MK-4: PROGRAMA PARA SÍNTESE DE FUNÇÕES MAJORITÁRIAS COM ATÉ QUATRO VARIÁVEIS DE ENTRADA.

Jeferson de Lima Muniz*, Evandro Catelani Ferraz*, 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: jeff_jl@hotmail.com, evandro.ferraz.07@gmail.com, gdueck@unb.ca, acrsilva@dee.feis.unesp.br

Abstract— With the evolution of technology and miniaturization the ICs (Integrated Circuits) with CMOS (Complementary Metal-Oxide Semicondutor) technology have become smaller and more efficient. To minimize further, new technologies are presented, such as the QCA technology, which together with the majority logic can reduce the size of a circuit. In this work the MK-4 program was implemented, with the purpose of performing the synthesis of the majority functions with up to four variables, using the Karnaugh map. In order to evaluate the developed program in relation to the minimized function cost, the obtained results were compared in terms of number of levels, number of majority gates, number of inputs and number of inverters, with the results obtained by the program *Exact*. All of the 65.536 4 variables functions were generated and MK-4 was able to generate 43.57% functions of lower cost, 13.97% equivalent cost functions and 42.46% higher cost functions when compared to the functions generated by *Exact*.
Keywords— Majority Logic, Synthesis of majority circuits, QCA.

Resumo— Com a evolução da tecnologia e miniaturização os CIs (Circuitos Integrados) com tecnologia CMOS (Complementary Metal-Oxide Semicondutor) têm se tornado cada vez menores e mais eficientes. Para minimizar ainda mais os circuitos digitais, novas tecnologias são apresentadas, como por exemplo a tecnologia QCA, que em conjunto com a lógica majoritária consegue diminuir o tamanho de um circuito. Neste trabalho implementou-se o programa denominado MK-4 que tem como proposta realizar a síntese de funções majoritárias com até quatro variáveis, utilizando o mapa de Karnaugh. A fim de avaliar o programa desenvolvido em relação ao custo da função minimizada, os resultados obtidos foram comparados em termos de número de níveis, número de portas majoritárias, número de entradas e número de inversores, com os resultados obtidos pelo programa *Exact*. Foram geradas todas as 65.536 funções de 4 variáveis e o programa MK-4 foi capaz de gerar 43, 57% funções de menor custo, 13, 97% funções de custo equivalente e 42, 46% funções de maior custo quando comparadas com as funções geradas pelo *Exact*.

Palavras-chave— Lógica Majoritária, Síntese de circuitos majoritários, QCA.

1 Introdução

A procura por tecnologias e algoritmos de síntese de funções booleanas para obter funções minímas é necessária para diminuir o custo de um circuito. Circuitos baseados em Autômatos Celulares com pontos Quânticos (QCA) têm a proposta de minimizar ainda mais os circuitos integrados quando comparados aos os circuitos da tecnologia CMOS (Kong et al., 2010).

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 unidade básica. Em contrapartida, existem tecnologias, como por exemplo 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).

Alguns dos métodos de síntese de funções booleanas mais conhecidos são os métodos da simplificação algébrica e do mapa-K (Karnaugh, 1953). Na lógica majoritária, diversos algoritmos de síntese focados na porta majoritária foram desenvolvidos como por exemplo os trabalhos de Zhang et al. (2004), Walus et al. (2004), Wang et al. (2013), Amarú et al. (2015) e Soeken et al. (2017). A implementação de novos métodos foi necessária pelo fato de que os métodos utilizados na síntese da lógica clássica não tem como foco a porta majoritária.

Em Zhang et al. (2004) foi proposto o método das treze funções padrões. O método faz uso de um cubo de 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 da função e as arestas representam as interações entre os mintermos da função. Na Figura 1 apresenta-se um cubo de três dimensões onde destacam-se um vértice, uma aresta e

uma face.

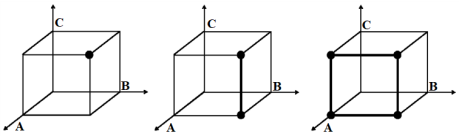


Figura 1: Vértice, aresta e face em um cubo de três dimensões.

