MANIPULAÇÃO DE FUNÇÕES PRIMITIVAS PARA SÍNTESE DE FUNÇÕES MAJORITÁRIAS

Evandro Catelani Ferraz*, Jeferson de Lima Muniz*, Gerhard W. Dueck†, Alexandre César Rodrigues da Silva*

*Departamento de Engenharia Elétrica
Universidade Estadual Paulista
Ilha Solteira, São Paulo, Brasil

†Departamento de Ciências da Computação
Universidade de New Brunswick
Fredericton, New Brunswick, Canadá

Emails: evandro.ferraz.07@gmail.com, jeff_jl@hotmail.com, gdueck@unb.ca, acrsilva@dee.feis.unesp.br

Abstract— Due to the great technology advancement and circuits miniaturization, the study of logics that can be applied to nanotechnology has been carried out comprehensively. Among the creation of nanoeletronic circuits the reversible and majority logic stands out. This paper proposes the MPL (*Majority Primitives Listing*) algorithm, utilized for majority logic synthesis. The algorithm receives a truth table as input and returns a majority function that covers the same set of minterms. The formulation of a valid output function is made with the combination of previously optimized primitive functions. As cost criteria the algorithm searches for a function with the least amount of levels, followed by the least amount of gates and the least amount of inverters. In this paper it's also presented a comparison between the MPL and the *Exact*, currently considered the best algorithm for majority logic synthesis. Tests have shown that both algorithms return optimal solutions for all functions with 3 input variables. For functions with 4 variables, with a total of 65.536 minterms sets, and also considering inverters as a cost criteria, the MPL achieves better results for 41.498 (64%) sets, achieves equal results for 11.489 (17%) sets and inferior results for the remaining 12.549 (19%) sets.

Keywords— Algorithm, Synthesis, Majority Logic, Primitive Functions.

Resumo— Devido ao grande avanço da tecnologia e à miniaturização de circuitos, o estudo de lógicas que podem ser aplicadas à nanotecnologia vem sendo realizado de forma abrangente. Para criação de circuitos nanoeletrônicos destacam-se a lógica reversível e a lógica majoritária. Neste artigo é proposto o algoritmo MPL (*Majority Primitives Listing*), utilizado para síntese de lógica majoritária. O algoritmo recebe uma tabela verdade como entrada e retorna uma função majoritária que cobre o mesmo conjunto de mintermos. A criação de uma função de saída válida é realizada a partir da combinação entre funções primitivas previamente otimizadas. Como critério de custo busca-se a geração de funções que tenham a menor quantidade de níveis, seguida da menor quantidade de operadores e inversores. Nesse artigo também é realizada a comparação do MPL com o *Exact*, considerado o melhor algoritmo para síntese de funções majoritárias atualmente. Testes mostraram que ambos os algoritmos retornam soluções ótimas para todas as funções com 3 variáveis de entrada. Para funções com 4 variáveis, com um total de 65.536 conjuntos de mintermos, e considerando também inversores como critério de custo, o MPL consegue melhores resultados para 41.498 (64%) conjuntos, resultados iguais para 11.489 (17%) conjuntos e resultados inferiores para os 12, 549 (19%) conjuntos restantes.

Palavras-chave— Algoritmo, Síntese, Lógica Majoritária, Funções Primitivas.

1 Introdução

A lógica majoritária permite a criação de circuitos nanoeletrônicos para diversas tecnologias diferentes, o que justifica a pesquisa por algoritmos de síntese majoritária que gerem circuitos cada vez mais otimizados.

Entre os primeiros trabalhos que abordam o conceito de lógica majoritária, destacam-se os autores Richard Lindaman (Lindaman, 1960), Marius Cohn (Cohn and Lindaman, 1961) e Sheldon B. Akers (Akers, 1961). Lindaman (1960) propôs o primeiro teorema para aplicação da lógica majoritária em problemas de decisão binária, introduzindo um operador majoritário à álgebra booleana clássica. O teorema propõe uma função booleana equivalente à uma operação majoritária. Na Equação 1, apresenta-se esse teorema.

$$M(A, B, C) = A \cdot B + A \cdot C + B \cdot C \quad (1)$$

Posteriormente, Cohn e Lindaman (1961) propuseram um conjunto de axiomas que define a álgebra majoritária de forma independente à álgebra booleana clássica. O conjunto proposto foi a base para a axiomatização atual da álgebra majoritária, representada pelo símbolo Ω.

