UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Câmpus de Ilha Solteira

Programa de Pós-Graduação em Engenharia Elétrica

ALINE DE PAULA SANCHES

EMPREGO DO MÉTODO DE QUINE-MCCLUSKEY ESTENDIDO PARA GERAR CIRCUITO MÍNIMO COM ESTRUTURAS ESOP (XOR-XNOR)

Ilha Solteira
2017

ALINE DE PAULA SANCHES

EMPREGO DO MÉTODO DE QUINE-MCCLUSKEY ESTENDIDO PARA GERAR CIRCUITO MÍNIMO COM ESTRUTURAS ESOP (XOR-XNOR)

Dissertação apresentada à Faculdade de Engenharia – UNESP – Campus de Ilha Solteira, como requisito para obtenção do título de Mestre em Engenharia Elétrica.

Área de Conhecimento: Automação.

Prof. Dr. Alexandre César Rodrigues da Silva
Orientador

Ilha Solteira
2017

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

S211e

Sanches, Aline de Paula.
Emprego do método de Quine-Mccluskey estendido para gerar circuito mínimo com estruturas ESOP (XOR-XNOR) / Aline de Paula Sanches. -- Ilha Solteira: [s.n.], 2017
104 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2017

Orientador: Alexandre César Rodrigues da Silva
Inclui bibliografia

1. Geração de implicantes primos. 2. Método de Quine-Mccluskey. 3. Expressões AND-XOR-XNOR.

unesp

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Emprego do método de Quine-McCluskey estendido para gerar circuito mínimo com estrutura ESOP (XOR-XNOR)

AUTORA: ALINE DE PAULA SANCHES
ORIENTADOR: ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovada como parte das exigências para obtenção do Título de Mestra 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. TÉRCIO ALBERTO DOS SANTOS FILHO
Departamento de Ciências da Computação / Universidade Federal de Goiás

Ilha Solteira, 06 de julho de 2017

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus, por ouvir minhas orações e ter permitido que eu chegasse nesta grande etapa.

Aos meus pais, pelo incentivo, motivação e por terem me ajudado financeiramente a fazer uma pós-graduação à distância.

Em especial queria agradecer minha mãe, que sempre me deu muita força, incentivo e carinho, principalmente nesta grande etapa.

Ao orientador Dr. Alexandre César Rodrigues da Silva, por ter me proporcionado esta grande oportunidade de fazer um mestrado. Agradeço pela paciência, pelos ensinamentos, dedicação e incentivo que me proporcionava a cada progresso conquistado desde o início até este grande momento.

Agradeço também aos meus amigos e aos amigos do laboratório pela amizade e por me ajudarem em momentos de dúvidas.

.

RESUMO

Com a disseminação de dispositivos eletrônicos cada vez menores e o advento de novas tecnologias. A busca por métodos de minimização de funções booleanas tem sido a base para eletrônica digital. Neste trabalho apresenta-se a implementação da primeira fase do método Quine-McCluskey Estendido que utiliza-se de estruturas AND-XOR-XNOR para a geração de implicantes primos. O objetivo do trabalho foi comprovar que, na maioria das vezes, a implementação de uma função Booleana utilizando expressões AND-XOR-XNOR requerem menor quantidade de termos produtos, quando comparado com implementação com expressões AND-OR. A fase de cobertura dos mintermos em ambos os métodos foi formulada como um problema de programação linear inteira 0 e 1 que através do programa Lp_solve obteve a solução de menor custo. Na comparação da eficiência dos métodos foram analisados os custos dos circuitos mínimos gerados, a quantidade de memória utilizada e o tempo de execução. Com os resultados obtidos pode-se concluir que, para a maioria dos casos executados, o método Quine-McCluskey Estendido gera uma solução de menor custo. No entanto, com relação ao desempenho computacional (tempo de execução e memória), o método Quine-McCluskey Estendido apresentou-se inferior se comparado ao Quine-McCluskey.

Palavras-chave: Geração de implicantes primos. Método de Quine-McCluskey. Expressões AND-XOR-XNOR.

ABSTRACT

With the dissemination of smaller and smaller electronic devices and the advent of new technologies. The search for methods of minimizing Boolean function has been the basis for digital electronics. This work presents the implementation of the first phase of the Extended Quine-McCluskey method, which uses AND-XOR-XNOR structures to generate prime implicants. The goal of this work is to prove that, in most cases, the implementation of a Boolean function using the expressions AND-XOR-XNOR requires fewer product terms than the implementation with AND-OR expressions does. The stage of mini terms covering in both methods was formulated with the 0-1 integer linear programming problem, which obtained lower cost through the Lp_Solve program. While comparing the efficiency of these methods we analised: the costs of the minimum circuits generated, the amount of memory that has been used and the runtime. With the obtained results it is possible to conclude that, for most of the executed cases, the Extended Quine-McCluskey method generates a solution of lower cost. On the other hand, with regards to the computational performance (runtime and memory), the Extended Quine-McCluskey method has shown itself inferior when compared to the Quine-McCluskey method.

Keywords: Generation of prime implicants. Quine-McCluskey method. AND-XOR-XNOR expressions.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BDD	Binary Decision Diagrams
BDS	BDD-Based Logic Decomposition System
BOOM	Boolean Minimizer
CIs	Circuitos Integrados
CMOS	Complementary Metal-Oxide-Semicondutor
CPL	Complementary Pass-Transistor Logic
DNA	Deoxyribonucleic Acid
DPL Double Pass	Transistor Logic
EP-SOP	Exclusive-Or -Projected Sum of Products
ESOP Exclusive-or	Sum of Products Expressions
Exorcism-MV-2	Exorcism-Multiple-Valued-2
FC-Min	Find Coverage
FPGA	Field Programmable Gate Array
K-map	Mapa de Karnaugh
LP_SOLVE	Linear Programming Solve
MCNC	Multi-Channel Nonparametric Clustering
MILP	Problema de Programação Linear Inteira Mista (MILP)
NP-Hard	NP-Difícil
PTL	Pass Transistor Logic
QCA	Quantum-dot Cellular Automata
ROBDD	Reduced Ordered Binary Decision Diagram
SET	Single Electron Tunneling
SOP	Soma de Produtos
TPL	Tunneling Phase Logic
ULA	Unidade Lógica e Aritmética
XOR	Exclusive-Or

XNOR	Exclusive-Nor
XORBDS	Exclusive-Or BDD-Based Logic Decomposition System

SUMÁRIO

1 INTRODUÇÃO

Em decorrência dos crescentes desafios em se projetar circuitos integrados (CIs) contendo cada vez mais dispositivos eletrônicos e o advento das novas tecnologias de fabricação como, por exemplo, *quantum-dot cellular automata* (QCA), DNA, *single electron tunneling*(SET), *tunneling phase logic*(TPL) (WANG et al., 2015), a busca por métodos de minimização de funções Booleanas tornou-se incessante. Neste contexto a porta XOR tem sido a base para as novas abordagens com circuitos reversíveis. Além disso, a porta XOR interage naturalmente com a lógica majoritária (*majority logic*) (HAASWIJK et al., 2017).

Tradicionalmente, a especificação do comportamento de um circuito combinacional é realizada utilizando-se a denominada tabela verdade, onde se especificam todas as possíveis combinações das variáveis de entradas e qual deve ser a resposta para cada padrão de entrada. Geralmente, utilizam-se na representação algébrica de uma função Booleana as formas canônicas soma de produto ou produto de soma, conhecidas como implementação AND-OR.

Entretanto, além da tradicional implementação utilizando a estrutura AND-OR, existem outras formas de se representar algebricamente um circuito digital como, por exemplo, estruturas AND-XOR, lógica majoritária, e muitas outras.

Sasao (1990) mostrou que, na maioria das vezes, a implementação da estrutura AND-XOR requer menos termos produtos do que a implementação da estrutura AND-OR. Além disso, muitos dos projetos de sistemas utilizados no dia a dia são inerentemente soma módulo 2 como, por exemplo, sistemas lineares, sistemas de telecomunicações, circuitos aritméticos, circuitos usados no campo da teoria de codificação e muito outros. Uma das vantagens dos circuitos com estruturas AND-XOR se refere à testabilidade, ou seja, este tipo de estrutura é mais simples de ser testada (SASAO, 1997).

Dentro deste cenário, a implementação de funções Booleanas através de estruturas AND-XOR tem assumido cada vez mais importância. Cabe salientar que, com o advento da computação quântica, os circuitos lógicos podem ser implementados com portas XOR controladas, podendo assumir controle positivo, controle negativo ou controle misto (WILLE; DRECHSLER, 2010).

Além da minimização de funções Booleanas empregando soma de produto ou produto de soma, são descritos na literatura vários métodos de minimização cujo objetivo é a implementação da função Booleana utilizando a estrutura AND-XOR. Um dos métodos mais conhecidos para simplificar funções Booleanas visando implementação em estruturas XOR é

o denominado Exorcism-MV-2 (SONG; PERKOWSKI, 1993). Este método pode ser utilizado em funções com múltiplas saídas incompletamente especificadas e da mesma forma que o programa ESPRESSO, permite trabalhar com uma quantidade de variáveis teoricamente ilimitadas.

A literatura também apresenta métodos que utilizam de algoritmos genéticos (CHATTOPADHYAY; ROY; CHAUDHURI, 1996; PRADHAN; KUMAR; CHATTOPADHYAY, 2008), métodos tabulares (TINDER, 1995; KHALID; AWWAL, 1996), representação em diagrama de decisão binária (YE; ROY, 1997; YANG; SINGHAL; CIESIELSKI, 1999; YANG; CIESIELSKI, 2002), dentre outros.

Dentre os métodos estudados, um que se destacou, devido as suas características, foi o método de Quine-McCluskey Estendido (TURTON, 1996). Da mesma forma que o método de Quine-McCluskey tradicional (MCCLUSKEY, 1956), é um método tabular composto por duas fases. A primeira fase é denominada de geração dos implicantes primos e a segunda fase é denominada de cobertura dos mintermos. O método de Quine-McCluskey Estendido tem como característica a obtenção de estruturas AND-XOR, sempre que possível.

Neste trabalho desenvolveu-se, em programa de computador, a primeira fase do método de Quine-McCluskey Estendido com o objetivo de compará-la com os resultados obtidos pela primeira fase do método de Quine-McCluskey tradicional.

A fase de cobertura dos mintermos foi tratada como um problema de programação linear inteira 0 e 1 (SILVA,1993), em que a função objetiva é a soma ponderada das variáveis que representam os implicantes primos gerados na primeira fase do método e cada restrição é a soma das variáveis que cobrem cada um dos mintermos da função. Além disso, cada restrição gera uma desigualdade do tipo >= 1, garantindo assim que todos os mintermos da função sejam cobertos por pelo menos um dos implicantes primos gerados. Dessa forma têm-se tantas restrições quanto forem os mintermos da função. Os *don't care states* não geram nenhuma restrição, pois não precisam ser cobertos.

Para efeito de comparação foram avaliados os custos dos circuitos mínimos gerados, a quantidade de memória utilizada e o tempo de execução.

Com os resultados obtidos pode-se comprovar a afirmação de Sasao (1990) de que a implementação AND-XOR requer, na maioria das vezes, menos termos produtos do que a implementação tradicional empregando-se estruturas AND-OR. Entretanto, o desempenho

computacional (tempo e memória) do método de Quine-McCluskey Estendido mostrou-se inferior ao do método tradicional.

Na seção 1.1 apresenta-se o objetivo do trabalho. Na seção 1.2 apresenta-se a justificativa em que descreve-se os motivos que contribuíram para o desenvolvimento desta pesquisa, e na seção 1.3 uma revisão bibliográfica onde se descreve alguns métodos de minimização de funções Booleanas. Procurou-se destacar as características das diferentes abordagens empregadas e a eficiência da implementação computacional em relação ao tempo de execução e consumo de memória além do custo para a implementação dos circuitos obtidos.

1.1 OBJETIVO

O objetivo do trabalho é implementar a primeira fase do método Quine-McCluskey Estendido e comparar os resultados obtidos com a primeira fase do método Quine-McCluskey tradicional. Para avaliar a eficiência do método Quine-McCluskey Estendido foram analisados os parâmetros: custo do circuito gerado, tempo de execução e a quantidade de memória do computador utilizada.

1.2 JUSTIFICATIVA

Com o advento de novas tecnologias e a busca por métodos eficientes para minimização de funções booleanas tornou-se cada vez mais importante, o estudo de estruturas que possibilitem a redução no custo de circuitos lógicos. O uso de estruturas XOR/XNOR gera, na maioria das vezes, um circuito de menor custo, contribuindo com uma redução no consumo de energia e uma maior testabilidade.

1.3 REVISÃO BIBLIOGRÁFICA SOBRE MÉTODOS DE MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS

A álgebra de Boole ou álgebra Booleana de dois valores é definida sob um conjunto de dois elementos no conjunto B = {0, 1} com regras para dois operadores binários "+" e ".", além do operador complemento. Estes operadores são utilizados na simplificação das expressões Booleanas e são associados às operações lógicas OR, AND e NOT, utilizadas na

implementação de circuitos digitais. Com o emprego destes três operadores pode-se gerar outros operadores lógicos como o NAND, NOR, XOR e XNOR.

Dentre os métodos de minimização estudados para o desenvolvimento deste trabalho estão o Mapa de Karnaugh (KARNAUGH, 1953), o método de Quine-McCluskey (MCCLUSKEY, 1956), os algoritmos empregados no programa ESPRESSO (BRAYTON et al., 1984) e no BOOM (HLAVIZKA; FISER, 2001), além do método Quine-McCluskey Estendido (TURTON, 1996).

O Mapa de Karnaugh (KARNAUGH, 1953) é um método de minimização de funções Booleanas que se baseia na tabela verdade modificada, de forma que as células adjacentes diferem em apenas um bit e que representam cada um dos mintermos ou maxtermos da função. Define-se mintermo como sendo um termo produto contendo todas as variáveis na sua forma complementada ou não em que a função assume o valor lógico 1. Da mesma forma, um maxtermo é um termo produto contendo todas as variáveis na sua forma complementada ou não e que a função assume o valor lógico 0. Toda função Booleana pode ser representada pela soma dos mintermos ou produto dos maxtermos. O Mapa de Karnaugh é um método cujo resultado depende da habilidade do projetista, pois depende da identificação de alguns padrões de minimização, podendo ser utilizado para uma quantidade limitada de variáveis.

O método de Quine-McCluskey é um método tabular que gera todos os implicantes primos da função e, conforme já mencionado, consiste de duas fases. A primeira fase é denominada de geração de implicantes primos e a segunda fase é denominada de cobertura dos mintermos (MCCLUSKEY, 1956). Trata-se de um método muito ineficiente, pois o consumo de memória e o tempo de execução crescem exponencialmente com o aumento da quantidade de variáveis. Entretanto, tem como principal característica a obtenção de uma solução cujo custo de implementação do circuito representa um mínimo global. Este método é descrito em detalhes na Seção 2.6.4, pois foi utilizado na comparação com o algoritmo implementado.

Para contornar as limitações do Quine-McCluskey em relação ao crescimento exponencial de uso de memória e tempo de execução, Brayton et al. (1984), desenvolveu um método cujo algoritmo deu origem ao programa ESPRESSO-II, que por muito tempo foi considerado o método de minimização de funções Booleanas mais eficiente. Este método baseia-se nas operações de expansão e redução. A expansão consiste em aumentar ao máximo possível os implicantes primos gerados, de tal forma que os mintermos que são cobertos pelo

implicante expandido são removidos. A redução consiste em eliminar o número de implicantes primos, mantendo no conjunto solução apenas os implicantes primos que cobrem a função original. Por ser um algoritmo heurístico, não se garante que a solução obtida represente uma solução mínima global.

Além destes métodos, também foi estudado o método BOOM que foi desenvolvido por Hlavizka e Fiser (2001), que permite a minimização de funções Booleanas, combinando-se a fase de geração dos implicantes primos com o problema de cobertura. Assim como o ESPRESSO, esta estratégia permite reduzir a quantidade total de implicantes, levando a uma melhora significativa na geração de implicantes primos. Este método também permite simplificar funções Booleanas de múltiplas saídas.

Um método de destaque na minimização de funções Booleanas com múltiplas saídas é o método denominado FC-Min (*Find Coverage*) que emprega uma abordagem inversa aos dos anteriormente descritos, ou seja, inicia-se procurando por uma cobertura mínima da matriz que representa a função de saída para em seguida procurar por implicantes primos derivados da cobertura obtida, a partir dos vetores da matriz de entrada. Uma característica importante deste método, é que não necessita do conjunto de "*don't care states"* da função. Dessa forma, a função inicial é representada somente pelos mintermos e maxtermos. O método por apresentar uma abordagem inversa, é rápido e o consumo de memória é menor, podendo ser facilmente aplicado a problemas com maior número de variáveis de entrada sem crescimento rápido no tempo de execução (FISER; HLAVICKA; KUBATOVA, 2003).

Alguns métodos de minimização de funções Booleanas que utilizam portas XOR e XNOR, também foram estudados.

Yinshui, Xiexiong e Lunyao (2011), propuseram a lógica dual, que baseia na junção da lógica Booleana com a lógica *Reed-Muller*. Por definição a lógica Booleana se baseia nos operadores AND-OR-NOT e a lógica *Reed-Muller* nos operadores AND-XOR. A minimização utilizando a lógica dual inicia-se com a análise da função, verificando se é adequado aplicar a minimização da lógica dual. Caso o resultado da análise seja favorável, então a função é particionada em dois conjuntos. Cada um dos conjuntos é tratado em processos distintos, satisfazendo a expressão, ou seja, um conjunto é tratado pela lógica Booleana AND-OR e o outro conjunto pela lógica *Reed-Muller,* AND-XOR. Comparando os algoritmos de minimização que utilizam apenas lógica *Reed-Muller* e os algoritmos que utilizam a lógica Booleana, o algoritmo que utiliza lógica dual demonstrou gerar menos termos produtos para a maioria dos casos testados.

Cheng e Huang (1999) descreveram uma técnica para implementar funções XOR-XNOR utilizando-se a técnica do PTL (*Pass Transistor Logic*). Um PTL é um circuito projetado para um somador de alta velocidade e baixo consumo de potência, baseado no mapa de Karnaugh modificado. Utiliza-se do Mapa de Karnaugh (*K-map*) modificado, para obtenção dos circuitos PTL. A expressão Booleana para simplificar os sinais de controle e entrada do PTL é alterado, para implementar um somador total de um bit. Foram realizadas comparações do somador PTL, com o somador estático CMOS (*Complementary Metal-Oxide-Semicondutor*), somador de função de transmissão CPL (*Complementary Pass-Transistor Logic*), somador de portas de transmissão DPL (*Double Pass-Transistor Logic*) e somador de portas de transmissão CPL. De acordo com os resultados apresentados, o circuito PTL apresentou menor atraso e menor consumo de potência, ou seja, maior velocidade e menor consumo de energia.

A utilização de portas XOR contribui com a eficiência de implementação em FPGA (*Field Programmable Gate Array*) em relação ao custo e a lógica. No entanto, Muma e Ko (2005) propuseramo algoritmo denominado XORBDS que se baseia na minimização ESOP (*Exclusive-or Sum of Products Expressions*), seguido da decomposição Binária BDD (*Binary Decision Diagrams*). A minimização ESOP permitiu uma redução no número de termos produto, utilizadas para implementar a lógica XOR. A decomposição baseada em BDD divide o circuito minimizado e visita os nós compartilhados sempre que possível. O processo de minimização e decomposição consiste nas seguintes etapas: converter o circuito *benchmark*, em circuitos de previsão de paridade, que é quando se alteram os sinais da saída original, em sinais produzidos pela saída de paridade XOR, executa Exorcism 4, que é um circuito de previsão de paridade, para obter este no formato ESOP, e por fim executa o BDS, no resultado minimizado do passo 2 para decompor o circuito de previsão de paridade. Os testes foram realizados em 14 circuitos MCNC de *benchmark.* O algoritmo XORBDS apresentou melhoras significativas na utilização ESOP e a decomposição baseada em BDD, pois foram obtidos os circuitos de previsão de paridade minimizados antes de aplicar a decomposição BDD.

Bernasconi, Ciriani e Cordone (2006) propuseram uma rede de quatro níveis para soma de produtos XOR. Primeiramente define-se uma forma algébrica para representação de função Booleana XOR (EP-SOP) e então prova-se que derivar um EP-SOP ótimo a partir da forma SOP (soma de produtos) é um problema difícil *(NPhard*). A partir desta etapa, descreve-se um algoritmo de aproximação que retorna em tempo polinomial uma forma EP-SOP, em que o custo, assegura-se estar próximo do ótimo. Este algoritmo foi proposto de

forma a reduzir a área do circuito resultante. De acordo com os experimentos 35% dos *benchmarks* de síntese EP-SOP mostraram obter uma menor área e atraso com relação à forma ótima SOP, obtendo algumas vezes 40-50% da área resultante da rede.

Kohutka e Pistek (2014) propuseram um método de otimização para árvores multiplexadores, utilizando métodos BDD, variáveis residuais, tabela *hash* e abordagem *top-down*. E para obter melhores resultados foi utilizado portas lógicas básicas para substituição de alguns multiplexadores na árvore. O método BDD usa a ordenação de variáveis implícitas definido pela função booleana de entrada em uma forma de vetor, até que todas variáveis ordenadas possam ser testadas alterando o vetor de entrada e criando ROBDD (*Reduced Ordered Binary Decision Diagram*) a partir de cada vetor. As variáveis residuais utilizadas são multiplexadores que podem ser substituídos por variáveis de entrada direta ou variável de entrada negada e em alguns outros casos portas lógicas básicas. Outro método utilizado é a tabela *hash* em que cada nó criado, é adicionado à lista ligada, gera-se um índice deste nó por uma função *hash*. Compara-se o novo nó com os nós de mesma profundidade que estão no mesmo índice da tabela *hash*. E por último tem-se a abordagem *top-down* de cima para baixo, em que os nós BDD são criados do topo (começando pela raiz) até seus sucessores. Durante esta criação, as regras de redução "unicidade" e "não redundância" são verificadas em cada nó BDD antes da criação do nó. Quando uma dessas regras de redução é aplicada, o BDD é reduzido, remove-se o nó, esse nó não é criado.

De acordo com os resultados obtidos pode se comprovar que o uso da tabela *hash* e abordagem *top-down* contribuíram com a velocidade no processo de síntese em funções booleanas com maior número de variáveis de entrada. Pois o uso destes dois métodos faz com que os nós BDD sejam removidos o mais rápido possível, evitando a passagem por todos os caminhos do BDD. O segundo benefício é a possibilidade de substituir alguns multiplexadores com portas lógicas básicas AND, OR e XOR. Por exemplo, 41,67% dos multiplexadores podem ser substituídos em "rd53", o que resulta em 15% menos transistores na tecnologia CMOS.

Callegaro et al. (2015) propôs uma abordagem para a decomposição booleana com base na diferença booleana e na análise do cofator, em que utilizou-se de multiplexadores com duas entradas e uma saída. Dois algoritmos foram suficientes para identificar as decomposições AND e XOR. Um dos algoritmos para obter funções de decomposição utilizou de análise de gráficos de decomposição. Enquanto que o algoritmo proposto, utilizou cofatores $O\ (n \log n)$ e $O\ (n)$ para obter funções de decomposição AND e XOR e além disto,

este algoritmo forneceu condições suficientes e necessárias para obter multiplexadores com número arbitrário de entrada. Foram realizados testes com os dois algoritmos, de acordo com os resultados obtidos o algoritmo proposto demonstrou uma maior eficiência quando comparado ao algoritmo que utilizou gráficos de decomposição.