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 encontrada pode não ser a mínima (Wang et al., 2013).

O método do mapa-K de três variáveis, desenvolvido em Walus et al. (2004), tem como base quarenta funções primitivas, em que a combinação de até três delas implementa qualquer função booleana com no máximo três variáveis de entrada. Entende-se como funções primitivas as funções que podem ser implementadas utilizando uma única ou nenhuma porta majoritária. Dada a tabela verdade de uma função F, o método combina três mapas-K com diferentes funções de forma que o mapa resultante cubra todos os mintermos de F pelo menos duas vezes.

No trabalho de Zhang et al. (2007) foi desenvolvido o software *Majority Logic Synthesizer* (MALS). Foi o primeiro software a minimizar funções com mais de três variáveis de entrada. O algoritmo do MALS começa reduzindo o número de variáveis de uma função de forma que a função resultante tenha apenas nós com 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 decomposição, realiza-se a minimização algébrica na função resultante e 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, aplica-se o método de minimização baseado no mapa-K para gerar uma função majoritária e o algoritmo é encerrado.

Em Wang et al. (2013) foi proposto o *Majority expressions Look-up Table* (MLUT). Trata-se de um software que realiza 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 65.536, então aplica-se o algoritmo do mapa-K de quatro variáveis, 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 et al. (2015) foi proposto um *framework* de síntese que faz o uso da inserção de erros propositais em nós de uma função majoritária. 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 síntese voltada para reduzir o número de níveis e outra voltada a reduzir o tamanho da função, ambas podendo gerar funções com resultados equivalentes mas com custos distintos.

Em Soeken et al. (2017) é proposta uma metodologia que tem o objetivo de realizar uma síntese exata de funções majoritárias com até seis variáveis. O algoritmo denominado *Exact* tem como base uma forma de representação de funções majoritárias conhecida como MIG (*Majority-Inverter Graph*). Um MIG gerado é transcrito como entrada em solucionador de módulo e teoria denominado Z3, apresentado em De Moura and Bjørner (2008), que tem o propósito de validar se a função gerada é uma solução válida (SAT) ou inválida (UNSAT). Ressalta-se que o *Exact* tem como critérios de custo o número de níveis e o número de portas majoritárias de uma função.

Neste trabalho implementou-se o programa denominado MK-4 que tem como proposta realizar a síntese de funções majoritárias com até quatro variáveis, utilizando o mapa de Karnaugh. Diferentemente do programa *Exact* que leva em consideração o número de níveis e o número de portas majoritárias, o MK-4 leva em consideração o número de níveis, o número de portas majoritárias, o número entradas e o número de inversores na contagem do custo da função gerada. O programa MK-4 também difere do programa MLUT visto que não é gerada uma tabela de busca para todas as 65.536 funções de quatro variáveis. O MK-4 possui como entrada uma tabela verdade e gera uma função minimizada que tenha como saída verdadeira os termos recebidos como entrada.

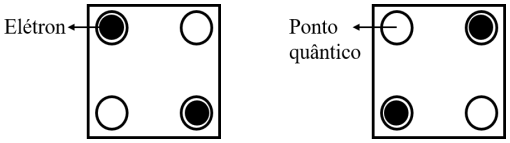
Este artigo está organizado de forma que na seção 2 são introduzidos os principais conceitos da tecnologia QCA, na seção 3 são apresentados os conceitos da lógica majoritária, na seção 4 é apresentado o método de síntese baseado no mapa-K e na seção 5 é apresentado o programa MK-4 e a comparação dos resultados obtidos pelo MK-4 e pelo programa *Exact*.

2 Tecnologia

Os QCA são células quânticas constituídas de quatro pontos quânticos e dois elétrons livres. Devido a lei de Coulomb, os elétrons ocupam as diagonais

opostas das células. A lei de Coulomb, proposta em 1785 por Charles Coulomb, descreve a força eletrostática entre partículas eletricamente carregadas, definindo que uma força $\overrightarrow{F}$, resultante da interação entre dois pontos carregados (q1 e q2), promove atração entre os dois pontos caso ambos possuam cargas diferentes e promove repulsão caso possuam cargas iguais (Sousa and de Oliveira, 2015).