Akers (1961) propôs o primeiro algoritmo para síntese de funções majoritárias que utiliza como entrada somente uma tabela verdade, sem que seja necessário a conversão prévia de funções majoritárias em funções booleanas clássicas. Nesse algoritmo, a função de saída é construída com a aplicação de variáveis complementadas e constantes à tabela verdade de entrada, com o objetivo de formular uma função lógica composta somente por

operadores majoritários.

Zhang et al. (2004) propuseram um método que realiza o mapeamento de funções booleanas de 3 variáveis a partir de um cubo de 3 dimensões, em que cada um dos 8 vértices representa um mintermo ou maxtermo de uma função. Dessa forma, as arestas representam a interação entre mintermos e uma face do cubo representa uma das variáveis de entrada em sua forma normal ou complementada. Podem ser obtidos 13 padrões de mapeamento a partir da combinação de vértices, arestas e faces. Cada padrão possui uma fórmula diferente para transcrever a função representada em uma função majoritária.

De forma semelhante, Wang et al. (2011) propuseram um método que utiliza um cubo de 4 dimensões para o mapeamento de funções com 4 variáveis de entrada, definindo um total de 143 padrões de representação. Os 143 padrões também possuem fórmulas específicas para encontrar suas funções majoritárias equivalentes.

Na álgebra majoritária, algoritmos de simplificação baseados em funções primitivas são utilizados de forma abrangente. Funções primitivas são funções que possuem no máximo 1 operador majoritário em sua forma otimizada. Walus et al. (2004) desenvolveram um algoritmo que efetua o mapeamento de cada uma das funções primitivas e utiliza os mapas obtidos para geração de funções majoritárias mais complexas. O mapeamento de funções é feito com a utilização de Mapas de Karnaugh, um método gráfico proposto por Maurice Karnaugh em 1953, que tem como objetivo a simplificação de uma função booleana clássica a partir do mapeamento de sua tabela verdade (Karnaugh, 1953).

Mishra and Thapliyal (2017) desenvolveram um algoritmo de síntese semelhante, o algoritmo $B2M$ (*Boolean to Majority*). O $B2M$ recebe uma função booleana como entrada e gera uma função majoritária que cobre o mesmo conjunto de mintermos. Para formular a função de saída também é realizada a combinação de funções primitivas, selecionadas a partir do valor MLD (*Modified Levenshtein Distance*) entre todas as funções primitivas e a função booleana de entrada. Na Equação 2 apresenta-se o cálculo do valor MLD para X e Y, sendo X o número de mintermos cobertos pela função de entrada, Y o número de mintermos cobertos por uma função primitiva qualquer e $X \cap Y$ o número de mintermos cobertos por ambas as funções. Nota-se que o valor MLD é igual ao maior valor entre as duas subtrações.

$$MLD(X,Y) = max\{X - (X \cap Y), Y - (X \cap Y)\} \quad (2)$$

Soeken et al. (2017) desenvolveram o *Exact*, apresentado como o melhor algoritmo para síntese de funções majoritárias no estado atual da arte. O *Exact* recebe uma tabela verdade e retorna uma função que faz a cobertura de todos os mintermos dessa tabela, funcionando para até 6 variáveis de entrada. A principal característica desse algoritmo é a proposta de uma síntese exata para a geração de funções majoritárias, construída a partir de um conjunto de restrições (K) que moldam um determinado problema de acordo com as definições da álgebra booleana majoritária. A função majoritária de saída é gerada com a aplicação de K ao solucionador de satisfatibilidade $Z3$ (De Moura and Bjørner, 2008). Como critério de custo o *Exact* leva em consideração a quantidade de níveis e de operadores na função de saída, possibilitando a escolha de qual desses critérios será priorizado.

Neste trabalho é proposto o algoritmo MPL, que utiliza a listagem de funções primitivas para construção de funções majoritárias a partir de uma tabela verdade de entrada. O algoritmo verifica todas as combinações possíveis entre funções primitivas e cria uma nova tabela para armazená-las. Para cada função, armazena-se também o conjunto de mintermos que foi coberto. Caso existam duas funções que cubram o mesmo conjunto de mintermos, a função de menor custo permance na tabela e a outra é descartada. Como resultado tem-se uma tabela (M_2) que lista todos os conjuntos cobertos por funções majoritárias de 2 níveis. Considera-se como critério de custo a quantidade de níveis da função, seguida por sua quantidade de operadores e por sua quantidade de inversores.