A minimização de circuitos têm sido importante em circuitos digitais, pois possibilita uma redução no consumo de energia, controle de potência e uma diminuição nos danos dos circuitos. Com o objetivo de obter a minimização de circuitos, Mbock (2015) propôs uma abordagem inovadora que consiste na utilização de uma matriz *n x n*, em que aplica-se um vetor de bits específicos. Esta abordagem explora recursos do System Generator e o poder computacional dos cálculos das matrizes. Esta ferramenta System Generator fornece muitos recursos lógicos e especificamente (portas) que permiti a construção de circuito aproximado. De acordo com os resultados obtidos com o uso do System Generator há uma grande redução no número de portas, de 1000 portas lógicas houve uma redução de 630.

Dhivyakala, Selvan e Kavitha (2016), propuseram o projeto de um circuito de compressão 4-2 usando o módulo XOR-XNOR e o multiplexador 2-1. O compressor é utilizado para realizar multiplicações de forma a reduzir os produtos parciais. O circuito de compressão 4-2 possui cinco entradas e três saídas. As cinco entradas são compressas em três saídas, onde X1, X2, X3, X4, Cin são as entradas e Cout, Carry, Sum são as saídas. Este compressor é projetado para área de baixa potência. Há diferentes arquiteturas e projetos utilizados para circuitos compressores 4-2. A lógica CMOS complementar é usada para implementar circuito XOR-XNOR. É uma técnica que requer grande número de transistores e maior consumo de energia. Além desta, tem-se o projeto 8T XOR-XNOR oferece melhor capacidade de condução, mas requer um grande número de transistores e grande área de layout. O projeto 6T XOR-XNOR, que apresenta a degradação do balanço da tensão de saída e sua capacidade de condução é fraca.

O circuito de compressão 4-2 foi comparado com estas diferentes arquiteturas e apresentou um menor número de transistores e consumo de energia.

Alguns estudos foram realizados visando obter uma redução no consumo de energia. Com base neste propósito, Usha, Rajendiran e Kavitha (2016) propuseram uma ULA (Unidade Lógica e Aritmética), de baixa potência, utilizando o operador XNOR. A ULA é responsável em realizar operações aritméticas como ADD, SUB, MULT, DIV, etc e operações lógicas como AND, OR, XOR, XNOR, etc. O principal componente de uma ULA

é o somador total, ou seja, se reduzir a potência consumida no somador, isto implica na redução da potência da ULA.

Ilynin et al. (2017), descreve sobre algoritmos evolutivos para minimização de funções booleanas soma de produtos (XOR). A minimização é baseada na decomposição da soma de três funções booleanas. Entre as 3 funções escolhe-se uma arbitrariamente. A minimização de funções de *n*-variável pode ser reduzida para encontrar *(n-1)* variável da função *f3* em que o valor *L(f1)+L(f2)+L(f3)* possa ser o mínimo. Várias restrições de pesquisa de espaço para a função *f3* produz vários algoritmos de minimização de aproximação. Estes algoritmos dão o limite superior para o complexidade do polinômio. Dentre os algoritmos utilizados tem-se: o algoritmo genético, colônia de abelhas, algoritmo da mosca e o algoritmo do macaco.

O limite superior do algoritmo genético foi funções com 8 variáveis em que o limite é calculado através de uma classe de funções, das funções são obtidos os respectivos polinômios e a complexidade é igual ao limite superior destas funções. Na colônia de abelhas com o aumento no número de abelhas e a disperção da distância entre elas, aumenta a qualidade do algoritmo para minimizar funções com 6 variáveis, obtendo na maioria dos casos o mínimo exato. Algoritmo da mosca devido à enumeração completa dos pontos dos bairros, o número de candidatos é relativamente pequeno, parâmetros de função 10-30. No entanto ocorre um grande efeito sobre o resultado final que é o número de bits que “inspeciona” a “mosca” no espaço. No algoritmo do macaco conforme o número de “macacos” aumenta, encontra soluções exatas rapidamente. E mudança nos parâmetros do algoritmo como: o número de bits para o “salto” do macaco, o número de “saltos” bit, para novas ascensões do macaco, que de fato não exercem influência significativa.

A minimização de funções Booleanas utilizando portas XOR, tem contribuído na redução do número de portas lógicas. A utilização de métodos heurísticos para sintetizar funções incompletamente especificadas (assumem tanto valor 0 ou 1) para circuitos XOR-AND-OR, foi proposto por Schaeffer (2017).

Dada a importância da estrutura XOR/XNOR nas implementações atuais, decorrente da novas tecnologias de fabricação, neste trabalho estudou-se um algoritmo de minimização que gera, sempre que possível, estruturas XOR/XNOR, visando a obtenção de uma solução de custo mínimo.

Neste capítulo foram apresentados os métodos de minimização de função booleanas encontrados na literatura, que utilizam operadores AND, OR, XOR e XNOR.

Este trabalho está estruturado na seguinte forma: no Capítulo 2 apresentam-se, os conceitos básicos da álgebra de Boole, enfatizando os teoremas e propriedades que envolvem a operação XOR. Apresentam-se também as diferentes formas de representação de uma função Booleana, bem como os teoremas e propriedades algébricas utilizadas na simplificação das funções Booleanas. Para que o comportamento dos métodos comparados pudessem ser visualizados apresenta-se na Seção 2.6.3 o Mapa de Karnaugh, utilizado na simplificação de função Booleana. Visando oferecer o embasamento teórico para o entendimento do Método de Quine-McCluskey Estendido, descreve-se na Seção 2.6.4 o tradicional método de Quine-McCluskey.

No Capítulo 3 apresentam-se as características e o algoritmo que foi utilizado na implementação em programa de computador do método de Quine-McCluskey Estendido.

No Capítulo 4 descreve-se o fluxograma do programa desenvolvido, algumas rotinas utilizadas para se obter dados utilizados na avaliação do desempenho, além da metodologia de teste utilizada.

Por fim, a comparação dos resultados e os comentários são apresentados no Capítulo 5.

2 ÁLGEBRA DE BOOLE

A álgebra de Boole foi desenvolvida em 1854, pelo matemático inglês George Boole, que publicou um livro clássico intitulado "*An investigation into the Laws of Thought*", em que foi descrito o modo como se toma decisões lógicas com base em circunstâncias verdadeiras ou falsas (BOOLE, 1854).

Em 1938, o engenheiro americano, Claude Elwood Shannon, aplicou esta álgebra para mostrar que as propriedades de circuitos elétricos de chaveamento podem ser representadas por uma álgebra Booleana com dois valores 0 e 1 (SHANNON, 1938). Shannon (1938) publicou o trabalho denominado "*A Symbolic Analysis of Relay and Switching Circuits*", que introduziu na área tecnológica o campo da eletrônica digital.

A álgebra de Boole é utilizada como um grande instrumento para análise e simplificação de sistemas lógicos, como também uma excelente ferramenta de projeto para circuitos lógicos.

2.1 OPERADORES DA ÁLGEBRA BOOLEANA

Os operadores fundamentais da álgebra Booleana são basicamente: OR, AND e NOT. Além destes tem-se os operadores XOR e XNOR. Todas as funções Booleanas podem ser representadas em termos destas operações básicas, portanto é necessário entender o funcionamento de cada um dos conectivos (TOCCI; WIDMER; MOSS, 2011).

2.1.1 Operador OR (Adição)

O operador OR é definido como saída igual a 1 se pelo menos uma das variáveis de entrada for 1, caso contrário a saída será 0. Este operador é representado pelo símbolo "+" ou "$\vee$", a expressão pode ser resumida como: Z = X+Y, sendo X e Y variáveis de entrada e Z função de saída.

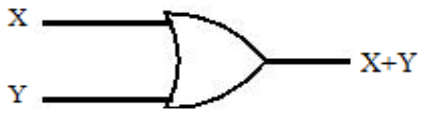
Na Tabela 1 apresenta-se a tabela verdade do operador OR e na Figura 1, o respectivo símbolo representativo.

Tabela 1- Operador OR.

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Fonte: Próprio autor.

Figura 1 - Simbologia da porta OR.



Fonte: Próprio autor.

2.1.2 Operador AND (Multiplicação)

No operador AND, ao contrário do operador OR, a saída é 0 se pelo menos uma das variáveis de entrada for 0, caso contrário a saída será 1. O operador AND é representado pelo símbolo "." ou "$\wedge$", a expressão pode ser resumida como: Z = X .Y, sendo X e Y variáveis de entrada e Z função de saída.

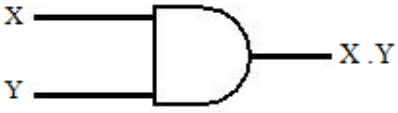
Na Tabela 2 apresenta-se a tabela verdade do operador AND e na Figura 2, o respectivo símbolo representativo.

Tabela 2 - Operador AND.

X	**Y**	**AND**
0	0	0
0	1	0
1	0	0
1	1	1

Fonte: Próprio autor.

Figura 2 - Simbologia da porta AND.



Fonte: Próprio autor.

2.1.3 Complementação NOT (Negação ou Inversão)

O operador NOT é a operação cujo resultado é o valor complementar ao valor assumido pela variável de entrada. Também devido ao fato de uma variável Booleana assumir um entre somente dois valores, o valor complementar será 1 se a variável vale 0 e será 0 se a variável vale 1.

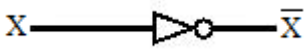
Os símbolos utilizados para representar a operação complementação em uma variável Booleana X são: $\overline{X}$, ~X e X' (lê-se X negado). Na Tabela 3 apresenta-se a tabela verdade do operador NOT e na Figura 3, o respectivo símbolo representativo.

Tabela 3- Operador NOT.

X	$\overline{X}$
0	1
1	0

Fonte: Próprio autor

Figura 3 - Simbologia da porta NOT.



Fonte: Próprio autor.

Diferente dos operadores OR e AND, a complementação é definida sobre uma variável, ou sobre o resultado de uma expressão, ou seja, o operador complementação é definido unário.

2.1.4 Operador XOR ou *Exclusive-Or*

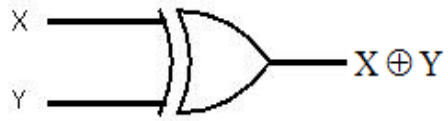
O operador XOR, também conhecido como *Exclusive-or*, apresenta função de saída igual a 1, se as variáveis de entrada são diferentes e 0 se as variáveis de entradas são iguais. O operador XOR é representado pelo símbolo $\oplus$, a expressão pode ser resumida como: $X \oplus Y = \overline{X}Y + X\overline{Y}$, sendo X e Y variáveis Booleanas. Na Tabela 4 apresenta-se a tabela verdade do operador XOR e na Figura 4, respectivo símbolo representativo.

Tabela 4 - Operador XOR.

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Fonte: Próprio autor.

Figura 4 - Simbologia da porta XOR.



Fonte: Próprio autor.

A expressão $X \oplus Y$ pode ser interpretada como “X não é igual a Y” e pode ser generalizada para permitir qualquer quantidade de operadores, pois trata-se de uma função com a propriedade associativa.

Em decorrência da propriedade associativa pode-se escrever a expressão como $X_1 \oplus X_2 \oplus \ldots \oplus X_n$. Esta forma de representá-la é muito útil, pois pode-se operar com pares de variáveis e substituir cada par com o resultado da operação binária realizada. Este processo de substituir o par de variáveis pelo resultado da operação binária pode ser repetido até que todas as variáveis tenham sido tratadas.

O método que foi implementado em linguagem de computador, denominado de Quine-McCluskey Estendido, é baseado nas propriedades desta função, por isso, algumas propriedades da função XOR são apresentadas a seguir (UNGER, 1997):

$0 \oplus 0 = 1 \oplus 1 = 0$; (1)

$0 \oplus 1 = 1 \oplus 0 = 1$; (2)

$X \oplus 0 = X$; (identidade elemento é 0) (3)

$X \oplus X = 0$; ($\oplus$ comportamento da subtração) (4)

$X \oplus \overline{X} = 1$; (5)

$X \oplus 1 = \overline{X}$; (6)

$\overline{X \oplus Y} = \overline{X} \oplus Y = X \oplus \overline{Y} = 1 \oplus X \oplus Y = \overline{X}\,\overline{Y} + XY$ (7)
(complemento da função XOR);

$X \oplus Y = Y \oplus X$ (propriedade comutativa); (8)

$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$ (propriedade associativa); (9)

$X(Y \oplus Z) = XY \oplus XZ$ (lei distributiva); (10)

$X \oplus Y = X \oplus Z$ implica que $Y = Z$. (11)
(propriedade de cancelamento)

2.1.5 Operador XNOR ou *Exclusive-Nor*

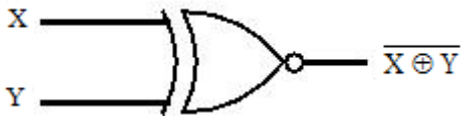
O operador XNOR é a porta lógica XOR complementada. Também conhecida como *Exclusive-Nor* ou função coincidência, em que a saída é igual a 1, se as variáveis de entradas são iguais, caso contrário a função de saída é igual a 0. O operador XNOR é representado pelo operador XOR invertido, sendo: $\overline{X \oplus Y} = XY + \overline{X}\,\overline{Y}$. Na Tabela 5 apresenta-se a tabela verdade do operador XNOR e na Figura 5, o respectivo símbolo representativo.

Tabela 5 - Operador XNOR.

X	Y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Fonte: Próprio autor.

Figura 5 - Simbologia da porta XNOR.



Fonte: Próprio autor.

2.2 LEIS E PROPRIEDADES DA ÁLGEBRA BOOLEANA

Expressão Booleana é a sentença matemática composta de termos cujas variáveis são Booleanas, da mesma forma, podendo assumir como resultado final 0 ou 1 (CAPUANO; IDOETA, 2002). Teorema é um conjunto de regras que são utilizadas para simplificar expressões lógicas e circuitos digitais.

2.2.1 Teoremas

A seguir são apresentados os seguintes teoremas: adição (OR), produto (AND) e complemento ou inversão lógica (NOT) e propriedades da álgebra Booleana (CAPUANO; IDOETA, 2002).

a)Teorema da Adição

$$A + 0 = A \quad (12)$$

$$A + 1 = 1 \quad (13)$$

$$A + \overline{A} = 1 \quad (14)$$

$$A + A = A \quad (15)$$

b) Teorema do Produto

$$A \cdot 0 = 0 \quad (16)$$

$$A \cdot 1 = A \quad (17)$$

$$A \cdot \overline{A} = 0 \quad (18)$$

$$A \cdot A = A \quad (19)$$

c) Teorema do Complemento ou da Inversão Lógica

$$\overline{A} = A \quad (20)$$

2.2.2 Propriedades

As principais propriedades algébricas, usadas no manuseio e simplificação de expressões são:

a) Propriedade Comutativa

É aplicada nas operações de adição e multiplicação, onde a ordem dos fatores, não altera o resultado.

$$\text{Adição: } A + B = B + A \quad (21)$$

$$\text{Multiplicação: } A \cdot B = B \cdot A \quad (22)$$

b) Propriedade Associativa

Da mesma forma que a propriedade comutativa, é aplicada nas operações de adição e multiplicação, em que a ordem dos agrupamentos, não altera o resultado.

$$\text{Adição: } A + (B + C) = (A + B) + C = A + B + C \quad (23)$$

$$\text{Multiplicação: } A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C \quad (24)$$

c) Propriedade Distributiva

Na propriedade distributiva, uma expressão é expandida multiplicando cada termo da expressão, sendo que uma expressão pode conter mais de um tipo de operação, diferentemente da associativa que é obrigatório a expressão conter apenas um tipo de operação.

$$A \cdot (B + C) = A \cdot B + A \cdot C \quad (25)$$

$$A + (B \cdot C) = (A + B) \cdot (A + C) \quad (26)$$

d) Identidades Auxiliares

Algumas identidades auxiliares, utilizadas na simplificação de expressões, são:

$$A + A \cdot B = A \quad (27)$$

$$(A + B) \cdot (A + C) = A + B \cdot C \quad (28)$$

$$A + \overline{A} \cdot B = A + B \quad (29)$$

2.3 TEOREMAS DE DE MORGAN

O primeiro teorema de De Morgan define que o complemento do produto é igual a soma dos complementos de cada variável, ou seja, ele converte a operação AND (.) em uma operação OR (+), logo:

$$\overline{A.B.C. \ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots \quad (30)$$

O segundo teorema é dual do primeiro, pois define que o complemento da soma é igual ao produto dos complementos de cada variável, ou seja, realiza-se a operação inversa do primeiro, convertendo a operação OR (+) em uma operação AND (.), portanto tem-se:

$$\overline{A + B + C + \ldots} = \overline{A}.\overline{B}.\overline{C}\ldots \quad (31)$$

2.4 SIMPLIFICAÇÃO DE EXPRESSÕES BOOLEANAS

O processo de redução de operações equivalentemente é denominado simplificação. Com o uso da álgebra de Boole, pode-se simplificar expressões e circuitos. Para realizar as simplificações, existem basicamente dois processos: o primeiro que é a simplificação através de álgebra de Boole e o segundo pelo mapa de Karnaugh. Neste exemplo apresenta-se a simplificação por álgebra de Boole, e nos próximos tópicos, apresenta-se a simplificação pelo mapa de Karnaugh. Considere a expressão:

$$S = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}C \quad (32)$$

Analisando a expressão, pode-se observar termos em comum como: $\overline{A}\overline{C}$, portanto aplica-se a propriedade distributiva, cujo os termos em comum coloca-se em evidência:

$$S = \overline{A}\overline{C}(\overline{B}+B)+A\overline{B}C \quad (33)$$

Conforme a expressão acima, como tem-se variáveis que se diferenciam em apenas um literal B e $\overline{B}$, aplica-se a identidade ($B + \overline{B} = 1$):

$$S = \overline{A}\overline{C}(1)+A\overline{B}C \quad (34)$$

Logo:

$$S = \overline{A}\overline{C}+A\overline{B}C \quad (35)$$

2.5 FUNÇÕES BOOLEANAS

A função booleana é uma função que depende da combinação das variáveis de entrada e dos operadores, para produzir um resultado (saída). No caso do operador XOR, a saída é 1, se a combinação dos valores de entradas forem diferentes e a saída é 0, se a combinação dos valores de entradas forem iguais.

A função Booleana pode ser representada por uma tabela com linhas e colunas, onde cada linha da tabela representa uma combinação diferente de valores de entrada, esta tabela é denominada tabela-verdade, sendo 2^n o número de combinações possíveis para as variáveis a serem consideradas na determinação do valor da função.

Para uma melhor interpretação, sobre funções Booleanas, apresenta-se um exemplo de tabela verdade para uma função Booleana de 2 variáveis, tem-se:

$$F(A,B) = A + B \quad (36)$$

Para determinar a quantidade de combinações aplica-se a formula do número de combinações 2^n, onde *n* é o número de entradas. No exemplo dado tem-se 2 entradas, logo $2^2 = 4$, portanto foram obtidos 4 combinações, conforme apresentado na Tabela 6.

Tabela 6 - Tabela verdade para a função F(A,B) = A + B.

A	**B**	**F(A,B)**
0	0	0
0	1	1
1	0	1
1	1	1

Fonte: Próprio autor.

2.6 FORMAS CANÔNICAS

As expressões Booleanas podem ser escritas em uma forma padronizada, denominada forma normal ou forma canônica. Uma função F(A,B,C) esta na forma canônica se F pode ser representada como soma de produtos ou produto de somas, em que todas as variáveis da soma de produtos ou produto de somas estão presentes. A função Booleana F(A,B,C) =$\sum m$(0, 2, 5, 7) pode ser representada na forma canônica soma de produtos ou produto da somas:

- Soma de produtos: $F(A,B,C) = \overline{A}\overline{B}\overline{C} + \overline{A}BC + A\overline{B}C + ABC$ (37)
- Produto de somas: $F(A,B,C) = (A + B + \overline{C}).(A + \overline{B} + \overline{C}).(\overline{A} + B + C).(\overline{A} + \overline{B} + C)$ (38)

As vantagens de se utilizar a forma canônica, é que se permite uma maior independência, facilitando a simplificação e a identificação de expressões para representar a mesma função lógica (NOGUEIRA, 2011).

2.6.1 Soma de Produtos

A soma de produtos é uma forma padrão de representação de funções Booleanas constituída pela aplicação da operação lógica OR sobre um conjunto de termos formados pela operação AND.

A cada combinação de entradas pode-se associar um termo produto, em que todas as variáveis da função estão presentes. Se a variável correspondente vale 0, deve-se representá-la na forma barrada e se a variável vale 1, deve-se representá-la na forma não barrada.

Uma função Booleana pode ser expressa algebricamente, como uma soma de produtos, obtida diretamente a partir da tabela verdade. A soma inclui todos os mintermos para os quais a função vale 1. Na Tabela 7 apresenta-se um exemplo da soma de produtos:

Tabela 7 - Forma canônica soma de produtos.

Decimal	A	B	C	F	
0	0	0	0	1	$\rightarrow \overline{A}\overline{B}\overline{C}$
1	0	0	1	0	
2	0	1	0	0	
3	0	1	1	1	$\rightarrow \overline{A}BC$
4	1	0	0	1	$\rightarrow A\overline{B}\overline{C}$
5	1	0	1	0	
6	1	1	0	0	
7	1	1	1	1	$\rightarrow ABC$

Fonte: Próprio autor.

O primeiro passo para se obter a expressão soma de produto é encontrar os mintermos. Os mintermos correspondem a cada linha da tabela onde a função de saída assume valor 1. Pode-se notar na Tabela 7 para F = 1, foram obtidos os seguintes termos: $\overline{A}\overline{B}\overline{C}$, $\overline{A}BC$, $A\overline{B}\overline{C}$ e ABC. Como o exemplo estudado nesta seção é soma de produtos, utiliza-se o operador OR entre os termos produtos:

$$F(ABC) = \overline{A}\overline{B}\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + ABC \tag{39}$$

Representando a função em binário tem-se:

Binário: 0 0 0, 0 1 1, 1 0 0, 1 1 1.

Transformando, cada número binário em decimal, tem-se os seguintes mintermos:

Mintermos:(0), (3), (4), (7).

Logo:

$F(ABC) = \sum m(0, 3, 4, 7)$

2.6.2 Produto de Somas

O produto de somas é outra forma padrão de representação de funções Booleanas caracterizada pela aplicação da operação lógica AND sobre um conjunto de termos formados pela operação OR.

A cada combinação de entradas pode-se associar um termo produto, em que todas as variáveis da função estão presentes. Se a variável correspondente vale 0, deve-se representá-la na forma não barrada e se a variável vale 1, deve-se representá-la na forma barrada.

Uma função Booleana pode ser expressa algebricamente, como produto de somas, diretamente a partir da tabela de verdade. O produto inclui todos os mintermos para os quais a função vale 0. Na Tabela 8 apresenta-se um exemplo do produto de somas.