Pela ação da lei de Coulomb, também chamada de interação Coulombiana, os elétrons podem se mover de um extremo a outro de uma célula QCA, sempre na diagonal e gerando duas possíveis configurações denotadas como polarização celular $P = +1$ e $P = -1$. Utiliza-se a polarização celular para definir o valor lógico de uma célula. Quando $P = +1$ a célula possui o valor lógico 1. Quando $P = -1$ a célula possui o valor lógico 0 (Chung et al., 2017). Representa-se na Figura 2 duas células QCA polarizadas.



(a) $P = -1$ (b) $P = +1$

Figura 2: Exemplo de células QCA com diferentes polarizações.

Dispositivos QCA utilizam a lei de Coulomb para transmitir informação lógica por meio de um fio QCA. A polarização de uma célula "A" altera o estado de uma célula "B" colocada próxima a ela, assim propagando o valor lógico de "A" (Lent and Tougaw, 1997).

Na Figura 3 apresenta-se um fio QCA transmitindo um sinal lógico 1, de uma ponto X a um ponto Y por meio da interação coulombiana.

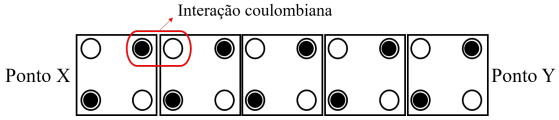


Figura 3: Fio QCA propagando o sinal de um ponto X até um ponto Y.

Um inversor QCA é feito arranjando as células como apresentado na Figura 4, a lei de Coulomb inverte a polarização das células, invertendo o sinal de entrada.

Uma porta majoritária é gerada ao arranjar cinco células QCA de forma que a interação Coulombiana entre elas faça com que a saída tenha valor lógico verdadeiro se, e somente se, pelo menos duas entradas possuírem valor lógico verdadeiro. Na Figura 5 apresenta-se uma porta majoritária de três entradas.

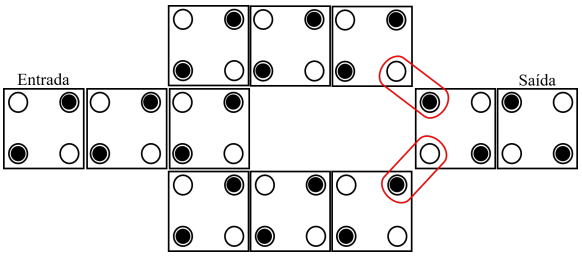


Figura 4: Inversor QCA.

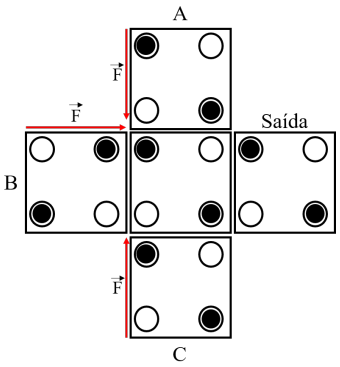


Figura 5: Porta majoritária de três entradas feita em QCA.

Conceitos sobre a lógica majoritária são apresentados na próxima seção.

3 Lógica Majoritária

Quando a álgebra boolena é definida utilizando o conjunto ($\mathbb{B}$,M_3,$\overline{\ }$,0,1), sujeita ao conjunto de axiomas Ω (Maioria, Comutatividade, Associatividade, Distributiva e Propagação de inversão), é chamada de lógica majoritária. O M_3 representa o operador majoritário entre três variáveis. Desse modo, 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 $F(A, B, C) = \mathrm{AB} + \mathrm{AC} + \mathrm{BC}$. Utilizando a lógica booleana clássica, é necessário aplicar três portas E e duas portas lógicas OU para descrever esse circuito. Em contrapartida, quando se aplica a lógica majoritária, uma única porta majoritária é o suficiente para implementar F e é escrita como M(A,B,C), em que lê-se maioria entre A, B e C.