O MPL pode ser utilizado para síntese de funções booleanas que possuem 3 ou 4 variáveis de entrada. Para 3 variáveis de entrada o algoritmo retorna uma solução ótima para todos os possíveis conjuntos de mintermos. Para 4 variáveis de entrada o algoritmo garante a solução ótima dos conjuntos cobertos pela tabela M_2 e utiliza a combinação dessas funções para a cobertura dos conjuntos restantes.

Este artigo está organizado da seguinte forma. Na seção 2 apresenta-se uma explicação sobre os elementos que constituem a álgebra majoritária, incluindo sua axiomatização e o conceito de funções primitivas majoritárias. Na seção 3 apresenta-se uma explicação detalhada sobre o algoritmo MPL. Na seção 4 apresenta-se os resultados obtidos na comparação entre o MPL e o *Exact*. Na seção 5 são apresentadas as conclusões do artigo.

2 Álgebra Booleana Majoritária

A álgebra booleana majoritária é composta pelo conjunto $\{\mathbb{B}, \neg, M\}$. Os elementos $\mathbb{B}$ e $\neg$, assim como na álgebra clássica, representam os valores binários $\{0, 1\}$ e o operador de inversão, respectivamente. O elemento restante do conjunto, M, representa o operador majoritário (Chattopadhyay et al., 2016).

Uma função majoritária retornará como saída o valor binário que for maioria entre suas variáveis de entrada. Dessa forma, um operador M que tenha como entrada um total de 3 variáveis retornará como saída um valor verdadeiro apenas se duas ou mais entradas forem verdadeiras. A tabela verdade apresentada na Tabela 1 exemplifica uma operação majoritária para as variáveis X, Y e Z.

Tabela 1: Exemplo de uma operação majoritária.

X	Y	Z	$M(X,Y,Z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A partir de uma operação majoritária também é possível a simulação de funções *AND* e *OR*, realizada com a fixação de uma das variáveis de entrada em um valor binário constante.

Como exemplo, considera-se a função $M(A,B,C)$. Fixando o valor de A em 0 tem-se uma função *AND* entre B e C. Fixando o valor de A em 1 tem-se uma função *OR* entre B e C. Na Tabela 2 é apresentada uma tabela verdade que prova essa relação.

Tabela 2: Simulação de funções *AND* e *OR*.

B	C	$B \cdot C$	$M(0,B,C)$	$B+C$	$M(1,B,C)$
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	1	1	1

2.1 Axiomatização de Funções Majoritárias (Ω)

O conjunto de axiomas que define a álgebra majoritária é chamado de Ω e pode ser dividido em axiomas de Comutatividade, Associatividade, Distribuição, Propagação de Inversão e Maioria (Amaru et al., 2016). O axioma de Comutatividade (Ω.C), representado na Equação 3, determina que a ordem das entradas não influencia no valor de saída da função.

$$M(X,Y,Z) = M(X,Z,Y) = M(Z,Y,X) \quad (3)$$

O axioma de Associatividade (Ω.A) define que a troca de variáveis entre duas funções é possível, desde que elas estejam em níveis sequênciais e possuam uma variável em comum. Na Equação 4, tem-se um exemplo da aplicação desse axioma.

$$M(X,U,M(Y,U,Z)) = M(Z,U,M(Y,U,X)) \quad (4)$$

Nota-se que a variável compartilhada entre os 2 níveis é a variável U. Dessa forma, é possível trocar a variável restante no nível superior por uma das variáveis do nível subsequente. No exemplo apresentado foi efetuada a troca entre as variáveis X e Z. O axioma de Distribuição (Ω.D) determina que é possível a distribuição de um conjunto de variáveis para funções que estejam em níveis sequenciais. Na Equação 5 apresenta-se um exemplo desse teorema, em que o conjunto $\{X,Y\}$ foi distribuído para as variáveis no nível seguinte.

$$\text{M(X,Y,M(U,V,Z))} = \text{M(M(X,Y,U), M(X,Y,V), Z)} \quad (5)$$

O axioma de Propagação de Inversão (Ω.I), representado na Equação 6, determina que o complemento de uma função pode ser propagado para todas suas entradas. Da mesma forma, uma função que possui todas suas entradas complementadas pode substituir seus inversores por um único inversor aplicado ao seu valor de saída.

$$\overline{M}(X,Y,Z) = M(\overline{X},\overline{Y},\overline{Z}) \quad (6)$$

O axioma de Maioria (Ω.M) pode ser dividido em duas equações. A Equação 7 define que a saída de uma função majoritária é igual ao valor que for maioria entre suas entradas. A Equação 8 define que o valor de saída será igual à variável de desempate em funções que possuem uma quantidade igual de valores verdadeiros e falsos.

$$M(X,X,Y) = X \quad (7)$$

$$M(X,\overline{X},Y) = Y \quad (8)$$

2.2 Funções Majoritárias Primitivas

Funções primitivas são funções formadas por um único ou por nenhum operador. Na álgebra majoritária, funções primitivas podem ser utilizadas como base para a construção de funções mais complexas. Todas as funções majoritárias primitivas podem ser obtidas a partir dos conjuntos V, G e T, sendo cada conjunto correspondente à funções com características específicas. A quantidade total de funções majoritárias primitivas é obtida pela soma das funções em V, G e T (Wang et al., 2015).