Tabela 8 - Forma canônica produto de somas.

Decimal	**A**	**B**	**C**	**F**	
0	0	0	0	1	
1	0	0	1	0	$\rightarrow A + B + \overline{C}$
2	0	1	0	0	$\rightarrow A + \overline{B} + C$
3	0	1	1	1	
4	1	0	0	1	
5	1	0	1	0	$\rightarrow \overline{A} + B + \overline{C}$
6	1	1	0	0	$\rightarrow \overline{A} + \overline{B} + C$
7	1	1	1	1	

Fonte: Próprio autor.

Inicialmente encontra-se os maxtermos, que corresponde a cada linha da tabela em que a função de saída é igual a 0. Para F = 0, foram obtidos os seguintes termos: $A + B + \overline{C}$, $A + \overline{B} + C$, $\overline{A} + B + \overline{C}$ e $\overline{A} + \overline{B} + C$. Após esta etapa, cria-se a função. Como o exemplo estudado é produto de somas, a operação a ser utilizada será AND entre os termos somas:

$$F(ABC) = (A + B + \overline{C}).(A + \overline{B} + C).(\overline{A} + B + \overline{C}).(\overline{A} + \overline{B} + C) \quad (40)$$

Representando a função em binário tem-se:

Binário: (0 0 1), (0 1 0), (1 0 1), (1 1 0).

Transformando, cada número binário em decimal, tem-se os seguintes maxtermos:

Maxtermos: (1), (3), (5), (6).

Logo:

F(ABC) = M(1, 3, 5, 6)

Nas Seções 2.6.3 e 2.6.4, descrevem-se alguns métodos de minimização de funções Booleanas como mapa de Karnaugh e o método de Quine-McCluskey.

2.6.3 Mapa de Karnaugh

Em 1953, foi desenvolvido o mapa de Karnaugh, pelo matemático e físico Maurice Karnaugh, enquanto trabalhava no grupo de pesquisas da empresa Bell (KARNAUGH, 1953).

O mapa de Karnaugh é uma ferramenta gráfica para a simplificação de expressões Booleanas, de forma rápida e prática se comparado com a aplicação dos teoremas da álgebra de Boole. É constituído por células, cada uma das quais é representada de um mintermo ou maxtermo ou *dont'care states*. Os *don't care states* são variáveis para as quais algumas combinações de valores das entradas nunca ocorrem. Como por exemplo: a umidade do ar está acima de 30° graus e abaixo de 10° graus. Os *dont' care states* são representados nos mapas de Karnaugh com o símbolo X e pode assumir o valor 0 ou 1. O valor a ser assumido depende da possibilidade de obter se uma simplificação máxima da função Booleana.

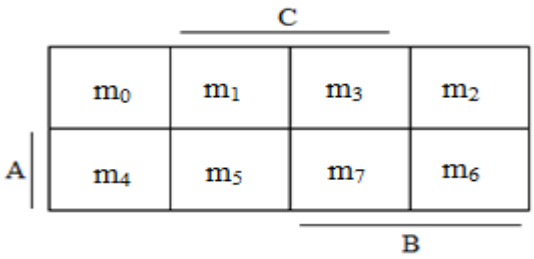
A simplificação de uma função usando o mapa de Karnaugh de uma função pode ser representada de duas formas: soma de produtos ou produto de somas. Na soma de produtos (soma de mintermos, onde a função F é igual a 1), as células correspondentes aos mintermos serão preenchidas com o valor 1 e as demais com o valor 0.

Um mapa de Karnaugh é uma matriz com 2^n células, em que *n* é o número de variáveis ou entradas do circuito, em que cada célula está associada a um mintermo ou maxtermo ou *don't care states* (DAGHLIAN, 1995). Para três variáveis, por exemplo, o mapa de Karnaugh é um conjunto de 8 células.

Para realizar a simplificação pelo mapa de Karnaugh, primeiramente cria-se uma tabela. Os valores da função que serão simplificados são preenchidos conforme a nova ordem. Posteriormente, identificam-se todos os grupos de mintermos-1 adjacentes entre si. Cada grupo origina um termo produto, em que somente as variáveis comuns a todos os mintermos-1 permanecem.

Os agrupamentos adjacentes são aplicados apenas na horizontal e na vertical, nunca na diagonal, conforme apresentado no Quadro 1. Observe que no Quadro 1, m_0 e m_5 não possuem adjacência, pois não se encontra nem na mesma linha e nem na mesma coluna.

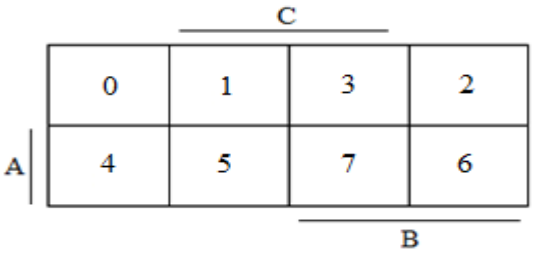
Quadro 1 - Simplificações possíveis entre os mintermos de uma função de 3 variáveis.



Fonte: Próprio autor.

Para um melhor entendimento sobre o emprego do mapa de Karnaugh, na minimização da função apresenta-se um exemplo explicando as etapas passo a passo. Na Figura 6 apresenta-se um exemplo de mapa de Karnaugh para 3 variáveis. O primeiro passo é criar uma tabela, e saber os valores decimais que representarão cada célula, para assim identificar qual célula será preenchido com o mintermo ou maxtermo associado.

Figura 6 - Mapa de Karnaugh para funções de 3 variáveis F(ABC).



Fonte: Próprio autor.

O mapa de Karnaugh representa a linha da tabela verdade modificada para se identificar os padrões de minimização ou seja, a propriedade $AB + A\overline{B} = A$.

Considere a função F(ABC) = $\sum m(0, 1, 2, 3, 6, 7)$, apresentada na Tabela 9.

Tabela 9- Representação da função F(ABC) = $\sum m(0, 1, 2, 3, 6, 7)$ na tabela verdade.

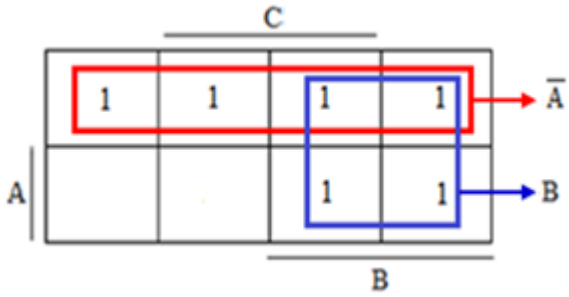
Decimal	**A**	**B**	**C**	**F**	
0	0	0	0	1	$\rightarrow \overline{A}\overline{B}\overline{C}$
1	0	0	1	1	$\rightarrow \overline{A}\overline{B}C$
2	0	1	0	1	$\rightarrow \overline{A}B\overline{C}$
3	0	1	1	1	$\rightarrow \overline{A}BC$
4	1	0	0	0	
5	1	0	1	0	
6	1	1	0	1	$\rightarrow AB\overline{C}$
7	1	1	1	1	$\rightarrow ABC$

Fonte: Próprio autor.

Após criar a tabela, como esta sendo usada a soma de produtos (soma de mintermos), cada linha da tabela correspondente a F = 1, corresponde a um mintermo. Identificado os mintermos na tabela, o próximo passo é colocar cada mintermo no mapa de Karnaugh, portanto é necessário identificar o valor decimal correspondente a cada mintermo.

Preenchido o mapa de Karnaugh, realizam-se os agrupamentos dos mintermos adjacentes, conforme apresentado na Figura 7. Foram obtidos dois agrupamentos: o primeiro agrupamento na horizontal, deu origem ao implicante $\overline{A}$ e o segundo agrupamento na vertical originou o implicante B, portanto a solução é F(ABC) = $\overline{A} + B$.

Figura 7 - Mapa de Karnaugh para F(ABC) =$\sum m(0, 1, 2, 3, 6, 7)$.



Fonte: Próprio autor

Analisando o mapa de Karnaugh, observa-se que o processo de simplificação é mais rápido e eficiente do que aquele realizado por álgebra de Boole. Independentemente dos processos a ser aplicados, ambos obtém os mesmos resultados. A única diferença está no mecanismo utilizado. A simplificação por álgebra de Boole consiste na aplicação de vários teoremas até se obter a simplificação, diferentemente do mapa de Karnaugh que é realizado pelo reconhecimento de padrões entre células adjacentes.

F(ABC) = $\overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} + ABC$ (41)

$\overline{A}\overline{B}(\overline{C} + C) + \overline{A}B(\overline{C} + C) + AB(\overline{C} + C)$ (Propriedade Distributiva) (42)

$\overline{A}\overline{B}(1) + \overline{A}B(1) + AB(1)$ (Identidade: $\overline{C} + C = 1$) (43)

$\overline{A}(\overline{B} + B) + B(\overline{A} + A)$ (Propriedade Distributiva) (44)

$\overline{A}(1) + B(1)$ (Identidade: $\overline{B} + B = 1$ e $\overline{A} + A = 1$) (45)

$\overline{A} + B$. (46)

Pode-se notar que em ambos os métodos utilizado obteve-se a mesma solução.

2.6.4 Método de Quine-McCluskey

O método clássico de Quine-McCluskey é também conhecido como método tabular e iterativo. É composto de duas fases: na primeira fase geram-se todos os implicantes primos e na segunda fase realiza-se uma cobertura mínima encontrando os implicantes primos que cobrem os mintermos da função (McCluskey, 1956).

O teorema fundamental do método de Quine-McCluskey é similar ao emprego no mapa de Karnaugh, em que os mintermos que diferem em apenas uma variável podem ser combinados pelo uso do teorema:

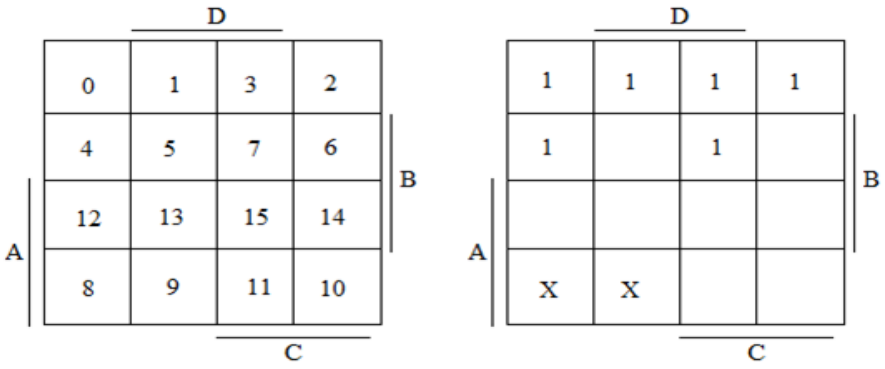
$$\mathbf{A}\mathbf{B} + \mathbf{A}\overline{\mathbf{B}} = \mathbf{A}. \tag{47}$$

No método de Quine-McCluskey a função além de conter mintermos, pode conter *don't care states* que são combinações de valores de entradas que nunca ocorrem. Na primeira fase do método, que se geram os implicantes primos, deve-se utilizar os *don't care states* quando existentes. Enquanto que na segunda fase que é realizada a cobertura, os *don't care states* não devem aparecer na tabela de cobertura, pois não necessitam ser cobertos por implicantes primos e incluídos na expressão mínima. Na expressão mínima, deve-se conter obrigatoriamente os implicantes primos essenciais e/ou implicantes primos com menor custo, ou seja, aqueles que cobrem um maior número de mintermos.

2.6.4.1 1ª Fase - Algoritmo para a geração de implicantes primos.

O método de Quine-McCluskey, é uma forma sistemática para gerar implicantes primos e cobrir os mintermos. Na Figura 8 apresenta-se um exemplo da função F(ABCD) = $\sum m(0, 1, 2, 3, 4, 7) + \sum d(8, 9)$, representada graficamente no mapa de Karnaugh.

Figura 8 - Mapa de Karnaugh com representação dos mintermos e *don't care states* da função F(ABCD) = $\sum m(0, 1, 2, 3, 4, 7) + \sum d(8, 9)$.



Fonte: Próprio autor.

Como pode-se observar na Figura 8, tem-se dois mapas de Karnaugh: no primeiro é apresentado os valores decimais correspondente a cada célula e no segundo as células preenchidas com os respectivos mintermos e *dont'care states*.

A primeira fase do método Quine-McCluskey, é composta dos seguintes passos:

1. Representar os mintermos e *don't care states* em uma tabela, na forma binária:

   Na Tabela 10, tem-se os mintermos ou *dont'care states* em cada linha, representados na forma binária. Exemplo: mintermo (1) na forma binária é 0001.

Tabela 10 - Representação binária dos mintermos e *dont'care states.*

Mintermo/ *dont'care states*
0000 (0)
0001 (1)
0010 (2)
0011 (3)
0100 (4)
0111 (7)
1000 (8)
1001 (9)

Fonte: Próprio autor.

2. Separar os mintermos e *don't care states* em caixas de acordo com o número de 1's da forma binária, esta operação define-se o grupo 0;

Na Tabela 11, é similar à Tabela 10, exceto na ordenação dos mintermos e *don't care states* em caixas apenas foi acrescentado a coluna caixa. Nesta etapa, gera-se o grupo 0 que é definido pela quantidade de X na representação binária. Verifica-se a quantidade de 1's na representação binária, que indicará a caixa que o mintermo pertence. Como por exemplo, o mintermo 2, representado na forma binária é 0010, como a quantidade de 1's é 1, portanto pertencerá a caixa 1.

Tabela 11 - Grupo 0.

Caixa	**Mintermo/** ***dont'care states***	**Marca**
0	0 0 0 0 (0)	*
1	0 0 0 1 (1)	*
1	0 0 1 0 (2)	*
1	0 1 0 0 (4)	*
1	1 0 0 0 (8)	*
2	0 0 1 1 (3)	*
2	1 0 0 1 (9)	*
3	0 1 1 1 (7)	*

Fonte: Próprio autor.

3. Comparar todos os elementos da caixa i com os elementos da caixa i+1. Marcar com *, os elementos que foram combinados. Dois elementos são combinados se diferem em apenas um elemento, um bit, conforme o teorema: $\mathbf{AB + A\overline{B} = A.}$ Na representação binária, a posição do bit que se difere é substituído por X, portanto gera-se um novo grupo.

Na Tabela 12 apresenta-se o grupo 1. Define-se grupo como o número de combinações que ocorreram, ou pela quantidade de X na representação binária. O resultado deste grupo, deve-se a combinação dos mintermos ou *dont'care states* do grupo 0, representado na Tabela 11.

Tabela 12 - Grupo 1.

Caixa	**Implicante**	**Marca**
0	0 0 0 X (0, 1)	*
0	0 0 X 0 (0, 2)	*
0	0 X 0 0 (0, 4)	*
0	X 0 0 0 (0, 8)	
1	0 0 X 1 (1, 3)	*
1	X 0 0 1 (1, 9)	*
1	0 0 1 X (2, 3)	*
1	1 0 0 X (8, 9)	*
2	0 X 1 1 (3, 7)	

Fonte: Próprio autor.

Como de fato, neste grupo tem-se o X, para os elementos serem combinados, X deve estar na mesma posição, e os elementos devem se diferenciar em apenas uma posição que é 0 ou 1. No entanto, 000X da caixa 0, só pode ser combinado com o único elemento da caixa 1 que é 001X. O resultado da combinação gera o implicante 00XX, conforme apresentado na Tabela 13. Neste exemplo, os únicos elementos que não foram combinados são: 0X00(0,4) e 0X11(3,7), portanto estes serão incluídos no resultado final.

Tabela 13 - Grupo 2.

Caixa	**Implicante**	**Marca**
0	0 0 X X (0, 1, 2, 3) X 0 0 X (0, 1, 8, 9)	

Fonte: Próprio autor.

3. Combinar grupos gerados da mesma forma que foi realizado no processo anterior, até que nenhuma combinação seja possível;

Na Tabela 13 apresenta-se o novo grupo gerado, a partir das combinações do grupo 1. Nesta tabela como há apenas uma única caixa, não tem como realizar comparação, portanto os elementos não são combinados e não são marcados.

4. Listar os implicantes primos, que são aqueles que não foram marcados por *.

Como na Tabela 12, os implicantes 0X00 e 0X11 não foram combinados, não foram marcados. O mesmo ocorre na Tabela 13, com os implicantes 00XX e X00X. Logo o conjunto de implicantes primos da função F(ABCD) = $\sum m(0, 1, 2, 3, 4, 7) + \sum d(8, 9)$, são:

$$0X00 + 0X11 + 00XX + X00X = \overline{A}\overline{C}\overline{D} + \overline{A}CD + \overline{A}\overline{B} + \overline{B}\overline{C}. \quad (48)$$

2.6.4.2 2ª Fase –Cobertura dos mintermos

Após obter os implicantes primos da função Booleana na primeira fase, a segunda fase do método Quine-McCluskey consiste em encontrar a cobertura dos mintermos, ou seja, o número mínimo de implicantes primos que cobrem os mintermos. Nesta fase, os *dont'care states* não entram na tabela, pois não precisam ser cobertos pelos implicantes. A segunda fase consiste dos seguintes passos:

1. Cria-se uma tabela em que as linhas representam os implicantes primos obtidos e as colunas os mintermos. Se existir *dont'care states*, não devem ser incluído na tabela.

   Na Tabela 14 apresentam-se os implicantes primos da função F(ABCD) = ∑*m*(0, 1, 2, 3, 4, 7) + ∑*d*(8,9), formada a partir dos implicantes primos obtidos do grupo 1 e do grupo 2, da primeira fase do método.

Tabela 14 - Cobertura.

Implicantes primos	**Mintermos**					
	0	**1**	**2**	**3**	**4**	**7**
0 X 0 0 (0, 4)						
0 X 1 1 (3, 7)						
0 0 X X (0, 1, 2, 3)						
X 0 0 X (0, 1, 8, 9)						

Fonte: Próprio autor.

2. Marcar as colunas com X, quando o implicante primo da linha cobre o mintermo da coluna.

   Na Tabela 15 apresenta-se a tabela de cobertura dos mintermos.

Tabela 15 - Tabela de implicantes primos e respectivas marcas.

Implicantes primos	**Mintermos**					
	0	**1**	**2**	**3**	**4**	**7**
0 X 0 0 (0, 4)	X				X	
0 X 1 1 (3, 7)				X		X
0 0 X X (0, 1, 2, 3)	X	X	X	X		
X 0 0 X (0, 1, 8, 9)	X	X				

Fonte: Próprio autor.

Nota-se que o implicante 00XX cobre os mintermos, 0, 1, 2 e 3. Marcam-se, dessa forma, com um X, as colunas destes mintermos.

3. Identificar na tabela as colunas que contém apenas uma marca X. A linha na qual estas colunas contém a marca X corresponde a um implicante primo essencial.

Na Tabela 16, a linha e a coluna correspondente ao implicante primo essencial está representada em negrito, logo os implicantes primos essenciais são: 0X00, pois o mintermo 4 tem-se apenas uma marca X e o 0X11, devido o mintermo 7 apresentar apenas uma marca X. Eliminam-se as linhas dos implicantes primos essenciais e as colunas dos mintermos cobertos por respectivos implicantes.

Tabela 16 - Tabela de implicantes primos e implicantes primos essenciais.

Implicantes primos	**Mintermos**					
	0	**1**	**2**	**3**	**4**	**7**
0 X 0 0 (0, 4)	X				**X**	
0 X 1 1 (3, 7)				X		**X**
0 0 X X (0, 1, 2, 3)	X	X	**X**	X		
X 0 0 X (0, 1, 8, 9)	X	X				

Fonte: Próprio autor.

Neste caso as linhas que contém os implicantes primos essenciais são: 0X00, 0X11, e 00XX os mintermos que são cobertos são: 0, 4, 3, 7, 0, 1, 2, 3. Portanto estas linhas e colunas são eliminadas, restando apenas o implicante primo X00X. Como pode-se observar todos os mintermos foram cobertos.

4. Encontrar o conjunto mínimo de implicantes primos, ou seja, o número mínimo de literais que cobrem as colunas remanescentes. Lembrando que na solução mínima obrigatoriamente devem aparecer os implicantes primos essenciais.

   Analisando a Tabela 16, não há nenhum mintermo a ser coberto, restou o implicante primo X00X, como não há mintermo para este implicante primo cobrir, portanto não faz parte do conjunto solução. No entanto o conjunto solução é composto obrigatoriamente pelos implicantes primos essenciais e os implicantes primos que apresentam o menor custo. Para este exemplo apenas os implicantes primos essenciais que devem apresentar no conjunto solução. Portanto tem-se como solução mínima a seguinte expressão:

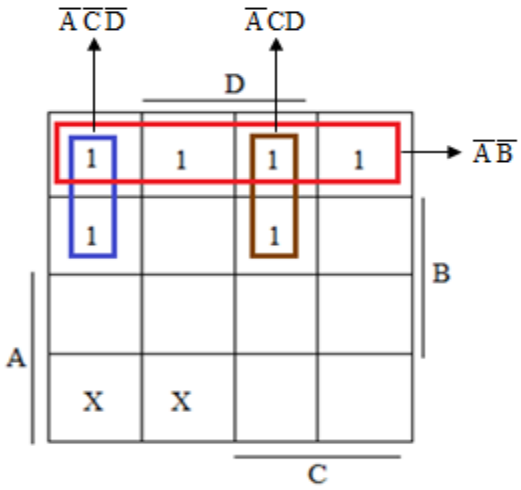
$$F(ABC) = 0X00 + 0X11 + 00XX = \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + \overline{A}\,\overline{B} \qquad (49)$$

Adotou-se como critério de custo a quantidade de entradas das portas AND e OR.

Nota-se que o custo inicial da função F(ABCD) = ∑*m*(0, 1, 2, 3, 4, 7) + ∑*d*(8, 9) é 30, que é a soma aritmética ponderada de todos os implicantes da função, lembrando que *don't care states* não são contados no custo. Após aplicar o método tabular de Quine-McCluskey, houve uma redução no custo para 11.

Na Figura 9 representa-se a segunda fase do método de Quine-McCluskey no mapa de Karnaugh.

Figura 9 - Representação da solução obtida pelo método Quine-McCluskey no mapa de Karnaugh.



Fonte: Próprio autor.

Neste capítulo são apresentados conceitos relacionados à álgebra booleana, operadores, teoremas e propriedades, enfatizando o operador XOR. E também as diferentes formas de simplificações de expressões booleanas utilizando teoremas da álgebra de Boole ou mapa de Karnaugh. Apresenta-se também neste capítulo o conceito teórico e o algoritmo do método clássico que gera implicantes primos denominado Quine-McCluskey, que será o método base para a comparação com o método proposto, que é descrito no Capítulo 3.

O método descrito no Capítulo 3, denominado de Quine-McCluskey Estendido tem um algoritmo semelhante ao método apresentado nesta seção. Entretanto, na primeira fase utilizam-se alguns conceitos que envolvem a operação XOR. A segunda fase, a fase de cobertura, é semelhante ao método original.

Cabe observar, conforme já descrito anteriormente, que a abordagem dada à fase de cobertura dos mintermos é o emprego da programação linear inteira 0 e 1, ou seja, o problema de cobertura dos mintermos é formulado como um problema de programação matemática.

