoritária visando a cobertura otimizada de mintermos tornam-se necessários.

Em 1961 no trabalho de Akers (1961) apresentou-se o primeiro método manual de

manipulação de expressões majoritárias.

Em Zhang *et al.* (2004) foi proposto o método das treze funções padrões. Neste método é utilizado um cubo (três dimensões) para mapear as funções booleanas de três variáveis. Dessa forma, cada dimensão do cubo tem a finalidade de representar uma variável de entrada. Os oito vértices representam os mintermos, as arestas representam os implicantes da função e as faces representam uma variável em sua forma complementar ou não complementar. Na Figura 1 apresenta-se um cubo onde destacam-se um vértice (mintermo), uma aresta (implicante) e uma face (variável).

Figura 1 – Vértice, aresta e face em um cubo.



Fonte: Adaptado de Zhang *et al.* (2004).

Treze padrões de funções são obtidos através da combinação de vértices, arestas e faces do cubo. Todas as funções com três variáveis de entrada se encaixam em um desses treze padrões que se encontram em uma forma reduzida.

O método das treze funções padrões foi um dos primeiros métodos de simplificação para lógica majoritária, porém o método só funciona para funções booleanas de três variáveis, além do que, mesmo conseguindo reduzir uma função, a função gerada pode não ser a mínima (WANG *et al.*, 2013).

Em Walus *et al.* (2004) foi desenvolvido um método de conversão de funções booleanas que emprega os operadores *E*, *NÃO* e *OU* em funções majoritárias que empregam o operador majoritário. Dada a tabela verdade de uma função F, o método combina três mapas de Karnaugh com diferentes funções de forma que, a fim de atingir a maioria, o mapa resultante cubra todos os mintermos de F pelo menos duas vezes. O método proposto teve um grande impacto na literatura, pois outros métodos tiveram como base as ideias propostas por Walus e, devido a sua importância, foi discutido em maiores detalhes no capítulo 5.

No trabalho de Zhang, Gupta e Jha (2007) foi desenvolvido o programa *Majority Logic Synthesizer* (MALS). Trata-se do primeiro programa a minimizar funções majoritárias

com mais de três variáveis de entrada. O algoritmo do MALS inicia reduzindo o número de variáveis de uma função de forma que a função resultante tenha no máximo três variáveis de entrada. Para decompor uma função em outra equivalente com apenas três variáveis de entrada é utilizada a ferramenta *SIS TOOL* desenvolvida por Sentovich *et al.* (1992).

Após a redução, é realizada a minimização algébrica na função resultante, então verifica-se se a função reduzida pode ser estruturada utilizando apenas três portas majoritárias. Caso seja possível, é gerada a função majoritária e o algoritmo é finalizado. Se não, o método do mapa de Karnaugh é aplicado para gerar uma função majoritária e o algoritmo é encerrado.

O programa MALS representou significantes avanços nas pesquisas de minimização de funções majoritárias, porém não garante uma solução miníma (WANG, 2014).

Em Wang *et al.* (2013) foi proposto o *Majority Expressions Look-up Table* (MLUT). Trata-se de um método exaustivo para encontrar o mínimo de todas as funções majoritárias com até quatro variáveis de entrada. O método parte do princípio de que o número total de funções geradas pela combinação de até dezesseis mintermos é igual a 2^{16} (65.536) funções, então aplica-se o conceito do método do mapa de Karnaugh de quatro variáveis, apresentado na seção 5.2, para encontrar o mínimo de todas as funções e gerar uma tabela de busca com todas as combinações de mintermos e seu mínimo correspondente.

Em Amarú, Gaillardon e Micheli (2015) foi proposto um *framework* de minimização que faz o uso da inserção de erros propositais em uma forma de representação de funções majoritárias conhecida como *Majority-Inverter Graph* (MIG). Entende-se como erro proposital um erro que estenda ou modifique uma função de forma que o resultado da função original não seja alterado. A partir da função resultante pode-se aplicar uma forma de minimização voltada para reduzir o número de níveis e outra voltada a reduzir o número de portas majoritárias da função, ambas podendo gerar funções com resultados equivalentes mas com custos distintos.

Em Mishra e Thapliyal (2017) foi proposto um algoritmo denominado *Boolean to Majority* (B2M) que possui como entrada uma função booleana e a partir dela é gerada uma função majoritária equivalente. Para gerar a função majoritária, o algoritmo combina até três funções primitivas que quando combinadas devem ter saída igual a da função booleana de entrada. Define-se função primitiva como uma função que possui no máximo uma porta majoritária para cobrir seu respectivo conjunto de mintermos. Porém, este

algoritmo funciona somente para funções com até 3 variáveis de entrada. Para funções com mais entradas a função é decomposta para que tenha no máximo três entradas.

Em Soeken *et al.* (2017) foi proposta uma metodologia que tem o objetivo de realizar uma síntese exata de funções majoritárias com até seis variáveis. O programa *exact_mig* tem como base o MIG de uma função. O MIG é transcrito como entrada em um solucionador de módulo e teoria (SMT) que tem como propósito verificar se o MIG gerado pelo *exact_mig* tem um resultado válido ou não.

Define-se solucionador de módulo e teoria como um algoritmo que valida se uma fórmula, expressão ou função de entrada possui uma solução válida ou não. Para isso um SMT possui diversas teorias que servem de auxílio para realizar a validação de uma entrada (CORDEIRO; FISCHER; MARQUES-SILVA, 2012).

Destaca-se que o programa *exact_mig* possui como critério de custo somente o número de níveis e de portas majoritárias de uma função, achando uma função miníma apenas para esses critérios.

Em Ferraz *et al.* (2018) foi desenvolvido o programa denominado *Majority Primitives Combination* (MPC) que funciona de forma otimizada para até quatro variáveis de entrada, mas também faz a minimização para até cinco variáveis. O programa recebe como entrada uma tabela verdade e retorna como saída uma função majoritária otimizada que cobre o mesmo conjunto de mintermos da tabela verdade de entrada.

A geração de uma função otimizada é realizada através da combinação de funções previamente otimizadas. A lista de funções previamente otimizadas é obtida a partir da combinação de até três funções primitivas.

Em Chu *et al.* (2018) foi proposta uma forma de decomposição de funções booleanas complexas em subfunções de tamanho reduzido. Para realizar a decomposição o algoritmo combina diferentes formas de decomposição baseados em *Disjoint-support decomposition* (DSD) (BERTACCO; DAMIANI, 1997) na lógica majoritária e na teoria de expansão de Shannon (SHANNON, 1949).

Destaca-se que esse algoritmo possui como entrada uma função booleana representada por sua tabela verdade. Como saída é gerado um *XOR-Majority Graph* (XMG) que representa a função de entrada decomposta.

Um *XOR-Majority Graph* é uma extensão do *Majority-Inverter Graph* em que é adicionado o operador XOR.

1.1 OBJETIVO

O objetivo deste trabalho é implementar um método de minimização de funções booleanas empregando a lógica majoritária tendo como base as características do mapa de Karnaugh para 4 variáveis de entrada e comparar os resultados obtidos com os resultados apresentados pelo programa de minimização *exact_mig* (SOEKEN *et al.*, 2017). Como critério de custo, foram adotados os critérios propostos por Kong, Shang e Lu (2010), apresentados no Capítulo 3. Também foram incorporados alguns dos critérios adotados pelo *exact_mig* que foram abordados por Soeken apresentados no Capítulo 4.

Assim foi implementado o programa denominado MK4, que possui como base uma tabela de 90 funções primitivas de 4 variáveis, em que cada função possui no máximo uma porta majoritária e cobre um conjunto específico de mintermos. As combinações dessas 90 funções primitivas conseguem gerar qualquer função majoritária de 4 variáveis de entrada.

1.2 ESTRUTURA DO TRABALHO

No Capítulo 2 são apresentados os principais conceitos da álgebra booleana e as portas lógicas. Também é abordada a minimização de funções booleanas por meio do mapa de Karnaugh.

No Capítulo 3 apresentam-se as propriedades da lógica majoritária, da porta majoritária de três entradas e uma forma gráfica para representação de funções majoritárias por meio dos *Majority Inverter Graphs*.

No Capítulo 4 é apresentado o programa *exact_mig*.

No Capítulo 5 é apresentado em detalhes o método do mapa de Karnaugh para geração de funções majoritárias.

No Capítulo 6 apresenta-se o programa desenvolvido, os resultados obtidos e a comparação com os resultados gerados pelo programa *exact_mig*.

No Capítulo 7 tem-se as conclusões e as sugestões para trabalhos futuros.

2 ÁLGEBRA BOOLEANA

Embora este trabalho tenha como foco a lógica majoritária, neste capítulo são apresentados os principais conceitos da álgebra booleana tendo como intuito prover uma base teórica para o entendimento da lógica majoritária abordada no Capítulo 3.

Os conceitos da álgebra booleana, propostos em Boole (1854), podem ser utilizados para representar circuitos digitais.

No domínio dos números binários booleanos $(\mathbb{B},.,+,\overline{\ })$, o símbolo $\mathbb{B}$ representa o conjunto dos números binários $\{0,1\}$, o símbolo "." (multiplicação) representa o operador de conjunção (E) e o símbolo "+" (adição) representa o operador de disjunção (OU). O símbolo "$\overline{\ }$" representa a complementação ou inversão (NÃO). Quando utilizada a lógica positiva, o valor binário 0 representa o valor lógico falso e o valor binário 1 representa o valor lógico verdadeiro (AMARÚ; GAILLARDON; MICHELI, 2014).

2.1 PRINCIPAIS OPERADORES BOOLEANOS

Qualquer circuito lógico pode ser gerado utilizando a álgebra booleana. Para isso são utilizados os operadores booleanos E, OU e $N\tilde{A}O$.

A conjunção booleana corresponde a uma porta lógica E, cuja saída é igual a 1 se todas as variáveis de entrada possuírem valor lógico verdadeiro. Se uma ou mais variáveis possuírem o valor lógico falso, a saída assume o valor lógico falso. Na Equação 1 apresenta-se a função de uma porta E, em que se lê A e B.

$$F(A,B) = A.B \tag{1}$$

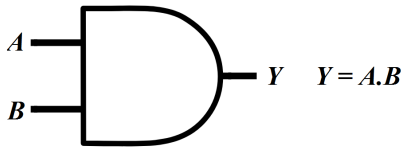
Na Tabela 1 apresenta-se a tabela verdade da porta lógica E e na Figura 2 apresenta-se o símbolo lógico e a expressão de uma porta E.

Tabela 1 – Tabela verdade de uma porta E.

A	***B***	**Função**	**Saída**	**Descrição**
0	0	0.0	0	Falso
0	1	0.1	0	Falso
1	0	1.0	0	Falso
1	1	1.1	1	Verdadeiro

Fonte: Dados do próprio autor.

Figura 2 – Símbolo representativo e a expressão de uma porta E.



Fonte: Dados do próprio autor.

A disjunção booleana, também chamada de soma lógica, corresponde à porta lógica OU, que possui saída lógica igual a 1 se uma ou mais variáveis de entrada possuírem valor lógico verdadeiro. Na Equação 2 apresenta-se a função de uma porta OU em que se lê A ou B.

$$F(A, B) = A + B \tag{2}$$

Na Tabela 2 apresenta-se a tabela verdade de uma porta lógica OU.

Tabela 2 – Tabela verdade de uma porta OU.

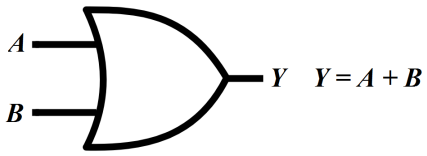
A	***B***	**Função**	**Saída**	**Descrição**
0	0	0+0	0	Falso
0	1	0+1	1	Verdadeiro
1	0	1+0	1	Verdadeiro
1	1	1+1	1	Verdadeiro

Fonte: Dados do próprio autor.

Na Figura 3 representa-se o símbolo lógico e a expressão de uma porta OU.

A complementação booleana corresponde à porta $N\tilde{A}O$ e tem como resultado o complemento de determinada variável ou expressão. Tem-se como exemplo uma variável A. Se $A = 1$ então $\overline{A}$ (lê-se A negado ou A barrado) $= 0$. Na Tabela 3 apresenta-se o comportamento de uma porta $N\tilde{A}O$.

Figura 3 – Símbolo representativo e a expressão de uma porta *OU*.



Fonte: Dados do próprio autor.

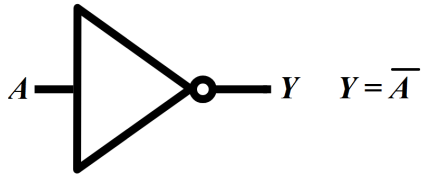
Tabela 3 – Tabela verdade de uma porta *NÃO*.

A	**Função**	**Saída**	**Descrição**
0	$\overline{0}$	1	Verdadeiro
1	$\overline{1}$	0	Falso

Fonte: Dados do próprio autor.

Na Figura 4 apresenta-se o símbolo lógico e a expressão de uma porta *NÃO*.

Figura 4 – Símbolo representativo e a expressão de uma porta *NÃO*.



Fonte: Dados do próprio autor.

É possível combinar as portas *Es* ou portas *OUs* com a porta *NÃO* para gerar os operadores *NÃO E* e *NÃO OU*. A porta *NÃO E* possui saída verdadeira se pelo menos uma das entradas possuir valor lógico falso. Na Equação 3 representa-se a função de uma porta *NÃO E*.

$$F(A, B) = \overline{A.B} \tag{3}$$

Na Tabela 4 apresenta-se o comportamento de uma porta *NÃO E*.

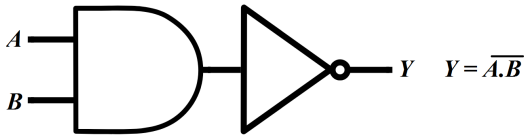
Tabela 4 – Tabela verdade de uma porta *NÃO E*.

A	***B***	**Função**	**Saída**	**Descrição**
0	0	$\overline{0.0}$	1	Verdadeiro
0	1	$\overline{0.1}$	1	Verdadeiro
1	0	$\overline{1.0}$	1	Verdadeiro
1	1	$\overline{1.1}$	0	Falso

Fonte: Dados do próprio autor.

Na Figura 5 apresenta-se o símbolo lógico e a expressão de uma porta *NÃO E*.

Figura 5 – Símbolo representativo de uma porta *NÃO E*.



Fonte: Dados do próprio autor.

A porta *NÃO OU*, representada pela Equação 4, possui saída verdadeira se, e somente se, todas as entradas possuírem valor lógico falso.

$$F(A, B) = \overline{A + B} \tag{4}$$

Na Tabela 5 apresenta-se o comportamento de uma porta *NÃO OU*.

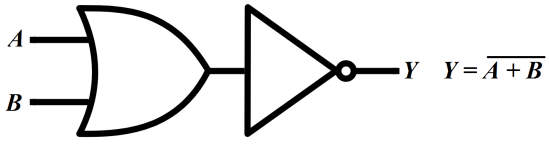
Tabela 5 – Tabela verdade de uma porta *NÃO OU*.

A	***B***	**Função**	**Saída**	**Descrição**
0	0	$\overline{0+0}$	1	Verdadeiro
0	1	$\overline{0+1}$	0	Falso
1	0	$\overline{1+0}$	0	Falso
1	1	$\overline{1+1}$	0	Falso

Fonte: Dados do próprio autor.

Na Figura 6 apresenta-se o símbolo lógico e a expressão de uma porta *NÃO OU*.

Figura 6 – Símbolo representativo de uma porta *NÃO OU*.



Fonte: Dados do próprio autor.

2.2 POSTULADOS E AXIOMAS DA ÁLGEBRA BOOLEANA

A álgebra booleana é compreendida por postulados e axiomas que permitem a manipulação e simplificação de funções booleanas.

2.2.1 Postulados

O primeiro postulado é definido pela Equação 5. Por se tratar de uma operação com uma porta *OU* (disjunção), se a variável A assumir valor lógico verdadeiro o resultado obtido será verdadeiro. Por outro lado, se A assumir o valor lógico falso, o resultado será falso.

$$A + 0 = A \tag{5}$$

Na Tabela 6 apresenta-se a tabela verdade para a função $F(A) = A + 0$.

Tabela 6 – Tabela verdade da função $F(A) = A + 0$.

A	**Função**	**Saída**	**Descrição**
0	0+0	0	Falso
1	1+0	1	Verdadeiro

Fonte: Dados do próprio autor.

O segundo postulado é definido pela Equação 6. Por se tratar de uma operação com uma porta *OU*, independentemente do valor lógico de A, o resultado sempre será verdadeiro.

$$A + 1 = 1 \tag{6}$$

Na Tabela 7 apresenta-se a tabela verdade para a função $F(A) = A + 1$.

Tabela 7 – Tabela verdade da função $F(A) = A + 1$.

A	**Função**	**Saída**	**Descrição**
0	0+1	1	Verdadeiro
1	1+1	1	Verdadeiro

Fonte: Dados do próprio autor.

O terceiro postulado é definido pela Equação 7. Por se tratar de uma operação com uma porta *OU* entre duas variáveis iguais, o resultado terá o mesmo valor lógico da variável de entrada.

$$A + A = A \tag{7}$$

Na Tabela 8 apresenta-se a tabela verdade para a função $F(A) = A + A$.

O quarto postulado é definido pela Equação 8. Nessa situação, a função terá pelo menos um termo com valor lógico verdadeiro, logo sempre será obtida uma saída com valor lógico verdadeiro.

$$A + \overline{A} = 1 \tag{8}$$

Tabela 8 – Tabela verdade da função $F(A) = A + A$.

A	Função	Saída	Descrição
0	0+0	0	Falso
1	1+1	1	Verdadeiro

Fonte: Dados do próprio autor.

Na Tabela 9 apresenta-se a tabela verdade para a função $F(A) = A + \overline{A}$.

Tabela 9 – Tabela verdade da função $F(A) = A + \overline{A}$.

A	Função	Saída	Descrição
0	0+1	1	Verdadeiro
1	1+0	1	Verdadeiro

Fonte: Dados do próprio autor.

O quinto postulado é definido pela Equação 9. Por ser uma operação com uma porta E (conjunção) independente do valor lógico assumido por A, o resultado sempre será uma saída com o valor lógico falso.

$$A.0 = 0 \tag{9}$$

Na Tabela 10 apresenta-se a tabela verdade para a função $F(A) = A.0$.

Tabela 10 – Tabela verdade da função $F(A) = A.0$

A	Função	Saída	Descrição
0	0.0	0	Falso
1	1.0	0	Falso

Fonte: Dados do próprio autor.

O sexto postulado é definido pela Equação 10. Por ser uma operação com a porta E, o valor lógico assumido por A será o mesmo resultado obtido na saída.

$$A.1 = A \tag{10}$$

Na Tabela 11 apresenta-se a tabela verdade para a função $F(A) = A.1$.

O sétimo postulado é definido pela Equação 11. Por se tratar de uma operação com uma porta E entre duas variáveis iguais, o resultado terá o mesmo valor lógico da variável de entrada.

$$A.A = A \tag{11}$$

Na Tabela 12 apresenta-se a tabela verdade para a função $F(A) = A.A$.

Tabela 11 – Tabela verdade da função $F(A) = A.1$

A	**Função**	**Saída**	**Descrição**
0	0.1	0	Falso
1	1.1	1	Verdadeiro

Fonte: Dados do próprio autor.

Tabela 12 – Tabela verdade da função $F(A) = A.A$

A	**Função**	**Saída**	**Descrição**
0	0.0	0	Falso
1	1.1	1	Verdadeiro

Fonte: Dados do próprio autor.

O oitavo postulado é definido pela Equação 12. Nessa equação, pelo menos um dos termos conterá um valor lógico falso. Por se tratar de uma porta *E*, o resultado sempre será uma saída com o valor lógico falso.

$$A.\overline{A} = 0 \tag{12}$$

Na Tabela 13 apresenta-se a tabela verdade para a função $F(A) = A.\overline{A}$.

Tabela 13 – Tabela verdade da função $F(A) = A.\overline{A}$

A	**Função**	**Saída**	**Descrição**
0	0.1	0	Falso
1	1.0	0	Falso

Fonte: Dados do próprio autor.

O nono postulado é definido pela Equação 13. A negação de uma negação sempre trará o resultado da função original.

$$\overline{\overline{A}} = A \tag{13}$$

Tem-se como exemplo a Tabela 14 onde apresenta-se a tabela verdade para a função $F(A) = \overline{\overline{A}}$.

Tabela 14 – Tabela verdade da função $F(A) = \overline{\overline{A}}$

A	**Função**	**Saída**	**Descrição**
0	$\overline{\overline{0}}$	0	Falso
1	$\overline{\overline{1}}$	1	Verdadeiro

Fonte: Dados do próprio autor.

2.2.2 Axiomas da álgebra booleana

Os três axiomas da álgebra booleana são: associatividade, comutatividade e distribuição.

O axioma da associatividade define que a forma de agrupamento adotada em uma função não altera o resultado. Tem-se como exemplo as funções:

- $F1(A, B, C) = A.(B.C)$;
- $F2(A, B, C) = B.(C.A)$;
- $F3(A, B, C) = A + (C + B)$;
- $F4(A, B, C) = C + (B + A)$.

Assim $F1 = F2$, ou seja $A.(B.C) = B.(C.A)$ e $F3 = F4$, ou seja $A + (C + B) = C + (B + A)$.

O axioma da comutatividade define que a ordem das variáveis em uma expressão não altera o resultado da operação. Tem-se como exemplo as funções:

- $F1(A, B) = A + B$;
- $F2(A, B) = B + A$;
- $F3(A, B) = A.B$;
- $F4(A, B) = B.A$

Pode-se afirmar que $F1 = F2$, ou seja $A + B = B + A$, e que $F3 = F4$ ou seja $A.B = B.A$.

O axioma da distribuição define que uma expressão pode ser expandida ou reduzida, fatorando uma expressão que contenha uma ou mais variáveis que se repetem. Tem-se como exemplo a função $F(A, B, C)$ representada pela Equação 14,

$$F(A, B, C) = A.(B + C) \tag{14}$$

Pode-se expandir a função $F(A, B, C)$ distribuindo a variável A gerando assim a função $F1(A, B, C)$ representada na Equação 15.

$$F1(A, B, C) = A.B + A.C \tag{15}$$

O inverso também é possível a partir de $F1$. Como a variável A é um literal

comum em ambos os termos, pode-se colocá-la em evidência gerando a função $F(A, B, C)$ representada na Equação 14.

2.2.3 Teoremas de De Morgan

Os teoremas de De Morgan são essenciais para o entendimento da álgebra booleana e para o entendimento de diversos métodos de minimização de funções booleanas.

O primeiro teorema define que o resultado da complementação de um produto lógico de variáveis é o mesmo resultado da soma das variáveis negadas do mesmo produto.

Tem-se como exemplo a função $F(A, B, C) = \overline{A.B.C}$. Aplicando o primeiro teorema de De Morgan, obtém-se a função $F1(A, B, C) = \overline{A}+\overline{B}+\overline{C}$, sendo F e $F1$ equivalentes.

Apresenta-se na Tabela 15 a tabela verdade para as funções F e $F1$.

Tabela 15 – Tabela verdade para as funções F e $F1$.

$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{C}$	$\boldsymbol{F = \overline{A.B.C}}$	**Saída**	**F1=** $\boldsymbol{\overline{A}+\overline{B}+\overline{C}}$	**Saída**	**Descrição**
0	0	0	$\overline{0.0.0}$	1	$\overline{0}+\overline{0}+\overline{0}$	1	Verdadeiro
0	0	1	$\overline{0.0.1}$	1	$\overline{0}+\overline{0}+\overline{1}$	1	Verdadeiro
0	1	0	$\overline{0.1.0}$	1	$\overline{0}+\overline{1}+\overline{0}$	1	Verdadeiro
0	1	1	$\overline{0.1.1}$	1	$\overline{0}+\overline{1}+\overline{1}$	1	Verdadeiro
1	0	0	$\overline{1.0.0}$	1	$\overline{1}+\overline{0}+\overline{0}$	1	Verdadeiro
1	0	1	$\overline{1.0.1}$	1	$\overline{1}+\overline{0}+\overline{1}$	1	Verdadeiro
1	1	0	$\overline{1.1.0}$	1	$\overline{1}+\overline{1}+\overline{0}$	1	Verdadeiro
1	1	1	$\overline{1.1.1}$	0	$\overline{1}+\overline{1}+\overline{1}$	0	Falso

Fonte: Dados do próprio autor.