Assim como na álgebra booleana, na lógica majoritária há axiomas que auxiliam na simplificação e na minimização de funções majoritárias, sendo os axiomas da comutatividade, da maioria, da associatividade, da distribuição, e do complemento (Amaru et al., 2016).

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

O axioma da comutatividade (Ω.C) define que como 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 $M(A, B, C) = M(A, C, B) = M(C, A, B)$.

De acordo com o axioma da associatividade (Ω.A), é possível trocar uma variável entre os níveis de uma função. Um novo nível é identificado quando tem-se um nó com uma função majoritária conectado a outro nó. 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 do segundo nível, gerando novas funções equivalentes. Assim, $M(A, B, M(A, C, D)) = M(A, C, M(A, B, D)) = M(A, D, M(A, C, B))$.

O axioma da distribuição (Ω.D) define que pode-se rearranjar variáveis que se repetem em níveis para evitar redundâncias. Tem-se como exemplo a função $M(M(A, B, C), M(A, B, D), E)$. Como A e B se repetem, é possível colocá-las em evidência e gerar a função $M(A, B, M(C, D, E))$.

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

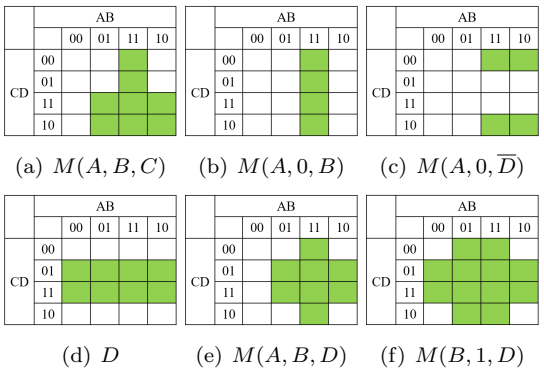
A porta majoritária de três entradas pode se comportar como uma porta E ou como uma porta OU. Fixando uma de suas entradas em 0, a condição de maioria só é atingida caso as outras duas variáveis de entrada sejam verdadeiras. Tem-se como exemplo a função $F(A, B) = \text{AB}$, sendo $M(A, 0, B)$ a representação dessa função em lógica majoritária, lendo-se maioria de A e B.

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

4 Método do mapa-K para quatro variáveis de entrada

Em Wang et al. (2011) foi proposto um método que utiliza o mapa-K para resolver problemas de até quatro variáveis de entrada. Tem-se como base a combinação de noventa funções primitivas, que representam todos os arranjos possíveis entre as quatro variáveis de entrada e a porta majoritária.

Na Figura 6 apresenta-se um exemplo de seis funções primitivas implementadas em um mapa-K de quatro variáveis, destacam-se em verde os mintermos cobertos por cada função.



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

(d) D (e) $M(A, B, D)$ (f) $M(B, 1, D)$

Figura 6: Exemplo de funções primitivas implementadas em mapas-K de quatro variáveis.

Para exemplificar o funcionamento do método, tem-se como exemplo os mintermos Σm(0,10,11). O primeiro passo é desenhar o mapa-K da função destacando os mintermos que devem estar na solução. Na Figura 7 apresenta-se um mapa-K com os mintermos Σm(0,10,11) destacados em verde.

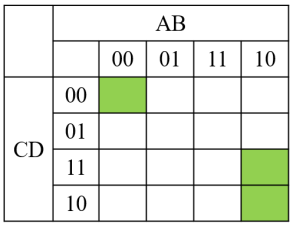
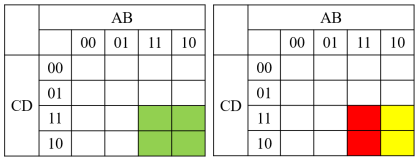


Figura 7: Mapa-K com os mintermos Σm(0,10,11) destacados.

O segundo passo é o de encontrar uma combinação de no máximo três funções primitivas de forma que os mintermos, destacados no passo 1, sejam cobertos duas ou mais vezes e que os maxtermos da função sejam cobertos no máximo uma vez.