3 MÉTODO DE QUINE-MCCLUSKEY ESTENDIDO

O método Quine-McCluskey Estendido é baseado no algoritmo empregado no método de Quine-McCluskey original. Entretanto, sempre que possível, as portas XOR e XNOR são utilizadas no processo de minimização juntamente com as portas OR e AND. A utilização de portas lógicas XOR e XNOR possibilita redução no custo, na maioria das vezes.

No algoritmo do método Quine-McCluskey, os mintermos e *dont'care states* são representados na forma binária, e as portas lógicas utilizadas são as portas AND e OR. Entretanto, no Quine-McCluskey Estendido cada dígito binário é representado na forma de potência, em que 1, é representado pela potência não barrada e o 0 pela potência barrada. Como exemplo considere o mintermo 5 = 0101 (representado na forma binária convencional) e mintermo 5 = $\bar{8}.4.\bar{2}.1$ (representado na forma de potência). A representação na forma de potência é utilizada apenas para operador XOR, caso contrário utiliza-se a forma convencional.

Como o objetivo do método é minimizar funções Booleanas, ou seja, reduzir o custo do circuito é necessário definir algum critério de custo. De acordo com o apresentado por Silva (1989) alguns critérios de custo são apresentados a seguir:

- Menor quantidade de variáveis na forma complementada ou não;
- Menor quantidade de variáveis na forma soma de produtos ou produto de somas;
- Menor quantidade de termos em uma expressão.

Neste trabalho adotou-se como critério de custo a menor quantidade de variáveis na forma soma de produto ou produto de somas.

3.1 DEFINIÇÕES

Para uma maior compreensão sobre o método é necessário entender algumas propriedades da álgebra Booleana dos operadores XOR e XNOR, que são os operadores a serem utilizados nas combinações entre os grupos. As propriedades utilizadas no método são descritas a seguir:

Propriedade 1- XOR: $X \oplus Y = \overline{X}Y + X\overline{Y}$. (50)

Propriedade 2- XNOR: $\overline{X \oplus Y} = XY + \overline{X}\overline{Y}$. (51)

Exemplo de aplicação da Propriedade 1, XOR: esta notação ocorre para qualquer par de mintermos que contenham as mesmas potências, em que um dos pares é complemento do outro (TURTON, 1996):

Mintermo 29: $16.8.4.\overline{2}.1$

Mintermo 30: $16.8.4.2.\overline{1}$

Resultado = $16.8.4.3.3$

Destaca-se no resultado, este par como sendo $\overline{2}.1 + 2.\overline{1} =$ (50). Como estas potências estão representadas na penúltima e última posição, então no resultado, estas posições serão substituídas pela soma da potência, 2 + 1 = 3, enquanto que nas posições que corresponde às potências iguais, primeira, segunda e terceira posição, as representações são conservadas.

Exemplo de aplicação da Propriedade 2, XNOR: utiliza-se quando há dois pares de mintermos, em que um par é complemento do outro par (TURTON, 1996):

Mintermo 20: $16.\overline{8}.4.\overline{2}.\overline{1}$

Mintermo 23: $16.\overline{8}.4.2.1$

Resultado: $16.\overline{8}.4.\overline{3}.\overline{3}$

Destaca-se no resultado, este par como sendo $\overline{2}.\overline{1} + 2.1 =$ (51), como estas potências estão representadas na penúltima e última posição, então no resultado, estas posições serão substituídas pela soma da potência, 2 + 1 = 3, e como o operador utilizado é XNOR, a potência deve ser barrada. Com relação às posições que corresponde as potências iguais: primeira, segunda e terceira posição, as representações são conservadas.

Estas propriedades são utilizadas na aplicação do método Quine-McCluskey Estendido. Além destas propriedades, tem-se a regra estabelecida em que as caixas devem conter o mesmo número de variáveis não complementadas, *dont´care states* e *exclusive-or*.

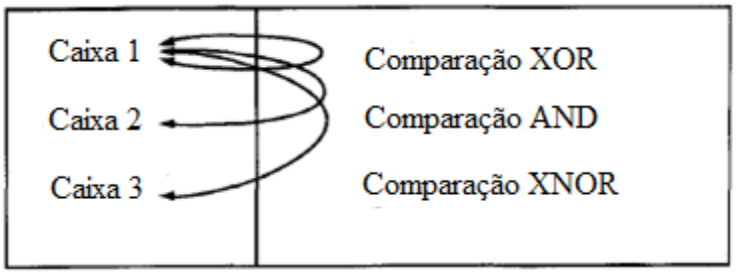
Há duas formas de realizar as combinações. Na primeira utiliza-se da quantidade de potências não barradas de um mintermo em relação a outro mintermo, conforme se apresenta na Tabela 17. Na Figura 10 apresenta-se a segunda forma para realizar as combinações, que consiste em utilizar a definição de caixas.

Tabela 17 - Comparação entre caixas.

Diferença	Operações
0	XOR
1	AND
2	XNOR

Fonte: Próprio autor.

Figura 10 - Tipos de operadores utilizados na comparação entre caixas.



Fonte: Adaptado de (TURTON, 1996).

Nota-se que as comparações são realizadas na própria caixa e com as duas caixas subsequentes, em que estas caixas devem se diferenciar em 1 ou 2 variáveis não complementadas.

O método apresenta duas características. A primeira, por ser similar ao Quine-McCluskey produz uma grande quantidade de comparações visando as combinações. Segundo BOLTON (1990), o problema é NP-difícil. Enquanto Brayton et al. (1984), afirma que pode se ter implicantes primos,em que *n* é o número de variáveis. A segunda característica é que por utilizar operador XOR pode ser implementado para múltiplos níveis, sendo 2 entradas e uma porta lógica XOR, ao invés de múltiplas entradas e uma porta AND (TURTON, 1996).

3.1.1 Algoritmo de Quine-McCluskey Estendido

O algoritmo do método de Quine-McCluskey Estendido apresenta as seguintes etapas:

1. Representar os mintermos e *dont'care states* em uma coluna na forma de potência;

2. Separar os mintermos em caixas de acordo com a quantidade de 1´s (as potências não barradas indicam a quantidade de 1´s que possuí cada elemento), esta operação define-se o grupo 0;

3. Execute os seguintes critérios:

a) Comparar todos os elementos da própria caixa e com as duas caixas subsequentes, em que estes devem se diferenciar em 1 ou 2 variáveis não complementadas. Marcar com * os elementos combinados. Dois elementos são combinados se diferir em um ou dois bits. Substituir as posições que diferem pela soma das potências correspondentes as posições. Caso seja, apenas uma posição que difere substitua por X;

b) Ao executar o passo 3, analisar também os elementos que foram combinados e aplicar as operações, descritas na Tabela 18 lembrando que é a quantidade de potências não barradas de um termo para o outro que indicará o tipo de operação a ser realizada.

Tabela 18 - Tipos de operadores utilizados na combinação das caixas.

Diferença	**Operações**	**Caixa**
0	Exclusive-or	Caixa 1 - Caixa 1
1	And	Caixa 1 - Caixa 2
2	NOT Exclusive-or	Caixa 1 - Caixa 3

Fonte: Próprio autor.

4. Combinar caixas geradas da mesma forma que no processo anterior, até que nenhuma combinação seja possível;

5. Listar os implicantes primos, que são os não marcados por *, ordenar em caixas de acordo com a quantidade de 1's.

Como exemplo de aplicação considere a função Booleana $F_1(ABC) = \sum m(1, 3, 4, 6, 7)$. Através de passos descreve-se o exemplo a seguir:

1. Representar os mintermos e *dont'care states* em uma coluna na forma de potência;

Na Tabela 19 tem-se os mintermos ou *dont'care states* em cada linha, representados na forma de potência. Exemplo: mintermo (1) na forma de potência $\bar{4}.\bar{2}.1 = 2^2.2^1.2^0$.

Tabela 19 - Representação potência dos mintermos e *dont'care states* da função F_1.

Mintermos/*dont'care states*
$\bar{4}.\bar{2}.1(1)$
$\bar{4}.2.1(3)$
$4.\bar{2}.\bar{1}(4)$
$4.2.\bar{1}(6)$
$4.2.1(7)$

Fonte: Próprio autor.

2. Separar os mintermos em caixas de acordo com a quantidade de 1´s (as potências não barradas indicam a quantidade de 1´s que possuí cada elemento), esta operação define-se o grupo 0;

Na Tabela 20, que é similar a Tabela 19, apenas foi acrescentado a coluna caixa. Nesta etapa gera-se o grupo 0 que é definido pela quantidade de X na representação binária. A caixa é determinada pela quantidade de potências não barradas que o mintermo apresenta. Como por exemplo, o mintermo 1 representado na forma de potência é $\bar{4}.\bar{2}.1$, como a quantidade de potências não barradas é 1, portanto pertence a caixa 1.

Tabela 20 - Agrupar em caixas de acordo com o número de 1's da função F_1.

Grupo 0	
Caixa	**Mintermos/*dont'care states***
1	$\bar{4}.\bar{2}.1$ (1) $4.\bar{2}.\bar{1}$(4)
2	$\bar{4}.2.1$(3) $4.2.\bar{1}$(6)
3	$4.2.1$(7)

Fonte: Próprio autor.

3. Execute os seguintes critérios:

a) Comparar todos os elementos da própria caixa e com as duas caixas subsequentes, em que estes devem se diferenciar em 1 ou 2 variáveis não complementadas. Marcar com * os elementos combinados. Dois elementos são combinados se diferir em um ou dois bits. Substituir as posições que diferem pela soma das potências correspondentes as posições. Caso seja, apenas uma posição que difere substitua por X;

b) Ao executar o passo 3, analisar também os elementos que foram combinados e aplicar as operações, descritas na Tabela 21 lembrando que é a quantidade de potências não barradas de um termo para o outro que indicará o tipo de operação a ser realizada.

Tabela 21 - Tipos de operadores utilizados na combinação das caixas da função F_1.

Diferença	Operações	Caixa
0	Exclusive-or	Caixa 1- Caixa 1
1	And	Caixa 1- Caixa 2
2	NOT Exclusive-or	Caixa 1 - Caixa 3

Fonte: Próprio autor.

Na Tabela 22, que é similar a Tabela 20, apenas foi criado a coluna combinações. Aplica-se primeiramente o critério 3.a). Neste exemplo o mintermo 1 que pertence a caixa 1, só pode ser comparado com os elementos da própria caixa, e com as caixas subsequentes que diferenciam em 1 ou 2 variáveis não complementadas. Portanto as caixas subsequentes são as caixas 2 e 3.

Tabela 22 - Combinação entre as caixas da função F_1.

Grupo 1		
Caixa	**Implicante**	**Combinações**
1	$\bar{4}.\bar{2}.1$ (1)*	$5.\bar{2}.5$(1,4)
	$4.\bar{2}.\bar{1}$(4)*	$\bar{4}.X.1$(1,3)
2	$\bar{4}.2.1$(3)*	$\bar{6}.\bar{6}.1$(1,7)
	$4.2.\bar{1}$(6)*	$4.X.\bar{1}$(4,6)
3	4.2.1(7)*	$4.\bar{3}.\bar{3}$(4,7)
		5.2.5(3,6)
		X.2.1(3,7)
		4.2.X(6,7)

Fonte: Próprio autor.

Comparam-se os elementos da própria caixa: (mintermo 1: $\bar{4}.\bar{2}.1$) e (mintermo 4: $4.\bar{2}.\bar{1}$), nota-se que os mintermos diferem em 2 bits, pois na primeira e última posição dos mintermos 1 e 4, tem-se potência barrada e não barrada. Como diferenciam em 2 bits, podem ser combinados. Aplica-se a soma das potências, nas posições que diferem. A primeira e última 4 + 1= 5. Portanto o resultado é $5.\bar{2}.5$(1,4) a primeira e última posição são substituídas pela potência 5, e a segunda posição por apresentar potências iguais, são conservadas. Os números apresentados no parentes indica os mintermos que foram combinados.

Após este processo, aplica-se o critério 3.b), em que verifica-se o tipo de operador, que é determinado pela quantidade de potências não barradas das posições que diferem. No mintermo 1 na primeira e última posição tem-se apenas 1 potência não barrada. No mintermo 4, também 1, então aplica-se a diferença (1-1= 0). Como a diferença é 0, conforme apresenta-se a Tabela 21, o operador a ser utilizado é o XOR.

4. Combinar caixas geradas da mesma forma que foi realizado no processo anterior, até que nenhuma combinação seja possível.

Na Tabela 23 compara-se a caixa 1, com todos os elementos da própria caixa, e com as caixas subsequentes que diferem em 1 ou 2 variáveis não complementadas. A única caixa subsequente que há nesta tabela, é a caixa 2. Compara-se então primeiro elemento da caixa 1: $(\bar{4}.X.1)$ e o segundo elemento da mesma caixa $(\bar{6}.\bar{6}.1)$. Pode-se observar que os valores das potências são diferentes, logo não podem ser combinados.

Tabela 23 - Combinação das caixas geradas da função F_1.

Grupo 2		
Caixa	**Implicante**	**Combinações**
1	$5.\bar{2}.5(1,4)*$ $\bar{4}.X.1(1,3)*$ $\bar{6}.\bar{6}.1(1,7)$ $4.X.\bar{1}(4,6)*$ $4.\bar{3}.\bar{3}(4,7)$	$5.X.5(1,3,4,6)$
2	$5.2.5(3,6)*$ $X.2.1(3,7)$ $4.2.X(6,7)$	

Fonte: Próprio autor.

Realiza-se então a comparação com todos os elementos desta caixa. Ao comparar-se o primeiro elemento $(\bar{4}.X.1)$ com o terceiro $(4.X.\bar{1})$ da mesma caixa, nota-se que as posições das potências são iguais, diferenciando em dois bits. Portanto realiza-se a combinação. Como as posições que diferem é a primeira e a terceira, então estas posições serão substituídas pela soma das potências, (4 + 1 = 5), e as potências iguais são conservadas. Portanto o resultado é $5.X.5(1,3,4,6)$.

Analisando estes dois elementos, pode-se observar que a quantidade de variáveis não barradas de um mintermo para o outro é (1 – 1 = 0). Portanto o operador utilizado será XOR.

5. Listar os implicantes primos, que são os que não foram marcados por *, ordenar em caixas de acordo com a quantidade de 1's.

Como na Tabela 23, não houve combinação, portanto não foram marcados, logo o conjunto de implicantes primos da função $F_1(ABC)= \sum m(1, 3, 4, 6, 7)$, são:

$$\bar{6}.\bar{6}.1 + 4.\bar{3}.\bar{3} + X.2.1 + 4.2.X + 5.X.5 = \quad (52)$$
$$= (\overline{A \oplus B})C + A(\overline{B \oplus C}) + BC + AB + (A \oplus C).$$

A partir dos implicantes primos obtidos na primeira fase do método Quine-McCluskey Estendido, deve-se realizar a fase de cobertura, para obter uma solução mínima. Formula-se então a cobertura dos mintermos da função como sendo um problema de programação linear inteira 0 e 1.

Dessa forma, as variáveis são os implicantes primos gerados e as restrições são desigualdades do tipo "$\geq$", cujo lado esquerdo é a soma dos implicantes primos que cobrem cada mintermo (SILVA, 1993).

Considera-se a função $F_1(ABC) = \sum m(1, 3, 4, 6, 7)$ e os implicantes primos obtidos na primeira fase do método Quine-McCluskey Estendido: $(\overline{A \oplus B})C + A(\overline{B \oplus C}) + BC + AB + (A \oplus C)$. Para simplificar a apresentação da formulação do problema de cobertura como um problema programação linear inteira 0 e 1, atribui-se letras para representar cada um dos implicantes primos gerados na primeira fase do método de Quine-McCluskey Estendido. Dessa forma tem-se: $a = (\overline{A \oplus B})C$, $b = A(\overline{B \oplus C})$, $c = BC$, $d = AB$ e $e = (A \oplus C)$. O problema de programação linear formulado é descrito como:

Min 5a + 5b + 3c + 3d + 3e

Sujeito as restrições

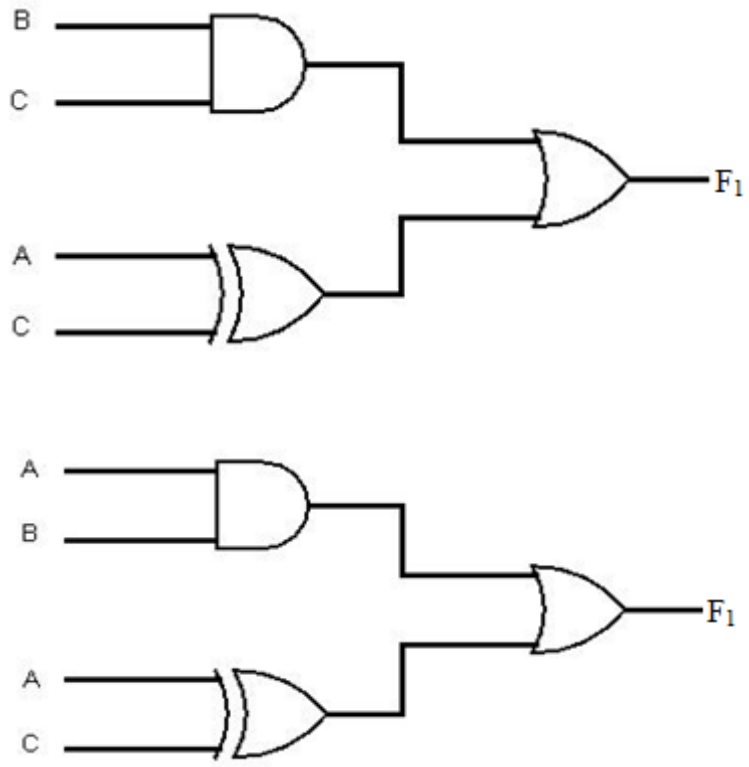
$a + e \geq 1$
$c + e \geq 1$
$b + e \geq 1$
$d + e \geq 1$
$a + b + c + d \geq 1$
$a, b, c, d, e \in \{0, 1\}$

Portanto o conjunto solução pode ser *c* = BC e *e* = $A \oplus C$ ou *d* = AB e *e* = $A \oplus C$ .Nota-se que a variável *e* deve estar contida nos dois conjuntos solução, pois é a única que cobre quase todos os mintermos, exceto o 7. Para cobrir o mintermo 7, tem-se as variáveis *a*, *b*, *c* e *d*. Como estas variáveis cobrem a mesma quantidade de mintermos e apresentam o mesmo custo, cada uma destas podem estar contidas na solução.

É importante destacar que o custo da função original $F_1(ABC) = \sum m\ (1, 3, 4, 6,7)$ era 20 e que após obter a solução mínima pelo método de Quine-McCluskey há uma redução do custo para 9. Entretanto, a solução mínima pelo método Quine-McCluskey Estendido, há uma redução do custo para 6.

Na Figura 11 apresenta-se a solução mínima obtida no método Quine-McCluskey Estendido, representado na forma de circuito. Têm-se dois conjuntos solução c = BC e e = $A \oplus C$ ou $d = AB$ e $e = A \oplus C$.

Figura 11 - Circuito referente à solução mínima do método Quine-McCluskey Estendido da função F_1.

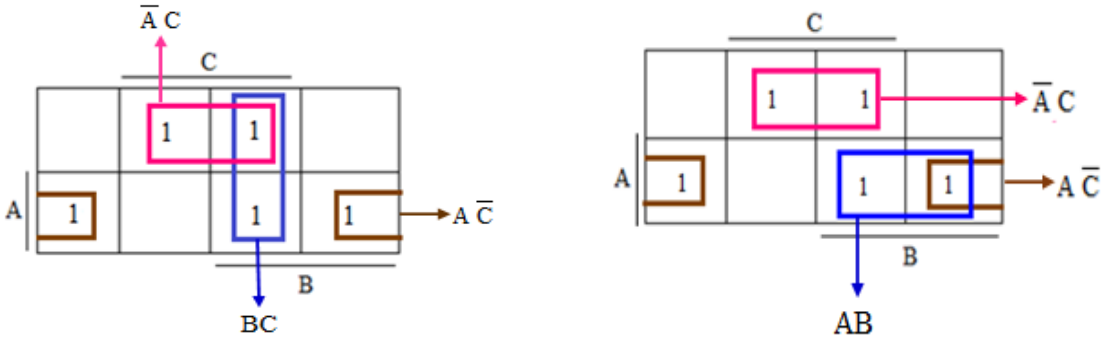


Fonte: Próprio autor.

Para verificar a diferença dos custos apresenta-se, na Figura 12, a solução mínima obtida no método de Quine-McCluskey representada no mapa de Karnaugh. Na Figura 13

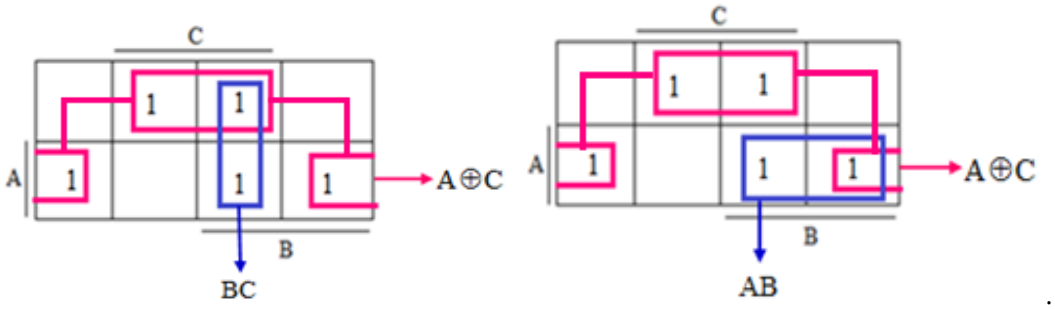
apresenta-se a solução mínima obtida no método de Quine-McCluskey Estendido também no mapa de Karnaugh.

Figura 12 - Mapa de Karnaugh da solução mínima obtida no método de Quine-McCluskey da função F_1.



Fonte: Próprio autor.

Figura 13 - Mapa de Karnaugh da solução mínima obtida no método de Quine-McCluskey Estendido da função F_1.



Fonte: Próprio autor.

Pode se notar que na Figura 12 os implicantes primos obtidos $\overline{A}C + A\overline{C}$, no método Quine-McCluskey, corresponde a um único agrupamento representado na Figura 13 pela expressão $A \oplus C$ em que os agrupamentos são representados na cor rosa.

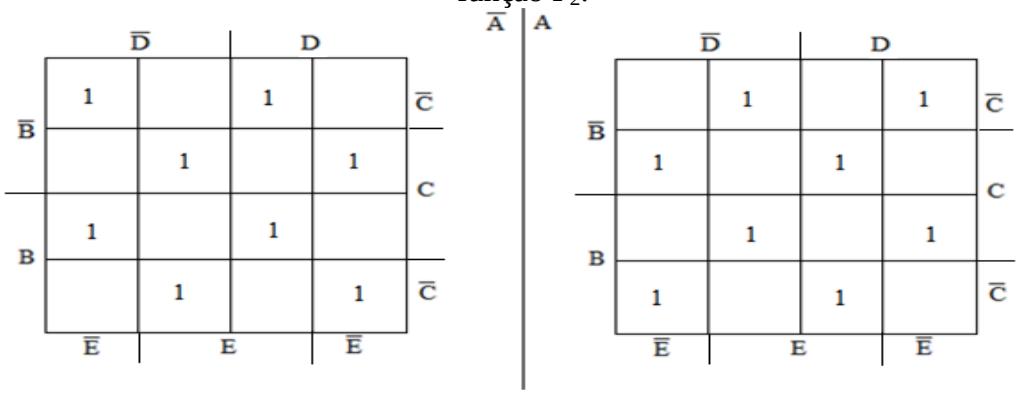
Na Figura 12, o custo obtido da solução mínima do método de Quine-McCluskey foi 9 e na Figura 13 o custo obtido da solução mínima do método Quine-McCluskey Estendido foi 6. Pode-se notar que houve uma redução do custo e que esta redução deve-se ao operador XOR. Portanto este é um dos casos apresentado, em que o custo apresentado pelo método Quine-McCluskey Estendido foi menor que o do método de Quine-McCluskey.