O segundo teorema define que a complementação de uma soma lógica de variáveis é equivalente ao produto das variáveis negadas dessa soma. Como exemplo tem-se a função $F(A, B, C) = \overline{A + B + C}$. Aplicando o segundo teorema, obtém-se a função $F1(A, B, C) = \overline{A}.\overline{B}.\overline{C}$, sendo F e $F1$ equivalentes.

2.3 SIMPLIFICAÇÃO DE EXPRESSÕES BOOLEANAS

Em circuitos digitais é de suma importância que as expressões lógicas estejam em sua forma mínima para reduzir o custo de implementação do circuito. Define-se como custo o número de portas lógicas e o número de entradas de um circuito.

Dada uma expressão lógica, é possível minimizá-la de forma que o número de variáveis e de portas lógicas sejam diminuídos, gerando uma nova expressão de menor custo, mas equivalente à original.

Alguns dos métodos de minimização mais conhecidos são o método de manipulação algébrica e o método do mapa de Karnaugh. Como neste trabalho implementou-se um programa que tem como base o mapa de Karnaugh, este método é abordado em detalhes na próxima seção.

2.4 MAPA DE KARNAUGH

Um dos mais conhecidos métodos de minimização é o mapa de Karnaugh (mapa-K). Trata-se de um método gráfico criado em 1953 por Maurice Karnaugh, utilizado para simplificar expressões lógicas ou converter tabelas verdade em circuitos lógicos equivalentes (KARNAUGH, 1953).

Um mapa-K pode ser usado para simplificar expressões com qualquer número de variáveis de entrada. Porém, quanto maior o número de variáveis, mais complexa torna-se a resolução. Sendo assim, é recomendado que se utilize este método somente com seis ou menos variáveis de entrada (WIDMER; MOSS; TOCCI, 2017).

Assim como uma tabela verdade, um mapa-K é formado por 2^n células, sendo n o número de variáveis de entrada da função. Cada uma das células pode representar um mintermo, um maxtermo ou um *don't care state* da função.

Um mintermo é um termo produto contendo todas as n variáveis na sua forma complementar ou não e representa uma linha da tabela verdade. Em um mintermo os termos que apresentam um valor lógico igual a 1 são representados por uma variável em sua forma não complementar. Do mesmo modo, variáveis que apresentam um valor lógico igual a 0 são representadas pela sua forma complementar (KOHAVI; JHA, 2009).

Tem-se como exemplo uma função $F(A, B, C)$. Algumas linhas da tabela verdade dessa função podem ser representadas pelos binários 111, 101, 010:

- O binário 111 é representado pelo mintermo $A.B.C$ também denominado como mintermo 7;
- O binário 101 é representado pelo mintermo $A.\overline{B}.C$ também denominado como mintermo 5;
- O binário 010 é representado pelo mintermo $\overline{A}.B.\overline{C}$ também denominado como

mintermo 2.

Uma função booleana na forma de soma de produtos também pode ser escrita como um somatório dos mintermos que possuem saída lógica igual a 1 em sua tabela verdade (KOHAVI; JHA, 2009). Tem-se como exemplo a tabela verdade da função $N\tilde{A}O$ E representada na Tabela 4. A função $N\tilde{A}O$ E pode ser escrita como representado pela Equação 16 ou como Σm(0,1,2).

$$F(A, B) = \overline{A}.\overline{B} + \overline{A}.B + A.\overline{B} \tag{16}$$

Um maxtermo é um termo soma contendo todas as n variáveis na sua forma complementar ou não e representa uma linha da tabela verdade. Diferentemente de um mintermo, em um maxtermo os termos que apresentam um valor lógico igual a 0 são representandos por uma variável em sua forma não complementar. Do mesmo modo, variáveis que apresentam um valor lógico igual a 1 são representadas pela sua forma complementar (KOHAVI; JHA, 2009).

Tem-se como exemplo uma função $F(A, B, C)$. Algumas linhas da tabela verdade dessa função podem ser representadas pelos binários 111, 101, 010:

- O binário 111 é representado pelo maxtermo $\overline{A} + \overline{B} + \overline{C}$ também denominado como maxtermo 7;
- O binário 101 é representado pelo maxtermo $\overline{A} + B + \overline{C}$ também denominado como maxtermo 5;
- O binário 010 é representado pelo maxtermo $A + \overline{B} + C$ também denominado como maxtermo 2.

Uma função booleana na forma produto de somas também pode ser escrita como um produto dos maxtermos que possuem saída lógica igual a 0 em sua tabela verdade (KOHAVI; JHA, 2009). Tem-se como exemplo a tabela verdade da função $N\tilde{A}O$ OU representada na Tabela 5. A função $N\tilde{A}O$ OU pode ser escrita como representado pela Equação 17 ou como ΠM(1,2,3).

$$F(A, B) = (A + \overline{B}).(\overline{A} + B).(\overline{A} + \overline{B}) \tag{17}$$

Na Tabela 16 apresenta-se a correspondência entre as linhas de uma tabela verdade e seus respectivos mintermos e maxtermos para uma função em que $n = 3$.

Tabela 16 – Tabela de correspondência entre linhas de uma tabela verdade com as células de um mapa-K.

Nº linha	**Entrada**	**Mintermo**	**Maxtermo**
0	000	$\overline{A}.\overline{B}.\overline{C}$	$A+B+C$
1	001	$\overline{A}.\overline{B}.C$	$A+B+\overline{C}$
2	010	$\overline{A}.B.\overline{C}$	$A+\overline{B}+C$
3	011	$\overline{A}.B.C$	$A+\overline{B}+\overline{C}$
4	100	$A.\overline{B}.\overline{C}$	$\overline{A}+B+C$
5	101	$A.\overline{B}.C$	$\overline{A}+B+\overline{C}$
6	110	$A.B.\overline{C}$	$\overline{A}+\overline{B}+C$
7	111	$A.B.C$	$\overline{A}+\overline{B}+\overline{C}$

Fonte: Dados do próprio autor.

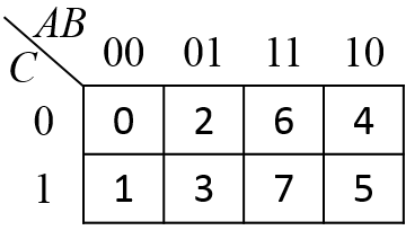
Por conveniência, neste trabalho o termo mintermo é utilizado para referenciar linhas da tabela verdade que possuem saída lógica igual a 1. Utiliza-se o termo maxtermo para referenciar as linhas da tabela verdade que tenham saída lógica igual a 0.

Em alguns circuitos lógicos pode haver condições em que determinados valores de uma função em uma tabela verdade nunca venham a ocorrer. Assim, não importa se a saída terá um valor lógico igual a 0 ou 1. Essas células são denominadas *don't care state* e são representadas com um "x" (WIDMER; MOSS; TOCCI, 2017).

Ao desenhar um mapa-K cuida-se para que as células adjacentes diferenciem-se apenas em uma variável, e deve-se levar em consideração que as células superiores são adjacentes as inferiores, assim como as células laterais esquerdas são adjacentes as células laterais direitas.

Na Figura 7 apresenta-se a disposição das células em um mapa-K de 3 variáveis. Nota-se que a variável A representa o *bit* mais significativo e a variável C representa o *bit* menos significativo.

Figura 7 – Exemplo de disposição das células em um mapa-K.



Fonte: Dados do próprio autor.

Os números das células representam as linhas da tabela verdade e estão dispostas de forma que uma célula diferencia-se de outra célula adjacente em apenas uma variável.

2.4.1 Conversão de uma tabela verdade em um Mapa-K

Para se desenhar um mapa-K de uma determinada função, primeiro deve-se conhecer sua tabela verdade. Tem-se como exemplo a função F representada na Equação 18.

$$F(A, B, C) = \overline{A}.C + \overline{A}.B + A\overline{B}.C + B.C \tag{18}$$

Como F possui três variáveis de entrada, a tabela verdade dessa função possui 2^3 linhas. Na Tabela 17 apresenta-se os oito possíveis resultados para F.

Tabela 17 – Tabela verdade da função $F(A, B, C) = \overline{A}$.C + $\overline{A}$.B + A.$\overline{B}$.C + B.C.

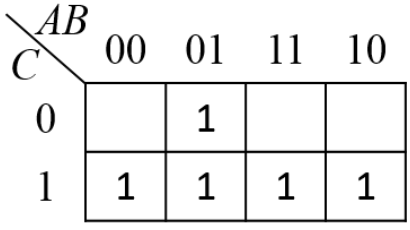
A	B	C	$F(A, B, C) = \overline{A}.C + \overline{A}.B + A.\overline{B}.C + B.C$	Saída	Descrição
0	0	0	$\overline{0}.0 + \overline{0}.0 + 0.\overline{0}.0 + 0.0$	0	Falso
0	0	1	$\overline{0}.1 + \overline{0}.0 + 0.\overline{0}.1 + 0.1$	1	Verdadeiro
0	1	0	$\overline{0}.0 + \overline{0}.1 + 0.\overline{1}.0 + 1.0$	1	Verdadeiro
0	1	1	$\overline{0}.1 + \overline{0}.1 + 0.\overline{1}.1 + 1.1$	1	Verdadeiro
1	0	0	$\overline{1}.0 + \overline{1}.0 + 1.\overline{0}.0 + 0.0$	0	Falso
1	0	1	$\overline{1}.1 + \overline{1}.0 + 1.\overline{0}.1 + 0.1$	1	Verdadeiro
1	1	0	$\overline{1}.0 + \overline{1}.1 + 1.\overline{1}.0 + 1.0$	0	Falso
1	1	1	$\overline{1}.1 + \overline{1}.1 + 1.\overline{1}.1 + 1.1$	1	Verdadeiro

Fonte: Dados do próprio autor.

A partir da tabela verdade monta-se o mapa-K colocando as saídas em suas respectivas células.

Na Figura 8 apresenta-se o mapa-K da função F, onde os mintermos são representados pelas células com digito 1 e os maxtermos são representados pelas células em branco.

Figura 8 – Mapa-K da função $F(A, B, C) = \overline{A}.C + \overline{A}.B + A.\overline{B}.C + B.C$



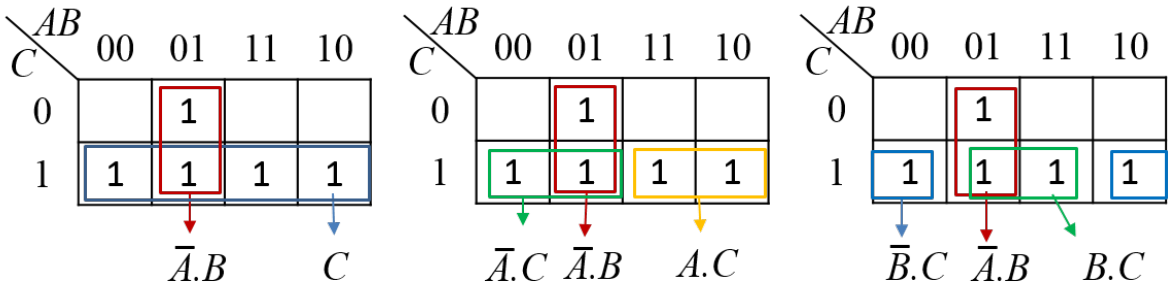
Fonte: Dados do próprio autor.

2.4.2 Minimização de uma função utilizando um Mapa-K

Para minimizar uma função utilizando o mapa-K, deve-se tentar agrupar mintermos adjacentes visando agrupar a maior quantidade de células possíveis respeitando a restrição de $2^0, 2^1, 2^2$, 2^3... 2^n células adjacentes. Cada agrupamento é chamado de implicante.

Após identificar os implicantes, é feita a soma de produtos entre eles. Se a escolha dos implicantes for efetuada de maneira otimizada, a função resultante estará na forma ótima, também chamada de função mínima, ou seja, com o mínimo uso de portas lógicas e de entradas. Na Figura 9 apresentam-se três exemplos de agrupamentos para o mapa-K apresentado na Figura 8.

Figura 9 – Exemplos de agrupamentos de implicantes.



(a) $\overline{A}.B + C$ (b) $\overline{A}.B + \overline{A}.C + A.C$ (c) $\overline{A}.B + \overline{B}.C + B.C$

Fonte: Dados do próprio autor.

Na Figura 9(a) apresenta-se a melhor escolha de implicantes, gerando a função $F1$ representada pela Equação 19, que é reduzida e equivalente à F apresentada na Figura 8.

$$F1(A, B, C) = \overline{A}.B + C \tag{19}$$

No agrupamento apresentado na Figura 9(b), obteve-se a função $F2$ representada pela Equação 20, que é reduzida e equivalente à F, mas não está na forma mínima.

$$F2(A, B, C) = \overline{A}.B + \overline{A}.C + A.C \tag{20}$$

Na Figura 9(c) apresenta-se a função $F3$ representada pela Equação 21, que é reduzida e equivalente à função F. Porém, não se encontra em sua forma mínima.

$$F3(A, B, C) = \overline{A}.B + \overline{B}.C + B.C \tag{21}$$

Observando o implicante $\overline{B}$.C apresentado na Figura 9(c), nota-se o uso da propriedade do mapa-K que define que as células a esquerda são adjacentes as células a direita.

2.4.3 Utilização de *don't care states* em um mapa-K

No mapa-K uma condição *don't care* é representada por um "x" e o projetista é livre para decidir se a saída terá valor lógico 0 ou 1. Esta propriedade pode ser útil para minimizar uma função (WIDMER; MOSS; TOCCI, 2017). Para exemplificar o uso dos *don't care states*, tem-se a tabela verdade apresentada na Tabela 18.

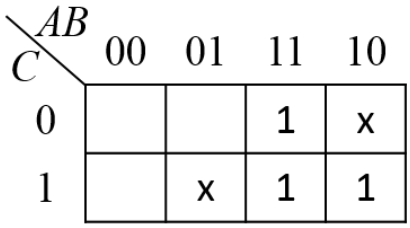
Tabela 18 – Tabela verdade com condições *don't care*.

A	***B***	***C***	**Saída**	**Descrição**
0	0	0	0	Falso
0	0	1	0	Falso
0	1	0	0	Falso
0	1	1	x	*don't care*
1	0	0	x	*don't care*
1	0	1	1	Verdadeiro
1	1	0	1	Verdadeiro
1	1	1	1	Verdadeiro

Fonte: Dados do próprio autor.

Na Figura 10 apresenta-se o mapa-K da tabela verdade apresentada na Tabela 18.

Figura 10 – Exemplo de *don't care states* em um mapa-K.



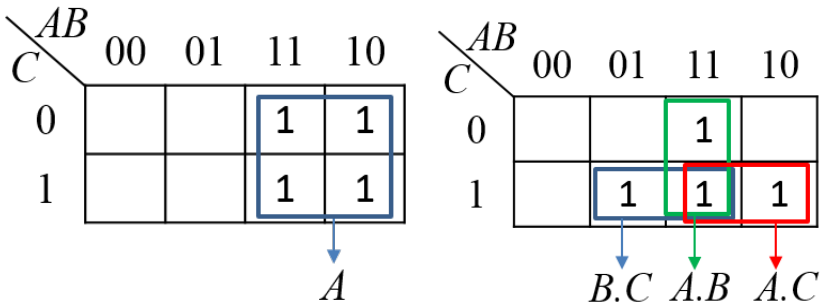
Fonte: Dados do próprio autor.

Na Figura 11(a) apresenta-se o melhor uso dos *don't care states* gerando apenas um implicante e a função F representada na Equação 22. Já na Figura 11(b) apresenta-se o pior uso, gerando três implicantes e a função $F1$ representada na Equação 23.

$$F(A, B, C) = A \tag{22}$$

$$F1(A, B, C) = A.B + A.C + B.C. \tag{23}$$

Figura 11 – Duas possíveis configurações para o mapa-K da Figura 10.

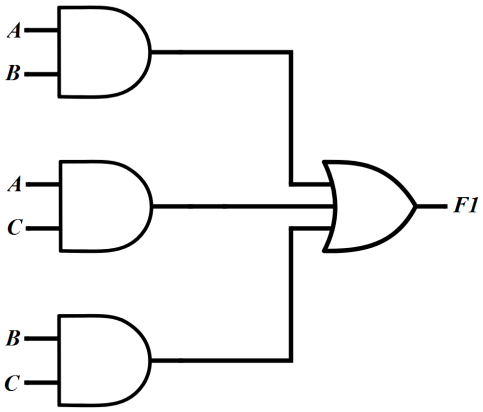


(a) Melhor configuração. (b) Pior configuração.

Fonte: Dados do próprio autor.

Apresenta-se na Figura 12 o circuito gerado pela função $F1$.

Figura 12 – Circuito gerado pela função $F1(A, B, C)$ = A.B + A.C + B.C.



Fonte: Dados do próprio autor.

Como apresentado, a aplicação dos *don't care states* pode reduzir consideravelmente o uso de portas lógicas de um circuito. O circuito apresentado na Figura 12 utiliza três portas E e uma porta OU. Se a aplicação dos *don't care states* for realizada de forma correta, o circuito resultante precisaria apenas de uma entrada A e nenhuma porta lógica.

2.5 CLASSIFICAÇÃO DE FUNÇÕES BOOLEANAS

No conjunto de funções booleanas, há funções com características específicas que são classificadas em subconjuntos, podendo ser: monotônicas, *unate*, *threshold*, *self-dual* e *majority* (AMARU, 2017).

Existem dois tipos de funções monotônicas: funções monotônicas positivas (crescentes) e funções monotônicas negativas (decrescentes).

Funções monotônicas positivas são funções encontradas na forma de soma de produtos e são representadas apenas por portas lógicas E e portas OU sem o uso de variáveis na forma complementar (SASAO, 2012). Na Equação 24 apresenta-se um exemplo de função monotônica positiva.

$$F(A, B, C) = A.B.C + A.B \tag{24}$$

As funções monotônicas negativas são funções encontradas na forma soma de produtos, mas somente com variáveis na forma complementar (SASAO, 2012). Na Equação 25 apresenta-se um exemplo de função monotônica negativa.

$$F(A, B, C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B} \tag{25}$$

O grupo das funções *unate* é o grupo generalização das funções monotônicas positivas e negativas. Uma função *unate* é representada na forma de soma de produtos, em que cada variável pode aparecer somente em sua forma complementar ou somente em sua forma não complementar (AMARU, 2017). Por exemplo, a função $F(A, B, C, D) = A.B + C.\overline{D}$, é uma função *unate* pois nenhuma variável aparece na forma complementar e não complementar ao mesmo tempo. Contudo, a função $F1(A, B, C, D) = A.B + \overline{B}.D$ contém a variável B na forma não complementar e na forma complementar, portanto $F1(A, B, C, D)$ não é uma função *unate*.

Uma função *threshold* pondera cada uma das entradas utilizando pesos específicos, e o resultado é verdadeiro se, e somente se, o somatório das entradas ponderadas for maior ou igual a uma marca *threshold* t, sendo t um número real. Funções *threshold* são importantes na área de inteligência artificial, sendo utilizadas como modelos de neurônios artificiais (SASAO, 2012).

Nas funções *self-dual* a complementação da saída é a mesma que a complementação das entradas. Por exemplo, a função $F(\overline{A}, \overline{B}, \overline{C})$ é equivalente à $\overline{F}(A, B, C)$.

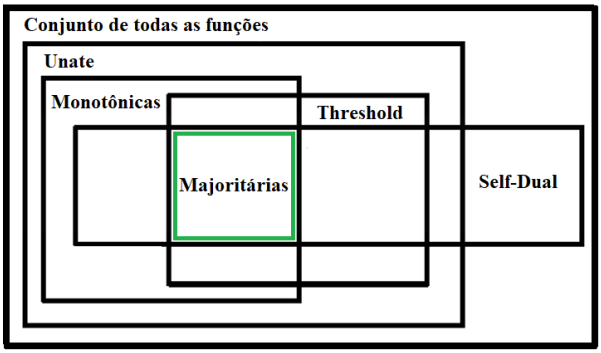
Uma função *majority*, também conhecida como função majoritária, possui uma saída verdadeira somente quando a maioria das entradas são verdadeiras. Dessa forma, uma função majoritária com n entradas, n sendo ímpar e maior ou igual a 3, terá a saída verdadeira se, e somente se, pelo menos $\lceil \frac{n}{2} \rceil$ entradas possuírem o valor lógico verdadeiro (AMARU *et al.*, 2016).

Tem-se como exemplo uma função onde $n = 3$, assim, $\lceil \frac{3}{2} \rceil = 2$, portanto, a função

terá saída verdadeira se, e somente se, o número de entradas com valor lógico verdadeiro for maior ou igual 2.

Na Figura 13 apresenta-se a interação entre os conjuntos de funções monotônicas, *unate*, *threshold*, *self-dual* e majoritárias.

Figura 13 – Interação entre os conjuntos de funções booleanas.



Fonte: Adaptado de Amaru (2017).

Observando a Figura 13 nota-se que:

- Todas as funções monotônicas, *threshold* e majoritárias são funções *unate*;
- Todas as funções majoritárias são funções *unate*, monotônicas, *threshold* e *self-dual*.

As funções majoritárias são o foco deste trabalho e foram abordadas com mais detalhes no Capítulo 3, motivo pelo qual estão sinalizadas em verde na Figura 13.

Neste capítulo foram abordados os principais conceitos da álgebra booleana e o método do mapa de Karnaugh para reduzir o custo de um circuito.

No Capítulo 3 aborda-se a lógica majoritária e os *Majority Inverter Graphs* utilizados para representação gráfica de funções majoritárias.

3 LÓGICA MAJORITÁRIA

Neste capítulo são apresentados os principais conceitos e axiomas da lógica majoritária. Também é apresentada a utilização de portas majoritárias para obter portas lógicas E e portas OU. Apresenta-se também uma forma de representação de funções majoritárias chamada de MIG (*Majority Inverter Graphs*) e o custo de implementação de um circuito de uma função majoritária.

A álgebra booleana utilizando o conjunto ($\mathbb{B}$,M_3,$\overline{\ }$), sujeita ao conjunto de axiomas Ω (Maioria, Comutatividade, Associatividade, Distributiva e Inversão), é chamada de lógica majoritária. O M_3 representa o operador majoritário entre três variáveis e define que funções majoritárias com três entradas possuem saídas verdadeiras se, e somente se, pelo menos duas entradas possuírem valores lógicos verdadeiros (AMARU *et al.*, 2016).

Em circuitos digitais, o uso de funções majoritárias pode minimizar uma função, reduzindo o uso de portas lógicas. Tem-se como exemplo a função majoritária apresentada na Equação 26.

$$F(A, B, C) = A.B + A.C + B.C \tag{26}$$

Utilizando a lógica booleana clássica, é necessário utilizar três portas E e uma porta lógica OU para descrever esse circuito. Por outro lado, aplicando-se a lógica majoritária, uma única porta majoritária é o suficiente para implementar F. A função apresentada na Equação 26 é equivalente a função majoritária apresentada na Equação 27, em que se lê, porta majoritária de A, B e C.

$$M(A, B, C) \tag{27}$$

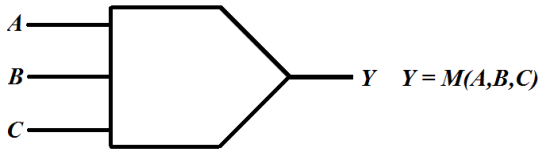
Na Tabela 19 apresenta-se a tabela verdade para uma porta majoritária $M(A, B, C)$ e na Figura 14 apresenta-se o símbolo lógico e a expressão de uma porta majoritária de três entradas.