O conjunto V representa todas as funções formadas por uma única variável de entrada, em sua forma complementada ou não, ou por um único valor constante. Na Equação 9, tem-se o cálculo da quantidade de funções que a listagem de V deve gerar.

$$V = 2 + (n \cdot 2) \quad (9)$$

Na Tabela 3, pode-se observar a listagem do conjunto V para 3 variáveis de entrada ($n = 3$).

Tabela 3: Listagem do conjunto V para $n = 3$.

Função Clássica	Função Majoritária
0	0
1	1
A	A
B	B
C	C
$\overline{A}$	$\overline{A}$
$\overline{B}$	$\overline{B}$
$\overline{C}$	$\overline{C}$

Nota-se que as funções clássicas e suas correspondentes majoritárias são iguais porque o conjunto V é composto apenas por funções sem operadores.

O conjunto G é formado por funções que possuem um único operador *AND* ou *OR*. Na Equação 10, tem-se o cálculo da quantidade de funções geradas pela listagem de G. As variáveis A e O representam o número de combinações possíveis entre as variáveis de entrada, para operações *AND* e *OR* respectivamente, considerando duas variáveis por combinação. Para $n = 3$, tem-se $A = \{A \cdot B, A \cdot C, B \cdot C\}$ e $O = \{A + B, A + C, B + C\}$. Cada combinação possui 4 variações de inversores, a combinação $A + B$ por exemplo, possui as variações $\{A + B, \overline{A} + B, A + \overline{B}, \overline{A} + \overline{B}\}$.

$$G = (A \cdot 4) + (O \cdot 4) \tag{10}$$

Na Tabela 4, tem-se a listagem do conjunto G para $n = 3$.

O conjunto T representa funções que possuem apenas um operador majoritário e que não possuem nenhum valor constante como entrada. Na Equação 11, apresenta-se o cálculo da quantidade de funções geradas pela listagem de T. A variável t representa a quantidade de combinações possíveis entre as variáveis de entrada, considerando 3 entradas por combinação. Nota-se que cada combinação possui 8 variações de polarização e, para $n = 3$, existe apenas uma combinação possível.

$$T = t \cdot 8 \tag{11}$$

Na Tabela 5, tem-se a listagem do conjunto T para $n = 3$.

3 Algoritmo MPL

Nesta seção apresenta-se o algoritmo MPL, proposto neste artigo. O MPL recebe como entrada uma tabela verdade f e retorna como saída uma função majoritária que cobre os mintermos de f. Para gerar uma função de saída válida utiliza-se a expressão $M(X_1, X_2, X_3)$. Cada variável X_c, sendo $1 \leq c \leq 3$, representa uma função majoritária primitiva ou uma função majoritária de 2 níveis. Por se tratar de um método exaustivo, o algoritmo funciona apenas para funções com 3 ou 4 variáveis de entrada.

Tabela 4: Listagem do conjunto G para $n = 3$.

Função Clássica	Função Majoritária
$A \cdot B$	$M(A, B, 0)$
$\overline{A} \cdot B$	$M(\overline{A}, B, 0)$
$A \cdot \overline{B}$	$M(A, \overline{B}, 0)$
$\overline{A} \cdot \overline{B}$	$\overline{M}(A, B, 1)$
$A \cdot C$	$M(A, 0, C)$
$\overline{A} \cdot C$	$M(\overline{A}, 0, C)$
$A \cdot \overline{C}$	$M(A, 0, \overline{C})$
$\overline{A} \cdot \overline{C}$	$\overline{M}(A, 1, C)$
$B \cdot C$	$M(0, B, C)$
$\overline{B} \cdot C$	$M(0, \overline{B}, C)$
$B \cdot \overline{C}$	$M(0, B, \overline{C})$
$\overline{B} \cdot \overline{C}$	$\overline{M}(1, B, C)$
$A + B$	$M(A, B, 1)$
$\overline{A} + B$	$M(\overline{A}, B, 1)$
$A + \overline{B}$	$M(A, \overline{B}, 1)$
$\overline{A} + \overline{B}$	$\overline{M}(A, B, 0)$
$A + C$	$M(A, 1, C)$
$\overline{A} + C$	$M(\overline{A}, 1, C)$
$A + \overline{C}$	$M(A, 1, \overline{C})$
$\overline{A} + \overline{C}$	$\overline{M}(A, 0, C)$
$B + C$	$M(1, B, C)$
$\overline{B} + C$	$M(1, \overline{B}, C)$
$B + \overline{C}$	$M(1, B, \overline{C})$
$\overline{B} + \overline{C}$	$\overline{M}(0, B, C)$