Além deste caso, apresenta-se uma classe de casos que se rotulou como o caso do xadrez. Para esta classe de exemplos o método implementado apresentou uma redução do custo surpreendente quando comparado com o método de Quine-McCluskey.

Na Figura 14 apresenta-se o caso do xadrez dado pela função $F_2(ABCDE) = \sum m\,(0, 3, 5, 6, 9, 10, 12, 15, 17, 18, 20, 23, 24, 27, 29, 30)$. Representando a solução mínima obtida pelo método de Quine-McCluskey tem-se como:

Resultado os termo produtos: $\overline{A}\overline{B}\overline{C}\overline{D}\overline{E} + \overline{A}\overline{B}\overline{C}DE + \overline{A}\overline{B}C\overline{D}\overline{E} + \overline{A}\overline{B}CD\overline{E} + \overline{A}B\overline{C}\overline{D}\overline{E} + \overline{A}B\overline{C}DE + \overline{A}BC\overline{D}\overline{E} + \overline{A}BCDE + A\overline{B}\overline{C}DE + A\overline{B}C\overline{D}\overline{E} + A\overline{B}CD\overline{E} + A\overline{B}CDE + AB\overline{C}\overline{D}\overline{E} + AB\overline{C}DE + ABC\overline{D}E + ABCD\overline{E}$ o custo obtido é de 96.

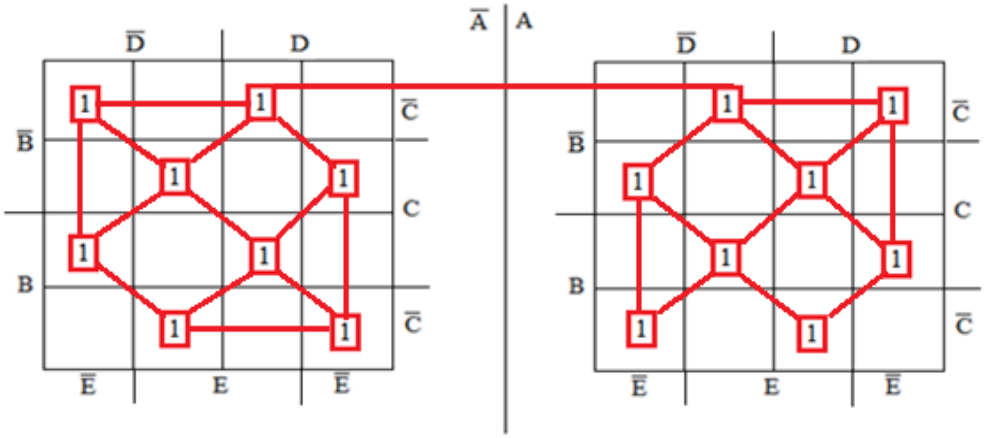
Figura 14 - Mapa de Karnaugh da solução mínima obtida no método de Quine-McCluskey da função F_2.



Fonte: Próprio autor.

Entretanto, aplicando-se o método de Quine-McCluskey Estendido a solução mínima obtida, é apresentada na Figura 15.

Figura 15 - Solução mínima obtida para o caso xadrez de cinco variáveis utilizando o método de Quine-McCluskey Estendido da função F_2.



Fonte: Próprio autor.

No entanto é importante destacar que o custo da função original $F_2(ABCDE) = \sum m$ (0, 3, 5, 6, 9, 10, 12, 15, 17, 18, 20, 23, 24, 27, 29, 30) era 96. Na Figura 14 após obter a solução mínima pelo método de Quine-McCluskey como não ocorreu nenhum agrupamento o custo continua sendo o mesmo da função original 96. E na Figura 15 representando a solução mínima pelo método Quine-McCluskey Estendido, houve uma combinação de mintermos, portanto o custo reduziu para 8. Dessa forma, pode-se observar que a redução do custo utilizando o método Quine-McCluskey Estendido foi significativa, comparada com o resultado obtido do Quine-McCluskey, representado na Figura 14. O resultado obtido da Figura 15 foi $\overline{A \oplus B \oplus C \oplus D \oplus E}$. Nota-se que utilizando o operador XNOR, pode-se representar todos os mintermos da função em um único agrupamento, permitindo uma redução no número de termos e consequentemente uma grande redução no custo.

O próximo exemplo a ser apresentado, mostra um caso em que os custos obtidos pelos dois métodos são iguais.

Considere a função $F_3(ABCD) = \sum m(1, 5, 7, 9)$. Aplicando-se o método Quine-McCluskey Estendido tem-se:

1. Representar os mintermos e *dont'care states* em uma coluna na forma de potência;

Na Tabela 24 apresentam-se os mintermos ou *dont'care states* em cada linha, representados na forma de potência. Como exemplo o mintermo (1) é representado na forma de potência $\bar{8}.\bar{4}.\bar{2}.1 = 2^3.2^2.2^1.2^0$.

Tabela 24 - Representação potência dos mintermos e *dont'care states* da função F_3.

Mintermos/*dont'care states*
$\bar{8}.\bar{4}.\bar{2}.1$ (1)
$\bar{8}.4.\bar{2}.1$ (5)
$\bar{8}.4.2.1$(7)
$8.\bar{4}.\bar{2}.1$ (9)

Fonte: Próprio autor.

2. Separar os mintermos em caixas de acordo com a quantidade de 1´s (as potências não barradas indicam a quantidade de 1´s que possuí cada elemento), esta operação define-se o grupo 0;

Na Tabela 25, que é similar a Tabela 24, apenas foi acrescentada a coluna caixa. Nesta etapa gera-se o grupo 0 que é definido pela quantidade de X na representação binária.

A caixa é determinada pela quantidade de potências não barradas, que o mintermo apresenta. Como, por exemplo, o mintermo 1 é representado na forma de potência é $\bar{8}.\bar{4}.\bar{2}.1$. Como a quantidade de potências não barradas é igual a 1, o mintermo pertence a caixa 1.

Tabela 25 - Agrupar em caixas de acordo com o número de 1's da função F_3.

Grupo 0	
Caixa	**Mintermos/** ***dont'care states***
1	$\bar{8}.\bar{4}.\bar{2}.1$ (1)
2	$\bar{8}.4.\bar{2}.1$(5) $8.\bar{4}.\bar{2}.1$(9)
3	$\bar{8}.4.2.1$ (7)

Fonte: Próprio autor.

3. Execute os seguintes critérios:
   a) Comparar todos os elementos da própria caixa e com as duas caixas subsequentes, em que estes devem se diferenciar em 1 ou 2 variáveis não complementadas. Marcar com * os elementos combinados. Dois elementos são combinados se diferir em um ou dois bits. Substituir as posições que diferem pela soma das potências correspondentes as posições. Caso seja, apenas uma posição que difere substitua por X;
   b) Ao executar o passo 3, analisar também os elementos que foram combinados e aplicar as operações, descritas na Tabela 26lembrando que é a quantidade de potências não complementadas de um termo para o outro que indicará o tipo de operação a ser realizada.

Tabela 26 - Tipos de operadores utilizados na combinação das caixas da função F_3.

Diferença	**Operações**	**Caixa**
0	XOR	Caixa 1- Caixa 1
1	AND	Caixa 1 -Caixa 2
2	XNOR	Caixa 1- Caixa 3

Fonte: Próprio autor.

Na Tabela 27, que é similar a Tabela 25, apenas foi criado a coluna combinações. Aplica-se primeiramente o critérios 3.a). Neste exemplo o mintermo 1, que pertence a caixa 1, só pode ser comparado com os elementos da própria caixa, e com as caixas subsequentes que

diferenciam em 1 ou 2 variáveis não complementadas. Portanto as caixas subsequentes são caixa 2 e 3.

Tabela 27 - Combinações entre as caixas da função F_3.

Grupo 1		
Caixa	**Representação Potência**	**Combinações**
1	$\bar{8}.\bar{4}.\bar{2}.1$ (1)*	$\bar{8}.X.\bar{2}.1$ (1,5)
2	$\bar{8}.4.\bar{2}.1$(5)* $8.\bar{4}.\bar{2}.1$(9)*	$X.\bar{4}.\bar{2}.1$(1,9) $\bar{8}.\bar{6}.\bar{6}.1$(1,7)
3	$\bar{8}.4.2.1$ (7)*	$12.12.\bar{2}.1$(5,9) $\bar{8}.4.X.1$(5,7)

Fonte: Próprio autor.

Como no grupo 1, não há nenhum elemento a ser comparado, compara-se a caixa 1 (mintermo 1: $\bar{8}.\bar{4}.\bar{2}.1$), com a caixa 2 (mintermo 5: $\bar{8}.4.\bar{2}.1$). Nota-se que os mintermos diferem em 1 bit, pois na segunda posição dos mintermos 1 e 5, tem-se potência barrada e não barrada. Como diferenciam em 1 bit, podem ser combinados. Substitui-se esta posição que difere pelo X. Portanto o resultado é $\bar{8}.X.\bar{2}.1(1,5)$ a segunda posição é substituída pelo X, e a primeira, terceira e quarta posição por apresentar potências iguais, são conservadas. Os números apresentados nos parentes indicam os mintermos que foram combinados.

Após este processo, aplica-se o critério 3.b), em que verifica-se o tipo de operador, que é determinado pela quantidade de potências não barradas das posições que diferem. No mintermo 1 na segunda posição tem-se 0 potência não barrada e mintermo 5, tem 1, então aplica-se a diferença (0 - 1 = - 1, considera o valor absoluto +1). Como a diferença é 1, conforme apresenta-se a Tabela 26, o operador a ser utilizado é o AND.

4. Combinar caixas geradas da mesma forma que no processo anterior, até que nenhuma combinação seja possível;

Na Tabela 28 apresenta-se a comparação da caixa 1, com todos os elementos da própria caixa, e com as caixas subsequentes que diferem em 1 ou 2 variáveis não complementadas. A única caixa subsequente que há nesta tabela, é a caixa 2. Compara-se então primeiro elemento da caixa 1: $(\bar{8}.X.\bar{2}.1)$ e o segundo elemento da mesma caixa $(X.\bar{4}.\bar{2}.1)$

. Pode-se observar que os valores das potências são diferentes, logo não podem ser combinados.

Tabela 28 - Combinação das caixas geradas da função F_3.

Caixa	Representação Potência	Combinações
1	$\bar{8}.X.\bar{2}.1\ (1,5)$ $X.\bar{4}.\bar{2}.1\ (1,9)$ $\bar{8}.\bar{6}.\bar{6}.1 (1,7)$	
2	$\bar{8}.4.X.1 (5,7)$ $12.12.\bar{2}.1 (5,9)$	

Fonte: Próprio autor.

5. Listar os implicantes primos, que são os não marcados por *, ordenar em caixas de acordo com a quantidade de 1's.

Como na Tabela 28 não houve combinação, portanto não foram marcados, logo o conjunto de implicantes primos da função $F_3(ABCD) = \sum m\ (1, 5, 7, 9)$, são:

$$\begin{aligned} &\bar{8}.X.\bar{2}.1 + X.\bar{4}.\bar{2}.1 + \bar{8}.\bar{6}.\bar{6}.1 + \bar{8}.4.X.1 + 12.12.\bar{2}.1 = \\ &= \bar{A}\bar{C}D + \bar{B}\bar{C}D + \bar{A}D(\overline{B \oplus C}) + \bar{A}BD + (A \oplus B)\bar{C}D. \end{aligned} \tag{53}$$

A partir dos implicantes primos obtidos no método Quine-McCluskey Estendido, deve-se realizar a fase de cobertura, para obter uma solução mínima. Conforme já mencionado a cobertura mínima é obtida através da solução de um problema de programação linear inteira 0 e 1. Dessa forma, as variáveis são os implicantes primos e as restrições são desigualdade do tipo "$\geq$", cujo lado esquerdo é a soma dos implicantes primos que cobrem cada mintermo.

Considera-se a função $F_3(ABCD) = \sum m(1, 5, 7, 9)$ e os implicantes primos obtidos no método Quine-McCluskey Estendido: $\bar{A}\bar{C}D + \bar{B}\bar{C}D + \bar{A}D(\overline{B \oplus C}) + \bar{A}BD + (A \oplus B)\bar{C}D.$ Sendo $a = \bar{A}\bar{C}D$, $b = \bar{B}\bar{C}D$, $c = \bar{A}D(\overline{B \oplus C})$, $d = \bar{A}BD$ e $e = (A \oplus B)\bar{C}D$, tem-se o problema de programação linear como:

Min 4a + 4b + 6c + 4d + 6e

Sujeito às restrições

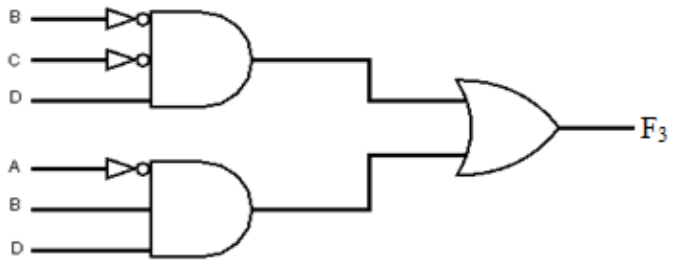
$$a + b + c \geq 1$$
$$a + d + e \geq 1$$
$$c + d \geq 1$$
$$b + e \geq 1$$
$$a, b, c, d, e \in \{0, 1\}$$

Portanto, o conjunto solução é *b* = $\overline{B}\overline{C}D$ *e d* = $\overline{A}BD$ nota-se que apenas com as variáveis *b* e *d* todos os mintermos são cobertos.

É importante destacar que o custo da função original $F_3(ABC) = \sum m\ (1, 5, 7, 9)$ era 16. Ao empregar o método de Quine-McCluskey tradicional obtém-se uma redução do custo para 8. A solução obtida pelo método de Quine-McCluskey Estendido, tem custo 8.

Na Figura 16 apresenta-se a solução mínima obtida no método Quine-McCluskey Estendido, representado na forma de circuito. Tem-se o conjunto solução b = $\overline{B}\overline{C}D$ e d = $\overline{A}BD$.

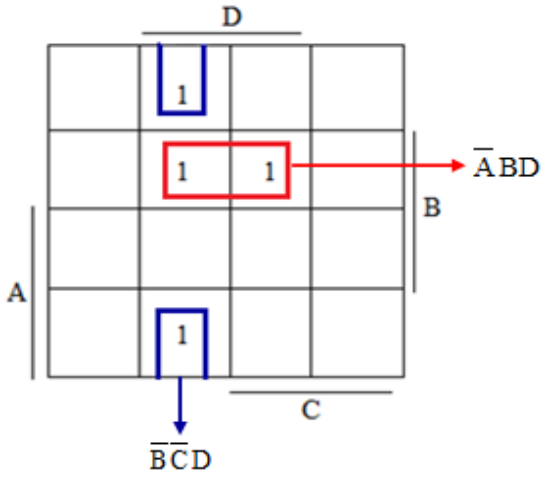
Figura 16 - Circuito referente à solução mínima no método Quine-McCluskey Estendido da função F_3.



Fonte: Próprio autor.

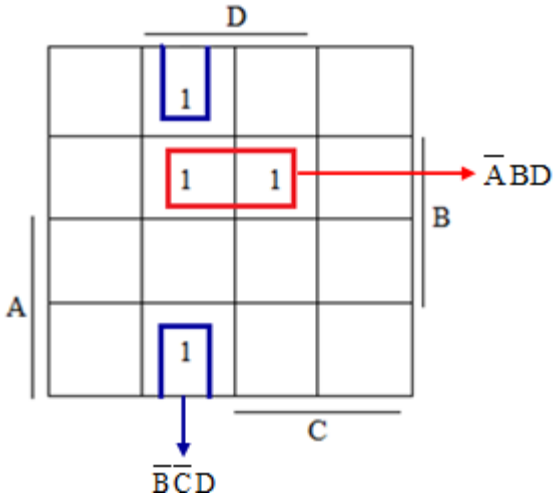
Para verificar a diferença dos custos, entre os dois métodos, apresenta-se na Figura 17a solução mínima obtida no método de Quine-McCluskey representada na forma de mapa de Karnaugh e na Figura 18 apresenta-se a solução mínima obtida no método Quine-McCluskey Estendido representada na forma de mapa de Karnaugh.

Figura 17 - Mapa de Karnaugh da solução mínima obtida pelo Quine-McCluskey da função F_3.



Fonte: Próprio autor.

Figura 18 - Mapa de Karnaugh da solução mínima obtida pelo Quine-McCluskey Estendido da função F_3.



Fonte: Próprio autor.

Pode-se observar que na Figura 17 e 18 os resultados obtidos foram o mesmo, o que implica com que os custos sejam iguais. Os custos foram iguais pois na solução obtida pelo método Quine-McCluskey Estendido não apresenta o operador XOR ou XNOR.

Além destes exemplos, também foram identificados exemplos em que o método Quine-McCluskey Estendido apresentou um custo maior comparado ao método de Quine-McCluskey.

Considere a função $F_4(ABCDE) = \sum m\ (11, 12, 20, 22)$. Aplicando-se o método Quine-McCluskey Estendido tem-se:

1. Representar os mintermos e *dont'care states* em uma coluna na forma de potência;

Na Tabela 29 apresentam-se os mintermos ou *dont'care states* em cada linha, representados na forma de potência. Exemplo: mintermo (11) na forma de potência $\overline{16}.8.\overline{4}.2.1 = 2^4.2^3.2^2.2^1.2^0$.

Tabela 29 - Representação potência dos mintermos e *dont'care states* da função F_4.

Mintermos/*dont'care states*
$\overline{16}.8.\overline{4}.2.1\,(11)$
$\overline{16}.8.4.\overline{2}.\overline{1}\,(12)$
$16.\overline{8}.4.\overline{2}.\overline{1}\,(20)$
$16.\overline{8}.4.2.\overline{1}\,(22)$

Fonte: Próprio autor.

2.Separar os mintermos em caixas de acordo com a quantidade de 1´s (as potências não barradas indicam a quantidade de 1´s que possuí cada elemento).

Na Tabela 30, que é similar a Tabela 29, apenas foi acrescentada a coluna caixa. Nesta etapa gera-se o grupo 0 que é definido pela quantidade de X na representação binária. A caixa é determinado pela quantidade de potências não barradas, que o mintermo apresenta. Como, por exemplo, o mintermo 12 é representado na forma de potência é $\overline{16}.8.4.\overline{2}.\overline{1}$. Como a quantidade de potências não barradas é igual a 2, o mintermo pertence a caixa 2.

Tabela 30 - Agrupar em caixas de acordo com o número de 1's da função F_4.

Grupo 0	
Caixa	**Mintermos/ *dont'care states***
2	$\overline{16}.8.4.\overline{2}.\overline{1}\,(12)$ $16.\overline{8}.4.\overline{2}.\overline{1}\,(20)$
3	$\overline{16}.8.\overline{4}.2.1\,(11)$ $16.\overline{8}.4.2.\overline{1}\,(22)$

Fonte: Próprio autor.

3. Execute os seguintes critérios:

a) Comparar todos os elementos da própria caixa e com as duas caixas subsequentes, em que estes devem se diferenciar em 1 ou 2 variáveis não complementadas. Marcar com * os elementos combinados. Dois elementos são combinados se diferir em um ou dois bits.

Substituir as posições que diferem pela soma das potências correspondentes as posições. Caso seja, apenas uma posição que difere substitua por X;

b) Ao executar o passo 3, analisar também os elementos que foram combinados e aplicar as seguintes operações, descritas na Tabela 31 lembrando que é a quantidade de potências não complementadas de um termo para o outro que indicará o tipo de operação a ser realizada.

Tabela 31 - Tipos de operadores utilizados na combinação das caixas da função F_4.

Diferença	Operações	Caixa
0	XOR	Caixa 1- Caixa 1
1	AND	Caixa 1 - Caixa 2
2	XNOR	Caixa 1 –Caixa 3

Fonte: Próprio autor.

Na Tabela 32, que é similar a Tabela 30, apenas foi criada a coluna combinações. Aplica-se o critério 3.a). Neste exemplo o mintermo 12 que pertence a caixa 2, só pode ser comparado com os elementos da própria caixa, e com as caixas subseqüentes que diferenciam em 1 ou 2 variáveis não complementadas. Portanto a caixa subsequente é a caixa 3.

Tabela 32 - Combinações entre as caixas da função F_4.

Grupo 1		
Caixa	**Representação Potência**	**Combinações**
2	$\overline{16}.8.4.\bar{2}.\bar{1}$ (12)* $16.\bar{8}.4.\bar{2}.\bar{1}$ (20)*	$24.24.4.\bar{2}.\bar{1}$ (12, 20) $16.\bar{8}.4.X.\bar{1}$ (20, 22)
3	$\overline{16}.8.\bar{4}.2.1$ (11) $16.\bar{8}.4.2.\bar{1}$ (22)*	

Fonte: Próprio autor.

Comparam-se os elementos da própria caixa: (mintermo 12: $\overline{16}.8.4.\bar{2}.\bar{1}$) e (mintermo 20: $16.\bar{8}.4.\bar{2}.\bar{1}$). Nota-se que os mintermos diferem em apenas 2 bits, pois na primeira e segunda posição dos mintermos 12 e 20, tem-se potência barrada e não barrada. Como diferenciam em 2 bits, podem ser combinados. Aplica-se então a soma das potências, nas posições que diferem: a primeira e segunda (16 + 8 = 24). Portanto o resultado é $24.24.4.\bar{2}.\bar{1}$ $(12,20)$ a primeira e segunda posição são substituídas pela potência 24, e a terceira, quarta e quinta posição por apresentar potências iguais, são conservadas. Os números apresentados no parentes indica os mintermos que foram combinados.

Após este processo, aplica-se o critério 3.b), em que verifica-se o tipo de operador, que é determinado pela quantidade de potências não barradas das posições que diferem. No mintermo 12 na primeira e segunda posição tem-se apenas 1 potência não barrada. E no mintermo 20, também 1, então aplica-se a diferença (1-1= 0). Como a diferença é 0, conforme apresenta-se a Tabela 31, o operador a ser utilizado é o XOR.

4.Combinar caixas geradas da mesma forma que no processo anterior, até que nenhuma combinação seja possível;

Na Tabela 33 compara-se a caixa 2, com todos os elementos da própria caixa, e com as caixas subsequentes que diferem em 1 ou 2 variáveis não complementadas. Nesta tabela não há nenhuma caixa subseqüente. Compara-se então primeiro elemento da caixa 2: $(24.24.4.\bar{2}.\bar{1})$ e o segundo elemento da mesma caixa $(16\bar{8}.4.X.\bar{1})$. Pode-se observar que os valores das potências são diferentes, logo não podem ser combinados.