Tabela 19 – Tabela verdade de uma porta majoritária de três entradas.

A	B	C	$M(A,B,C)$	Saída	Descrição
0	0	0	M(0,0,0)	0	Falso
0	0	1	M(0,0,1)	0	Falso
0	1	0	M(0,1,0)	0	Falso
0	1	1	M(0,1,1)	1	Verdadeiro
1	0	0	M(1,0,0)	0	Falso
1	0	1	M(1,0,1)	1	Verdadeiro
1	1	0	M(1,1,0)	1	Verdadeiro
1	1	1	M(1,1,1)	1	Verdadeiro

Fonte: Dados do próprio autor.

Figura 14 – Símbolo representativo e a expressão de uma porta majoritária de três entradas.



Fonte: Dados do próprio autor.

3.1 PORTA MAJORITÁRIA

Uma das principais características da porta majoritária é a de se comportar como portas E ou portas OU. Nas próximas seções essas características são descritas em detalhes.

3.1.1 Porta majoritária comportando-se com uma porta *E*

A porta majoritária de três entradas pode se comportar como uma porta E fixando-se uma de suas entradas em 0. A condição de maioria só é atingida caso as outras duas variáveis de entrada forem verdadeiras. Tem-se como exemplo a função $F(A,B) = A.B$, sendo $M(A,0,B)$ a representação da função em lógica majoritária, lendo-se maioria de A e B.

Nota-se que o valor fixo 0 está localizado no centro da porta majoritária. Neste trabalho, por convenção, as constantes sempre estão localizadas no meio da porta majoritária.

Na Tabela 20 apresenta-se a tabela verdade para a função E utilizando a porta

majoritária.

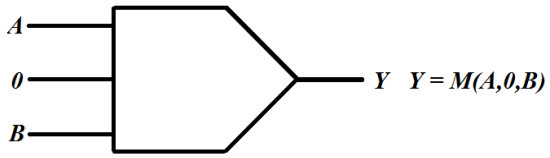
Tabela 20 – Tabela verdade de uma função E utilizando a porta majoritária.

A	B	$M(A,0,B)$	Saída	Descrição
0	0	M(0,0,0)	0	Falso
0	1	M(0,0,1)	0	Falso
1	0	M(1,0,0)	0	Falso
1	1	M(1,0,1)	1	Verdadeiro

Fonte: Dados do próprio autor.

Na Figura 15 apresenta-se o símbolo lógico e a expressão de uma porta majoritária de três entradas comportando-se como uma porta E.

Figura 15 – Símbolo representativo e a expressão de uma porta majoritária de três entradas comportando-se como uma porta E.



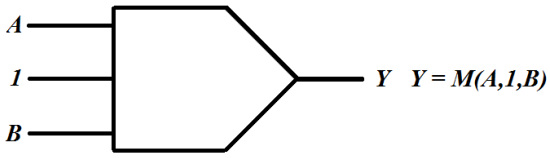
Fonte: Dados do próprio autor.

3.1.2 Porta majoritária comportando-se com uma porta *OU*

Fixando-se uma das entradas da porta majoritária em 1, tem-se uma porta OU. Assim a condição de maioria é atingida caso qualquer uma das demais variáveis de entrada possuírem valor lógico verdadeiro. Tem-se como exemplo a função $F(A,B) = A + B$. A representação da função F em lógica majoritária é $M(A,1,B)$, que é lida como maioria de A ou B.

Na Figura 16 apresenta-se o símbolo lógico e a expressão de uma porta majoritária de três entradas comportando-se como uma porta OU.

Figura 16 – Símbolo representativo e a expressão da porta majoritária de três entradas comportando-se como uma porta OU.



Fonte: Dados do próprio autor.

Na Tabela 21 apresenta-se a tabela verdade para a função *OU* utilizando uma porta majoritária.

Tabela 21 – Tabela verdade de uma função *OU* utilizando a porta majoritária.

A	B	$M(A, 1, B)$	Saída	Descrição
0	0	M(0,1,0)	0	Falso
0	1	M(0,1,1)	1	Verdadeiro
1	0	M(1,1,0)	1	Verdadeiro
1	1	M(1,1,1)	1	Verdadeiro

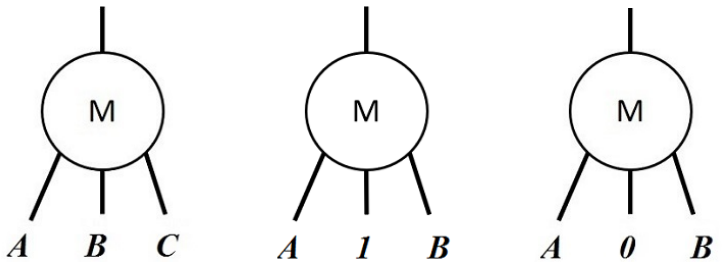
Fonte: Dados do próprio autor.

3.2 *MAJORITY INVERTER GRAPHS* - MIG

O *Majority Inverter Graph*, ou MIG, é uma representação gráfica de funções majoritárias proposta por Amarú, Gaillardon e Micheli (2014). O MIG possui o formato de árvore com três ramos (entradas), em que cada ramificação se conecta a uma variável ou a outro nó (porta majoritária) (AMARÚ; GAILLARDON; MICHELI, 2014).

Na Figura 17 apresentam-se, respectivamente, exemplos de MIGs para as funções $M(A, B, C)$, $M(A, 1, B)$ e $M(A, 0, B)$. Cada círculo representa uma porta majoritária. A linha superior representa a saída da porta majoritária e as linhas inferiores representam os ramos que se conectam nas entradas dessa porta.

Figura 17 – Exemplo de MIGs para diversas funções majoritárias.

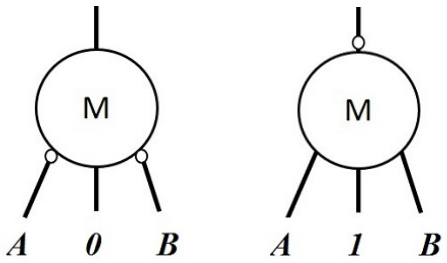


(a) $M(A, B, C)$ (b) $M(A, 1, B)$ (c) $M(A, 0, B)$

Fonte: Dados do próprio autor.

As funções majoritárias são funções *self-dual*, podendo-se transferir as negações das variáveis de entrada para a saída da função. Assim, em um MIG a negação pode ser representada de duas maneiras. Negando as variáveis de entrada ou negando a saída da função. Na Figura 18 apresentam-se as duas formas de representação de negação em um MIG.

Figura 18 – Exemplo de negações de funções em um MIG.



(a) $M(\overline{A}, 0, \overline{B})$ (b) $\overline{M}(A, 1, B)$

Fonte: Dados do próprio autor.

Quando uma função majoritária possui n nós, sendo n>1, é dito que a função possui mais de um nível. Tem-se como exemplo duas funções representadas pelas equações 28 e 29.

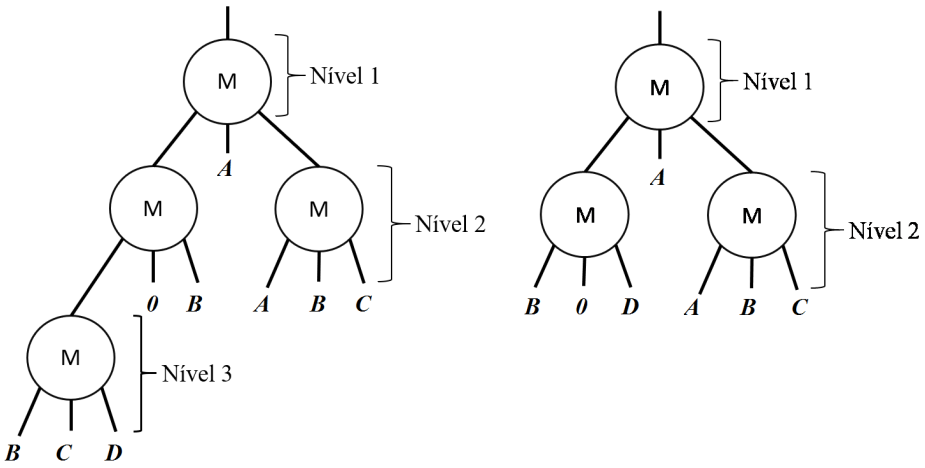
$$M(A, M(M(B, C, D), 0, B), M(A, B, C)) \tag{28}$$

$$M(A, M(B, 0, D), M(A, B, C)) \tag{29}$$

A função representada pela Equação 28 possui três níveis, enquanto a função representada pela Equação 29 possui dois. Um novo nível é identificado quando tem-se um nó com uma função majoritária conectado a outro nó.

Na Figura 19(a) apresentam-se o MIG da Equação 28 e na Figura 19(b) apresenta-se o MIG da Equação 29 destacando-se os níveis de cada uma.

Figura 19 – Exemplo de funções com mais de um nível em um MIG.



(a) MIG com três níveis (b) MIG com dois níveis

Fonte: Dados do próprio autor.

3.3 CONJUNTO DE AXIOMAS Ω DA LÓGICA MAJORITÁRIA

Na lógica majoritária existe um conjunto de axiomas (Ω) que auxiliam na minimização de funções majoritárias, sendo os axiomas da maioria, da comutatividade, da associatividade, da distribuição e da inversão (AMARU *et al.*, 2016).

Cohn e Lindaman (1961) provaram algebricamente os axiomas da lógica majoritária. Foi utilizado um conjunto de proposições de deduções booleanas (P) que fornecem a base para provar o conjunto Ω. Tem-se o conjunto de proposições:

- P.1: $X = X$;
- P.2: Se $X = Y$, então $Y = X$;
- P.3: Se $X = Y$ e $Y = Z$, então $X = Z$;
- P.4: Se $W = X$, então $M(X, Y, Z) = M(W, Z, Y)$;
- P.5: $X = Y$ ou $X = \overline{Y}$;
- P.6: X é diferente de $\overline{X}$;
- P.7: $M(X, Y, Z) = M(Y, X, Z)$;
- P.8: $M(X, X, Y) = X$.

O axioma da maioria (Ω.M) define que o resultado de uma função majoritária é verdadeiro caso a maioria das entradas possuírem valores lógicos verdadeiros. Tem-se, como exemplo, a função $M(A, B, A)$. Como o literal A está em maioria, o resultado da função sempre será A.

Outro exemplo é a função $M(A, B, \overline{A})$. Nota-se que o literal A aparece na forma complementar e não complementar, assim a função é definida como $M(1, B, 0)$ ou como $M(0, B, 1)$. O literal B é tido como o critério de desempate e a saída da função sempre será o valor lógico de B.

$M(A, B, \overline{A}) =$ B pode ser provado usando o conjunto P da seguinte forma:

- De acordo com $P.5$ $B = A$ ou $B = \overline{A}$;
  - Se $B = A$ então:
    - De acordo com $P.2$ se $B = A$ então $A = B$;
    - Então $M(A, B, \overline{A}) = M(B, B, \overline{A})$;
    - De acordo com $P.8$ $M(B, B, \overline{A}) = B$.
  - Se $B = \overline{A}$ então:

* De acordo com $P.2$ se $B = \overline{A}$ então $\overline{A} = B$;
* Então $M(A, B, \overline{A}) = M(A, B, B)$;
* De acordo com $P.8$ $M(A, B, B) = B$.

Na Figura 20 apresenta-se o uso do axioma Ω.M para as funções $M(A, B, A)$ e $M(A, B, \overline{A})$.

Figura 20 – Exemplo do axioma Ω.M para as funções $M(A, B, A)$ e $M(A, B, \overline{A})$ em um MIG.

Fonte: Dados do próprio autor.

O axioma da comutatividade (Ω.C) define que o resultado de uma função majoritária depende apenas da maioria. A ordem das variáveis de entrada não altera o resultado final de uma função. Assim, $M(A, B, C) = M(A, C, B) = M(C, A, B) = M(C, B, A) = M(B, C, A) = M(B, A, C)$.

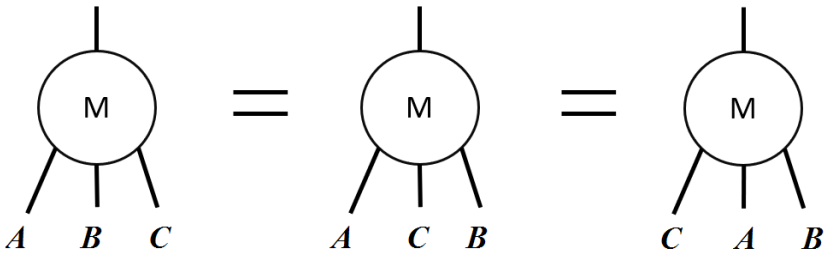
O axioma $\Omega.C$ pode ser provado da seguinte forma:

- De acordo com $P.4$ Se A = A então M(A,B,C) = M(A,C,B);
  - De acordo com $P.7$ $M(A, C, B) = M(C, A, B)$;
  - De acordo com $P.4$ Se $C = C$ então $M(C, A, B) = M(C, B, A)$;
  - De acordo com $P.7$ $M(C, B, A) = M(B, C, A)$;
  - De acordo com $P.4$ Se $B = B$ então $M(B, C, A) = M(B, A, C)$.

Na Figura 21 apresenta-se um exemplo do axioma Ω.C representando a igualdade entre as funções $M(A, B, C)$, $M(A, C, B)$ e $M(C, A, B)$ em um MIG.

O axioma da associatividade (Ω.A) define que é possível trocar uma variável entre os níveis de uma função. Para isso, deve haver ao menos um literal em comum em níveis que estejam em cascata. Por exemplo, na função $M(A, B, M(A, C, D))$, o literal A repete-se em ambos os níveis. Portanto, é possível trocar o literal B com o literal C ou D

Figura 21 – Exemplo de uso do axioma Ω.C.



Fonte: Dados do próprio autor.

do segundo nível, gerando novas funções equivalentes. Assim, $M(A, B, M(A, C, D)) = M(A, C, M(A, B, D)) = M(A, D, M(A, C, B))$.

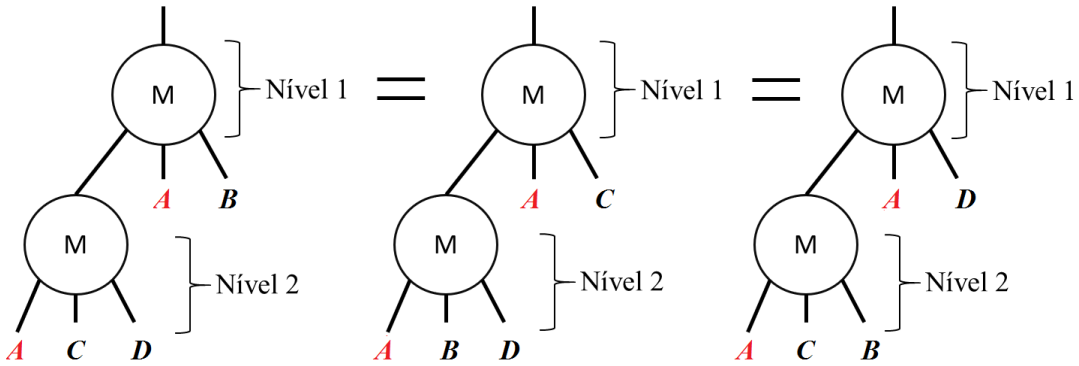
Utilizando a expressão $M(A, B, M(A, C, D)) = M(A, C, M(A, B, D))$, o axioma Ω.A pode ser provado usando o conjunto P da seguinte forma:

- De acordo com $P.5$ $A = D$ ou $A = \overline{D}$;
  - Se $A = D$ então:
    - De acordo com $P.2$ se $A = D$ então $D = A$;
    - Então $M(A, B, M(A, C, A)) = M(A, C, M(A, B, A))$;
    - De acordo com Ω.M $M(A, B, M(A, C, A)) = M(A, B, A)$;
    - De acordo com Ω.M $M(A, C, M(A, B, A)) = M(A, C, A)$;
    - Portanto $M(A, B, A) = M(A, C, A)$;
    - De acordo com Ω.M $M(A, B, A) = A$ e $M(A, C, A) = A$;
    - Portanto $A = A$.
  - Se $A = \overline{D}$ então:
    - Então $M(\overline{D}, B, M(\overline{D}, C, D)) = M(\overline{D}, C, M(\overline{D}, B, D))$;
    - De acordo com Ω.M $M(\overline{D}, B, M(\overline{D}, C, D)) = M(\overline{D}, B, C)$;
    - De acordo com Ω.M $M(\overline{D}, C, M(\overline{D}, B, D)) = M(\overline{D}, C, B)$;
    - Portanto de acordo com Ω.C $M(\overline{D}, B, C) = M(\overline{D}, C, B)$.

Na Figura 22 apresenta-se em um MIG o uso do axioma Ω.A.

O axioma da distribuição (Ω.D) define que as variáveis que se repetem em níveis, podem ser rearranjadas para evitar redundâncias. Tem-se como exemplo, a função $M(M(A, B, C), M(A, B, D), E)$.

Figura 22 – Exemplo de uso do axioma Ω.A para a função $M(A, B, M(A, C, D))$.



Fonte: Dados do próprio autor.

Como A e B se repetem, é possível colocá-las em evidência e gerar a função $M(A, B, M(C, D, E))$.

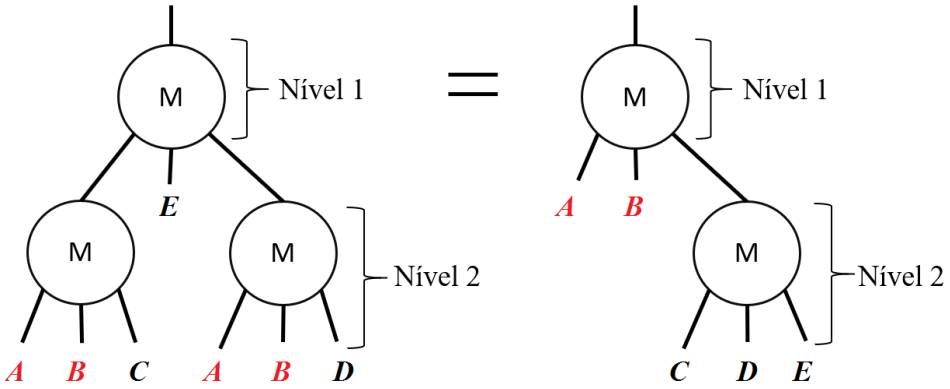
$M(M(A, B, C), M(A, B, D), E) = M(A, B, M(C, D, E))$ pode ser provado usando o conjunto P da seguinte forma:

- De acordo com $P.5$ $A = B$ ou $A = \overline{B}$;
  - Se $A = B$ então:
    - De acordo com $P.2$ se $A = B$ então $B = A$;
    - Então $M(E, M(A, A, C), M(A, A, D)) = M(A, A, M(C, D, E))$;
    - De acordo com Ω.M $M(E, M(A, A, C), M(A, A, D)) = M(E, A, A)$;
    - De acordo com Ω.M $M(A, A, M(C, D, E)) = A$;
    - De acordo com Ω.M $M(E, A, A) = A$;
    - Portanto $A = A$.
  - Se $A = \overline{B}$ então:
    - Então $M(E, M(\overline{B}, B, C), M(\overline{B}, B, D)) = M(\overline{B}, B, M(C, D, E))$;
    - De acordo com Ω.M $M(E, M(\overline{B}, B, C), M(\overline{B}, B, D)) = M(E, C, D)$;
    - De acordo com Ω.M $M(\overline{B}, B, M(C, D, E)) = M(C, D, E)$;
    - Portanto de acordo com Ω.C $M(E, C, D) = M(C, D, E)$.

Na Figura 23 apresenta-se em um MIG o uso do axioma Ω.D.

O axioma da inversão (Ω.I) define que pode-se propagar a negação das variáveis de entrada de uma função para a saída. Tem-se como exemplo, as funções $M(\overline{A}, \overline{B}, \overline{C})$,

Figura 23 – Exemplo de uso do axioma Ω.D para a função $M(M(A,B,C),M(A,B,D),E)$.



Fonte: Dados do próprio autor.

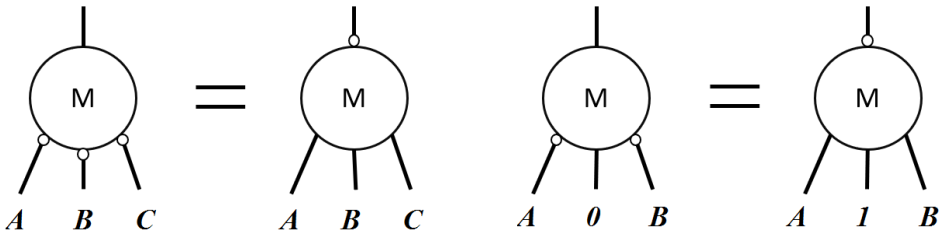
$M(\overline{A},0,\overline{B})$ e $M(\overline{A},1,\overline{B})$ que podem ser escritas respectivamente como $\overline{M}(A,B,C)$, $\overline{M}(A,1,B)$ e $\overline{M}(A,0,B)$.

A igualdade $M(\overline{A},\overline{B},\overline{C}) = \overline{M}(A,B,C)$ pode ser provada usando o conjunto P da seguinte forma:

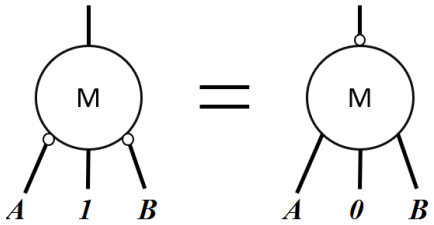
- De acordo com $P.5$ $A = B$ ou $A = \overline{B}$;
  - Se $A = B$ então $\overline{A} = \overline{B}$ assim:
    - De acordo com $P.2$ se $A = B$ e $B = A$ então $\overline{A} = \overline{B}$ e $\overline{B} = \overline{A}$;
    - Então $\overline{M}(A,A,C) = M(\overline{A},\overline{A},\overline{C})$;
    - De acordo com Ω.M $\overline{M}(A,A,C) = \overline{A}$;
    - De acordo com Ω.M $M(\overline{A},\overline{A},\overline{C}) = \overline{A}$;
    - Portanto $\overline{A} = \overline{A}$.
  - Se $A = \overline{B}$ então:
    - De acordo com $P.2$ se $A = \overline{B}$ e $\overline{B} = A$ então $\overline{A} = B$ e $B = \overline{A}$;
    - Então $\overline{M}(\overline{B},B,C) = M(B,\overline{B},\overline{C})$;
    - De acordo com Ω.M $\overline{M}(\overline{B},B,C) = \overline{C}$;
    - De acordo com Ω.M $M(A,\overline{A},\overline{C}) = \overline{C}$;
    - Portanto $\overline{C} = \overline{C}$.

Na Figura 24 apresentam-se exemplos de aplicação do axioma Ω.I em um MIG.

Figura 24 – Exemplos do axioma Ω.I para diferentes funções em um MIG.



(a) $M(\overline{A}, \overline{B}, \overline{C}) = \overline{M}(A, B, C)$ (b) $M(\overline{A}, 0, \overline{B}) = \overline{M}(A, 1, B)$



(c) $M(\overline{A}, 1, \overline{B}) = \overline{M}(A, 0, B)$

Fonte: Dados do próprio autor.