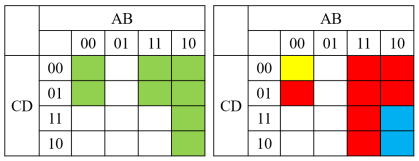
Escolhendo a função primitiva $M(A, 0, C)$, os mintermos Σm(11,10) e os maxtermos ΣM(14,15) são cobertos uma vez. Na Figura 8(a) apresenta-se o mapa-K da função escolhida. Na Figura 8(b) apresenta-se o mapa resultante após a escolha da primeira função primitiva. Em amarelo são destacados os mintermos que foram cobertos ao menos uma vez. Em vermelho são destacados os maxtermos que foram cobertos uma vez. Portanto não podem ser cobertos novamente por nenhuma outra função primitiva.



(a) $M(A, 0, C)$. (b) Mapa resultado.

Figura 8: Mapa resultado após a escolha do primeiro primitivo.

A segunda função primitiva deve ser encontrada. Na Figura 9(a) apresenta-se a escolha da função $M(A, \overline{B}, \overline{C})$ que cobre o mintermo Σm(0,10,11) e os maxtermos ΣM(1,8,9,12,13).

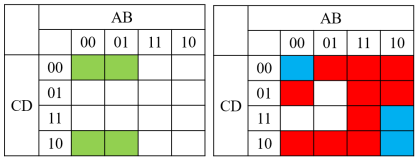


(a) $\overline{M}(\overline{A}, B, C)$. (b) Mapa resultado.

Figura 9: Mapa resultado após a escolha da segunda função primitiva.

A Figura 9(b) representa o mapa resultante após a escolha da segunda função primitiva. Em azul destacam-se os mintermos que foram cobertos duas vezes portanto estão presentes na solução final. Nada impede que um mintermo que já foi coberto duas vezes seja coberto novamente por uma terceira função. Mintermos já cobertos ao menos duas vezes passam a ser considerados *don't care states*.

Como o mintermo 0 ainda não foi coberto duas vezes, um terceiro primitivo que o cubra é selecionado. Deve-se garantir que os maxtermos cobertos uma vez não sejam cobertos pela nova função escolhida. Na Figura 10(a) apresenta-se a escolha da função $\overline{M}(A, 1, D)$ que cobre o mintermo Σm(0) e os maxtermos ΣM(2,4,6)



(a) $\overline{M}(A, 1, D)$. (b) Mapa resultado.

Figura 10: Mapa resultado após a escolha da terceira função primitiva.

No final do método, todas as funções primitivas são conectados a uma porta majoritária a fim de formar a função majoritária:

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

A função tem como saída verdadeira os mintermos Σm(0,10,11).

Caso não seja possível encontrar três funções primitivas que cubram os mintermos desejados, é adicionado um nível na função a fim de atingir a maioria. Tem-se como exemplo os mintermos Σm$(1, 5, 8, 15)$. Na Figura 11 apresenta-se o mapa-K em que destacam-se em verde os mintermos.

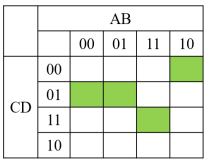
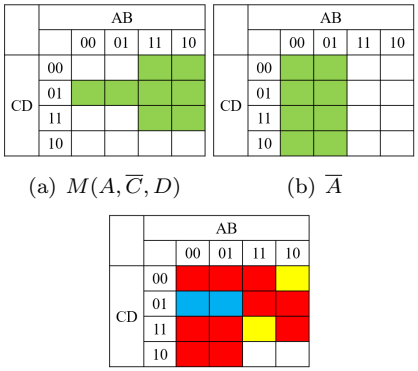


Figura 11: Mapa-K para os mintermos Σm$(1, 5, 8, 15)$.

Não é possível encontrar uma solução para essa função utilizando apenas três funções primitivas. Portanto, encontram-se até duas funções primitivas 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 e é necessário garantir que nenhum maxtermo seja coberto duas vezes (Wang et al., 2011).