Tabela 5: Listagem do conjunto T para $n = 3$.

Função Clássica	Função Majoritária
$AB + AC + BC$	$M(A, B, C)$
$\overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{C} + \overline{B} \cdot \overline{C}$	$\overline{M}(A, B, C)$
$\overline{A} \cdot B + \overline{A} \cdot C + B \cdot C$	$M(\overline{A}, B, C)$
$A \cdot \overline{B} + A \cdot \overline{C} + \overline{B} \cdot \overline{C}$	$M(A, \overline{B}, \overline{C})$
$A \cdot \overline{B} + A \cdot C + \overline{B} \cdot C$	$M(A, \overline{B}, C)$
$\overline{A} \cdot B + \overline{A} \cdot \overline{C} + B \cdot \overline{C}$	$M(\overline{A}, B, \overline{C})$
$A \cdot B + A \cdot \overline{C} + B \cdot \overline{C}$	$M(A, B, \overline{C})$
$\overline{A} \cdot \overline{B} + \overline{A} \cdot C + \overline{B} \cdot C$	$M(\overline{A}, \overline{B}, C)$

3.1 Formação de Tabelas

O primeiro passo do MPL é a fase de formação de tabelas, em que são construídas a tabela de funções primitivas e a tabela M_2. O algoritmo recebe como entrada uma tabela verdade f, indentifica a quantidade de entradas n, e gera a tabela de funções primitivas a partir dos conjuntos V, G e T. Também são registrados os mintermos cobertos por cada função primitiva, sendo todas as funções primitivas a solução ótima para seu respectivo conjunto de mintermos.

A tabela M_2 é formada pela aplicação de todas as combinações possíveis de funções primitivas à formula $M(X_1, X_2, X_3)$. Para cada função gerada, também registra-se o conjunto de mintermos coberto. Caso existam duas funções que cu-

bram os mesmos mintermos, a função de menor custo é mantida e a outra é descartada. Dessa forma, a tabela M_2 lista todos os conjuntos de mintermos que devem ser cobertos por uma função majoritária de 2 níveis. Como exemplo de uma função da tabela M_2, tem-se $M(X_1, X_2, X_3) = M(A, M(A, \overline{B}, 0), \overline{M}(A, B, C))$.

Por serem obtidas com um método exaustivo, as funções da tabela M_2, assim como as funções primitivas, serão sempre a solução ótima para seus respectivos conjuntos de mintermos.

O total de conjuntos, representado pela variável S, é calculado por 2^m, sendo m a quantidade de mintermos e maxtermos em f. Nota-se que $m = 2^n$.

Para $n = 3$, tem-se $S = 256$. A tabela de primitivos cobre 40 desses conjuntos. Os 216 conjuntos restantes são cobertos pelas funções da tabela M_2. Sendo assim, S pode ser coberto completamente por funções majoritárias de 2 ou menos níveis, o que torna a fase de formação de tabelas o suficiente para a obtenção de todas as soluções ótimas para $n = 3$.

Para $n = 4$, tem-se $S = 65.536$, sendo 90 desses conjuntos referentes à funções com um único ou nenhum operador majoritário, que são cobertas pela tabela de funções primitivas. Na formação da tabela M_2, apenas 10.260 conjuntos podem ser cobertos por funções majoritárias de 2 níveis. Para os 55.186 conjuntos restantes, 55.184 podem ser cobertos por funções majoritárias de 3 níveis. Apenas 2 conjuntos precisam de uma função majoritária de 4 níveis. A cobertura de cada um desses casos é feita com a aplicação de uma síntese específica.

3.2 Síntese para Funções Majoritárias de 3 Níveis

Esta seção apresenta a síntese utilizada pelo algoritmo MPL na construção de funções majoritárias de 3 níveis. O objetivo dessa síntese é montar uma função majoritária $M(X_1, X_2, X_3)$ a partir da combinação de funções primitivas ou da tabela M_2, que cubra o conjunto de mintermos da tabela verdade de entrada f.