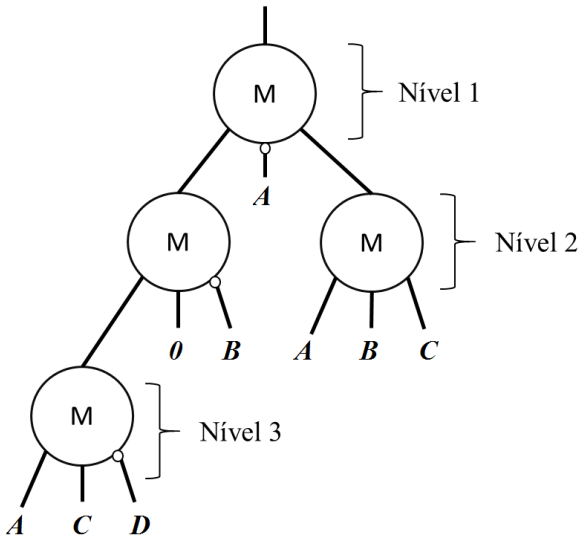
3.4 CUSTO E MINIMIZAÇÃO DE FUNÇÕES MAJORITÁRIAS

Kong, Shang e Lu (2010) propuseram um critério de custo específico para a tecnologia *Quantum-Dot Cellular Automata* (QCA). Esta tecnologia possui características que tornam propicia a implementação de circuitos majoritários. Sendo assim, quatro critérios são considerados:

- O atraso na propagação de um sinal lógico na tecnologia QCA aumenta conforme o número de níveis de uma função. Dessa forma, o número de níveis é considerado como critério primário de custo;
- O número de portas é considerado como critério secundário de custo;
- As entradas de uma função são consideradas como o terceiro critério de custo. Porém apenas as variáveis de entrada e as portas majoritárias que são utilizadas como entradas são computadas no custo. Valores lógicos fixados em 0 ou em 1 possuem custo zero;
- Inversores são o quarto critério de custo.

Tem-se como exemplo a função $M(\overline{A}, M(\overline{B}, 0, M(A, C, \overline{D})), M(A, B, C))$ representada em um MIG na Figura 25.

Figura 25 – MIG da função $M(\overline{A}, M(\overline{B}, 0, M(A, C, \overline{D})), M(A, B, C))$.



Fonte: Dados do próprio autor.

Assim, de acordo com os critérios de custo propostos por Kong, Shang e Lu (2010), a função $M(\overline{A}, M(\overline{B}, 0, M(A, C, \overline{D})), M(A, B, C))$ possui três níveis, quatro portas majoritárias, onze entradas, e três inversores. Nota-se que valores fixos têm custo zero.

Para exemplificar a diferença de custo de um circuito reproduzido pela lógica clássica em relação a um circuito implementando com a lógica majoritária, tem-se como exemplo a função representada na Equação 30 que possui saída verdadeira para os mintermos 0, 1, 3, 5, 7, 9, 11 e 15.

$$F(A, B, C, D) = \overline{A}.D + \overline{B}.D + C.D + \overline{A}.\overline{B}.\overline{C} \tag{30}$$

O circuito gerado pela expressão representada na Equação 30, possui dois níveis, cinco portas lógicas, treze entradas e cinco inversores.

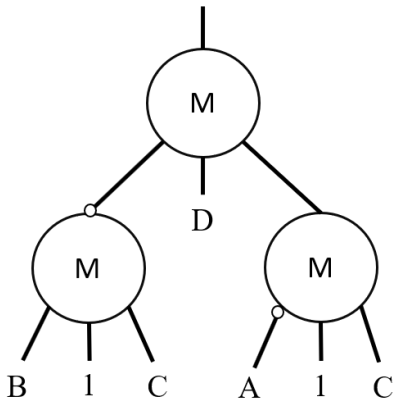
Na Equação 31 apresenta-se a função majoritária minimizada que representa a função Σm(0,1,3,5,7,9,11,15).

$$M(\overline{M}(B, 1, C), D, M(\overline{A}, 1, C)) \tag{31}$$

O circuito gerado pela expressão representada na Equação 31 possui, 2 níveis, 3 portas majoritárias, 7 entradas e 2 inversores, possuindo um custo menor do que o circuito gerado pela lógica clássica.

Na Figura 26 apresenta-se o MIG da função $\overline{M}(B, 1, C), D, M(\overline{A}, 1, C))$.

Figura 26 – MIG da função $\overline{M}(B, 1, C), D, M(\overline{A}, 1, C))$.



Fonte: Dados do próprio autor.

Neste capítulo foram apresentados os principais conceitos sobre a lógica majoritária e sobre os MIGs. No Capítulo 4 é apresentado o programa denominado *exact_mig* que tem como objetivo a síntese de funções majoritárias com até 6 variáveis de entrada.

4 PROGRAMA *EXACT_MIG*

Neste capítulo é apresentado o programa denominado *exact_mig*, que é um dos melhores programas de síntese de funções majoritárias. O *exact_mig* foi o programa utilizado para comparar os resultados com os do programa MK4 desenvolvido neste trabalho.

Em Soeken *et al.* (2017) foi desenvolvido um método denominado *exact_mig* que funciona para funções com até 6 variáveis de entrada e possui como objetivo a geração de funções majoritárias minimizadas. O programa foi desenvolvido na linguagem de programação C++ e incorporado ao *framework* de síntese de funções denominado *Cirkit* (SOEKEN *et al.*, 2017).

O *exact_mig* pode ser executado de duas principais formas com critérios de custos distintos. Quando executado com o comando "*exact_mig -o 1*", o programa possui como critério de custo, respectivamente, o menor número de portas majoritárias e o menor número de níveis possível. Quando executado com o comando "*exact_mig -o 2*", os critérios de custo passam a ser o número de níveis seguido pelo número de portas majoritárias.

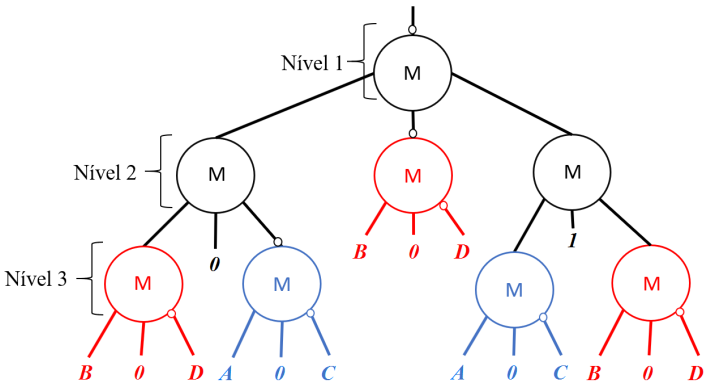
O *exact_mig* possui restrições na contagem de portas majoritárias que definem que uma porta majoritária que se repete em uma função é contada somente uma vez. Tem-se como exemplo a função F apresentada na Equação 32.

$$\overline{M}(\overline{M}(B,0,\overline{D}),M(0,M(B,0,\overline{D}),\overline{M}(A,0,\overline{C})),M(1,M(B,0,\overline{D}),M(A,0,\overline{C}))). \quad (32)$$

Nota-se que na Equação 32, desconsiderando-se os inversores das saídas das portas majoritárias, as portas M(B,0,$\overline{D}$) e M(A,0,$\overline{C}$) se repetem. Assim, são contadas como portas somente a primeira vez que aparecem.

Na Figura 27 apresenta-se o MIG da função F que possui 3 níveis, 5 portas majoritárias, 11 entradas e 5 inversores. Em vermelho e azul destacam-se as portas majoritárias que se repetem.

Figura 27 – MIG destacando as portas majoritárias repetidas.



Fonte: Dados do próprio autor.

De acordo com os critérios de custo apresentados, o programa *exact_mig* é conhecido como o programa que realiza a melhor minimização de funções majoritárias no atual estado da arte (SOEKEN *et al.*, 2017).

4.1 ALGORITMO *EXACT_MIG*

Para entender o funcionamento do *exact_mig*, é necessário conhecer as propostas dos solucionadores de satisfatibilidade (*SAT solver*) e solucionadores de módulo e teoria (*SMT solver*).

Um solucionador de satisfatibilidade booleana é definido como um algoritmo que possui como entrada uma expressão booleana e tem como objetivo validar se a expressão de entrada é uma expressão valida ou invalida. Em um solucionador de satisfatibilidade, uma solução válida é chamada de SAT e uma solução inválida é chamada de UNSAT (GANZINGER *et al.*, 2004).

Um solucionador de satisfatibilidade que possui restrições de uma teoria T, por exemplo, T sendo a teoria da lógica majoritária, passa a ser considerado um solucionador de módulo e teoria que possui como objetivo encontrar um resultado SAT para uma expressão, porém, levando em consideração todas as restrições e características da teoria T (MOURA; DUTERTRE; SHANKAR, 2007).

Para verificar se uma solução gerada é uma solução SAT, o *exact_mig* utiliza um *SMT solver* desenvolvido pela empresa Microsoft e denominado Z3 (MOURA; BJØRNER, 2008).

4.1.1 Descrição do algoritmo

Para a execução, o *exact_mig* utiliza as variáveis f, r e d e o conceito de MIGs. A variável f representa uma tabela verdade com n variáveis de entrada. A variável r representa o número de portas majoritárias que a função pode possuir e d representa o número máximo de níveis que a função gerada pode possuir.

Na Figura 28 apresenta-se o pseudocódigo do programa *exact_mig* quando executado com o comando "*exact_mig -o 1*", ou seja, com o algoritmo priorizando o menor número de portas majoritárias seguido do menor número de níveis.

Figura 28 – Pseudocódigo do programa *exact_mig* executado com o comando "*exact_mig -o 1*".

```
1  Início{
2      f = tabelaVerdadeEntrada();
3      //f representa a tabela verdade de entrada
4      r = 0;
5      //r representa o número de portas
6      while(true){
7          if (Z3RetornouSAT(f,r)  == true){
8              //É verificado se existe uma solução SAT para f com r portas
9              melhorFunção = funçãoEncontrada;
10             imprimir(melhorFunção);
11             return melhorFunção;
12         }
13         else{
14             //Caso não exista uma solução SAT r é acrescentado em 1
15             r = r + 1;
16         }
17     }
18 }
```

Fonte: Adaptado de Soeken *et al.* (2017).

O algoritmo tem início com a variável f recebendo uma tabela verdade como entrada e com a variável r recebendo o valor 0. Assim, enquanto o *exact_mig* não encontrar um MIG que tenha r portas majoritárias e que retorne um resultado SAT pelo Z3, a variável r é atualizada para $r + 1$ até que um MIG com r portas e que retorne um resultado SAT seja encontrado.

Como a variável r é iniciada em 0 e tem seu valor incrementado em 1 a cada interação, o primeiro resultado SAT retornado pelo Z3 já estará otimizado em número de portas majoritárias.

Quando o *exact_mig* é executado pelo comando *exact_mig -o 2*, ou seja, priorizando a minimização de níveis seguido do número de portas majoritárias, o algoritmo utilizado é diferente do apresentado na Figura 28. Quando executado dessa forma, o algoritmo tem como base o número máximo de portas majoritárias que um MIG pode possuir. O total máximo de portas majoritárias que um MIG comporta é igual a $(3^d - 1)/2$ (SOEKEN *et al.*, 2017).

O algoritmo tem início com a variável f recebendo uma tabela verdade de entrada e com a variável d recebendo o valor 0. O *exact_mig* tenta gerar um MIG que tenha o número de níveis igual a d e com o menor valor possível de r. Caso um resultado UNSAT seja obtido, a variável d é atualizada em $d+1$. Esse processo se repete até que seja gerado um MIG que retorne um resultado SAT.

Para cada valor atribuído à variável d, a variável r recebe o valor 0 e é incrementada em 1 até que seja gerado um MIG com d níveis, r portas majoritárias e que retorne um resultado SAT pelo Z3 ou até que r seja maior que $(3^d - 1)/2$.

Na Figura 29 apresenta-se o pseudocódigo do algoritmo *exact_mig* quando executado com o comando "*exact_mig -o 2*".

Figura 29 – Pseudocódigo do programa *exact_mig* executado com o comando "*exact_mig -o 2*".

```
Início{
    f = tabelaVerdadeEntrada();
    //f representa a tabela verdade de entrada
    d = 0; //d representa o número de níveis
    while(true){
        r = 0;//r representa o número de portas
        while(r < (3^d -1)/2){
            //Enquanto r for menor menor que o número máximo
            //de portas que um MIG de d níveis pode conter
            if(Z3RetornouSAT(f,r,d) == true){
                //É verificado se existe uma solução SAT para f com d níveis e r portas
                melhorFunção = funçãoEncontrada;
                imprimir(melhorFunção);
                return melhorFunção;
            }
            else{
                //Caso não exista uma solução SAT r é acrescentado em 1
                r = r + 1;
            }
        }
        d = d + 1;
    // Caso r == (3^d -1)/2 e não exista uma solução SAT então d é acrescentado em 1
    }
}
```

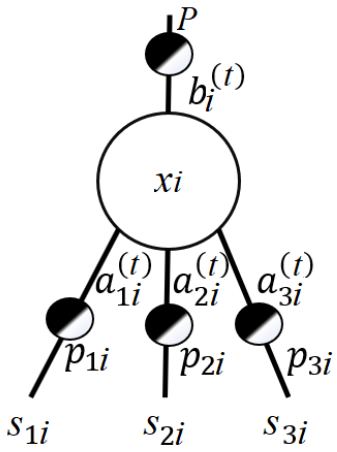
Fonte: Adaptado de Soeken *et al.* (2017).

Nota-se que o primeiro resultado SAT obtido pela função *Z3RetornouSAT(f,r,d)*, apresentada na Figura 29 está otimizado no número de níveis e no número de portas majoritárias.

4.2 GERAÇÃO DE UM MIG

Para exemplificar a geração de um MIG pelo *exact_mig* tem-se como exemplo a Figura 30 em que se representa um MIG *xi* de exemplo. Os círculos de duas cores representam as variáveis de polarização.

Figura 30 – MIG codificado.



Fonte: Adaptado de Soeken *et al.* (2017).

Para cada ramo do MIG, são adicionadas as variáveis s_{ci} e p_{ci}. As variáveis s_{ci} representam, respectivamente, as entradas da porta majoritária em que o índice c representa uma entrada e i representa a qual porta a entrada pertence (SOEKEN *et al.*, 2017).

As variáveis P e p_{ci} recebem valores binários e representam, respectivamente, a polarização final do MIG e a polarização das entradas de xi. Quando $P = 0$ a saída do MIG está invertida, e quando uma variável $p_{ci} = 0$ a entrada s_{ci} está invertida (SOEKEN *et al.*, 2017).

Para garantir um resultado ótimo válido, um MIG gerado pelo *exact_mig* deve respeitar algumas restrições como descrito a seguir.

Na Equação 33 apresenta-se a restrição que ordena as entradas das portas majoritárias.

$$(s_{1i} < s_{2i}).(s_{2i} < s_{3i}) \tag{33}$$

Visando diminuir a quantidade de inversores em um MIG, tem-se a restrição representada na Equação 34 que limita o número de inversores para que exista no máximo uma entrada invertida por porta majoritária.

$$(p_{1i} + p_{2i}).(p_{1i} + p_{3i}).(p_{2i} + p_{3i}) \tag{34}$$

Um MIG é considerado válido quando para cada termo t da tabela verdade f, a saída $b_i^{(t)}$, após passada pela polarização P representada pela Equação 35 , seja correspondente ao valor lógico de t, portanto, a variável $b_i^{(t)}$ representa a saída da porta majoritária xi e é definida pela Equação 36.

$$f^{(t)} = (b_i^{(t)} \oplus \overline{P}) \tag{35}$$

$$b_i^{(t)} = M(a_{1i}^{(t)}, a_{2i}^{(t)}, a_{3i}^{(t)}) \tag{36}$$

As variáveis $a_{ci}^{(t)}$ representam as entradas após serem modificadas pelas variáveis p_{ci} e são definidas pela Equação 37.

$$a_{ci}^{(t)} = (s_{ci} \oplus \overline{p_{ci}}) \tag{37}$$

O uso das restrições apresentadas somadas as restrições de níveis e de portas majoritárias possibilita que o *exact_mig* gere funções otimizadas no número de níveis e portas majoritárias, além de reduzir o número de inversores.

No apêndice B apresenta-se um manual de instalação e execução do programa *exact_mig*.

Neste capítulo foram apresentados os principais conceitos utilizados no programa *exact_mig*. No capítulo 5 é abordado o método do mapa-K para geração de funções majoritárias otimizadas e no capítulo 6 é descrito o programa MK4, desenvolvido neste trabalho, que tem como base o método do mapa-K, além de apresentar uma comparação entre os resultados obtidos pelo *exact_mig* e pelo MK4.

5 MÉTODO DO MAPA-K UTILIZANDO PORTAS MAJORITÁRIAS

O mapa-K pode ser utilizado para realizar a síntese de funções majoritárias. Neste trabalho foi implementado o programa denominado MK4 que utiliza as características do método do mapa-K para minimizar uma função booleana. Assim, neste capítulo é apresentado o funcionamento deste método. Na seção 5.1 é exemplificado o uso de um mapa-K de três variáveis para gerar uma função majoritária e na seção 5.2 é abordado o uso do mapa-K de quatro variáveis.

5.1 MÉTODO DO MAPA-K PARA TRÊS VARIÁVEIS

O método do mapa-K desenvolvido em Walus *et al.* (2004) tem como base quarenta funções primitivas. Entende-se como funções primitivas as funções que podem ser implementadas utilizando uma única ou nenhuma porta majoritária. As combinações de até três funções primitivas implementam qualquer função majoritária com no máximo três variáveis de entrada e que precisam de dois níveis para serem geradas.

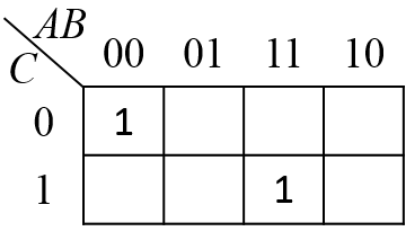
Para exemplificar o funcionamento do método, tem-se como exemplo a função F representada na Equação 38:

$$F(A, B, C) = A.B.C + \overline{A}.\overline{B}.\overline{C}. \tag{38}$$

A Equação 38 compreende os mintermos 0 e 7. O primeiro passo do método é desenhar o mapa-K da função destacando os mintermos que devem estar na solução. Na Figura 31 apresenta-se o mapa-K da função F.

O segundo passo é encontrar uma combinação de no máximo três funções primitivas de forma que os mintermos sejam cobertos duas ou mais vezes e que os maxtermos da função sejam cobertos no máximo uma vez.

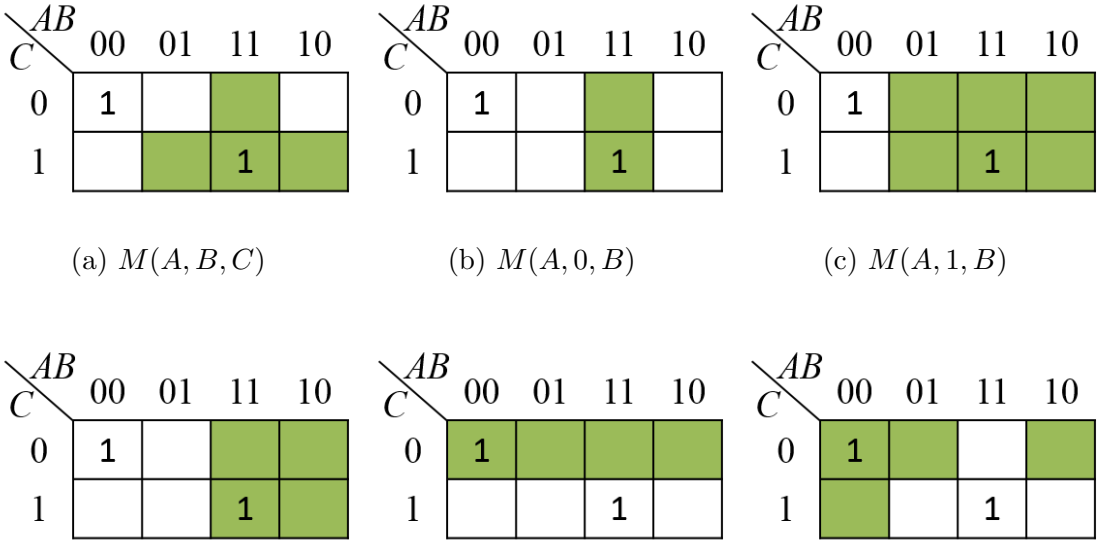
Figura 31 – Mapa-K da função $F(A, B, C) = A.B.C + \overline{A}.\overline{B}.\overline{C}$.



Fonte: Dados do próprio autor.

Na Figura 32 apresenta-se o exemplo de seis funções primitivas implementadas em um mapa-K de três variáveis. Destacam-se em verde os termos cobertos por cada função.

Figura 32 – Exemplo de funções primitivas implementadas em mapas-K de três variáveis.



(a) $M(A, B, C)$ (b) $M(A, 0, B)$ (c) $M(A, 1, B)$

(d) A (e) $\overline{C}$ (f) $\overline{M}(A, B, C)$

Fonte: Dados do próprio autor.

Na Tabela 22 apresentam-se as quarenta funções primitivas e os termos cobertos por cada uma.

Para exemplificar, escolhendo a função $M(\overline{A}, 1, B)$, os mintermos 0 e 7 e os maxtermos 1, 2, 3 e 6 são cobertos uma vez. Na Figura 33 apresenta-se o mapa-K da função escolhida.

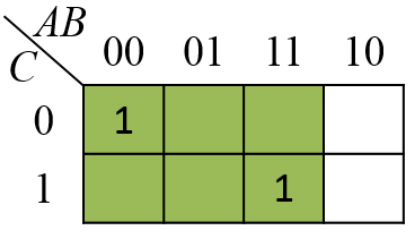
Os maxtermos que foram cobertos uma vez não podem ser cobertos novamente por nenhuma outra função primitiva, caso contrário possuirão valor lógico verdadeiro para a função gerada.

Tabela 22 – Tabela de 40 funções primitivas com no máximo três variáveis de entrada.

Função	Termos cobertos	Função	Termos cobertos
0	–	$M(\overline{A}, 1, B)$	$\Sigma m(0, 1, 2, 3, 6, 7)$
1	$\Sigma m(0, 1, 2, 3, 4, 5, 6, 7)$	$M(\overline{A}, 1, C)$	$\Sigma m(0, 1, 2, 3, 5, 7)$
A	$\Sigma m(4, 5, 6, 7)$	$M(\overline{B}, 1, C)$	$\Sigma m(0, 1, 3, 4, 5, 7)$
$\overline{A}$	$\Sigma m(0, 1, 2, 3)$	$M(A, 1, \overline{B})$	$\Sigma m(0, 1, 4, 5, 6, 7)$
B	$\Sigma m(2, 3, 6, 7)$	$M(A, 1, \overline{C})$	$\Sigma m(0, 2, 4, 5, 6, 7)$
$\overline{B}$	$\Sigma m(0, 1, 4, 5)$	$M(B, 1, \overline{C})$	$\Sigma m(0, 2, 3, 4, 6, 7)$
C	$\Sigma m(1, 3, 5, 7)$	$M(A, B, C)$	$\Sigma m(3, 5, 6, 7)$
$\overline{C}$	$\Sigma m(0, 2, 4, 6)$	$\overline{M}(A, B, C)$	$\Sigma m(0, 1, 2, 4)$
$M(A, 0, B)$	$\Sigma m(6, 7)$	$M(A, B, \overline{C})$	$\Sigma m(2, 4, 6, 7)$
$M(A, 0, C)$	$\Sigma m(5, 7)$	$M(A, \overline{B}, C)$	$\Sigma m(1, 4, 5, 7)$
$M(B, 0, C)$	$\Sigma m(3, 7)$	$M(\overline{A}, B, C)$	$\Sigma m(1, 2, 3, 7)$
$M(A, 1, B)$	$\Sigma m(2, 3, 4, 5, 6, 7)$	$\overline{M}(A, \overline{B}, C)$	$\Sigma m(0, 2, 3, 6)$
$M(A, 1, C)$	$\Sigma m(1, 3, 4, 5, 6, 7)$	$\overline{M}(A, B, \overline{C})$	$\Sigma m(0, 1, 3, 5)$
$M(B, 1, C)$	$\Sigma m(1, 2, 3, 5, 6, 7)$	$\overline{M}(\overline{A}, B, C)$	$\Sigma m(0, 4, 5, 6)$
$M(\overline{A}, 0, B)$	$\Sigma m(2, 3)$	$\overline{M}(A, 1, B)$	$\Sigma m(0, 1)$
$M(\overline{A}, 0, C)$	$\Sigma m(1, 3)$	$\overline{M}(A, 1, C)$	$\Sigma m(0, 2)$
$M(\overline{B}, 0, C)$	$\Sigma m(1, 5)$	$\overline{M}(B, 1, C)$	$\Sigma m(0, 4)$
$M(A, 0, \overline{B})$	$\Sigma m(4, 5)$	$\overline{M}(A, 0, B)$	$\Sigma m(0, 1, 2, 3, 4, 5)$
$M(A, 0, \overline{C})$	$\Sigma m(4, 6)$	$\overline{M}(A, 0, C)$	$\Sigma m(0, 1, 2, 3, 4, 6)$
$M(B, 0, \overline{C})$	$\Sigma m(2, 6)$	$\overline{M}(B, 0, C)$	$\Sigma m(0, 1, 2, 4, 5, 6)$

Fonte: Dados do próprio autor.