A combinação das funções $M(A, \overline{C}, D)$ e $\overline{A}$ cobrem os mintermos 1 e 5 duas vezes e os mintermos 8 e 15 uma vez. Na Figura 12(a) apresenta-se o mapa-K da função $M(A, \overline{C}, D)$ e na Figura 12(b) apresenta-se o mapa-K da variável $\overline{A}$.



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

(c) Mapa resultado.

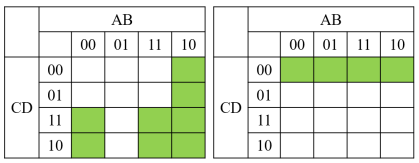
Figura 12: Funções primitivas escolhidas para cobertura do maior número de mintermos.

Na Figura 12(c) apresenta-se o mapa-K resultante da combinação dos mapas-K apresentados na Figura 12(a) e Figura 12(b). Em azul, destacam-se os mintermos cobertos duas ou mais vezes, portanto se encontram na solução da função e passam a ser considerados como *don't care states*. Em amarelo destacam-se os mintermos 8 e 15, que precisam ser cobertos mais uma vez para serem incluídos na solução. Em vermelho destacam-se os maxtermos que são cobertos uma vez, portanto não podem ser cobertos novamente.

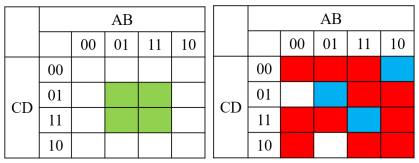
As funções $M(A, \overline{B}, C), \overline{M}(C, 1, D)$ e $M(B, 0, D)$ quando combinadas cobrem duas ve-

zes apenas os mintermos 8 e 15 e uma vez o *don't care state* 5. Os maxtermos 0, 2, 3, 4, 7, 9, 10, 11, 12, 13 e 4 são cobertos apenas uma vez, portanto não possuem uma saída verdadeira para a função $M(M(A,\overline{B},C),\overline{M}(C,1,D),M(B,0,D))$.

Na Figura 13(a) apresentam-se os mintermos cobertos pela função $M(A,\overline{B},C)$. Na Figura 13(b) apresentam-se os mintermos cobertos pela função $\overline{M}(C,1,D)$ e na Figura 13(c) apresentam-se os mintermos cobertos pela função $M(B,0,D)$.



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



(c) $M(B,0,D)$ (d) Mapa resultado.

Figura 13: Funções primitivas escolhidas para cobertura dos mintermos 8 e 15.

Na Figura 13(d) apresenta-se o mapa resultante da combinação das três funções primitivas escolhidas. No último passo do método, as duas funções majoritárias encontradas são conectadas a uma porta majoritária gerando a função:
M(M(A,$\overline{C}$,D),$\overline{A}$,M(M(A,$\overline{B}$,C),$\overline{M}(C,1,D)$),M(B,0,D))).

A função gerada possui saída verdadeira para os mintermos $\Sigma m(1,5,8,15)$.

5 Programa MK-4

O programa MK-4 proposto neste trabalho implementa o método do mapa-K de quatro variáveis. O MK-4 foi implementado na linguagem C++ utilizando a IDE Visual Studio 2017 no sistema operacional *Windows* 10. Como a porta majoritária é geralmente utilizada com a tecnologia QCA, em Kong et al. (2010) foram propostos critérios de custos, com o enfoque nesta tecnologia, que foram os mesmos critérios adotados neste trabalho. A seguir apresentam-se os critérios propostos:

- O número de níveis é considerado como critério primário de custo pois quanto maior a quantidade de níveis, maior é o atraso do circuito. Um novo nível é identificado quando tem-se uma porta majoritária usada como entrada em outra porta majoritária;
- O número de portas é considerado como critério secundário de custo. Portas que se repetem em uma função são computadas somente uma vez no 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.

Todas as 65.536 funções possíveis com até 4 variáveis de entrada foram geradas pelo MK-4. A fim de avaliar o programa desenvolvido em relação ao custo da função minimizada, os resultados obtidos foram comparados em termos de número de níveis, número de portas majoritárias, número de variáveis de entrada e número de inversores, com os resultados obtidos pelo programa *Exact*.