Tabela 33 - Combinação das caixas geradas da função F_4.

Grupo 2		
Caixa	**Representação Potência**	**Combinações**
2	$24.24.4.\bar{2}.\bar{1}$ (12, 20) $16\bar{8}.4.X.\bar{1}$ (20, 22)	

Fonte: Próprio autor.

5. Listar os implicantes primos, que são os não marcados por *, ordenar em caixas de acordo com a quantidade de 1's.

Como na Tabela 32, o implicante $\overline{16}.8.\bar{4}.2.1$ não foi combinado, portanto não foi marcado. O mesmo ocorre na Tabela 33, com os implicantes $24.24.4.\bar{2}.\bar{1}$ e $16\bar{8}.4.X.\bar{1}$. Logo o conjunto de implicantes primos da função $F_4(ABCDE) = \sum m\ (11, 12, 20, 22)$, são:

$$24.24.4.\bar{2}.\bar{1} + 16\bar{8}.4.X.\bar{1} + \overline{16}.8.\bar{4}.2.1 = (A \oplus B)C\overline{D}\overline{E} + A\overline{B}\overline{C}\overline{E} + \overline{A}B\overline{C}DE. \qquad (54)$$

A partir dos implicantes primos obtidos no método Quine-McCluskey Estendido, deve-se realizar a fase de cobertura, para obter uma solução mínima. Conforme já mencionado a cobertura mínima é obtida através da solução de um problema de programação linear inteira 0 e 1.

Dessa forma, as variáveis são os implicantes primos e as restrições são desigualdade do tipo "$\geq$", cujo lado esquerdo é a soma dos implicantes primos que cobrem cada mintermo.

Considera-se a função $F_4(ABCDE) = \sum m(11, 12, 20, 22)$ e os implicantes primos obtidos no método Quine-McCluskey Estendido: $(A \oplus B)C\overline{D}\overline{E} + A\overline{B}C\overline{E} + \overline{A}B\overline{C}DE$. Sendo $a = (A \oplus B)C\overline{D}\overline{E}$, $b = A\overline{B}C\overline{E}$ e $c = \overline{A}B\overline{C}DE$, tem-se o problema de programação linear como:

Min 7a + 5b + 6c

Sujeito às restrições

$a \geq 1$

$a + b \geq 1$

$c \geq 1$

$b \geq 1$

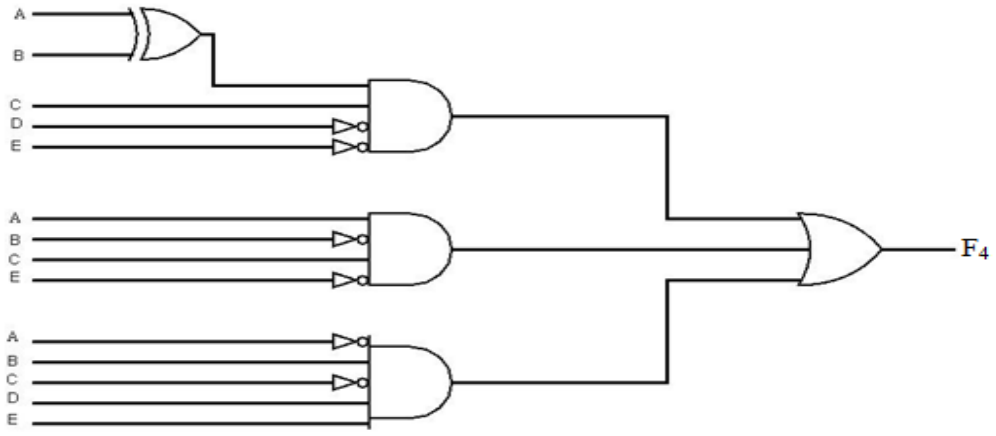
$a, b, c \in \{0, 1\}$

Portanto, o conjunto solução é $a = (A \oplus B)C\overline{D}\overline{E}$, $b = A\overline{B}C\overline{E}$ e $c = \overline{A}B\overline{C}DE$. Nota-se que com as variáveis *a*,*b* e *c* todos os mintermos são cobertos.

É importante destacar que o custo da função original $F_4(ABCDE) = \sum m(11, 12, 20, 22)$ era 28. Ao empregar o método tradicional obtém-se uma redução do custo para 17. A solução obtida pelo método de Quine-McCluskey Estendido, tem custo 18.

Na Figura 19 apresenta-se a solução mínima obtida no método Quine-McCluskey Estendido, representando na forma de circuito. Tem-se o conjunto solução $a = (A \oplus B)C\overline{D}\overline{E}$, $b = A\overline{B}C\overline{E}$ e $c = \overline{A}B\overline{C}DE$.

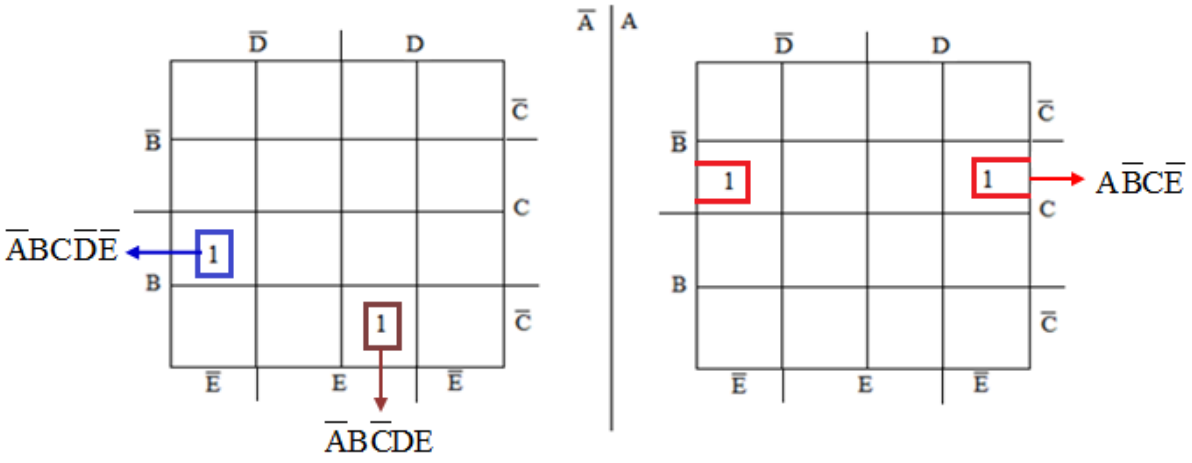
Figura 19 - Circuito referente à solução mínima no método Quine-McCluskey Estendido da função F_4.



Fonte: Próprio autor.

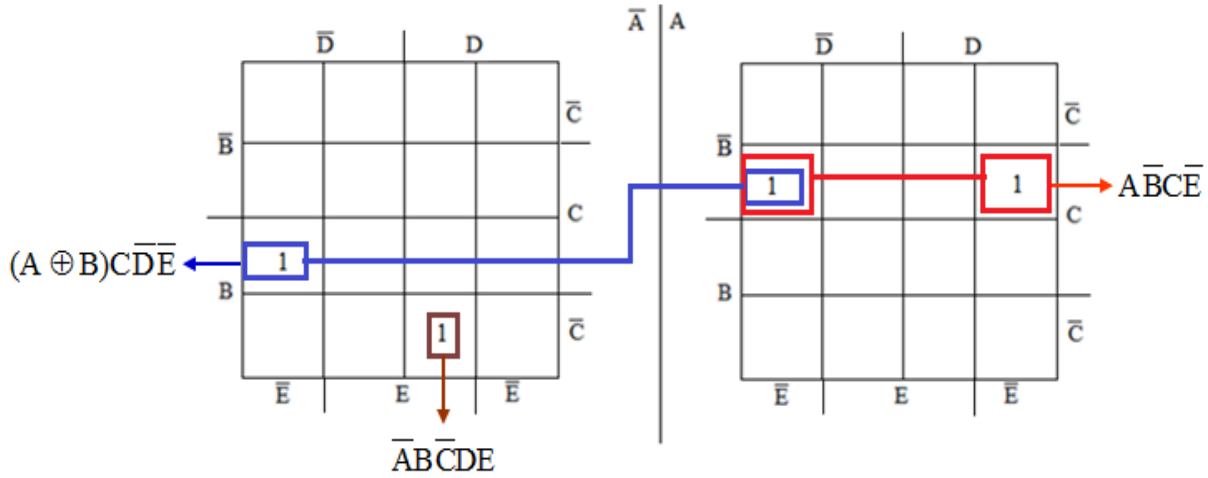
Para verificar a diferença dos custos, na Figura 20 apresenta-se a solução mínima obtida no método de Quine-McCluskey representada na forma de mapa de Karnaugh e na Figura 21 apresenta-se a solução mínima obtida no método Quine-McCluskey Estendido representada na forma de mapa de Karnaugh.

Figura 20 - Mapa de Karnaugh da solução mínima obtida pelo Quine-McCluskey da função F_4.



Fonte: Próprio autor.

Figura 21 - Mapa de Karnaugh da solução mínima obtida pelo Quine-McCluskey Estendido da função F_3.



Fonte: Próprio autor.

Pode-se notar que o custo obtido na Figura 21 foi 18, maior que o da Figura 20 que foi de 17, pois para o implicante primo obtido $\overline{A}BC\overline{D}\overline{E}$ da Figura 20, gerando-se este mesmo termo na Figura 21, utilizando-se o operador XOR, $(A \oplus B)C\overline{D}\overline{E}$ o custo se tornou maior.

Portanto, para este exemplo, o custo apresentado do método Quine-McCluskey Estendido foi maior quando comparado ao Quine-McCluskey.

Neste capítulo apresentou-se as características e conceitos teóricos sobre o método Quine-McCluskey Estendido que é um método que gera implicantes primos utilizando-se operadores XOR/XNOR. E para um melhor entendimento sobre o método foi descrito quatro exemplos que mostraram que em alguns casos o método de Quine-McCluskey Estendido obteve um custo menor, quando comparado com o método tradicional, devido à utilização do operador XOR e XNOR. No caso em que não foi possível aglutinar os mintermos em estrutura XOR, apresentou o mesmo custo. Entretanto, houveram caso em que o método Quine-McCluskey Estendido apresentou um custo maior quando comparado com o método tradicional.

Conforme mencionado, o algoritmo utilizado pelo método de Quine-McCluskey Estendido foi programado em linguagem C para que pudesse avaliar o desempenho computacional em termos de uso de memória e tempo de execução, além do custo do resultado obtido.

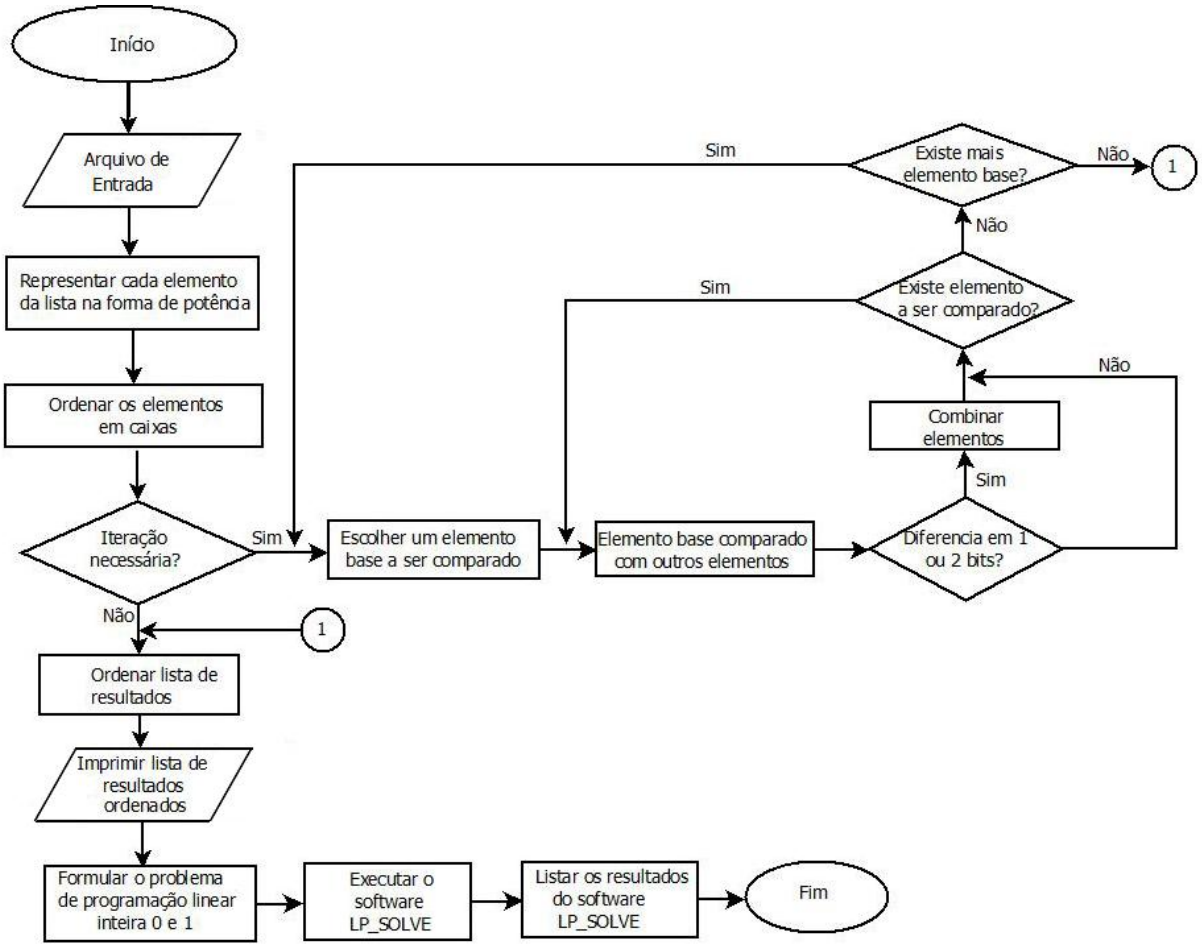
No capítulo 4 é apresentado o fluxograma do algoritmo implementado, a interface do programa com o usuário e a metodologia que foi utilizada para a realização dos testes de comparação entre o método original e o estendido.

4 DESENVOLVIMENTO E INTERFACE DO SOFTWARE

Neste capítulo é apresentado um fluxograma, que mostra passo a passo a implementação do software. A metodologia de testes utilizada, que são os arquivos de entrada e saída para que o usuário possa interagir com o software. E também como é realizado o cálculo do consumo de memória nos métodos Quine-McCluskey Estendido e Quine-McCluskey.

O programa que implementa o método Quine-McCluskey Estendido foi desenvolvido em linguagem de programação em C, utilizando-se lista simplesmente encadeada. Para um maior entendimento da implementação do algoritmo, apresenta-se na Figura 22 o fluxograma:

Figura 22 - Fluxograma do algoritmo do método de Quine-McCluskey Estendido.



Fonte:Próprio autor.

O primeiro passo do algoritmo é ler o arquivo de entrada, que descreve a função booleana a ser minimizada. Este arquivo contém os caracteres F que indica a quantidade de variáveis, o M para identificar os mintermos e o D para identificar os *dont'care states*.

Após realizar a leitura, transformam-se os elementos que são os mintermos ou *dont'care states* em forma de potência, na base 2^F.

A próxima etapa é ordenar os mintermos ou *dont'care states* em caixas. As caixas são calculados pela quantidade de potências não barradas. No programa as potências não barradas são representadas por * e as potências barradas por !. No entanto, a quantidade de * indica a caixa que o mintermo pertence. Após este processo realiza-se a ordenação das caixas para facilitar o processo de comparação. Como as caixas são ordenadas na primeira iteração, nas iterações seguintes permanecerão ordenadas.

Com as caixas ordenadas, realizam-se as comparações, logo verifica se existe uma nova iteração. Esta verificação é realizada através da variável x. Como a variável é inicializada na função *main*( ) com 1, o que força com que o laço *while* (x = 1) seja executado.

No laço *while* chama-se a função *comparação*, que apresenta-se um elemento base a ser comparado, e que este elemento será comparado com todos os elementos da própria caixa e com as duas próximas caixas. Nesta comparação é chamada a função *combina2elementos* que analisará cada um dos elementos que estão sendo comparados verificado se diferenciam em 0, 1 ou 2 bits. Caso atenda a condição, os elementos são combinados, caso contrário verifica-se há elementos da própria caixa ou das duas caixas subsequentes para serem comparados se não houver elemento a ser comparado, então verifica se existe elemento base caso exista repita o processo da comparação novamente, caso não exista, significa que não há mais elemento base e não há mais grupos a ser comparados.

Após este processo, verifica novamente se existe uma nova iteração. Caso x = 1 significa que existe, e novamente realiza-se o mesmo processo. Este procedimento é realizado até x = 0. Neste caso não há mais iteração, ou seja, não há mais elementos para serem combinados. Então ordena-se a lista *resultados*, lembrando que os resultados são compostos por elementos não marcados e mais resultados das combinações.

Nas combinações dos elementos, é analisado também o tipo de operador a ser utilizado, XOR, AND e XNOR. Utiliza-se o operador XOR quando a quantidade de potências não barradas das posições que diferem de um mintermo para o outro é 0, o operador AND quando a quantidade de potências não barradas é 1 e o operador XNOR quando a quantidade de potência não barradas é 2.

A lista dos resultados é ordenada e podem conter elementos não marcados. Estes elementos não marcados podem vir de iterações anteriores ou a atual, fazendo com que a lista fique desordenada, portanto é feita a ordenação.

Após este processo, imprime-se a lista dos resultados e formula-se o problema de cobertura como sendo um problema de programação linear inteira 0 e 1. Para que o problema de cobertura possa ser solucionado é necessário associar cada implicante primo gerado a uma variável. Dessa forma, pode-se definir a função objetivo que representa a soma ponderada das variáveis que representam os implicantes primos obtidos na primeira fase do método. Na linha abaixo da função objetivo tem-se a palavra-chave *subject to* ou *st,* e nas linhas seguintes as restrições, em que cada linha corresponde à cobertura de um mintermo. Para cada mintermo têm-se os implicantes primos que o cobre. Utilizou-se para obter a solução do problema o software LP_SOLVE (BERKELAAR; EIKLAND; NOTEBAERT, 2007).

O LP_SOLVE obtém a solução do problema de cobertura formulado como um problema de programação linear inteira 0 e 1 e retorna como resultado a solução do problema com valores binários. Realiza-se a leitura do arquivo para se obter as variáveis que obtiveram valores = 1, que representam os implicantes que pertencem a solução mínima. Realiza-se então o cálculo do custo total de todas as variáveis,do tempo e do consumo de memória, sendo estas informações acrescentadas no arquivo de saída do programa.

A metodologia empregada para a geração dos estudos de casos empregados nos testes, o formato do arquivo de entrada e do arquivo de saída estão descritos na seção 4.1.

4.1 METODOLOGIA EMPREGADA NOS TESTES

4.1.1 Arquivo de entrada e saída

Para a execução dos programas que implementam o algoritmo de Quine-McCluskey e de Quine-McCluskey Estendido, foi criado um arquivo de entrada padrão para cada programa, descrevendo a função Booleana a ser minimizada .

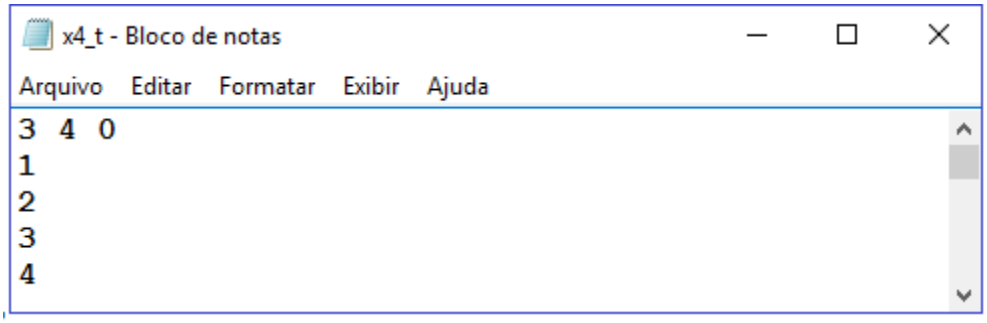
Na Figura 23 apresenta-se o arquivo de entrada para o programa Quine-McCluskey. Este arquivo deve apresentar as seguintes informações:

1. Na primeira linha deve conter 3 rótulos separados por um espaço, sendo o primeiro a quantidade de variáveis, o segundo a quantidade de mintermos e o terceiro a quantidade de *dont’care states*;

2. A segunda linha e nas linhas seguintes devem conter os mintermos da função, que são representados na forma decimal, e são calculados de acordo com a base 2^F, em que F é a quantidade de variáveis. Exemplo para F= 3, 2^3 = 8. Como o mintermo começa do 0, logo podem variar de 0 a 7;

3. Na linha seguinte, após o último mintermo são representados os *dont'care states*, que possui a mesma regra que os mintermos.

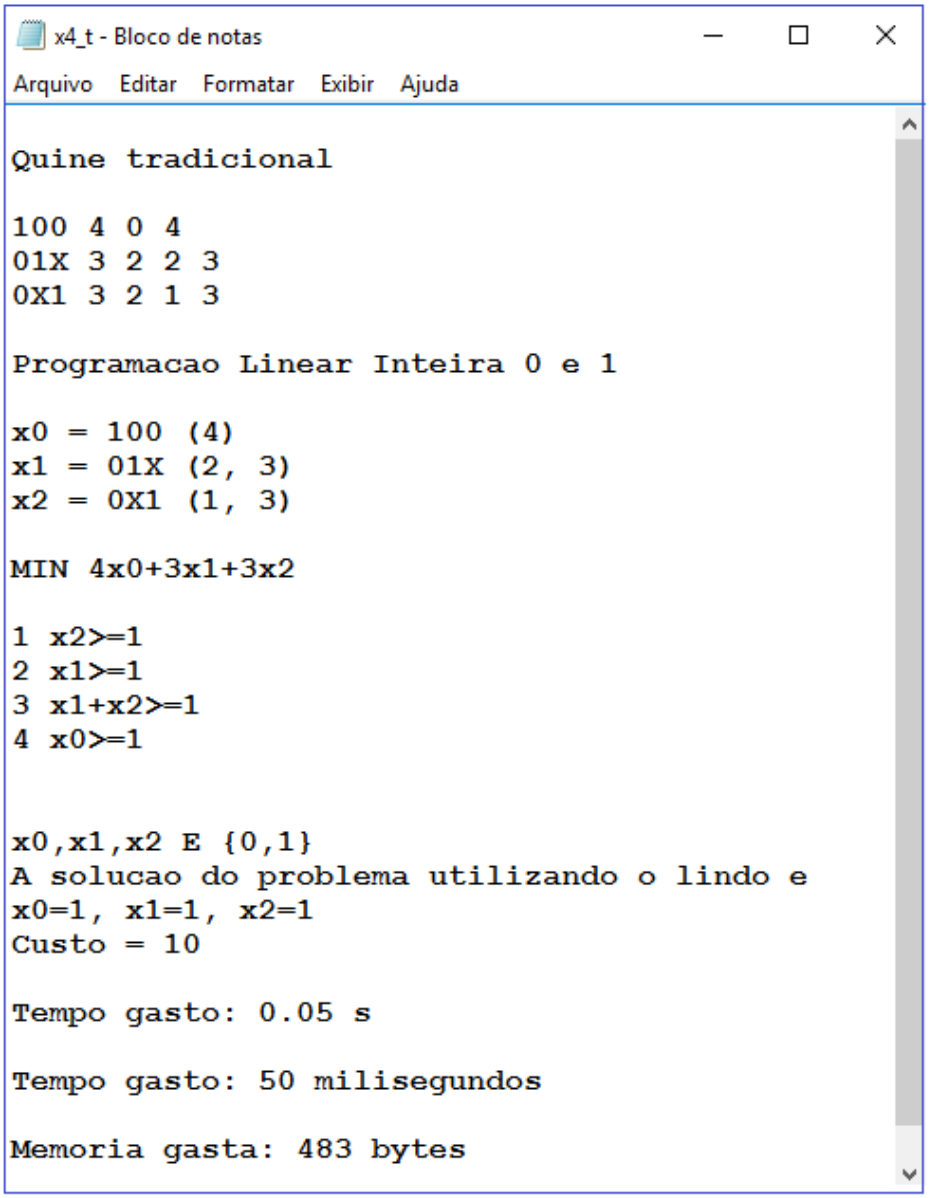
Figura 23 - Entrada para o programa Quine-McCluskey.