Figura 33 – Mapa-K da função $M(\overline{A}, 1, B)$.



Fonte: Dados do próprio autor.

Cada mintermo foi coberto somente uma vez, portanto, deve-se selecionar outra função primitiva que cubra pelo menos um dos mintermos (0 e 7), mas que não cubra nenhum dos maxtermos (1, 2, 3 e 6).

Na Tabela 23 apresenta-se a quantidade de vezes que cada mintermo e maxtermo foram cobertos depois da escolha da primeira função primitiva.

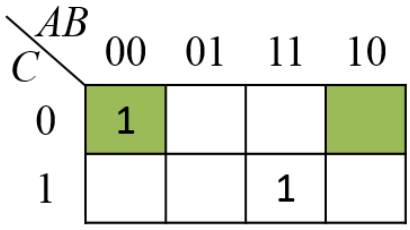
Tabela 23 – Termos cobertos após a escolha da primeira função primitiva.

Termo	Descrição	Total de vezes coberto
0	Mintermo	1
1	Maxtermo	1
2	Maxtermo	1
3	Maxtermo	1
4	Maxtermo	0
5	Maxtermo	0
6	Maxtermo	1
7	Mintermo	1

Fonte: Dados do próprio autor.

Na Figura 34 apresenta-se a escolha da função $M(\overline{B}, 0, \overline{C})$ que cobre o mintermo 0 e o maxtermo 4.

Figura 34 – Mapa-K da função $M(\overline{B}, 0, \overline{C})$.



Fonte: Dados do próprio autor.

Na Tabela 24 apresenta-se a quantidade de vezes que cada mintermo e maxtermo foram cobertos depois da escolha da segunda função primitiva.

Tabela 24 – Termos cobertos após a escolha da segunda função primitiva.

Termo	Descrição	Total de vezes coberto
0	Mintermo	2
1	Maxtermo	1
2	Maxtermo	1
3	Maxtermo	1
4	Maxtermo	1
5	Maxtermo	0
6	Maxtermo	1
7	Mintermo	1

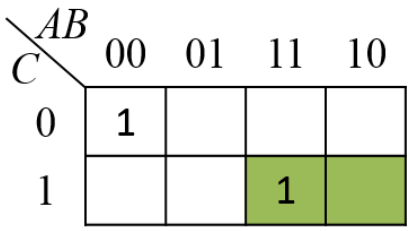
Fonte: Dados do próprio autor.

Nota-se que o mintermo 7 foi coberto uma única vez e os mintermos têm que ser

cobertos 2 ou mais vezes, assim, é necessário selecionar uma terceira função primitiva que o cubra. Além disso, deve-se garantir que todos os maxtermos sejam cobertos no máximo uma vez. Portanto, os maxtermos (1, 2, 3, 4 e 6) não podem ser cobertos pela nova função primitiva escolhida. O mintermo 0 já foi coberto duas vezes então não precisa ser coberto novamente.

Na Figura 35 apresenta-se a escolha da função $M(A, 0, C)$ que cobre o mintermo 7 e o maxtermo 5.

Figura 35 – Mapa-K da função $M(A, 0, C)$.



Fonte: Dados do próprio autor.

Na Tabela 25 apresenta-se a quantidade de vezes que cada mintermo e maxtermo foram cobertos depois da escolha da terceira função primitiva. Nota-se que ambos os mintermos foram cobertos duas vezes e que nenhum maxtermo foi coberto mais de uma vez.

Tabela 25 – Termos cobertos após a escolha da terceira função primitiva.

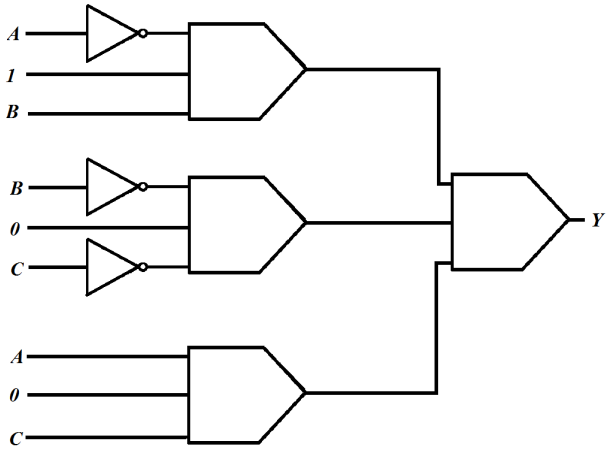
Termo	Descrição	Total de vezes coberto
0	Mintermo	2
1	Maxtermo	1
2	Maxtermo	1
3	Maxtermo	1
4	Maxtermo	1
5	Maxtermo	1
6	Maxtermo	1
7	Mintermo	2

Fonte: Dados do próprio autor.

No final do método do mapa de Karnaugh para 3 variáveis, todas as funções primitivas selecionadas são conectadas a uma porta majoritária a fim de formar a função $M(M(\overline{A}, 1, B), M(\overline{B}, 0, \overline{C}), M(A, 0, C))$ que tem como saída verdadeira os mintermos 0 e 7. Caso as funções primitivas tenham sido selecionadas de forma otimizada, a função resultante estará na forma ótima.

Na Figura 36 apresenta-se o circuito gerado pela aplicação do método.

Figura 36 – Circuito da função $M(M(\overline{A}, 1, B), M(\overline{B}, 0, \overline{C}), M(A, 0, C))$.



Fonte: Dados do próprio autor.

5.2 MÉTODO DO MAPA-K PARA QUATRO VARIÁVEIS

Em Wang, Niamat e Vemuru (2011) foi proposto um método que utiliza o mapa-K para resolver problemas de até quatro variáveis de entrada. Um mapa-K de quatro variáveis possui 2^4 células. Na Figura 37 apresenta-se a disposição das 16 células em um mapa-K de quatro variáveis.

Figura 37 – Disposição das células em um mapa-K de quatro variáveis.

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

Fonte: Dados do próprio autor.

O método tem como base a combinação de 90 funções primitivas. No apêndice A apresentam-se as Tabelas 40, 41 e 42 contendo todas as funções primitivas de 4 variáveis e os respectivos termos cobertos.

Diferente do método do mapa-K de três variáveis, esse método nem sempre encontra a solução ao combinar três funções primitivas. Há casos em que é necessário o acréscimo de

mais um nível para que uma solução seja encontrada (WANG; NIAMAT; VEMURU, 2011).

Existem 6 principais padrões para funções majoritárias de 4 variáveis de entrada. No primeiro padrão as funções podem possuir uma ou nenhuma porta majoritária, sendo uma função primitiva, como apresentado na Equação 39.

$$M(A, B, D) \tag{39}$$

No segundo padrão, podem possuir uma porta majoritária tendo como entradas três funções primitivas como apresentado na Equação 40 que possui dois níveis.

$$M(M(B, 0, \overline{C}), M(A, B, C), M(\overline{A}, B, D)) \tag{40}$$

No terceiro padrão, podem possuir uma porta majoritária tendo como entradas duas funções primitivas e um ramo contendo uma função majoritária não primitiva como apresentado na Equação 41 que possui três níveis.

$$M(A, \overline{M}(B, C, \overline{D}), M(B, M(C, 0, \overline{D}), 0)) \tag{41}$$

No quarto padrão, podem possuir uma porta majoritária tendo como entradas uma função primitiva e dois ramos contendo outras funções não primitivas como apresentado na Equação 42 que possui três níveis.

$$M(\overline{M}(A, 0, C), M(M(B, 0, D), A, C), M(M(A, 0, C), \overline{M}(B, 0, D), 0)) \tag{42}$$

No quinto padrão, podem possuir uma porta majoritária tendo como entradas três ramos com funções não primitivas como apresentado na Equação 43 que possui três níveis.

$$M(M(A, M(B, \overline{C}, D), 0), \overline{M}(\overline{C}, M(\overline{A}, B, D), 0), M(\overline{M}(A, 1, C), M(\overline{A}, B, D), \overline{M}(B, \overline{C}, D))) \tag{43}$$

Só existem duas funções de quatro variáveis que pertencem ao sexto padrão, sendo representadas pelas Equações 44 e 45. Ambas possuindo quatro níveis tratando-se de uma função OU Exclusivo e NÃO OU Exclusivo.

$$M(B, \overline{M}(B, A, M(C, \overline{M}(0, D, C), M(0, D, \overline{C}))), M(\overline{B}, A, M(C, \overline{M}(0, D, C), M(0, D, \overline{C})))) \tag{44}$$

$$\overline{M}(B, \overline{M}(B, A, M(C, \overline{M}(0, D, C), M(0, D, \overline{C}))), M(\overline{B}, A, M(C, \overline{M}(0, D, C), M(0, D, \overline{C})))) \tag{45}$$

Para funções de até dois níveis, o método do mapa-K de quatro variáveis funciona igual ao método do mapa-K de três variáveis, mas utilizando a tabela de 90 funções primitivas. Porém, para funções que precisam de três ou mais níveis o método se comporta de outra forma.

Para exemplificar o funcionamento do método do mapa-K de quatro variáveis para funções que precisam de três níveis, tem-se como exemplo a função $F(A, B, C, D) = \overline{A}.\overline{C}.D + A.B.C.D + A.\overline{B}.\overline{C}.\overline{D}$, que tem saída verdadeira para os mintermos 1, 5, 8 e 15.

Não é possível encontrar uma solução para essa função utilizando apenas três funções primitivas. Portanto, encontra-se até duas funções que cubram duas vezes o maior número de mintermos possíveis. Os mintermos que não foram cobertos duas vezes precisam ter sido cobertos pelo menos uma vez por uma das funções primitivas. É necessário garantir que nenhum maxtermo seja coberto duas ou mais vezes (WANG; NIAMAT; VEMURU, 2011).

Na Figura 38 apresenta-se o mapa-K da função $F(A, B, C, D) = \overline{A}.\overline{C}.D + A.B.C.D + A.\overline{B}.\overline{C}.\overline{D}$.

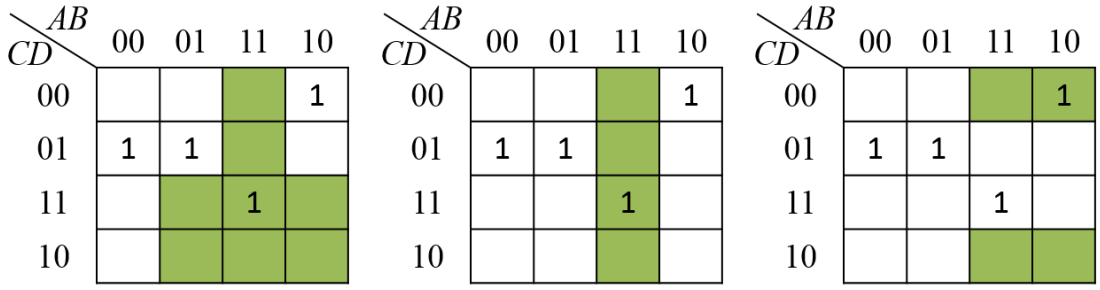
Figura 38 – Mapa-K da função $F(A, B, C, D) = \overline{A}.\overline{C}.D + A.B.C.D + A.\overline{B}.\overline{C}.\overline{D}$.

CD \ AB	00	01	11	10
00				1
01	1	1		
11			1	
10				

Fonte: Dados do próprio autor.

Na Figura 39 apresenta-se o exemplo de seis funções primitivas implementadas em um mapa-K de quatro variáveis. Destacam-se em verde os termos cobertos por cada função.

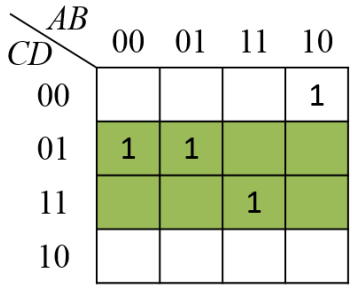
Figura 39 – Exemplo de funções primitivas implementadas em mapas-K de quatro variáveis.



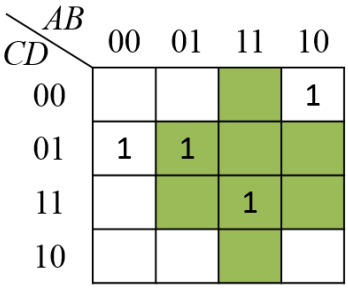
(a) $M(A, B, C)$

(b) $M(A, 0, B)$

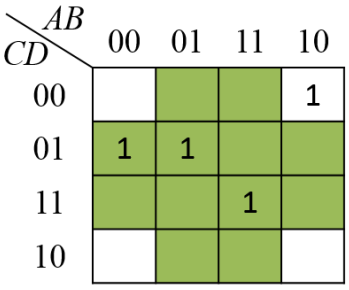
(c) $M(A, 0, \overline{D})$



(d) D



(e) $M(A, B, D)$

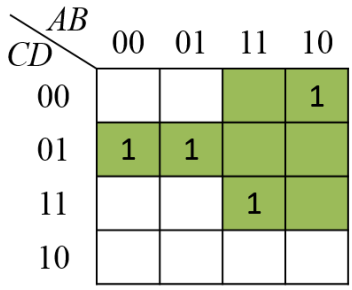


(f) $M(B, 1, D)$

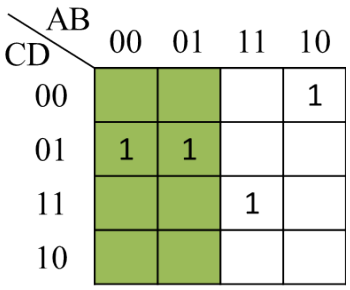
Fonte: Dados do próprio autor.

Combinando as funções $M(A, \overline{C}, D)$ e $\overline{A}$, os mintermos 1 e 5 são cobertos duas vezes e os mintermos 8 e 15 uma vez. Na Figura 40(a) apresenta-se o mapa-K da função $M(A, \overline{C}, D)$ e na Figura 40(b) apresenta-se o mapa-K da variável $\overline{A}$.

Figura 40 – Funções primitivas escolhidas para cobertura do maior número de mintermos.



(a) $M(A, \overline{C}, D)$



(b) $\overline{A}$

Fonte: Dados do próprio autor.

Os mintermos 1 e 5 foram cobertos duas vezes, assim já estão na solução e passam a ser considerados *don't care states* pois não precisam ser cobertos novamente por nenhuma outra função.

Os mintermos 8 e 15 foram cobertos somente uma vez, assim precisam ser cobertos novamente para serem incluídos na função. Os maxtermos cobertos uma vez não podem ser cobertos novamente.

Na Tabela 26 têm-se as quantidades de vezes que cada termo foi coberto pela combinação dos mapas-K apresentados na Figura 40(a) e Figura 40(b).

Tabela 26 – Termos cobertos após a escolha das funções primitivas $M(A, \overline{C}, D)$ e A.

Termo	**Descrição**	**Total de vezes coberto**	**Saída**
0	Maxtermo	1	0
1	Mintermo	2	1
2	Maxtermo	1	0
3	Maxtermo	1	0
4	Maxtermo	1	0
5	Mintermo	2	1
6	Maxtermo	1	0
7	Maxtermo	1	0
8	Mintermo	1	0
9	Maxtermo	1	0
10	Maxtermo	0	0
11	Maxtermo	1	0
12	Maxtermo	1	0
13	Maxtermo	1	0
14	Maxtermo	0	0
15	Mintermo	1	0

Fonte: Dados do próprio autor.

Não existe nenhuma função primitiva que cubra os mintermos 8 e 15 sem cobrir nenhum dos maxtermos que já foram cobertos uma vez. Assim, é necessário que seja adicionado um novo nível na função.

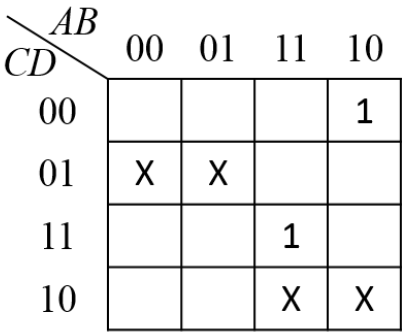
Aplica-se novamente o método do mapa-K de quatro variáveis, em que a função gerada cubra os mintermos 8 e 15. Como os termos 1 e 5 são mintermos na função original e foram cobertos duas vezes no passo anterior, esses termos passam a ser considerados como *don't care states* da nova função.

Os maxtermos do passo anterior que não foram cobertos nenhuma vez também são

considerados como *don't care states* pois mesmo que sejam cobertos duas vezes neste passo, quando conectados a uma porta majoritária com as funções primitivas previamente selecionadas, vão possuir uma saída com valor lógico falso.

Na Figura 41 apresenta-se o mapa-K do novo nível.

Figura 41 – Mapa-K com os mintermos 8 e 15 e os *don't care states* 1, 5, 10 e 14.

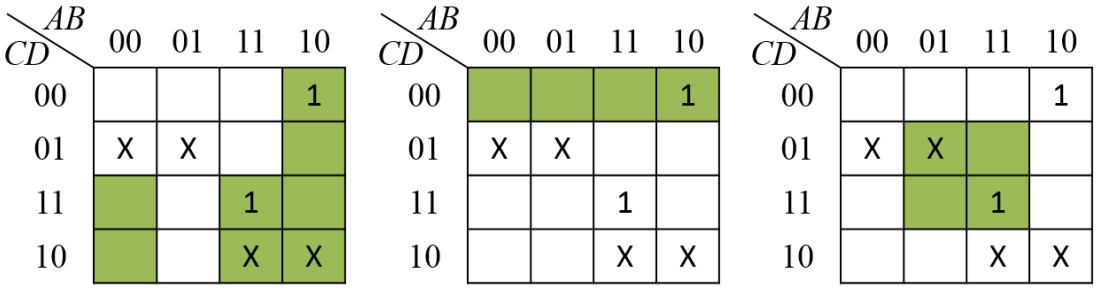


Fonte: Dados do próprio autor.

A função gerada pela junção das funções primitivas $M(A, \overline{B}, C), M(\overline{C}, 0, \overline{D})$ e $M(B, 0, D)$ é igual a $M(M(A, \overline{B}, C), M(\overline{C}, 0, \overline{D}), M(B, 0, D))$, que cobre duas vezes somente os mintermos 8 e 15.

Na Figura 42(a) apresentam-se os mintermos cobertos pela função $M(A, \overline{B}, C)$. Na Figura 42(b) apresentam-se os mintermos cobertos pela função $M(\overline{C}, 0, \overline{D})$ e na Figura 42(c) apresentam-se os mintermos cobertos pela função $M(B, 0, D)$.

Figura 42 – Funções escolhidas para cobertura dos mintermos 8 e 15.



(a) $M(A, \overline{B}, C)$ (b) $M(\overline{C}, 0, \overline{D})$ (c) $M(B, 0, D)$

Fonte: Dados do próprio autor.

Na Tabela 27 apresentam-se as quantidades que cada termo foi coberto pelo novo nível. Nota-se que somente os mintermos da função foram cobertos duas vezes.

Tabela 27 – Termos cobertos depois da escolha do novo nível.

Termo	Descrição	Total de vezes coberto	Saída
0	Maxtermo	1	0
1	*Don't care state*	0	0
2	Maxtermo	1	0
3	Maxtermo	1	0
4	Maxtermo	1	0
5	*Don't care state*	1	0
6	Maxtermo	0	0
7	Maxtermo	1	0
8	Mintermo	2	1
9	Maxtermo	1	0
10	*Don't care state*	1	0
11	Maxtermo	1	0
12	Maxtermo	1	0
13	Maxtermo	1	0
14	*Don't care state*	1	0
15	Mintermo	2	1

Fonte: Dados do próprio autor.

No último passo do método, as duas funções primitivas, previamente selecionadas, e a função majoritária gerada no segundo passo, são conectadas a uma porta majoritária gerando a função representada pela Equação 46.

$$M(M(A,\overline{C},D),\overline{A},M(M(A,\overline{B},C),M(\overline{C},0,\overline{D}),M(B,0,D))). \tag{46}$$

A função gerada possui saída verdadeira para os mintermos 1, 5, 8 e 15.

Porém, não é possível encontrar todas funções dessa maneira. Há casos em que é necessário encontrar até duas funções majoritárias não primitivas e uma função primitiva ou até três funções não primitivas e utilizá-las como ramos de uma porta majoritária para que os mintermos corretos sejam cobertos. Na próxima seção é descrito como o método se comporta para casos desse tipo.

5.2.1 Método do mapa-K para funções que precisam de duas ou mais funções não primitivas conectadas em uma porta majoritária

Para exemplificar casos em que não é possível gerar uma função válida combinando apenas três funções primitivas ou combinando duas funções primitivas com um novo nível contendo uma nova função majoritária, tem-se como exemplo a função $\Sigma m(0, 3, 5, 6, 7, 9, 10, 12, 15)$. Na Figura 43 apresenta-se o mapa-K desta função.

Figura 43 – Mapa-K da função $\Sigma m(0, 3, 5, 6, 7, 9, 10, 12, 15)$.

CD \ AB	00	01	11	10
00	1		1	
01		1		1
11	1	1	1	
10		1		1

Fonte: Dados do próprio autor.

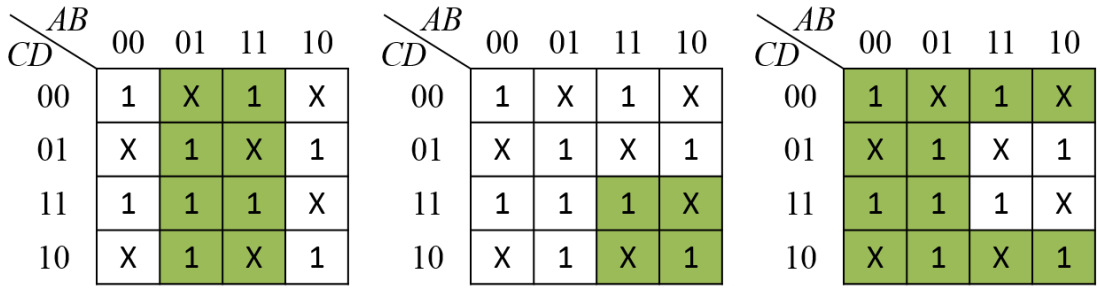
Para o mapa-K apresentado, não existe nenhuma combinação de três funções primitivas que cubra duas vezes todos os mintermos sem que os maxtermos sejam cobertos duas vezes. O método de adicionar uma única função não primitiva extra na função, também não consegue encontrar uma função que cubra todos os mintermos.

A solução é encontrar até três funções, primitivas ou não primitivas, distintas que quando combinadas como ramos em uma porta majoritária, consigam gerar uma saída verdadeira para os mintermos corretos.

Para exemplificar o problema, inicialmente encontra-se o primeiro ramo que cubra duas vezes o máximo de mintermos possíveis. Para o primeiro ramo, os maxtermos da função passam a ser considerados como *don't care states*. Tem-se como exemplo o ramo da função $f1 = M(B, M(A, 0, C), \overline{M}(A, 0, D))$.