Na Tabela 1 apresenta-se a comparação de resultados entre os dois programas. Nota-se que as funções estão separadas pelo total de mintermos cobertos. Para exemplificar tem-se a linha 12 da tabela. Existem no total 1.820 funções, de quatro variáveis, que cobrem 12 mintermos. O programa MK-4 gerou um resultado de menor custo que os gerados pelo *Exact* em 879 funções, também encontrou 256 resultados iguais e 685 resultados de maior custo.

Tabela 1: Comparação de resultados obtidos.

T. Mint.	T. Funções	MK-4 <Exact	MK4 = Exact	MK-4 >Exact
0	1	0	1	0
1	16	10	6	0
2	120	37	75	8
3	560	346	209	5
4	1.820	743	407	670
5	4.368	2.140	1.014	1.214
6	8.008	3.381	1.436	3.191
7	11.440	5.028	1.206	5.206
8	12.870	4.220	1.669	6.981
9	11.440	5.030	983	5.427
10	8.008	3.668	1.083	3.257
11	4.368	2.524	681	1.163
12	1.820	879	256	685
13	560	446	107	7
14	120	89	23	8
15	16	12	4	0
16	1	0	1	0
Total	65.536	28.553	9.161	27.822

T. Mint. : Total de mintermos.
T. Funções: Total de Funções.

Destaca-se que para todos os casos gerados pelo MK-4, ao todo $43,57\%$ das funções geradas possuem menor custo, $13,97\%$ das funções geradas possuem custo equivalente e $42,46\%$ das funções geradas possuem maior custo quando comparadas com as funções geradas pelo Exact.

A geração de funções de menor custo pelo programa MK-4 em comparação com as funções geradas pelo *Exact* foi possível pois o MK-4 leva em consideração o número de níveis, número de portas majoritárias, número entradas e o número de inversores de uma função. Já o *Exact* leva em consideração somente o número de níveis e o número de portas majoritárias, critérios para qual o *Exact* garante a solução miníma para todas as funções de até 6 entradas.

Para exemplificar um caso onde o programa MK-4 gerou um resultado de menor custo em relação ao *Exact*, a seguir tem-se duas funções que cobrem o conjunto de mintermos Σm(10,11,14), uma gerada pelo MK-4 e outra gerada pelo *Exact*.

A função $M(M(A,0,C),\overline{M}(B,0,D),0)$, gerada pelo MK-4, possui: 2 níveis, 3 portas majoritárias, 6 entradas e 1 inversor.

A função $M(M(A,0,C),\overline{M}(B,\overline{C},D),0)$, gerada pelo *Exact*, possui: 2 níveis 3 portas majoritárias, 7 entradas e 2 inversores.

Assim embora o programa *Exact*, garante o mínimo número de níveis e de portas majoritárias, o programa MK-4 foi capaz de gerar uma função de menor custo reduzindo o número de entradas e de inversores.

Entretanto, apesar do programa MK-4 ter gerado, na maioria dos casos, uma função de custo menor ou equivalente ao *Exact*, foram identificados casos onde o *Exact* gerou melhores resultados. Para exemplificar um dos casos, tem-se como exemplo a função $\overline{M}(f1,f2,f3)$, gerada pelo MK-4, que possui 3 níveis, 8 portas majoritárias, 19 entradas e 5 inversores. A função gerada cobre os mintermos Σm(0,3,5,10,12,15). As funções $f1$, $f2$ e $f3$ são definidas por:

- $f1 = M(A,1,M(B,C,\overline{D}))$;
- $f2 = \overline{M}(A,0,M(B,C,\overline{D}))$;
- $f3 = M(\overline{M}(B,1,C),M(C,1,D),M(B,0,\overline{D}))$;

Sabe-se que:

- $f1$ esta otimizada para o conjunto de mintermos Σm(2,4,6,7,8,9,10,11,12,13,14,15);
- $f2$ esta otimizada para o conjunto de mintermos Σm(0,1,2,3,4,5,6,7,8,9,11,13);
- $f3$ esta otimizada para o conjunto de mintermos Σm(1,6,9,14).