O cálculo do custo considerado pelo algoritmo é apresentado na Equação 12. As variáveis r e I representam a quantidade de operadores e de inversores da função, respectivamente. O valor 10 funciona como peso para respeitar o critério de custo do MPL, que leva em consideração primeiramente a minimização de níveis, seguida da minimização de operadores, seguida da minimização de inversores.

$$Custo = (10 \cdot r) + I \tag{12}$$

Nota-se que a quantidade de níveis não é levada em consideração no cálculo do custo porque no mínimo uma função sempre será retirada da tabela M_2, ou seja, uma das funções X_c sempre será uma função majoritária de 2 níveis. Uma função de 2 níveis aplicada à um operador $M(X_1, X_2, X_3)$ sempre vai gerar uma função majoritária de 3 níveis. A síntese só é aplicada se o conjunto de mintermos não puder ser coberto pelas funções (de 1 ou 2 níveis) encontradas na fase de formação de tabelas.

Para aplicação da síntese, tem-se as seguintes etapas:

1. Ordena-se todas as funções presentes na tabela de primitivos e na tabela M_2 de acordo com seu custo;

2. Seleciona-se como X_1, da tabela M_2, a função majoritária de menor custo, que cubra pelo menos um mintermo de f e que ainda não tenha sido escolhida como X_1. Após a seleção de X_1, a função escolhida é marcada como já utilizada pelo algoritmo. Nota-se que X_1 sempre é escolhida da tabela MPL porque no mínimo uma função X_c deve ser de 2 níveis e, como funções da tabela M_2 possuem um custo maior que as funções primitivas, prioriza-se a escolha de funções de 2 níveis com o menor custo possível;

3. Cria-se um vetor contador, representado pela variável v, que será utilizado para parametrizar os critérios de escolha para as funções seguintes. Um vetor contador possui 2^n posições i, sendo $0 \leq i \leq 2^n$, e cada posição i representa uma posição de f. Assim que uma função X_1 é escolhida, o vetor contador é atualizado de acordo com os mintermos cobertos por essa função. Para as posições em que i corresponde à um mintermo de f coberto também por X_1, adiciona-se o valor 1. Para posições correspondentes à mintermos cobertos somente pela função X_1, subtrai-se o valor 1. Nota-se que v possui inicialmente o valor 0 em todas as suas posições.

   Após a atualização de v, criam-se outros 3 vetores: v_1, v_0 e v_{-1}, que guardam as posições do vetor contador que possuem valor 1, 0 e -1, respectivamente;

4. Seleciona-se como X_2, primeiramente da tabela de funções primitivas, a função majoritária de menor custo que possua os seguintes requisitos:

   - Cubra todos os mintermos presentes em v_0;
   - Não cubra nenhum dos mintermos presentes em v_{-1};
   - Não esteja marcada como já utilizada pelo algoritmo.

Caso não exista uma função primitiva válida, a busca por X_2 é feita na tabela M_2. Após a seleção de X_2, a função escolhida é marcada como já utilizada e executa-se o Passo 5. Caso não exista uma função majoritária com esses requisitos em nenhuma das tabelas é necessário que seja escolhida uma nova função X_1 e retorna-se ao Passo 2;

5. Atualiza-se v a partir de X_2, formando assim novos vetores v_1, v_0 e v_{-1};

6. Seleciona-se como X_3, também considerando primeiramente a tabela de funções primitivas, a função majoritária de menor custo que possua os seguintes requisitos:

   - Cubra todos os mintermos presentes em v_1;
   - Não cubra nenhum dos mintermos presentes em v_{-1}.

   Se uma função primitiva válida não puder ser encontrada, busca-se X_3 na tabela M_2. Após a seleção de X_3, tem-se a formação completa da função $M(X_1, X_2, X_3)$ e executa-se o Passo 7. Caso não exista uma função majoritária com esses requisitos em nenhuma das tabelas, é necessário que seja escolhida uma nova função X_2 e retorna-se ao Passo 4;

7. Aplica-se $\Omega.I$ na função $M(X_1, X_2, X_3)$ encontrada, gerando a função $\overline{M}(\overline{X_1}, \overline{X_2}, \overline{X_3})$. A quantidade de inversores em ambas as funções é comparada e o algoritmo retorna como saída a função majoritária de menor custo.

De todos os 55.184 conjuntos de mintermos que devem ser cobertos por funções de 3 níveis, 50.016 são cobertos por funções em que 2 elementos de X_c são funções primitivas.