Na Figura 44 apresentam-se os mapas-K das funções primitivas utilizadas para gerar $f1$ e na Tabela 28 apresentam-se as quantidades de vezes que cada termo foi coberto pelo primeiro ramo escolhido.

Figura 44 – Funções primitivas utilizadas para gerar $f1$.



CD \ AB	00	01	11	10
00	1	X	1	X
01	X	1	X	1
11	1	1	1	X
10	X	1	X	1

(a) B

CD \ AB	00	01	11	10
00	1	X	1	X
01	X	1	X	1
11	1	1	1	X
10	X	1	X	1

(b) $M(A, 0, C)$

CD \ AB	00	01	11	10
00	1	X	1	X
01	X	1	X	1
11	1	1	1	X
10	X	1	X	1

(c) $\overline{M}(A, 0, D)$

Fonte: Dados do próprio autor.

Tabela 28 – Termos cobertos pelo primeiro ramo.

Termo	Descrição	Total de vezes coberto	Saída
0	Mintermo	1	0
1	*Don't care state*	1	0
2	*Don't care state*	1	0
3	Mintermo	1	0
4	*Don't care state*	2	1
5	Mintermo	2	1
6	Mintermo	2	1
7	Mintermo	2	1
8	*Don't care state*	1	0
9	Mintermo	1	0
10	Mintermo	2	1
11	*Don't care state*	1	0
12	Mintermo	2	1
13	*Don't care state*	1	0
14	*Don't care state*	3	1
15	Mintermo	2	1

Fonte: Dados do próprio autor.

Todos os *don't care states* que foram cobertos duas vezes passam a ser considerados maxtermos. Nota-se que os mintermos 0, 3 e 9 foram cobertos apenas uma vez e, portanto, obrigatoriamente todos precisam ser cobertos duas vezes pelo segundo ramo da função. O segundo ramo a ser encontrado deve satisfazer as restrições:

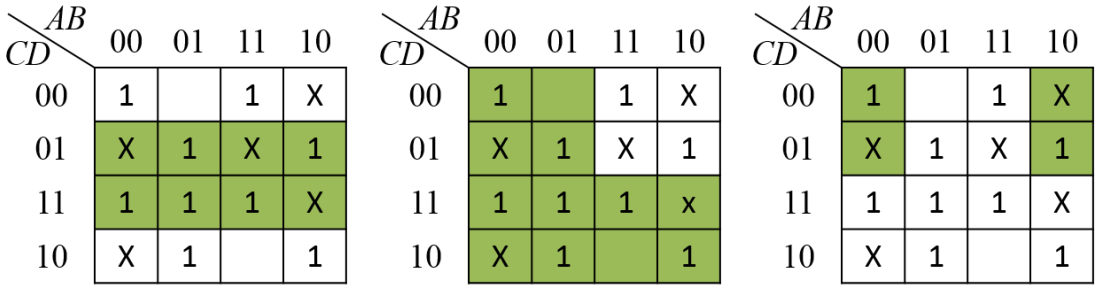
- Todos os mintermos que não possuem uma saída verdadeira na função gerada

anteriormente devem ser cobertos duas vezes;

- Nenhum maxtermo deve ser coberto duas ou mais vezes;
- Deve-se cobrir o menor número possível de *don't care states*;
- Ao menos um dos mintermos com saída verdadeira na função gerada anteriormente deve ser coberto duas vezes.

Tem-se como exemplo a função $f2 = M(D, M(\overline{A}, 1, C), \overline{M}(B, 1, C))$. Na Figura 45 apresentam-se os mapas-K das funções primitivas utilizadas para gerar $f2$.

Figura 45 – Funções primitivas utilizadas para gerar $f2$.



(a) D (b) $M(\overline{A}, 1, C)$ (c) $\overline{M}(B, 1, C)$

Fonte: Dados do próprio autor.

Na Tabela 29 apresentam-se as quantidades de vezes que cada termo foi coberto pelo segundo ramo.

Quando conectadas a uma porta majoritária funcionando como uma porta E ou como uma porta OU as funções $f1$ e $f2$ geram as tabelas verdades apresentadas na Tabela 30. Nota-se que para a função $M(f1, 1, f2)$, maxtermos passam a ter saída verdadeira. Portanto, a função $M(f1, 1, f2)$ não é valida. Para a função $M(f1, 0, f2)$ os mintermos 5, 7 e 15 são cobertos duas vezes, mas os outros mintermos são cobertos somente uma vez cada. Portanto, $M(f1, 0, f2)$ não é uma solução válida. Assim, é necessário gerar um terceiro ramo com uma função primitiva ou uma função $f3$ que cubra duas vezes os mintermos restantes.

Como os mintermos 5, 7 e 15 já foram cobertos duas vezes, por $f1$ e $f2$, são considerados *don't care states*. Os maxtermos que não foram cobertos nem por $f1$ e nem por $f2$ também são considerados *don't care states*.

Tabela 29 – Termos cobertos pelo segundo ramo.

Termo	**Descrição**	**Total de vezes coberto**	**Saída**
0	Mintermo	2	1
1	*Don't care state*	3	1
2	*Don't care state*	1	0
3	Mintermo	2	1
4	Maxtermo	1	0
5	Mintermo	2	1
6	Mintermo	1	0
7	Mintermo	2	1
8	*Don't care state*	1	0
9	Mintermo	2	1
10	Mintermo	1	0
11	*Don't care state*	2	1
12	Mintermo	0	0
13	*Don't care state*	1	0
14	Maxtermo	1	0
15	Mintermo	2	1

Fonte: Dados do próprio autor.

Tabela 30 – Tabelas verdades para as funções $M(f1, 0, f2)$ e $M(f1, 1, f2)$

Termo	**Descrição**	$\boldsymbol{f1}$	$\boldsymbol{f2}$	$\boldsymbol{M(f1, 0, f2)}$	$\boldsymbol{M(f1, 1, f2)}$
0	Mintermo	0	1	0	1
1	Maxtermo	0	1	0	1
2	Maxtermo	0	0	0	0
3	Mintermo	0	1	0	1
4	Maxtermo	1	0	0	1
5	Mintermo	1	1	1	1
6	Mintermo	1	0	0	1
7	Mintermo	1	1	1	1
8	Maxtermo	0	0	0	0
9	Mintermo	0	1	0	1
10	Mintermo	1	0	0	1
11	Maxtermo	0	1	0	1
12	Mintermo	1	0	0	1
13	Maxtermo	0	0	0	0
14	Maxtermo	1	0	0	1
15	Mintermo	1	1	1	1

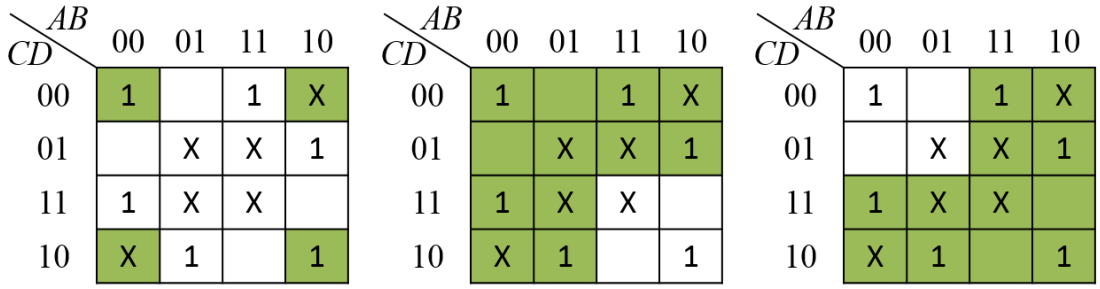
Fonte: Dados do próprio autor.

Não há nenhuma função primitiva que cobre somente os mintermos que faltam. Todas as funções primitivas que cobrem os mintermos 0, 3, 6, 9, 10 e 12 também cobrem algum maxtermo. Assim, uma função $f3$ que deve satisfazer as seguintes restrições é gerada:

- Todos os mintermos precisam ser cobertos duas vezes;
- Nenhum maxtermo deve ser coberto duas ou mais vezes.

Tem-se como exemplo a função $f3 = M(\overline{M}(B,1,D), \overline{M}(A,0,C), M(A,1,C))$. Na Figura 46 apresentam-se os mapas-K das funções primitivas utilizadas para gerar $f3$.

Figura 46 – Funções primitivas utilizadas para gerar $f3$.



(a) $\overline{M}(B,1,D)$ (b) $\overline{M}(A,0,C)$ (c) $M(A,1,C)$

Fonte: Dados do próprio autor.

Na Tabela 31 apresentam-se as quantidades que cada termo foi coberto pelo terceiro ramo escolhido.

O último passo do método é o de conectar todos os ramos encontrados em uma porta majoritária, gerando assim a função: $M(f1, f2, f3)$ que possui saída verdadeira para os mintermos 0, 3, 5, 6, 7, 9, 10, 12 e 15. Na Tabela 32 apresenta-se a tabela verdade da função $M(f1, f2, f3)$.

Caso não seja possível gerar uma função com até três níveis, um novo nível deve ser adicionado.

Como descrito, esse método torna-se complexo para gerar funções que necessitam de mais de dois níveis. Manualmente, as chances de encontrar uma função ótima são baixas, já que a escolha do primeiro ramo de uma função pode ser feita de forma equivocada, aumentando a dificuldade de encontrar outros ramos que consigam cobrir apenas os mintermos desejados.

Tabela 31 – Termos cobertos pelo terceiro ramo.

Termo	**Descrição**	**Total de vezes coberto**	**Saída**
0	Mintermo	2	1
1	Maxtermo	1	0
2	*Don't care state*	3	1
3	Mintermo	2	1
4	Maxtermo	1	0
5	*Don't care state*	1	0
6	Mintermo	2	1
7	*Don't care state*	2	1
8	*Don't care state*	3	1
9	Mintermo	2	1
10	Mintermo	2	1
11	Maxtermo	1	0
12	Mintermo	2	1
13	*Don't care state*	2	1
14	Maxtermo	1	0
15	*Don't care state*	1	0

Fonte: Dados do próprio autor.

Tabela 32 – Tabela verdade da função $M(f1, f2, f3)$.

Termo	**Descrição**	$\boldsymbol{f1}$	$\boldsymbol{f2}$	$\boldsymbol{f3}$	$\boldsymbol{M(f1, f2, f3)}$
0	Mintermo	0	1	1	1
1	Maxtermo	0	1	0	0
2	Maxtermo	0	0	1	0
3	Mintermo	0	1	1	1
4	Maxtermo	1	0	0	0
5	Mintermo	1	1	0	1
6	Mintermo	1	0	1	1
7	Mintermo	1	1	1	1
8	Maxtermo	0	0	1	0
9	Mintermo	0	1	1	1
10	Mintermo	1	0	1	1
11	Maxtermo	0	1	0	0
12	Mintermo	1	0	1	1
13	Maxtermo	0	0	1	0
14	Maxtermo	1	0	0	0
15	Mintermo	1	1	0	1

Fonte: Dados do próprio autor.

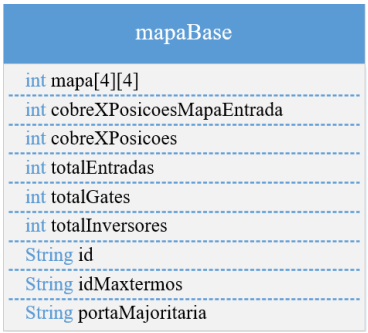
Para abstrair todo o potencial do método, um programa de computador torna-se necessário. Assim, foi desenvolvido neste trabalho, o programa denominado MK4 que utiliza como base o método do mapa-K para gerar uma função majoritária minimizada. No Capítulo 6 descreve-se o funcionamento do programa MK4.

6 PROGRAMA MK4

Neste capítulo apresenta-se o programa MK4 desenvolvido neste trabalho, que implementa o método do mapa-K de quatro variáveis. O MK4 foi implementado na linguagem C++ utilizando a IDE Visual Studio 2017 no sistema operacional *Windows* 10. Os critérios de custo do MK4 são os mesmos propostos por Kong, Shang e Lu (2010), apresentados na seção 3.4. Porém, o MK4 também leva em consideração o critério utilizado no *exact_mig*, onde uma porta majoritária repetida só é considerada uma vez na contagem do custo.

No MK4 as funções primitivas são representadas pela classe mapaBase conforme apresentado na Figura 47.

Figura 47 – Classe mapaBase.



Fonte: Dados do próprio autor.

Na Figura 48 apresenta-se o pseudocódigo do programa MK4. Para o entendimento do pseudocódigo as seguintes variáveis devem ser definidas:

- tabelaVerdade: Variável responsável por armazenar a tabela verdade de entrada;
- listaMB: Lista contendo todos os mapas das funções primitivas que realizam a cobertura de ao menos um mintermo da tabela verdade;
- melhorFuncao: A melhor função gerada para a tabela verdade de entrada;
- listaCB: Lista em que cada item representa a combinação de 3 mapas de funções primitivas ou um mapa de uma função primitiva;

- listaDCB: Lista em que cada item representa a combinação de 2 itens de listaCB.

Figura 48 – Pseudocódigo do programa MK4.

```
Início{
    tabelaVerdade = leituraEntrada();
    //Leitura da tabela verdade de entrada
    listaMB       = adicionaFunçõesPrimitivas();
    //Carrega as funções primitivas para o vetor listaMB.
    if(verificaFunçãoPrimitiva() == true){
        //Se uma função primitiva cobrir todos os mintermos de entrada:
        melhorFunção = funçãoPrimitiva;
    }
    else{
        if(combinaAteTresFunçõesPrimtivas() == true){
            // Se uma função válida for gerada ao combinar 3 funções primitivas:
            melhorFunção = melhorCombinação;
        }
        else{
                listaCB = listaMB + geraListaTodasCombinações();
                // São geradas todas as combinações de 3 funções que cobrem ao menos um mintermo
                for(int i=0 ;i < listaCB.tamanho();i++){
                    for(int j=i+1;j < listaCB.tamanho();j++){
                        // São combinados dois itens de ListaCB
                        if(verificaCombinação(listaCB[i],listaCB[j]) == true){
                            //Se a combinação resultante permitir a escolha de outro ramo:
                                listaDCB.add(listaCB[i],listaCB[j]);
                                if(listaDCB.tamanho == 15000){
                        // Para limitar o espaço de busca o tamanho de listaDCB é limitado para 15000 combinações.
                                    break;
                                }
                        }
                    }
                }
```

(a) Parte 1

```
                for(int i=0;i < listaDCB.tamanho();i++){
                        funcaoEncontrada = geraTerceiroRamo(listaDCB[i]);
                        //Para cada item de listaDCB é gerado um terceiro ramo que torne a função resultante válida
                        if(custo(funcaoEncontrada) < custo(melhorFuncao)){
                            //Se o custo da função gerada for inferior ao custo da melhor função até o momento
                            melhorFuncao = funcaoEncontrada;
                        }
                }
            }
        }
    }
    melhorFuncao = validaMaxtermos(melhorFuncao);
    //Verifica se é possível inverter a função para reduzir o número de inversores
    imprimir(melhorFunção);
}
```

(b) Parte 2

Fonte: Dados do próprio autor.

O programa MK4 é executado com as seguintes etapas:

1. Inicia-se o programa com a entrada de uma tabela verdade contendo os mintermos de uma função que se deseja minimizar. A tabela verdade de entrada pode ser informada no padrão binário ou no padrão hexadecimal.

Para exemplificar, tem-se a função de quatro variáveis Σm(0,10). O programa MK4 aceita a entrada dessa função como sendo "1000000000100000" em que cada dígito representa um termo da função. Assim o primeiro valor "1" representa o mintermo 0 e o outro valor "1" representa o mintermo 10. Também aceita-se o valor hexadecimal equivalente ao número binário "1000000000100000" sendo este 0x8020 ou 8020.

É verificado se todos os mintermos da tabela são cobertos por apenas uma função primitiva. Se uma única função primitiva cobrir todos os mintermos sem cobrir nenhum maxtermo, a função primitiva é selecionada e o Passo 4 é executado. Se nenhuma função primitiva for encontrada, então todos as funções primitivas que realizam a cobertura de ao menos um mintermo são adicionadas no vetor listaMB e executa-se o Passo 2;

2. São realizadas todas as combinações possíveis, de até três funções primitivas, entre as funções contidas em listaMB. É selecionada a combinação que cobrir duas vezes os mintermos de entrada e que tenha o menor custo. Nenhum maxtermo deve ser coberto mais de uma vez. Se uma função válida for gerada, é executado o Passo 4. Se não, executa-se o Passo 3.

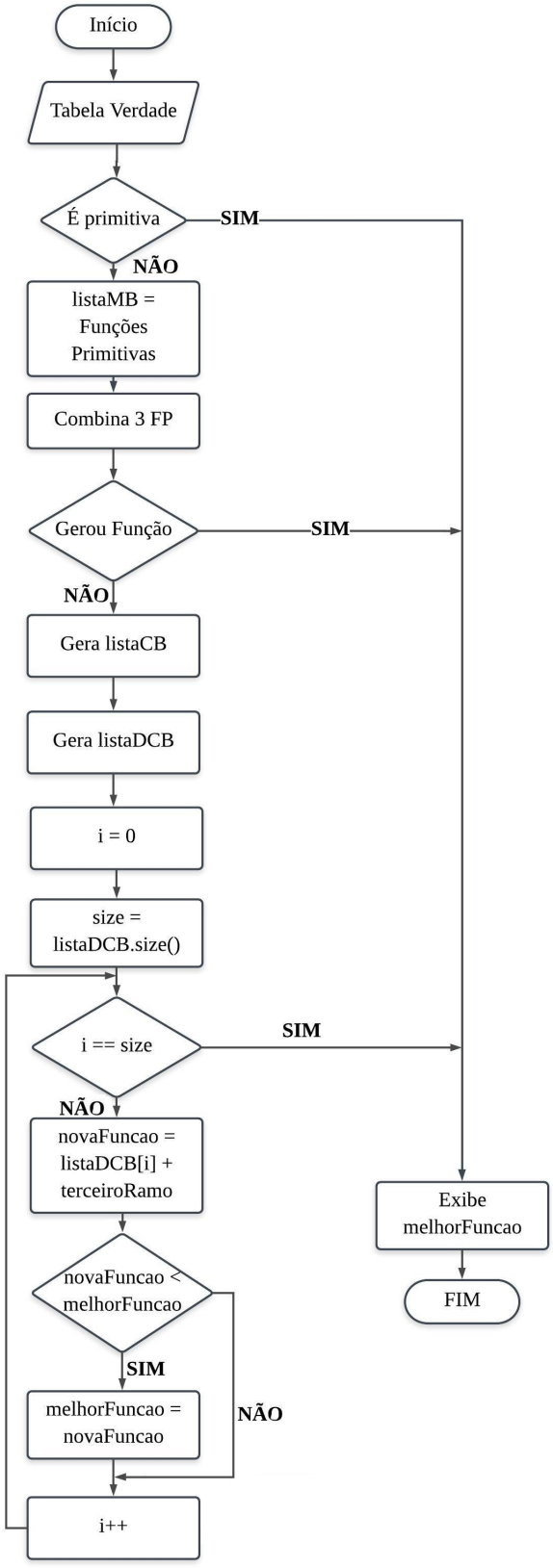
3. Gera-se o vetor listaCB contendo todas as combinações de 3 funções primitivas e todos os itens de listaMB. Então cada item de listaCB é combinado com outro item para gerar o vetor listaDCB. Para cada item de listaDCB é gerada uma terceira função que cubra todos os mintermos que ainda não foram cobertos duas vezes. A função de menor custo gerada é selecionada e executa-se o Passo 4;

4. Estrutura-se uma função majoritária utilizando as funções encontradas. É verificado se é possível aplicar o axioma de inversão na função para reduzir o número de inversores, se sim a função é atualizada. Executa-se o Passo 5;

5. A função gerada é exibida na tela e o programa é encerrado.

Na Figura 49 apresenta-se o fluxograma do programa MK4.

Figura 49 – Fluxograma do programa MK4.



Fonte: Dados do próprio autor.

6.1 RESULTADOS OBTIDOS

A fim de avaliar o programa desenvolvido em relação ao custo da função minimizada, foram coletados os resultados obtidos pelo MK4 para as 65.536 possíveis funções de 4 variáveis. Os resultados foram comparados em termos de níveis, número de portas majoritárias, número de entradas e número de inversores com os resultados obtidos pelo programa *exact_mig*.

Como descrito no Capítulo 4, o *exact_mig* possui dois parâmetros distintos de execução. Foram coletados os resultados gerados pelo *exact_mig* executado com o comando *exact_mig -o 2*, ou seja, com o objetivo primário a redução de níveis e o secundário a redução de portas majoritárias.

Para a comparação de resultados, todas as 65.536 funções foram divididas em 16 conjuntos. Cada conjunto possui x funções que cobrem p mintermos. Na Equação 47 apresenta-se a expressão utilizada para calcular quantas funções de p mintermos existem, em que n representa o total máximo de mintermos que uma função pode cobrir.

$$x = \frac{n!}{p!(n-p)!} \tag{47}$$

Na Tabela 33 apresenta-se a comparação de resultados entre os dois programas. Nota-se que as funções estão separadas pelo total de mintermos cobertos.

Para exemplificar considere a linha 12 da tabela. Existem no total 1.820 funções, de quatro variáveis, que cobrem 12 mintermos. O programa MK4 gerou 1.522 funções de menor custo, 251 funções de custo equivalente e 47 resultados de custo maior em relação as funções geradas pelo programa *exact_mig*.

Todas as funções foram otimizadas no número de níveis em ambos os programas. As melhores funções otimizadas pelo MK4 foram funções geradas com um menor número de entradas ou inversores.

De 65.536 funções comparadas, o programa MK4 gerou aproximadamente 76,36% funções com menor custo, 16,24% funções com custo igual e somente 7,40% das funções geradas tiveram custo superior aos das funções geradas pelo programa *exact_mig*.

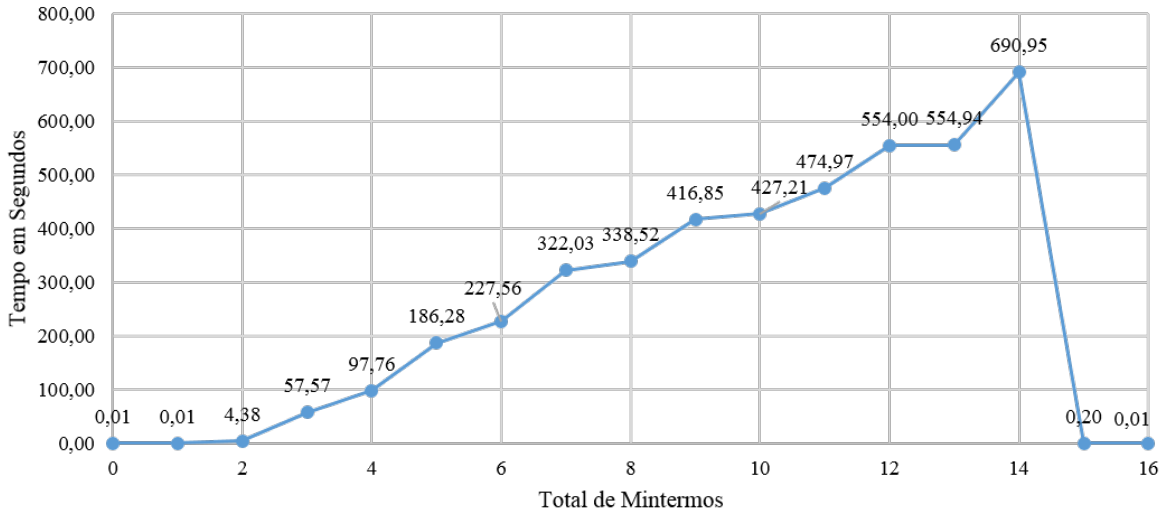
Na Figura 50 apresenta-se o gráfico de tempo médio, em segundos, gasto pelo programa MK4 para gerar cada função. Nota-se que o gráfico está dividido pelo total de mintermos que uma função possui.