As funções $f1$, $f2$ e $f3$ estão otimizadas para seus respectivos conjuntos de mintermos.

A função gerada pelo *Exact* para o conjunto de mintermos Σm(0,3,5,10,12,15) é igual a $\overline{M}(fx1,fx2,fx3)$ e possui 3 níveis, 7 portas majoritárias, 18 entradas e 7 inversores. As funções $fx1$, $fx2$ e $fx3$ são definidas por:

- $fx1 = \overline{M}(M(A,\overline{B},D),0,C)$;
- $fx2 = M(M(\overline{A},B,C),0,D)$;
- $fx3 = M(\overline{M}(\overline{C},0,D),M(A,\overline{B},D),M(\overline{A},B,C))$;

Sabe-se que:

- $fx1$ esta otimizada para o conjunto de mintermos Σm(0,1,2,4,5,6,7,8,9,12,13,14);
- $fx2$ esta otimizada para o conjunto de mintermos Σm(1,9,11,13);
- $fx3$ não esta otimizada para o conjunto de mintermos Σm(2,3,4,6,7,8,10,11,14,15).

A função $fx3$ gerada pelo *Exact* possui 2 níveis, 4 portas majoritárias, 11 entradas e 4 inversores. Para o conjunto de mintermos Σm(2,3,4,6,7,8,10,11,14,15) a função de menor custo é igual a $M(C,M(A,1,B),\overline{M}(A,B,D))$ que possui 2 níveis 3 portas majoritárias, 8 entradas e 1 inversor.

Optando por gerar uma função $fx3$ não otimizada para seu respectivo conjunto de mintermos, o *Exact* foi capaz de gerar a função $\overline{M}(fx1,fx2,fx3)$ com o custo menor que o gerado pelo MK-4 que gerou uma função otimizada para as três funções $f1$, $f2$ e $f3$.

Assim como trabalhos futuros pretende-se melhorar o programa MK-4 para que possa gerar funções $f3$ não otimizadas, mas que quando conectadas a uma porta majoritária com outras funções, seja possível gerar uma função otimizada.

6 Conclusões

Neste artigo foram apresentados os principais conceitos da lógica majoritária e sobre a tecnologia QCA. Também foi apresentado o programa MK-4, desenvolvido em C++, para síntese de funções majoritárias. Os resultados obtidos pelo MK-4 foram comparados com os resultados obtidos pelo programa *Exact*. Como o *Exact* não leva o número de entradas e o número de inversores na contagem do custo, o MK-4 que leva em consideração esses critérios foi capaz de gerar $43,57\%$ funções de menor custo, $13,97\%$ funções de custo equivalente e $42,46\%$ funções de maior custo quando comparadas com as funções geradas pelo *Exact*.

Agradecimentos

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

Referências

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.

Amarú, L., Gaillardon, P.-E. and De Micheli, G. (2015). Boolean logic optimization in majority-inverter graphs, *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, IEEE, pp. 1–6.

Chung, C.-C., Chen, Y.-C., Wang, C.-Y. and Wu, C.-C. (2017). Majority logic circuits optimisation by node merging, *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, IEEE, pp. 714–719.

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.

Kong, K., Shang, Y. and Lu, R. (2010). An optimized majority logic synthesis methodology for quantum-dot cellular automata, *IEEE Transactions on Nanotechnology* **9**(2): 170–183.

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

Sentovich, E. M., Singh, K. J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P. R., Brayton, R. K. and Sangiovanni-Vincentelli, A. (1992). Sis: A system for sequential circuit synthesis.

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* .

Sousa, W. and de Oliveira, R. (2015). Coulomb's law discretization method: A new methodology of spatial discretization for the radial point interpolation method, *IEEE Antennas and Propagation Magazine* **57**(2): 277–293.

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., Vemuru, S., Alam, M. and Killian, T. (2013). Comprehensive majority/minority logic synthesis method, *Nanotechnology (IEEE-NANO), 2013 13th IEEE Conference on*, IEEE, pp. 694–697.

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., Gupta, P. and Jha, N. K. (2007). 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* **26**(7): 1233–1245.

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.