Entre os 5.168 conjuntos restantes, 5.056 são cobertos por funções em que apenas 1 elemento de X_c é uma função primitiva. Os 112 conjuntos restantes só podem ser cobertos por funções em que todos os elementos de X_c são funções de 2 níveis da tabela M_2.

3.3 Síntese para Funções Majoritárias de 4 Níveis

Os 2 conjuntos de mintermos que podem ser cobertos apenas por uma função majoritária de 4 níveis são os conjuntos correspondentes à função booleana clássica em que todas as variáveis de entrada estão conectadas à um operador *XOR* e à essa função complementada. Na Tabela 6, é possível observar as duas funções clássicas e suas respectivas tabelas verdade.

O objetivo dessa síntese é formular $M(B, X_2, X_3)$. A atribuição $X_1 = B$ é justificada porque a tabela verdade de B pode ser utilizada para definir o conjunto de mintermos que deve ser coberto pelas funções X_2 e X_3. A tabela verdade de B é [0000111100001111].

Tabela 6: Conjuntos de mintermos cobertos apenas por funções majoritárias de 4 níveis.

Função Clássica	**Tabela Verdade**
$A \oplus B \oplus C \oplus D$	[0110100110010110]
$\overline{A \oplus B \oplus C \oplus D}$	[1001011001101001]

Como é definido pelo axioma $\Omega.M$, cada mintermo deve ser coberto por no mínimo duas das funções X_c. Tendo $\Omega.M$ e a tabela verdade de B como base, define-se a estrutura apresentada na Tabela 7. Nota-se que a estrutura divide cada tabela verdade em 4 subvetores, representados pela variável Q_j, em que $1 \leq j \leq 4$.

Tabela 7: Estrutura para formação da tabela verdade de X_2 e X_3.

	Q_1	Q_2	Q_3	Q_4
B	0000	1111	0000	1111
X_2	1111	$----$	$----$	0000
X_3	$----$	0000	1111	$----$

Pode-se observar que, a partir de B e de $\Omega.M$, foi possível definir 2 dos 4 subvetores da tabela verdade de X_2 e X_3. A outra metade da tabela é definida pelos subvetores da tabela verdade de entrada f.

Como exemplo considera-se a função $A \oplus B \oplus C \oplus D$, em que f = [0110100110010110] e seus subvetores são Q_1 = [0110], Q_2 = [1001], Q_3 = [1001] e Q_4 = [0110].

A tabela verdade de X_2, representada pela variável X_2f, é encontrada a partir da combinação de seus quadrantes predefinidos, Q_1 e Q_4, e dos quadrantes Q_2 e Q_3 de f. Sendo assim, X_2f = [1111100110010000].

De forma semelhante, a tabela verdade de X_3, representada por X_3f, pode ser obtida com a combinação de seus quadrantes predefinidos, Q_2 e Q_3, e dos quadrantes Q_1 e Q_4 de f, gerando X_3f = [0110000011110110].

A seleção de X_2 e X_3 é feita com a síntese para funções de 3 níveis aplicada à sua respectiva tabela verdade (X_2f e X_3f).

4 Testes

Nesta seção apresenta-se os resultados obtidos com a comparação entre os algoritmos MPL e *Exact*. Os testes foram feitos a partir de um arquivo *.bash* que executou ambos os algoritmos para todos os 65.536 conjuntos de mintermos possíveis. O critério de custo utilizado nessa comparação é o mesmo utilizado pelo MPL, apresentado na Equação 12. Na Tabela 8 tem-se em

cada coluna a quantidade de conjuntos pertencentes à um grupo S_i específico. Cada S_i, sendo $0 \leq i \leq 2^n$, representa um total de conjuntos que possui uma quantidade específica de mintermos. S_4, por exemplo, representa todos os conjuntos que possuem 4 mintermos. Para cada S_i a tabela mostra a quantidade de conjuntos em que o MPL gerou resultados com custo menor, maior, e igual ao *Exact*.