Tabela 33 – Comparação de resultados obtidos pelo MK4 e obtidos pelo *exact_mig*.

Tot. Mint.	Tot. Fun.	MK4 <exact.	MK4 = exact.	MK4 > exact.
0	1	0	1	0
1	16	10	6	0
2	120	44	75	1
3	560	363	197	0
4	1.820	1.257	494	69
5	4.368	3.504	835	29
6	8.008	5.562	1.818	628
7	11.440	9.216	1.208	1.016
8	12.870	9.014	2.979	877
9	11.440	9.367	914	1.159
10	8.008	5.807	1.269	932
11	4.368	3.809	473	86
12	1.820	1.522	251	47
13	560	461	97	2
14	120	96	24	0
15	16	12	4	0
16	1	0	1	0
Total	65.536	50.044	10.646	4.846

Fonte: Dados do próprio autor.

Figura 50 – Média de tempo gasto em execução pelo MK4 para gerar uma função de p mintermos.

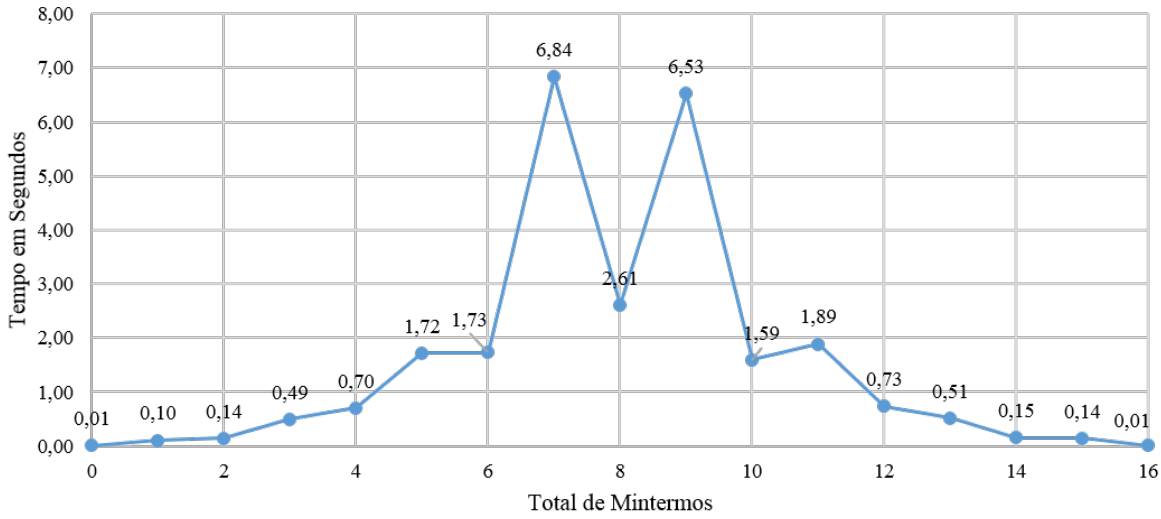


Fonte: Dados do próprio autor.

Nota-se que o tempo médio gasto com cada função aumenta conforme o aumento da quantidade de mintermos e diminui somente quando a função possui 15 ou 16 mintermos. A queda no tempo de processamento se deve ao fato de que existem poucas funções que cobrem de 15 a 16 mintermos e que precisam de três níveis para serem geradas.

Na Figura 51 apresenta-se o gráfico de tempo médio gasto pelo programa *exact_mig* para gerar cada função.

Figura 51 – Média de tempo gasto pelo *exact_mig* para gerar uma função de p mintermos.



Fonte: Dados do próprio autor.

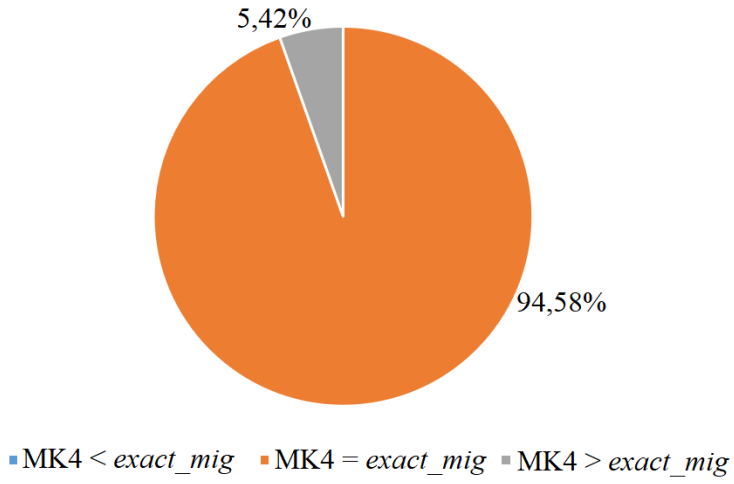
Nota-se que o tempo médio gasto pelo *exact_mig* é significativamente inferior ao tempo gasto pelo programa MK4. Isso se deve ao fato do *exact_mig* possuir menos critérios de custo.

As funções geradas por ambos os programas também foram comparadas considerando-se unicamente o número de níveis e portas majoritárias, o número de níveis, portas majoritárias e entradas. E também somente o número de níveis, portas majoritárias e inversores.

Na Figura 52 apresenta-se o gráfico de porcentagens de funções melhores otimizadas pelo programa MK4 e pelo programa *exact_mig* quando considerados como custo o número de níveis e de portas majoritárias.

Na Tabela 34 apresenta-se a comparação de resultado entre os dois métodos quando considerados o número de níveis e portas majoritárias na contagem do custo.

Figura 52 – Porcentagem de funções melhores otimizadas pelo MK4 e pelo *exact_mig* considerando apenas o número de níveis e portas majoritárias.



Fonte: Dados do próprio autor.

Tabela 34 – Comparação de resultados considerando o número de níveis e portas majoritárias.

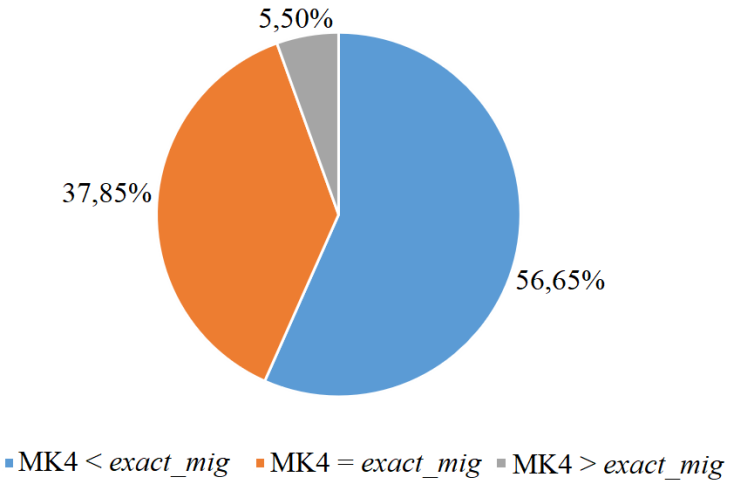
Tot. Mint.	Tot. Func.	MK4 < exact.	MK4 = exact.	MK4 > exact.
0	1	0	1	0
1	16	0	16	0
2	120	0	120	0
3	560	0	560	0
4	1.820	0	1.808	12
5	4.368	0	4.362	6
6	8.008	0	7.600	408
7	11.440	0	10.546	894
8	12.870	0	12.354	516
9	11.440	0	10.500	940
10	8.008	0	7.285	723
11	4.368	0	4326	42
12	1.820	0	1.808	12
13	560	0	560	0
14	120	0	120	0
15	16	0	16	0
16	1	0	1	0
Total	65.536	0	61.983	3.553

Fonte: Dados do próprio autor.

Na Figura 53 apresenta-se o gráfico de porcentagens de funções melhores otimizadas pelo programa MK4 e pelo programa *exact_mig* quando considerados como custo o número

de níveis, número de portas majoritárias e número de entradas.

Figura 53 – Porcentagem de funções melhores otimizadas pelo MK4 e pelo *exact_mig* considerando apenas o número de níveis, portas majoritárias e entradas.

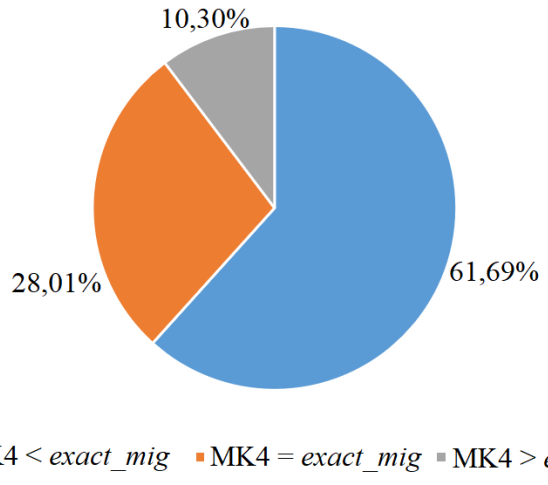


Fonte: Dados do próprio autor.

Na Tabela 35 apresenta-se a comparação de resultado entre os dois métodos quando considerados o número de níveis, portas majoritárias e entradas na contagem do custo.

Na Figura 54 apresenta-se o gráfico de porcentagens de funções melhores otimizadas pelo programa MK4 e pelo programa *exact_mig* quando considerados como custo o número de níveis, número de portas majoritárias e número de inversores.

Figura 54 – Porcentagem de funções melhores otimizadas pelo MK4 e pelo *exact_mig* considerando apenas o número de níveis, portas majoritárias e inversores.



Fonte: Dados do próprio autor.

Na Tabela 36 apresenta-se a comparação de resultado entre os dois métodos quando considerados o número de níveis, portas majoritárias e inversores na contagem do custo.

Tabela 35 – Comparação de resultados considerando o número de níveis, portas majoritárias e entradas.

Tot. Mint.	**Tot. Func.**	**MK4 < exact.**	**MK4 = exact.**	**MK4 > exact.**
0	1	0	1	0
1	16	8	8	0
2	120	14	106	0
3	560	278	282	0
4	1.820	1.047	761	12
5	4.368	2.991	1.367	10
6	8.008	3.699	3.901	408
7	11.440	7.774	2.762	904
8	12.870	5.666	6.670	534
9	11.440	7.720	2.766	954
10	8.008	3.616	3.665	727
11	4.368	2.966	1.357	45
12	1.820	1.048	760	12
13	560	278	282	0
14	120	14	106	0
15	16	8	8	0
16	1	0	1	0
Total	65.536	37.127	24.803	3.606

Fonte: Dados do próprio autor.

Na Figura 55 apresenta-se o gráfico de memória média gasta, em *megabytes* (MB), pelo programa MK4 e pelo programa *exact_mig* para gerar funções de p mintermos. Nota-se que a memória gasta pelo programa *exact_mig* é significativamente inferior a memória gasta pelo programa MK4. Isso se deve ao fato do *exact_mig* possuir menos critérios de custo e utilizar um MIG base para geração de uma função.

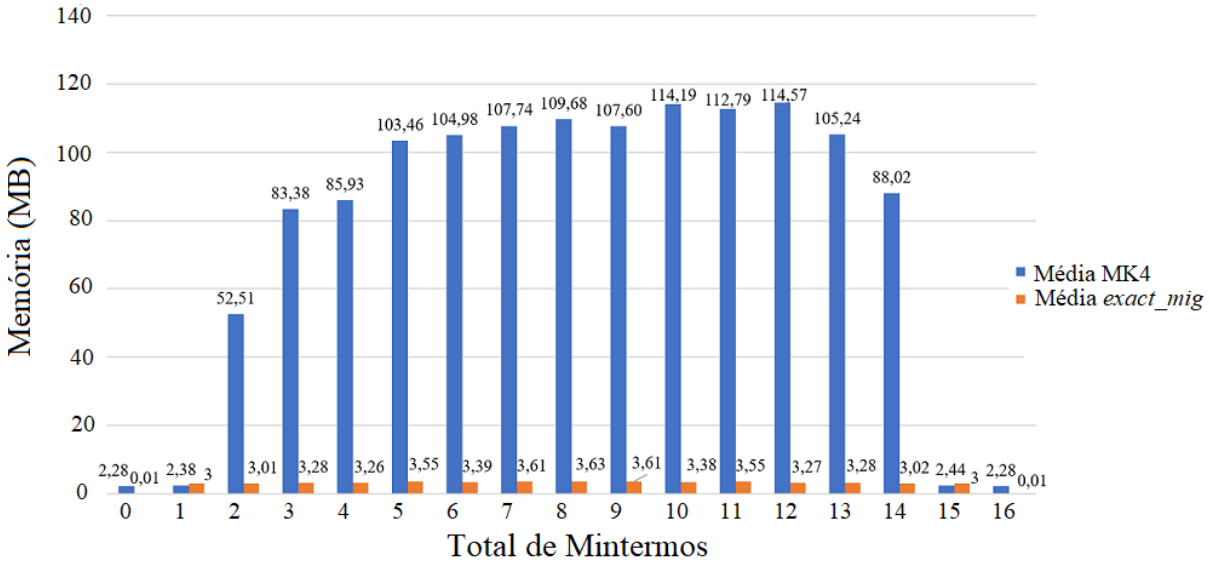
Com os resultados apresentados nota-se que o programa MK4 consegue gerar casos com melhores resultados que os do programa *exact_mig*, mas ainda não é capaz de gerar todos os resultados melhores ou iguais. Isso se deve ao fato de que o programa MK4 faz uma busca exaustiva ao combinar as funções primitivas para gerar o vetor ListaCB. Existem inúmeras funções que cobrem uma mesma tabela verdade que poderiam ser selecionadas como $f1$, $f2$ e $f3$. Assim, para que o tempo de execução não se tornasse excessivamente grande, as funções com tabelas verdade repetidas com um custo superior as demais são descartadas. Nesse processo algumas funções que poderiam gerar um resultado otimizado podem ser excluídas gerando os casos de maior custo.

Tabela 36 – Comparação de resultados considerando o número de níveis, portas majoritárias e inversores.

Tot. Mint.	Tot. Func.	MK4 <exact.	MK4 = exact.	MK4 >exact.
0	1	0	1	0
1	16	4	12	0
2	120	43	76	1
3	560	256	296	8
4	1.820	906	735	179
5	4.368	2.658	1.446	264
6	8.008	4.570	2.642	796
7	11.440	7.110	2.929	1.401
8	12.870	7.326	4.376	1.168
9	11.440	7.585	2.357	1.498
10	8.008	5.019	1.902	1.087
11	4.368	3.195	933	240
12	1.820	1.247	471	102
13	560	398	153	9
14	120	96	24	0
15	16	12	4	0
16	1	0	1	0
Total	65.536	40.425	18.358	6.753

Fonte: Dados do próprio autor.

Figura 55 – Média de memória gasta em execução pelos programas MK4 e *exact_mig* para gerar uma função de p mintermos.



Fonte: Dados do próprio autor.

6.2 EXPANSÃO DO PROGRAMA MK4 PARA 5 VARIÁVEIS

Para fazer com que o programa MK4 funcione para até 5 variáveis, o teorema de expansão de Shannon (1949) é utilizado.

O teorema define que qualquer função booleana $f(x1, x2, ...xn)$ pode ser expressa pela Equação 48, em que a variável com o *bit* mais significativo é isolada e são geradas duas outras funções com as variáveis restantes, denominadas funções resíduo.

$$f(x1, x2, ..., xn) = x1.f(1, x2, ...xn) + \overline{x1}.f(0, x2..., xn) \tag{48}$$

Utilizando a lógica majoritária, o teorema de expansão de Shannon pode ser escrito como representado pela Equação 49 em que $x1$ é isolado e são geradas duas funções majoritárias $f1$ e $f2$ geradas com os termos restantes $x2, ...xn$.

$$f(x1, x2, .., xn) = M(M(\overline{x1}, 0, f1), 1, M(x1, 0, f2)) \tag{49}$$

Assim, visando a expansão do programa MK4 para cinco variáveis, o teorema de expansão de Shannon foi combinado com o método do mapa-K para funções majoritárias.

Para exemplificar o método, tem-se como exemplo a função Σm(0,8,15,16,25,26,31). O primeiro passo é formar a tabela verdade dessa função, em que a variável com o *bit* mais significativo é rotulada de X. Na Tabela 37 apresenta-se a tabela verdade da função.

No segundo passo, divide-se a tabela verdade em duas, uma contendo os termos de 0 a 15 e outra contendo os termos de 16 a 31. Neste passo a variável X é desconsiderada.

Na Tabela 38 apresenta-se a tabela verdade gerada para os 16 primeiros termos.

Os mintermos com saída verdadeira são utilizados como entrada no programa MK4 que gera a função representada na Equação 50 e que possui saída verdadeira para os mintermos 0, 8 e 15.

$$M(M(\overline{B}, 1, C), \overline{M}(C, 1, D), M(A, M(B, 0, D), 0)) \tag{50}$$

Como a variável X possui valor lógico falso na primeira parte da tabela verdade original, conecta-se a função gerada com a variável $\overline{X}$ em uma porta majoritária funcionando como uma porta E, gerando a função $f1$ representada na Equação 51 que possui saída verdadeira para os mintermos 0, 8 e 15.

$$f1 = M(\overline{X}, 0, M(M(\overline{B}, 1, C), \overline{M}(C, 1, D), M(A, M(B, 0, D), 0))) \tag{51}$$

Tabela 37 – Tabela verdade da função Σm(0,8,15,16,25,26,31).

Termo	***X***	***A***	***B***	***C***	***D***	**Saída**	**Descrição**
0	0	0	0	0	0	1	Verdadeiro
1	0	0	0	0	1	0	Falso
2	0	0	0	1	0	0	Falso
3	0	0	0	1	1	0	Falso
4	0	0	1	0	0	0	Falso
5	0	0	1	0	1	0	Falso
6	0	0	1	1	0	0	Falso
7	0	0	1	1	1	0	Falso
8	0	1	0	0	0	1	Verdadeiro
9	0	1	0	0	1	0	Falso
10	0	1	0	1	0	0	Falso
11	0	1	0	1	1	0	Falso
12	0	1	1	0	0	0	Falso
13	0	1	1	0	1	0	Falso
14	0	1	1	1	0	0	Falso
15	0	1	1	1	1	1	Verdadeiro
16	1	0	0	0	0	1	Verdadeiro
17	1	0	0	0	1	0	Falso
18	1	0	0	1	0	0	Falso
19	1	0	0	1	1	0	Falso
20	1	0	1	0	0	0	Falso
21	1	0	1	0	1	0	Falso
22	1	0	1	1	0	0	Falso
23	1	0	1	1	1	0	Falso
24	1	1	0	0	0	0	Falso
25	1	1	0	0	1	1	Verdadeiro
26	1	1	0	1	0	1	Verdadeiro
27	1	1	0	1	1	0	Falso
28	1	1	1	0	0	0	Falso
29	1	1	1	0	1	0	Falso
30	1	1	1	1	0	0	Falso
31	1	1	1	1	1	1	Verdadeiro

Fonte: Dados do próprio autor.

Então gera-se a segunda parte da tabela verdade original, excluindo a variável X e os termos de 16 a 31 são tratados como se fossem de 0 a 15.

Tabela 38 – Tabela verdade para os 16 primeiros termos.

Termo	***A***	***B***	***C***	***D***	**Saída**	**Descrição**
0	0	0	0	0	1	Verdadeiro
1	0	0	0	1	0	Falso
2	0	0	1	0	0	Falso
3	0	0	1	1	0	Falso
4	0	1	0	0	0	Falso
5	0	1	0	1	0	Falso
6	0	1	1	0	0	Falso
7	0	1	1	1	0	Falso
8	1	0	0	0	1	Verdadeiro
9	1	0	0	1	0	Falso
10	1	0	1	0	0	Falso
11	1	0	1	1	0	Falso
12	1	1	0	0	0	Falso
13	1	1	0	1	0	Falso
14	1	1	1	0	0	Falso
15	1	1	1	1	1	Verdadeiro

Fonte: Dados do próprio autor.

Assim na Tabela 39 apresenta-se a tabela verdade para os 16 últimos termos.

Como os termos 0, 9, 10 e 15 correspondem, respectivamente, aos mintermos 16, 25, 26 e 31 da tabela verdade, possuem saída verdadeira, são considerados os mintermos de entrada para o programa MK4 que gera a função representada na Equação 52.

$$\overline{M}(M(B,\overline{C},M(A,0,\overline{D})),M(C,\overline{A},M(\overline{B},0,D)),1) \tag{52}$$

Como a variável X possui valor lógico verdadeiro na segunda parte da tabela verdade original, conecta-se a função gerada com a variável X em uma porta majoritária funcionando como uma porta E, gerando a função $f2$ representada na Equação 53 e que possui saída verdadeira para os mintermos 16, 25, 26, e 31.

$$f2 = M(X,0,\overline{M}(M(B,\overline{C},M(A,0,\overline{D})),M(C,\overline{A},M(\overline{B},0,D)),1)) \tag{53}$$

No último passo, as duas funções geradas são conectadas a uma porta majoritária funcionando como uma porta OU gerando a função $M(f1,1,f2)$ que cobre os mintermos 0, 8, 15, 16, 25, 26 e 31.

Tabela 39 – Tabela verdade para os 16 últimos termos.

Termo	***A***	***B***	***C***	***D***	**Saída**	**Descrição**
0	0	0	0	0	1	Verdadeiro
1	0	0	0	1	0	Falso
2	0	0	1	0	0	Falso
3	0	0	1	1	0	Falso
4	0	1	0	0	0	Falso
5	0	1	0	1	0	Falso
6	0	1	1	0	0	Falso
7	0	1	1	1	0	Falso
8	1	0	0	0	0	Falso
9	1	0	0	1	1	Verdadeiro
10	1	0	1	0	1	Verdadeiro
11	1	0	1	1	0	Falso
12	1	1	0	0	0	Falso
13	1	1	0	1	0	Falso
14	1	1	1	0	0	Falso
15	1	1	1	1	1	Verdadeiro

Fonte: Dados do próprio autor.

Contudo, destaca-se que a geração de uma função majoritária utilizando o teorema de expansão de Shannon não gera uma função ótima pois, como nota-se na Equação 49, para aplicar o teorema de forma correta é necessário adicionar dois níveis e 3 portas majoritárias para conectar as funções $f1$ e $f2$. Apesar disso, esta abordagem permite que funções majoritárias com 6, 7 ou mais variáveis possam ser otimizadas, possibilitando a geração de funções com custo menor que uma determinada função inicial.

Neste capítulo foi apresentado o programa MK4 e a comparação entre os resultados gerados pelo MK4 e os resultados gerados pelo programa *exact_mig*. No capítulo 7 é apresentado a conclusão deste trabalho e as propostas para trabalhos futuros.

7 CONCLUSÕES

Neste trabalho foram apresentados os principais conceitos sobre a lógica majoritária, com foco em métodos de minimização. O algoritmo do mapa-K, para quatro variáveis, foi implementando em um programa de computador denominado MK4. Foram testados todos os 65.536 casos de funções de quatro variáveis. Os resultados obtidos pelo programa MK4 foram comparados com os resultados gerados pelo programa *exact_mig*. O MK4 gerou 50.044 funções de menor custo, 10.646 funções equivalentes e 4.846 funções de custo maior quando comparadas ao *exact_mig*. Assim confirma-se que o programa MK4 é competitivo em relação ao estado atual da arte, gerando 92,60% das funções de custo igual ou inferior as funções geradas pelo programa *exact_mig*.

Para trabalhos futuros são sugeridos os seguintes tópicos:

- Melhorar o tempo de execução do programa MK4 testando o uso de meta-heurísticas;
- Verificar e documentar se a ocorrência de *spikes* que ocorrem em circuitos utilizando a lógica convencional é solucionada com o uso da lógica majoritária;
- Elaborar um estudo aprofundado sobre a tecnologia QCA;
- Elaborar um estudo aprofundado sobre o uso de solucionadores de satisfatibilidade booleana e sobre a síntese exata de funções;
- Validar o uso da lógica majoritária acrescentando o operador XOR;
- Expandir o programa MK4 para que funcione para funções com mais entradas e com o número de níveis otimizado.