Fonte: Próprio autor.

Na Figura 24 apresentam-se os resultados gerados pelos programas. Na primeira linha tem-se o resultado do método de Quine-McCluskey tradicional. Na segunda linha uma linha em branco, na terceira e nas linhas seguintes os resultados que são 100 4 0 4, em que 100 é o implicante primo, 4 é o custo, 0 indica que esta sendo coberto pelo próprio mintermo o implicante primo, e o 4 é o mintermo que esta sendo coberto pelo implicante primo 100, 01X 3 2 2 3 e 0X1 3 2 1 3, também fazem parte dos resultados. A partir dos resultados obtidos, na segunda linha, após a frase da "Programação Linear Inteira 0 e 1", formula-se o problema de programação linear inteira 0 e 1.

Figura 24 - Saída para o programa Quine-McCluskey.



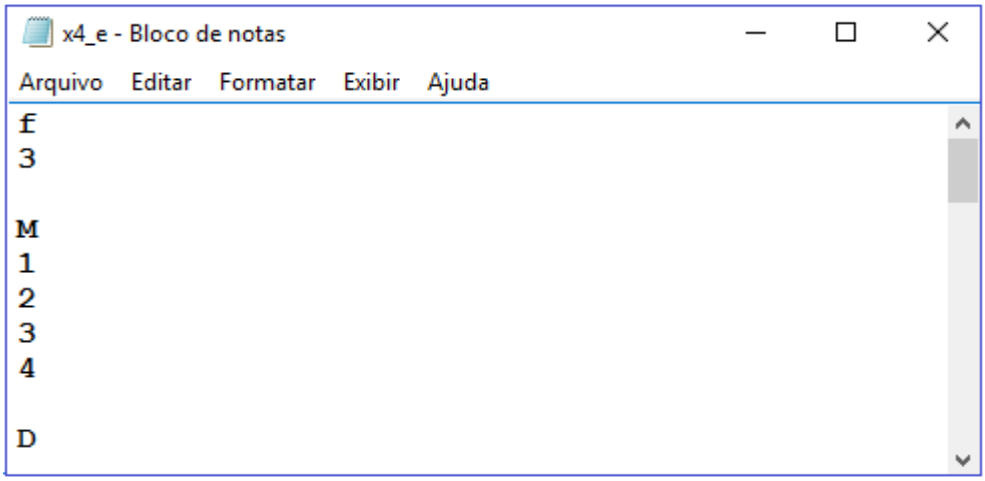
Fonte: Próprio autor.

Após este processo, faz o uso de um software comercial que resolve problema de programação linear inteira mista (MILP), conhecido como LP_SOLVE (BERKELAAR; EIKLAND; NOTEBAERT, 2007). O resultado obteve-se x0 = 1, x1 = 1 e x2 = 1, custo 10, que são impressos no arquivo, isto significa que x0, x1 e x2, correspondem aos seguintes implicantes primos x0 = 100, x1 = 01X e x2 = 0X1. Nota-se que o custo total de todos os implicantes primos é 10. Além destas informações, são impressas o tempo de processamento e o consumo de memória.

Para a execução do programa Quine-McCluskey Estendido, foi criado outro arquivo de entrada padrão, que também contém a descrição da função a ser minimizada (soma de mintermos). Na Figura 25 apresenta-se um exemplo do arquivo de entrada, que deve conter as seguintes instruções:

1. A primeira linha deve conter o caractere F que significa função;
2. Na segunda linha deve conter o dígito, que indica a quantidade de variáveis;
3. A terceira linha deve permanecer em branco;
4. A quarta linha deve conter o caractere M;
5. A quinta linha e as linhas seguintes contém os mintermos, que são representados na forma decimal;
6. A linha seguinte do último mintermo deve permanecer em branco;
7. A próxima linha deve conter o caractere D, que são os *dont'care states*;
8. Na linha abaixo do caractere D, deve conter os dígitos que são os *dont'care states*, caso haja, para saber o número máximo que os *dont'care states* podem ter, segue-se a regra estabelecida para os mintermos.

Figura 25 - Entrada para o programa Quine-McCluskey Estendido.

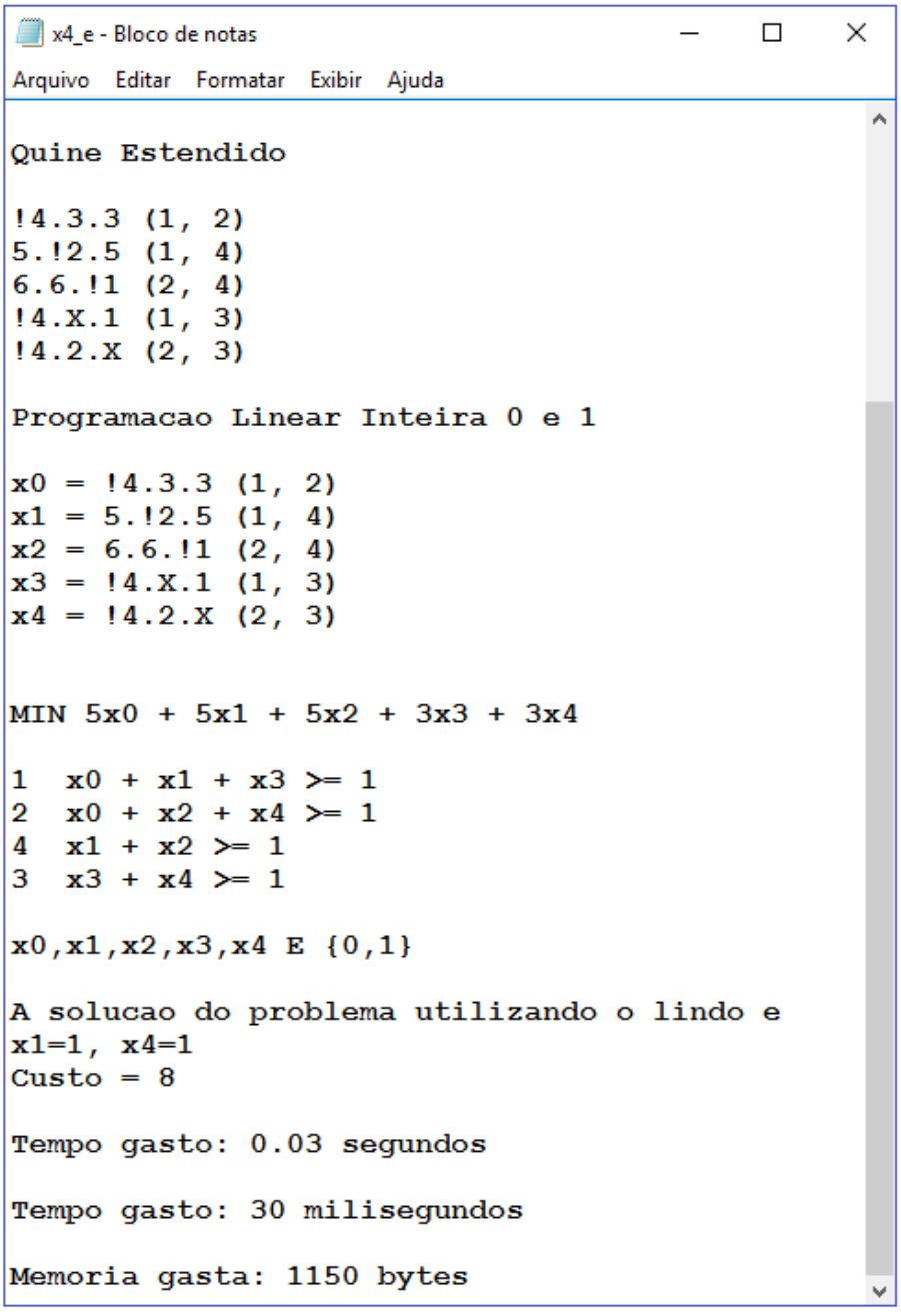


Fonte: Próprio autor.

Após executar o programa, é gerada a saída no mesmo arquivo de entrada. Na Figura 26 apresentam-se os resultados do método. Na primeira linha tem-se a solução do método Quine Estendido, na segunda uma linha em branco e na terceira, quarta, quinta, sexta e sétima os resultado que são !4.3.3 (1, 2), 5.!2.5 (1, 4), 6.6.!1 (2, 4), !4.X.1 (1, 3) e !4.2.X (2, 3). As

potências representadas antes dos parentes são os implicantes primos e os números contidos nos parentes, são os mintermos que foram cobertos pelos implicantes primos.

Figura 26 - Saída para o programa Quine-McCluskey Estendido.



Fonte: Próprio autor.

A partir dos resultados obtidos, formula-se também o problema de programação linear inteira 0 e 1, assim como foi realizado no arquivo de entrada do Quine-McCluskey. Chama-se então o LP_SOLVE que gera os seguintes resultados: x1 = 1 e x4 = 1, cujo custo é

8. Esta solução correspondem aos seguintes implicantes primos x1 = 5.!2.5 e x4 = !4.2.X. Além destas informações, são impressas o tempo de processamento e o consumo de memória.

Na Seção 4.2 descreve-se como é realizado o cálculo do consumo de memória nos métodos Quine-McCluskey Estendido e Quine-McCluskey.

4.2 CÁLCULO DO CONSUMO DE MEMÓRIA

4.2.1 Método de Quine-McCluskey Estendido

Um dos parâmetros a ser analisado nos métodos de Quine-McCluskey Estendido e de Quine-McCluskey é o consumo de memória. Nesta seção explica-se detalhadamente como é realizado o seu cálculo.

O primeiro passo é calcular a quantidade de bytes de cada registro da *struct* utilizada. Cada variável do tipo inteiro contém 4 bytes, e cada variável do tipo caractere contém 1 byte. As variáveis são calculadas pela quantidade de bytes de cada registro, e os ponteiros pela quantidade de bytes que aloca somado a quantidade de bytes do registro, multiplicado pela quantidade de elementos que o registro aloca. Lembrando que cada ponteiro aloca 8 bytes. A *struct* utilizada é descrita a seguir:

```
typedef struct no{
        int marca;
        int caixa;
        int qtdmintermo;
        int qtdmin;
        int caixa_avaliada;
        char verifica;
        int *potencia;
        int *mintermo;
        char *barrado;
        int *potenciamin;
        char *barradomin;
struct no *prox;
} elemento;
```

Variáveis:

int marca - 4 bytes

int caixa - 4 bytes

int qtdmintermo - 4 bytes

int qtdmin - 4 bytes

int caixa_avaliada - 4 bytes

char verifica - 1 byte

Total: 21 bytes

Ponteiros:

Quantidade de bytes que aloca somado a quantidade de bytes de cada registro multiplicado pelo tamanho de alocação para o ponteiro.

Como o ponteiro mintermo aloca a variável qtdmintermo e esta variável armazena mintermos e *dont'care states* que estão sendo cobertos pelo elemento naquele momento, e o que necessita ser calculado é a quantidade de mintermos e *dont'care states* a serem utilizados do início até o final do programa, então utiliza-se a variável cnt_inicio. Além desta variável, tem a variável qtd que indica quantidade de variáveis, que é alocada pelos ponteiros potencia, barrado, potenciamin e barradomin. Este cálculo é apresentado a seguir:

int *potencia; 8 + 4 x qtd

int *mintermo; 8 + 4 x qtdmintermo

char *barrado; 8 + 1 x qtd

int *potenciamin; 8 + 4 x qtd

char *barradomin; 8 + 1 x qtd

struct no *prox; 8

Total: 48 + 10 x qtd + 4 x cnt_inicio

Variáveis + Ponteiros:

21 + 48 + 14 x qtd

69+10 x qtd + 4 x cnt_inicio

Além de calcular a quantidade de bytes de cada estrutura, calcula-se o maior tamanho que a lista pode apresentar.Então cria-se uma variável contadora, que será incrementada a medida que um elemento da lista é criado. Realiza-se a comparação se a variável contadora >*max*, caso atenda a condição, *max = cont_total*, portanto a variável *max* armazenará sempre a maior quantidade de elementos que foram criados. Como este elemento

é liberado, insere-se uma variável que decrementa *cont_total*, a medida que o elemento da lista é liberado.

O cálculo do consumo de memória da estrutura do Quine-McCluskey Estendido é:

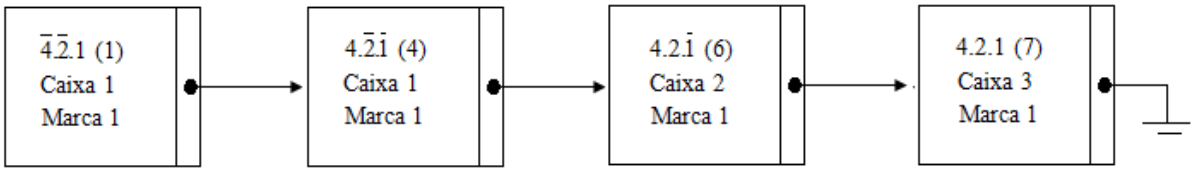
$$(69 + 10 \times qtd + 4 \times cnt\_inicio) \times (max) \quad (55)$$

Como exemplo considere uma função com 5 variáveis tem-se:

$$(69 + 10 \times 5 + 4 \times 32) \times (912) = 225.264 \text{ bytes}$$

Na Figura 27, apresenta-se um exemplo da lista simplesmente encadeada.

Figura 27- Lista simplesmente encadeada.



Fonte: Próprio autor.

Na Seção 4.2.2 descreve-se detalhadamente como é realizado o cálculo do consumo de memória do método Quine-McCluskey.

4.2.2 Método de Quine-McCluskey

A idéia empregada é semelhante ao cálculo do método Quine-McCluskey Estendido, em que cada variável do tipo inteiro tem 4 bytes, cada variável do tipo caractere tem 1 byte e cada ponteiro tem 8 bytes. Então o cálculo é: quantidade de bytes que cada variável aloca somado a quantidade de bytes do registro multiplicada a quantidade de elementos que aqueles registro aloca.

```
struct TERMOITEM{
    int caixa, decimal;
    char *elemento;
    char tipoElemento;
};
```

Variáveis:

int caixa, decimal; 8 bytes

char tipoElemento; 1 byte

Total: 9 bytes

Ponteiros:

Quantidade de bytes que aloca somado a quantidade de bytes de cada registro multiplicado ao tamanho de alocação para o ponteiro.

A variável *numbits* indica a quantidade de variáveis e o numElemento indica quantidade de mintermos e *dont'care states*.

char *elemento; 8bytes + *numbits* + 1

Total: 8bytes + *numbits* + 1

Variáveis + Ponteiros:

Fórmula: 9 + 8 + *numbits* + 1

18 + *numbits*

struct LTERMO{

char *min; 8 byte + *numbits* + 1

int marca; 4 bytes

int decimal; 4 bytes

int numC; 4 bytes

int *vetorElemCobertos; 8 + 4 x numElemento

char tipoElemento; 1 byte

struct LTERMO *prox; 8

};

Variáveis:

int marca; 4 bytes

int decimal; 4 bytes

int numC; 4 bytes

char tipoElemento; 1 byte

Total: 13 bytes

Ponteiros:

Quantidade de bytes que aloca somado a quantidade de bytes de cada registro multiplicado ao tamanho de alocação para o ponteiro.

char *min; 8 + *numbits* + 1

int *vetorElemCobertos; 8 + 4 x numElemento

struct LTERMO *prox; 8

Total: 25 + *numbits* + 4 xnumElemento

Variáveis + Ponteiros

25 + 13 + *numbits* + 4 x numElemento

38 + *numbits* + 4 x numElemento

Assim como o método Quine-McCluskey Estendido, insere-se uma variável contadora que será incrementada à medida que o elemento da lista é criado. Realiza-se a variável contadora >*max*, caso atenda a condição *max = cont_total*, portando a variável, *max* armazenará sempre a maior quantidade de elementos que foram criados.

Portanto a fórmula para calcular o consumo de memória da estrutura do Quine-McCluskey é:

((18 + numbits) x numElemento) + (38 + numbits + 4 x numElemento) x (max)) (56)

Exemplo: 5 variáveis : ((18 + 5) x 32) + (38 + 5 + 4 x 32) x (254)) =
=((23 x 32) + (43 + 4 x 32) x (254)) =
= (736 + (43 + 128) x (254)) =
= (736 + 171 x 254) = 44.170 bytes

No Capítulo 5 apresentam-se os resultados obtidos com os testes realizados. Pode-se observar o tempo de execução, o consumo de memória e realiza a comparação em relação aos custos obtidos nos dois métodos.

.

5 RESULTADOS OBTIDOS

5.1 COMPARAÇÕES DOS MÉTODOS QUINE-MCCLUSKEY ESTENDIDO E O MÉTODO DE QUINE-MCCLUSKEY

Com o objetivo de avaliar a eficiência dos métodos que geram implicantes primos foi realizada uma série de testes, de forma a comparar o método de Quine-McCluskey Estendido com o método de Quine-McCluskey. Os parâmetros analisados foram o custo, tempo de execução e consumo de memória.

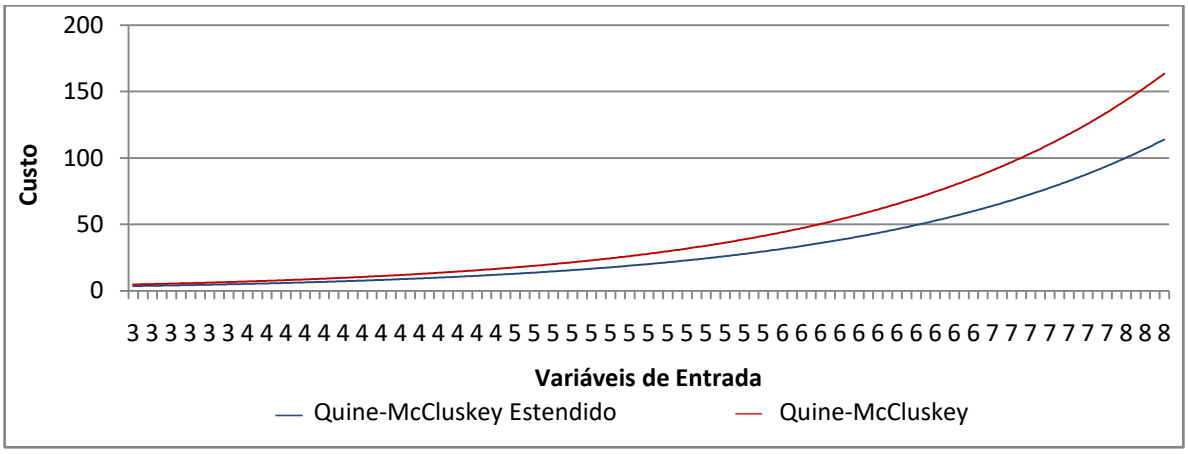
O algoritmo Quine-McCluskey Estendido foi implementado em C, e a estrutura de dados utilizada foi a lista simplesmente encadeada. Para o algoritmo Quine-McCluskey, desenvolvido por (FRANSCICANI, 2016), foi utilizado a mesma estrutura. O uso desta estrutura deve-se o fato de que a cada iteração o número de combinações variam. A lista simplesmente encadeada permite trabalhar com estruturas dinâmicas. Os programas foram executados na plataforma Windows 8, computador com 6 Gigabytes de memória, processador Core i5, 2,60 Gigahertz.

A metodologia de testes utilizada para analisar os parâmetros custo, tempo de execução e consumo de memória foi selecionar casos com mintermos múltiplos de 2, até a quantidade máxima 2^n e calcular a média. Para 3 e 4 variáveis de entrada, foram selecionados múltiplos de 2, a medida que a quantidade de variáveis aumentava, a quantidade de mintermos selecionados, foi o dobro.

Para os gráficos que representam o pior caso que são as funções contendo a quantidade máxima de mintermos, ou seja, 2^n mintermos, ocorrem todas as comparações e combinações possíveis. A quantidade de casos testados para cada variável de entrada foram de 12. Dos resultados obtidos, eliminou-se as extremidades e calculou-se a média.

Com o objetivo de entender o comportamento dos métodos apresentou-se os resultados obtidos em forma de gráficos. Na Figura 28 apresenta-se a comparação do custo da solução mínima obtido pelo método de Quine-McCluskey Estendido e pelo método de Quine-McCluskey. Pode-se observar que o custo no método de Quine-McCluskey Estendido é menor que o do método de Quine-McCluskey, e a medida que se aumenta a quantidade de variáveis de entrada, a diferença entre os custos se tornam mais significativa.

Figura 28 - Gráfico de comparação do custo entre os métodos analisados.



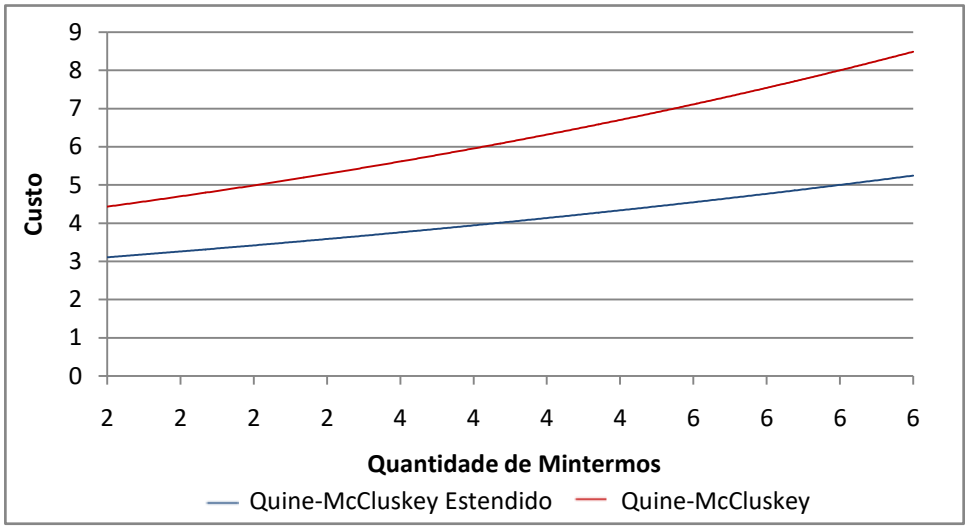
Fonte: Próprio autor.