Tabela 8: Comparação entre MPL e *Exact*.

i	Si	$MPL <$ ***Exact***	$MPL >$ ***Exact***	$MPL =$ ***Exact***
0	1	0	0	1
1	16	4	1	11
2	120	40	7	73
3	560	236	57	267
4	1.820	987	354	479
5	4.368	2.735	393	1.240
6	8.008	4.523	1.719	1.766
7	11.440	7.930	1.859	1.651
8	12.870	7.034	3.703	2.133
9	11.440	8.100	1.925	1.415
10	8.008	4.998	1.673	1.337
11	4.368	3.224	423	721
12	1.820	1.206	366	248
13	560	375	60	125
14	120	94	8	18
15	16	12	1	3
16	1	0	0	1
TOTAL	**65.536**	**41.498**	**12.549**	**11.489**

Dessa forma, o algoritmo MPL gera uma melhoria na cobertura de 41.498 (64 %) conjuntos de mintermos, gera resultados inferiores para 12.549 (19 %) conjuntos e resultados iguais para os outros 11.489 (17 %). Nota-se que o MPL consegue gerar melhores resultados porque o *Exact* não busca a otimização de inversores, em sua síntese exata considera-se apenas a quantidade de níveis e operadores como critério de custo. Nessa comparação, as funções do *Exact* foram geradas com a priorização da quantidade de níveis seguida da quantidade de operadores, diferindo do critério de custo do MPL apenas pela quantidade de inversores como terceiro parâmetro. Resultados sobre a comparação de funções com 3 variáveis de entrada não são apresentados porque ambos os algoritmos retornam soluções ótimas para todos os 256 conjuntos.

5 Conclusões

Neste artigo foi apresentado o algoritmo MPL, que tem como objetivo a geração de funções majoritárias a partir de uma tabela verdade de entrada. Também foi apresentado um estudo sobre os principais conceitos da álgebra booleana majoritária e funções primitivas.

Utilizando o critério de custo proposto, o MPL apresentou, em sua maioria, resultados superiores ou equivalentes ao *Exact*. Para funções com 3 variáveis de entrada, ambos os algoritmos retornam soluções ótimas para todos os 256 conjuntos de mintermos. Para funções com 4 variáveis, de um total de 65.536 conjuntos, o MPL conseguiu melhores resultados para 41.498 (64%), resultados iguais para 11.489 (17%) e resultados inferiores para os 12.549 (19%) conjuntos restantes.

Agradecimentos

Os autores agradecem à CAPES, CNPq nº309193/2015-0, e ao Programa de Pós Graduação em Engenharia Elétrica da Universidade Estadual Paulista (UNESP), Ilha Solteira.

Referências

Akers, S. B. (1961). A truth table method for the synthesis of combinational logic, *IRE Transactions on Electronic Computers* (4): 604–615.

Amaru, L., Gaillardon, P.-E., Chattopadhyay, A. and De Micheli, G. (2016). A sound and complete axiomatization of majority-n logic, *IEEE Transactions on Computers* **65**(9): 2889–2895.

Chattopadhyay, A., Amarú, L., Soeken, M., Gaillardon, P.-E. and De Micheli, G. (2016). Notes on majority boolean algebra, *Multiple-Valued Logic (ISMVL), 2016 IEEE 46th International Symposium on*, IEEE, pp. 50–55.

Cohn, M. and Lindaman, R. (1961). Axiomatic majority-decision logic, *IRE Transactions on Electronic Computers* (1): 17–21.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver, *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 337–340.

Karnaugh, M. (1953). The map method for synthesis of combinational logic circuits, *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics* **72**(5): 593–599.

Lindaman, R. (1960). A theorem for deriving majority-logic networks within an augmented boolean algebra, *IRE Transactions on Electronic Computers* (3): 338–342.

Mishra, V. K. and Thapliyal, H. (2017). Heuristic based majority/minority logic synthesis for emerging technologies, *VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID), 2017 30th International Conference on*, IEEE, pp. 295–300.

Soeken, M., Amaru, L. G., Gaillardon, P.-E. and De Micheli, G. (2017). Exact synthesis of majority-inverter graphs and its applications, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* .

Walus, K., Schulhof, G., Jullien, G., Zhang, R. and Wang, W. (2004). Circuit design based on majority gates for applications with quantum-dot cellular automata, *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, Vol. 2, IEEE, pp. 1354–1357.

Wang, P., Niamat, M. and Vemuru, S. (2011). Minimal majority gate mapping of 4-variable functions for quantum cellular automata, *Nanotechnology (IEEE-NANO), 2011 11th IEEE Conference on*, IEEE, pp. 1307–1312.

Wang, P., Niamat, M. Y., Vemuru, S. R., Alam, M. and Killian, T. (2015). Synthesis of majority/minority logic networks, *IEEE Transactions on Nanotechnology* **14**(3): 473–483.

Zhang, R., Walus, K., Wang, W. and Jullien, G. A. (2004). A method of majority logic reduction for quantum cellular automata, *IEEE Transactions on Nanotechnology* **3**(4): 443–450.