REFERÊNCIAS

AKERS, S. B. On the algebraic manipulation of majority logic. **IRE Transactions on Electronic Computers**, New York, EC-10, n. 4, p. 779, 1961.

AMARU, L. *et al.* A sound and complete axiomatization of majority-n logic. **IEEE Transactions on Computers**, Washington, v. 65, n. 9, p. 2889–2895, 2016.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Majority-inverter graph: a novel data-structure and algorithms for efficient logic optimization. In: ANNUAL DESIGN AUTOMATION CONFERENCE, 51., 2014, San Francisco. **Proceedings...** San Francisco: IEEE, 2014. p. 1–6.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Boolean logic optimization in majority-inverter graphs. In: DESIGN AUTOMATION CONFERENCE- DAC, 52., 2015, Portland. **Proceedings...** Portland: IEEE, 2015. p. 1–6.

AMARU, L. G. **New data structures and algorithms for logic synthesis and verification**. Lausana: Springer, 2017. 156 p.

BERTACCO, V.; DAMIANI, M. The disjunctive decomposition of logic functions. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN- ICCAD, 15., 1997, San Jose. **Proceedings...** San Jose: IEEE, 1997. p. 78–82.

BOOLE, G. **An investigation of the laws of thought**: on which are founded the mathematical theories of logic and probabilities. Cork: Dover Publications, 1854. 424 p.

BRAYTON, R. K. *et al.* **Logic minimization algorithms for VLSI synthesis**. Boston: Springer Science & Business Media, 1984. 194 p.

CHU, Z. *et al.* Functional decomposition using majority. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 23., 2018, Jeju. **Proceedings...** Jeju: IEEE, 2018. p. 676–681.

COHN, M.; LINDAMAN, R. Axiomatic majority-decision logic. **IRE Transactions on Electronic Computers**, New York, EC-10, n. 1, p. 17–21, 1961.

CORDEIRO, L.; FISCHER, B.; MARQUES-SILVA, J. Smt-based bounded model checking for embedded ansi-c software. **IEEE Transactions on Software Engineering**, Washington, v. 38, n. 4, p. 957–974, 2012.

DAGENAIS, M. R.; AGARWAL, V. K.; RUMIN, N. C. Mcboole: A new procedure for exact logic minimization. **IEEE transactions on computer-aided design of integrated circuits and systems**, New York, v. 5, n. 1, p. 229–238, 1986.

FERRAZ, E. *et al.* Synthesis of majority expressions through primitive function manipulation. In: INTERNATIONAL WORKSHOP ON BOOLEAN PROBLEMS, 13., 2018, Bremen. **Proceedings...** Bremen: IWSBP, 2018. p. 117–132.

GANZINGER, H. *et al.* Dpll (t): Fast decision procedures. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, 16., 2004, Boston. **Proceedings...** Boston: Springer, 2004. p. 175–188.

HLAVICKA, J.; FISER, P. Boom-a heuristic boolean minimizer. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN., 2., 2001, Amsterdam. **Proceedings...** Amsterdam: IEEE, 2001. p. 439–442.

KARNAUGH, M. The map method for synthesis of combinational logic circuits. **Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics**, New York, v. 72, n. 5, p. 593–599, 1953.

KOHAVI, Z.; JHA, N. K. **Switching and finite automata theory**. New York: Cambridge University Press, 2009. 658 p.

KONG, K.; SHANG, Y.; LU, R. An optimized majority logic synthesis methodology for quantum-dot cellular automata. **IEEE Transactions on Nanotechnology**, Piscataway, v. 9, n. 2, p. 170–183, 2010.

LINDAMAN, R. A theorem for deriving majority-logic networks within an augmented boolean algebra. **IRE Transactions on Electronic Computers**, IEEE, New York, v. EC-9, n. 3, p. 338–342, 1960.

MCCLUSKEY, E. J. Minimization of boolean functions. **Bell Labs Technical Journal**, Kansas, v. 35, n. 6, p. 1417–1444, 1956.

MISHRA, V. K.; THAPLIYAL, H. Heuristic based majority/minority logic synthesis for emerging technologies. In: INTERNATIONAL CONFERENCE ON EMBEDDED SYSTEMS- VLSID, 16., 2017, Hyderabad. **Proceedings...** Hyderabad: IEEE, 2017. p. 295–300.

MOURA, L. D.; BJØRNER, N. Z3: an efficient smt solver. In: INTERNATIONAL CONFERENCE ON TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEM, 13., 2008, Budapeste. **Proceedings...** Budapeste: Springer, 2008. p. 337–340.

MOURA, L. de; DUTERTRE, B.; SHANKAR, N. A tutorial on satisfiability modulo theories. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, 18., 2007, Berlin. **Proceedings...** Berlin: Springer, 2007. p. 20–36.

SASAO, T. **Switching theory for logic synthesis**. Berlim: Springer Science & Business Media, 2012. 362 p.

SENTOVICH, E. M. *et al.* **SIS**: a system for sequential circuit synthesis. *[S.l.: s.n]*, 1992. Disponível em: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/2010.html. Acesso em: 28 ago. 2018.

SHANNON, C. E. The synthesis of two-terminal switching circuits. **Bell System Technical Journal**, Kansas, v. 28, n. 1, p. 59–98, 1949.

SOEKEN, M. **CirKit**: documentation. *[S.l.: s.n.]*, 2017. Disponível em: http://cirkit.readthedocs.io/en/latest/. Acesso em: 10 nov 2018.

SOEKEN, M. *et al.* Exact synthesis of majority-inverter graphs and its applications. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, La Jolla, v. 36, n. 11, p. 1842–1855, 2017.

WALUS, K. *et al.* Circuit design based on majority gates for applications with quantum-dot cellular automata. In: SIGNALS, SYSTEMS AND COMPUTERS., 38., 2004, Pacific Grove. **Proceedings...** Pacific Grove: IEEE, 2004. v. 2, p. 1354–1357.

WANG, P. **A comprehensive technique for majority/minority logic synthesis with applications in nanotechnology**. 2014. 136 f. Tese (Doctor of Philosophy) — College of Electrical Engineering, The University of Toledo, Toledo, 2014.

WANG, P.; NIAMAT, M.; VEMURU, S. Minimal majority gate mapping of 4-variable functions for quantum cellular automata. In: NANOTECHNOLOGY- IEEE-NANO, 11., 2011, Portland. **Proceedings...** Portland: IEEE, 2011. p. 1307–1312.

WANG, P. *et al.* Comprehensive majority/minority logic synthesis method. In: NANOTECHNOLOGY- IEEE-NANO, 13., 2013, Beijing. **Proceedings...** Beijing: IEEE, 2013. p. 694–697.

WANG, P. *et al.* Synthesis of majority/minority logic networks. **IEEE Transactions on Nanotechnology**, Piscataway, v. 14, n. 3, p. 473–483, 2015.

WIDMER, N. S.; MOSS, G. L.; TOCCI, R. J. **Digital systems: principles and applications**. 12. ed. [*S.l.*]: Pearson, 2017. 1026 p.

ZHANG, R.; GUPTA, P.; JHA, N. K. Majority and minority network synthesis with application to qca-, set-, and tpl-based nanotechnologies. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Piscataway, v. 26, n. 7, p. 1233–1245, 2007.

ZHANG, R. *et al.* A method of majority logic reduction for quantum cellular automata. **IEEE Transactions on Nanotechnology**, Piscataway, v. 3, n. 4, p. 443–450, 2004.

APÊNDICE A – FUNÇÕES PRIMITIVAS DE QUATRO VARIÁVEIS

Tabela 40 – Funções primitivas de 4 variáveis parte 1.

Função	**Termos cobertos**
0	-
1	Σm(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
A	Σm(8,9,10,11,12,13,14,15)
B	Σm(4,5,6,7,12,13,14,15)
C	Σm(2,3,6,7,10,11,14,15)
D	Σm(1,3,5,7,9,11,13,15)
$\overline{A}$	Σm(0,1,2,3,4,5,6,7)
$\overline{B}$	Σm(0,1,2,3,8,9,10,11)
$\overline{C}$	Σm(0,1,4,5,8,9,12,13)
$\overline{D}$	Σm(0,2,4,6,8,10,12,14)
M(A,0,B)	Σm(12,13,14,15)
M(A,0,C)	Σm(10,11,14,15)
M(A,0,D)	Σm(9,11,13,15)
M(B,0,C)	Σm(6,7,14,15)
M(B,0,D)	Σm(5,7,13,15)
M(C,0,D)	Σm(3,7,11,15)
M(A,1,B)	Σm(4,5,6,7,8,9,10,11,12,13,14,15)
M(A,1,C)	Σm(2,3,6,7,8,9,10,11,12,13,14,15)
M(A,1,D)	Σm(1,3,5,7,8,9,10,11,12,13,14,15)
M(B,1,C)	Σm(2,3,4,5,6,7,10,11,12,13,14,15)
M(B,1,D)	Σm(1,3,4,5,6,7,9,11,12,13,14,15)
M(C,1,D)	Σm(1,2,3,5,6,7,9,10,11,13,14,15)
M(A,0,$\overline{B}$)	Σm(8,9,10,11)
M(A,0,$\overline{C}$)	Σm(8,9,12,13)

Tabela 41 – Funções primitivas de 4 variáveis parte 2.

Função	**Termos cobertos**
M(A,0,$\overline{D}$)	Σm(8,10,12,14)
M(B,0,$\overline{C}$)	Σm(4,5,12,13)
M(B,0,$\overline{D}$)	Σm(4,6,12,14)
M($\overline{A}$,0,B)	Σm(4,5,6,7)
M($\overline{A}$,0,C)	Σm(2,3,6,7)
M($\overline{B}$,0,C)	Σm(2,3,10,11)
M(C,0,$\overline{D}$)	Σm(2,6,10,14)
M($\overline{A}$,0,D)	Σm(1,3,5,7)
M($\overline{B}$,0,D)	Σm(1,3,9,11)
M($\overline{C}$,0,D)	Σm(1,5,9,13)
M($\overline{A}$,1,B)	Σm(0,1,2,3,4,5,6,7,12,13,14,15)
M($\overline{A}$,1,C)	Σm(0,1,2,3,4,5,6,7,10,11,14,15)
M($\overline{A}$,1,D)	Σm(0,1,2,3,4,5,6,7,9,11,13,15)
M(A,1,$\overline{B}$)	Σm(0,1,2,3,8,9,10,11,12,13,14,15)
M($\overline{B}$,1,C)	Σm(0,1,2,3,6,7,8,9,10,11,14,15)
M($\overline{B}$,1,D)	Σm(0,1,2,3,5,7,8,9,10,11,13,15)
M(A,1,$\overline{C}$)	Σm(0,1,4,5,8,9,10,11,12,13,14,15)
M(B,1,$\overline{C}$)	Σm(0,1,4,5,6,7,8,9,12,13,14,15)
M($\overline{C}$,1,D)	Σm(0,1,3,4,5,7,8,9,11,12,13,15)
M(A,1,$\overline{D}$)	Σm(0,2,4,6,8,9,10,11,12,13,14,15)
M(B,1,$\overline{D}$)	Σm(0,2,4,5,6,7,8,10,12,13,14,15)
M(C,1,$\overline{D}$)	Σm(0,2,3,4,6,7,8,10,11,12,14,15)
$\overline{M}$(A,0,B)	Σm(0,1,2,3,4,5,6,7,8,9,10,11)
$\overline{M}$(A,0,C)	Σm(0,1,2,3,4,5,6,7,8,9,12,13)
$\overline{M}$(A,0,D)	Σm(0,1,2,3,4,5,6,7,8,10,12,14)
$\overline{M}$(B,0,C)	Σm(0,1,2,3,4,5,8,9,10,11,12,13)
$\overline{M}$(B,0,D)	Σm(0,1,2,3,4,6,8,9,10,11,12,14)
$\overline{M}$(C,0,D)	Σm(0,1,2,4,5,6,8,9,10,12,13,14)
$\overline{M}$(A,1,B)	Σm(0,1,2,3)
$\overline{M}$(A,1,C)	Σm(0,1,4,5)
$\overline{M}$(A,1,D)	Σm(0,2,4,6)
$\overline{M}$(B,1,C)	Σm(0,1,8,9)
$\overline{M}$(B,1,D)	Σm(0,2,8,10)
$\overline{M}$(C,1,D)	Σm(0,4,8,12)
M(A,B,C)	Σm(6,7,10,11,12,13,14,15)
M(A,B,D)	Σm(5,7,9,11,12,13,14,15)

Tabela 42 – Funções primitivas de 4 variáveis parte 3.

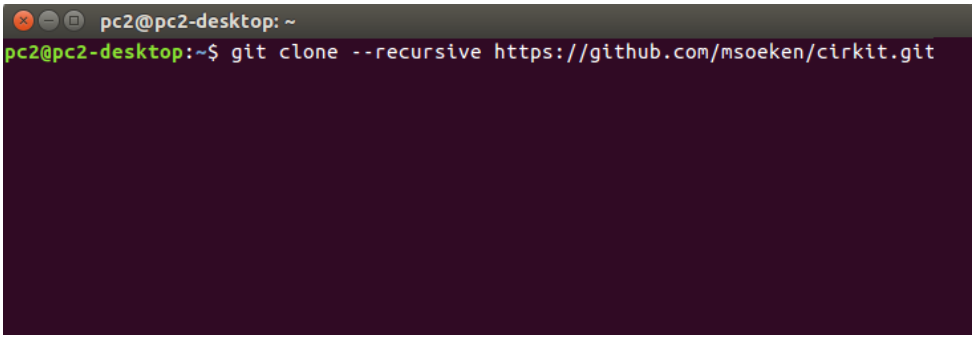
Função	**Termos cobertos**
M(A,C,D)	Σm(3,7,9,10,11,13,14,15)
M(B,C,D)	Σm(3,5,6,7,11,13,14,15)
$\overline{M}$(A,B,C)	Σm(0,1,2,3,4,5,8,9)
M($\overline{A}$,B,C)	Σm(2,3,4,5,6,7,14,15)
M(A,$\overline{B}$,C)	Σm(2,3,8,9,10,11,14,15)
M(A,B,$\overline{C}$)	Σm(4,5,8,9,12,13,14,15)
$\overline{M}$(A,B,D)	Σm(0,1,2,3,4,6,8,10)
M($\overline{A}$,B,D)	Σm(1,3,4,5,6,7,13,15)
M(A,$\overline{B}$,D)	Σm(1,3,8,9,10,11,13,15)
M(A,B,$\overline{D}$)	Σm(4,6,8,10,12,13,14,15)
$\overline{M}$(A,C,D)	Σm(0,1,2,4,5,6,8,12)
M($\overline{A}$,C,D)	Σm(1,2,3,5,6,7,11,15)
M(A,$\overline{C}$,D)	Σm(1,5,8,9,11,12,13,15)
M(A,C,$\overline{D}$)	Σm(2,6,8,10,11,12,14,15)
$\overline{M}$(B,C,D)	Σm(0,1,2,4,8,9,10,12)
M($\overline{B}$,C,D)	Σm(1,2,3,7,9,10,11,15)
M(B,$\overline{C}$,D)	Σm(1,4,5,7,9,12,13,15)
M(B,C,$\overline{D}$)	Σm(2,4,6,7,10,12,14,15)
$\overline{M}$($\overline{A}$,B,C)	Σm(0,1,8,9,10,11,12,13)
$\overline{M}$(A,$\overline{B}$,C)	Σm(0,1,4,5,6,7,12,13)
$\overline{M}$(A,B,$\overline{C}$)	Σm(0,1,2,3,6,7,10,11)
$\overline{M}$($\overline{A}$,B,D)	Σm(0,2,8,9,10,11,12,14)
$\overline{M}$(A,$\overline{B}$,D)	Σm(0,2,4,5,6,7,12,14)
$\overline{M}$(A,B,$\overline{D}$)	Σm(0,1,2,3,5,7,9,11)
$\overline{M}$($\overline{A}$,C,D)	Σm(0,4,8,9,10,12,13,14)
$\overline{M}$(A,$\overline{C}$,D)	Σm(0,2,3,4,6,7,10,14)
$\overline{M}$(A,C,$\overline{D}$)	Σm(0,1,3,4,5,7,9,13)
$\overline{M}$($\overline{B}$,C,D)	Σm(0,4,5,6,8,12,13,14)
$\overline{M}$(B,$\overline{C}$,D)	Σm(0,2,3,6,8,10,11,14)
$\overline{M}$(B,C,$\overline{D}$)	Σm(0,1,3,5,8,9,11,13)

APÊNDICE B – INSTRUÇÕES PARA INSTALAÇÃO E EXECUÇÃO DO PROGRAMA *EXACT_MIG*

O programa *exact_mig* esta disponível no *framework* de minimização denominado *Cirkit* (SOEKEN, 2017). A seguir apresenta-se o passo a passo de instalação do *Cirkit* no sistema operacional Ubuntu.

O primeiro passo é realizar o *download* do *Cirkit*. Para isso, executa-se o comando: *git clone --recursive https://github.com/msoeken/cirkit.git* no terminal do Ubuntu como apresentado na Figura 56.

Figura 56 – *Download* do *framework Cirkit*.



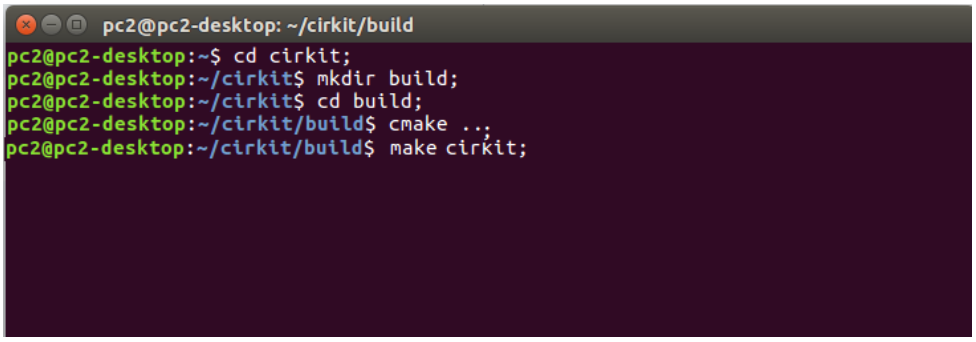
Fonte: Dados do próprio autor.

Após realizar o *download*, é necessário instalar o *framework*. Para isso, dentro da pasta do *Cirkit* executa-se os comandos: *mkdir build; cd build; cmake ..; make cirkit;* como apresentado na Figura 57.

Com o *Cirkit* instalado, utiliza-se o comando *./cirkit* para iniciar a execução do *framework* e obter acesso ao programa *exact_mig*. Para executar o *exact_mig* é necessário informar uma tabela verdade de entrada. Tem-se como exemplo a função $\Sigma m(0, 2, 4, 8, 12, 14, 15)$.

A tabela verdade de entrada deve ser inserida de forma que os termos (decimais) estejam ordenados do maior para o menor, assim a tabela verdade é inserida com o comando *tt* 1101000100010101

Figura 57 – Instalação do *Cirkit*.



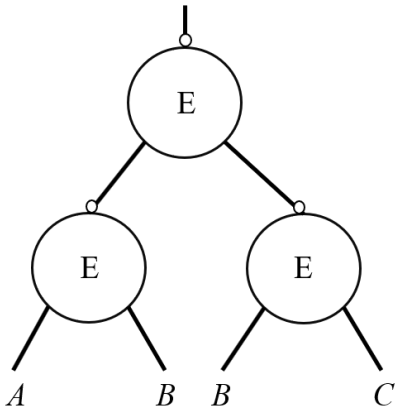
Fonte: Dados do próprio autor.

Depois, visando a conversão da tabela verdade em um MIG, executa-se a sequência de comandos: *convert --tt_to_aig; convert --aig_to_mig*. Após a conversão, o programa *exact_mig* pode ser executado.

Nota-se que é necessário converter a tabela verdade em um AIG e depois convertê-la em um MIG antes de começar a utilizar o programa *exact_mig*.

Os AIGs (*And Inverter Graphs*) são formas de representação de funções booleanas utilizando apenas portas E e $N\tilde{A}O$. A porta OU é simulada utilizando uma porta E com suas entradas e saídas negadas. Na Figura 58 apresenta-se o exemplo de um AIG para a função $f(A, B, C) = A.B + B.C$. Nota-se que o círculo maior representa uma porta E e os círculos menores representam os inversores.

Figura 58 – Exemplo de AIG para a função $f(A, B, C) = A.B + B.C$.



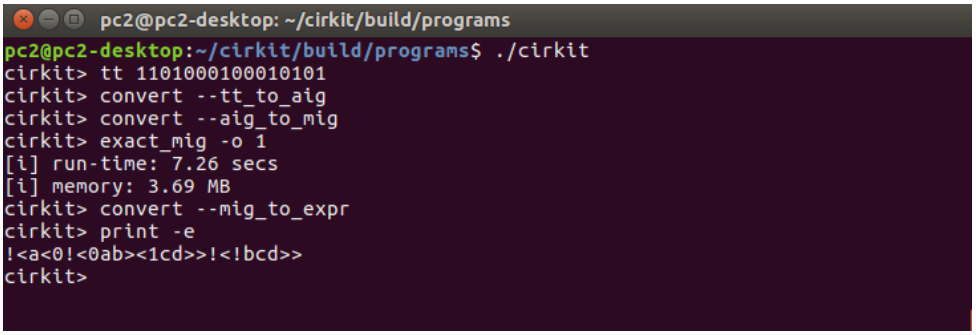
Fonte: Dados do próprio autor.

Existem duas formas de execução do *exact_mig*: quando executado com o mando *exact_mig -o 1* o *exact_mig* irá gerar uma função majoritária otimizada nos termos de portas majoritárias e no número de níveis. Quando executado com o comando *exact_mig -o 2*, o programa irá gerar uma função majoritária otimizada nos termos de níveis seguido do número de portas majoritárias.

Após a execução do *exact_mig*, deve-se executar o comando *convert --mig_to_expr* visando converter o MIG gerado em uma expressão lógica. Utiliza-se o comando *print -e* para exibir a expressão gerada.

Na Figura 59 apresenta-se a execução do programa *exact_mig* utilizando o comando *exact_mig -o 1*.

Figura 59 – Execução do *exact_mig* com o comando *exact_mig -o 1*.

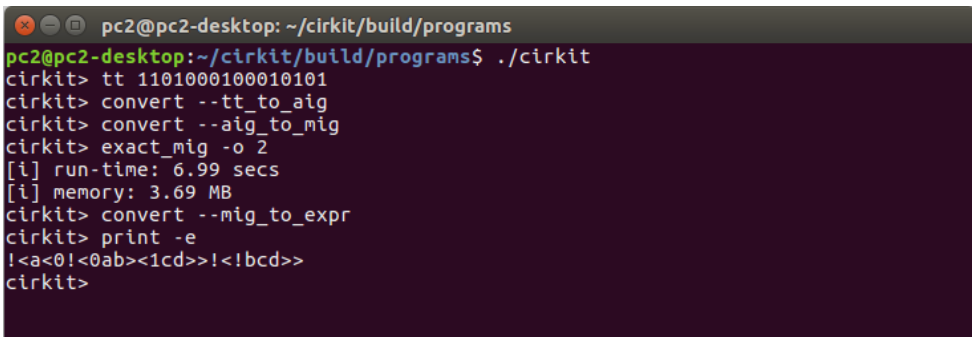


Fonte: Dados do próprio autor.

Destaca-se que no *exact_mig* a variável F representa o *bit* mais significativo, a variável A o *bit* menos significativo, o símbolo $<$ representa uma porta majoritária e o simbolo ! representa a negação.

Na Figura 60 apresenta-se a execução do programa *exact_mig* utilizando o comando *exact_mig -o 2*.

Figura 60 – Execução do *exact_mig* com o comando *exact_mig -o 2*.



Fonte: Dados do próprio autor.