Esta redução ocorre devido a utilização do operador XOR. No gráfico da Figura 28, o eixo da abscissa, contém até 8 variáveis de entrada, pois no método proposto, gerar implicantes primos para funções acima de 826 variáveis, faz com que o programa lp_solve, que resolve o problema de cobertura, torna-se inadequado, pois o tempo de processamento chega a exceder 24 horas.

Nas Figuras 29, 30, 31, 32, 33 e 34 apresentam-se os gráficos dos custos para cada variável de entrada e seus respectivos mintermos. Pode-se notar através destes gráficos que o Quine-McCluskey Estendido apresenta custo menor quando comparado com o método de Quine-McCluskey. Nas Figuras 30 a 33, observa-se que quando aumenta-se a quantidade de mintermos a curva do custo diminui. Esta característica é decorrente do fato de que a medida em que a quantidade de mintermo aumenta o método Quine-McCluskey Estendido consegue identificar mais estruturas XOR ou XNOR, que leva a uma solução com menor custo quando comparando com a solução utilizando somente estruturas AND-OR.

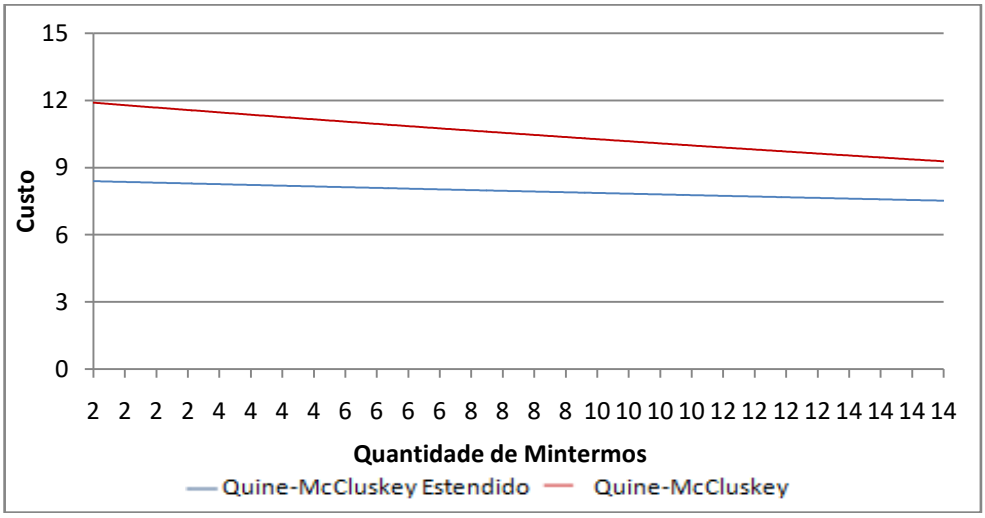
Esta característica pode ser comprovada pela curva da Figura 34, onde estudou-se funções com 8 variáveis. Com 8 variáveis existem 256 diferentes mintermos, entretanto trabalhou-se com funções de no máximo 96 mintermos, ou seja, 37% da quantidade máxima. Nota-se ainda que a diferença entre os custos das soluções mínima é bastante acentuada.

Figura 29 - Gráfico de comparação do custo entre os métodos analisados para funções com 3 variáveis de entrada.



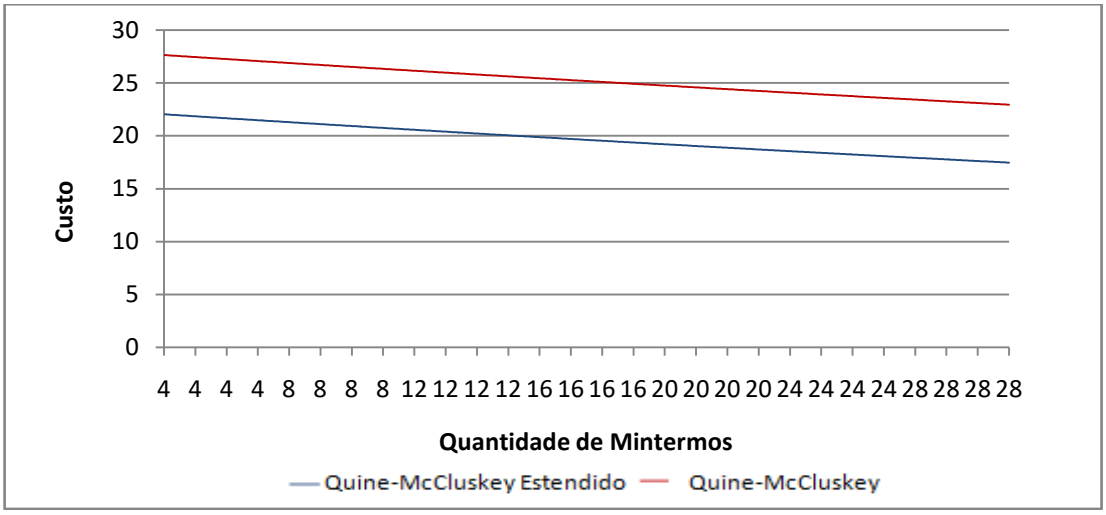
Fonte: Próprio autor.

Figura 30 - Gráfico de comparação do custo entre os métodos analisados para funções com 4 variáveis de entrada.



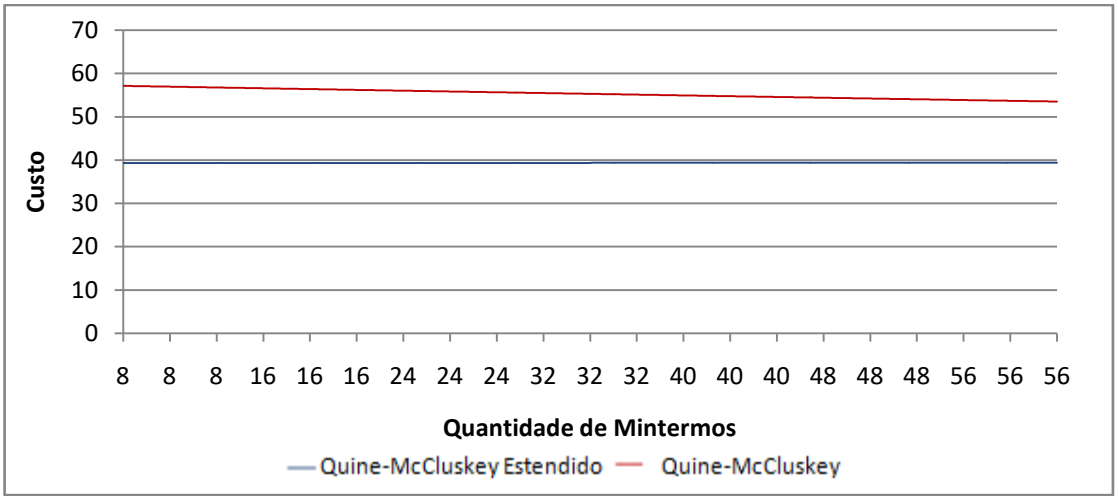
Fonte: Próprio autor.

Figura 31- Gráfico de comparação do custo entre os métodos analisados para funções com 5 variáveis de entrada.



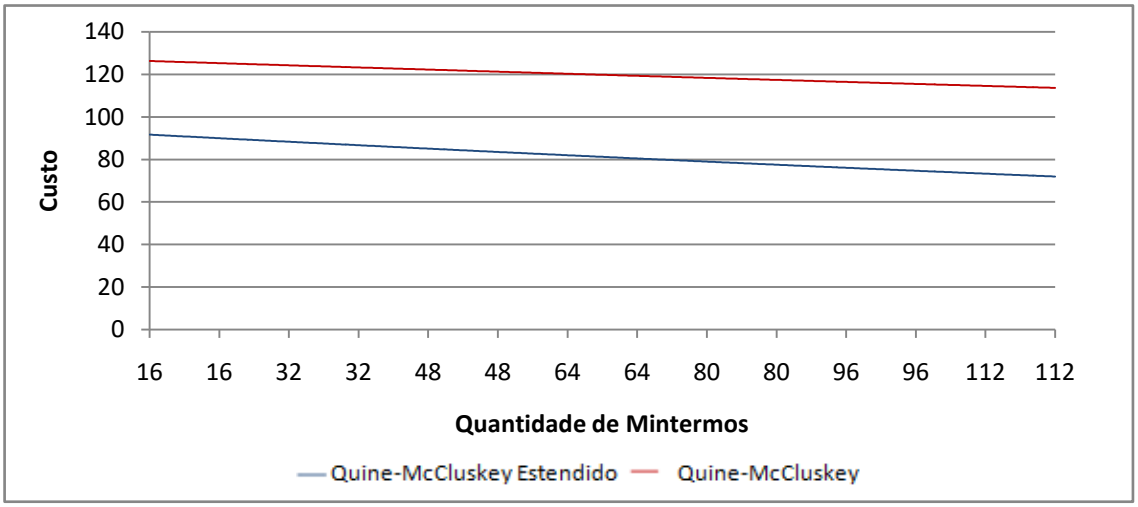
Fonte: Próprio autor.

Figura 32- Gráfico de comparação do custo entre os métodos analisados para funções com 6 variáveis de entrada.



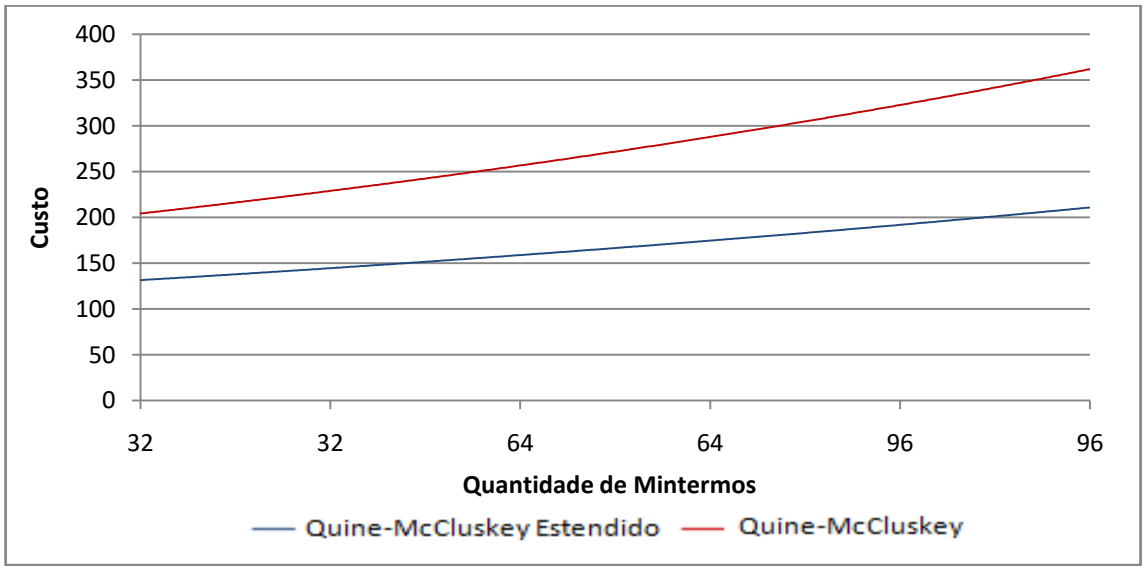
Fonte: Próprio autor.

Figura 33 - Gráfico de comparação do custo entre os métodos analisados para funções com 7 variáveis de entrada



Fonte: Próprio autor.

Figura 34 - Gráfico de comparação do custo entre os métodos analisados para funções com 8 variáveis de entrada.



Fonte: Próprio autor.

Para analisar a superioridade de um dos métodos em relação ao outro calculou-se a porcentagem da diferença entre os custos obtidos, conforme apresenta-se na Tabela 34. As funções Booleanas F_1 a F_n, são apresentadas no Apêndice A. Pode-se observar que para 1,83% dos casos (percentual negativo) o método de Quine-McCluskey obteve um custo menor que o custo do Quine-McCluskey Estendido. Para 9,17 % dos casos (percentual igual

0,01) os custos obtidos pelos dois métodos foram iguais. Entretanto, para 88,99 % (percentual positivo) dos casos, o método de Quine-McCluskey Estendido apresentou um custo menor quando comparado com o método de Quine-McCluskey.

Tabela 34 - Relação entre o custo obtido pelo método de Quine-McCluskey Estendido e o método de Quine-McCluskey.

(continua)

Função	Variáveis de Entrada	Mintermos	Custo Quine-McCluskey Estendido	Custo Quine-McCluskey Tradicional	%
F_1	3	2	4	8	50,00
F_2	3	2	2	2	0,00
F_3	3	2	4	8	50,00
F_4	3	2	2	2	0,00
F_5	3	4	8	10	20,00
F_6	3	4	2	6	66,67
F_7	3	4	6	6	0,00
F_8	3	4	8	10	20,00
F_9	3	6	4	7	42,86
F_{10}	3	6	6	9	33,33
F_{11}	3	6	4	7	42,86
F_{12}	3	6	4	7	42,86
F_{13}	4	2	5	10	50,00
F_{14}	4	2	3	3	0,00
F_{15}	4	2	5	10	50,00
F_{16}	4	2	3	3	0,00
F_{17}	4	4	10	13	23,08
F_{18}	4	4	10	13	23,08
F_{19}	4	4	12	20	40,00
F_{20}	4	4	10	14	28,57
F_{21}	4	6	12	12	0,00
F_{22}	4	6	9	12	25,00
F_{23}	4	6	11	22	50,00
F_{24}	4	6	14	16	12,50
F_{25}	4	8	8	11	27,27
F_{26}	4	8	13	15	13,33
F_{27}	4	8	14	20	30,00
F_{28}	4	8	14	15	6,67
F_{29}	4	10	13	18	27,78
F_{30}	4	10	13	14	7,14
F_{31}	4	10	10	20	50,00
F_{32}	4	10	13	15	13,33

Tabela 34 - Relação entre o custo obtido pelo método de Quine-McCluskey Estendido e o método de Quine-McCluskey.

(continuação)

Função	Variáveis de Entrada	Mintermos	Custo Quine-McCluskey Estendido	Custo Quine-McCluskey Tradicional	%
F_{33}	4	12	13	20	35,00
F_{34}	4	12	7	7	0,00
F_{35}	4	12	4	7	42,86
F_{36}	4	12	7	7	0,00
F_{37}	4	14	3	3	0,00
F_{38}	4	14	7	10	30,00
F_{39}	4	14	3	3	0,00
F_{40}	4	14	5	8	37,50
F_{41}	5	4	18	17	-5,88
F_{42}	5	4	12	17	29,41
F_{43}	5	4	12	16	25,00
F_{44}	5	4	18	17	-5,88
F_{45}	5	8	24	32	25,00
F_{46}	5	8	24	26	7,69
F_{47}	5	8	21	33	36,36
F_{48}	5	8	20	33	39,39
F_{49}	5	12	25	41	39,02
F_{50}	5	12	25	30	16,67
F_{51}	5	12	23	27	14,81
F_{52}	5	12	23	36	36,11
F_{53}	5	16	30	44	31,82
F_{54}	5	16	20	25	20,00
F_{55}	5	16	28	32	12,50
F_{56}	5	16	26	30	13,33
F_{57}	5	20	24	25	4,00
F_{58}	5	20	23	30	23,33
F_{59}	5	20	23	29	20,69
F_{60}	5	20	24	32	25,00
F_{61}	5	24	24	32	25,00
F_{62}	5	24	18	25	28,00
F_{63}	5	24	18	25	28,00
F_{64}	5	24	21	22	4,55
F_{65}	5	28	16	25	36,00
F_{66}	5	28	12	16	25,00
F_{67}	5	28	10	12	16,67
F_{68}	5	28	10	13	23,08
F_{69}	6	8	30	48	37,50

Tabela 34 - Relação entre o custo obtido pelo método de Quine-McCluskey Estendido e o método de Quine-McCluskey.

(continuação)

Função	Variáveis de Entrada	Mintermos	Custo Quine-McCluskey Estendido	Custo Quine-McCluskey Tradicional	%
F_{70}	6	8	31	47	34,04
F_{71}	6	8	23	39	41,03
F_{72}	6	16	44	48	8,33
F_{73}	6	16	39	61	36,07
F_{74}	6	16	42	61	31,15
F_{75}	6	24	51	64	20,31
F_{76}	6	24	44	50	12,00
F_{77}	6	24	52	69	24,64
F_{78}	6	32	46	70	34,29
F_{79}	6	32	45	51	11,76
F_{80}	6	32	47	64	26,56
F_{81}	6	40	48	84	42,86
F_{82}	6	40	46	77	40,26
F_{83}	6	40	49	73	32,88
F_{84}	6	48	42	71	40,85
F_{85}	6	48	45	71	36,62
F_{86}	6	48	36	41	12,20
F_{87}	6	56	30	37	18,92
F_{88}	6	56	30	39	23,08
F_{89}	6	56	27	35	22,86
F_{90}	7	16	66	90	26,67
F_{91}	7	16	64	74	13,51
F_{92}	7	32	97	143	32,17
F_{93}	7	32	89	117	23,93
F_{94}	7	48	102	140	27,14
F_{95}	7	48	97	153	36,60
F_{96}	7	64	101	186	45,70
F_{97}	7	64	102	182	43,96
F_{98}	7	80	96	136	29,41
F_{99}	7	80	89	140	36,43
F_{100}	7	96	78	118	33,90
F_{101}	7	96	79	115	31,30
F_{102}	7	112	55	82	32,93
F_{103}	7	112	51	74	31,08
F_{104}	8	32	131	196	33,16
F_{105}	8	32	135	201	32,84
F_{106}	8	64	172	296	41,89

Tabela 34 - Relação entre o custo obtido pelo método de Quine-McCluskey Estendido e o método de Quine-McCluskey.

(conclusão)

Função	Variáveis de Entrada	Mintermos	Custo Quine-McCluskey Estendido	Custo Quine-McCluskey Tradicional	%
F_{107}	8	64	179	327	45,26
F_{108}	8	96	194	334	41,92
F_{109}	8	96	202	316	36,08

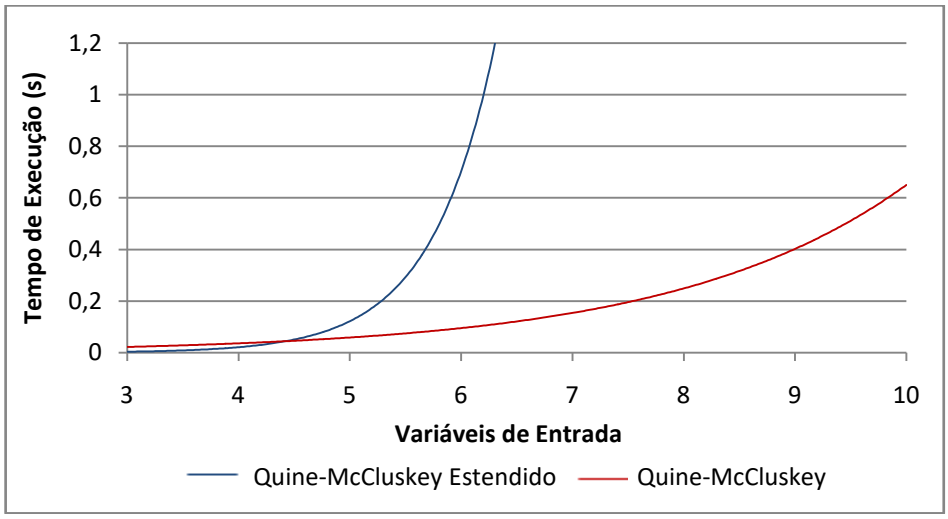
Fonte: Próprio autor.

Os gráficos apresentados nas Figuras 35 e 36 apresentam uma análise nos parâmetros tempo de execução e consumo de memória, respectivamente. Os gráficos representam funções com até 10 variáveis de entrada e 832 mintermos, pois para quantidade de mintermos a partir de 896, o método de Quine-McCluskey Estendido se tornava inadequado, demorando mais de 24 horas para terminar a execução.

Durante o processo de execução, no intervalo de 24 horas, pode-se notar que o arquivo apresentava o tamanho máximo 55.533 KB, quase o dobro quando comparado com o tamanho máximo 32.156KB, executado em apenas 3,16 segundos.

Na Figura 35 apresenta-se uma comparação no tempo de execução do método de Quine-McCluskey Estendido com o método de Quine-McCluskey. Pode-se notar que o método Quine-McCluskey Estendido e o método de Quine-McCluskey, crescem exponencialmente a medida que aumenta a quantidade de variáveis de entrada.Nota-se, também, que o Quine-McCluskey Estendido apresenta um tempo de execução maior quando comparado com o Quine-McCluskey. Este crescimento deve-se a quantidade de comparações e combinações.

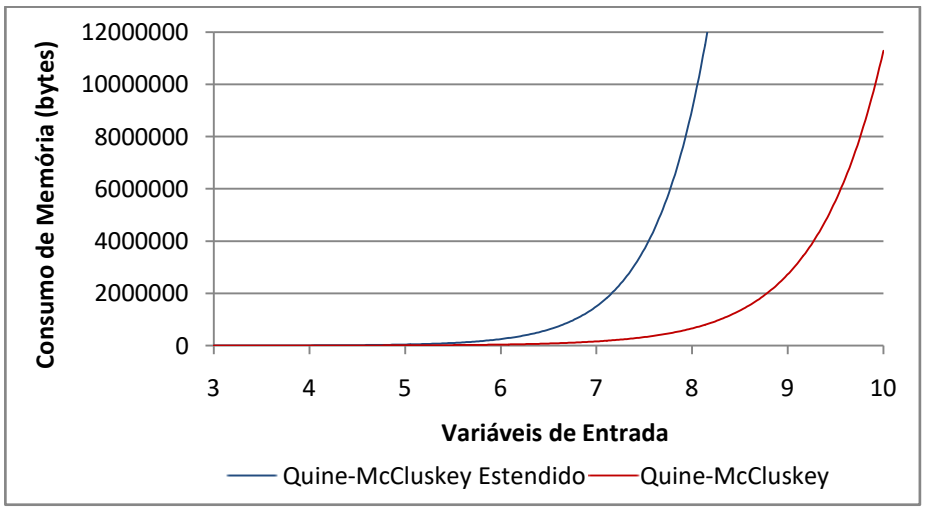
Figura 35- Gráfico de comparação do tempo de execução.



Fonte: Próprio autor.

Na Figura 36 apresenta-se uma comparação no consumo de memória. Pode-se observar que o método de Quine-McCluskey Estendido e o método de Quine-McCluskey crescem exponencialmente a medida que aumenta a quantidade de variáveis de entrada. Nota-se claramente que o método Quine-McCluskey Estendido apresenta um maior consumo de memória quando comparado com o método de Quine-McCluskey. Entende-se que isto ocorre devido a necessidade de se armazenar mais informações na *struct* e apresentar um maior tamanho de lista devido a quantidade de combinações necessárias.

Figura 36- Gráfico de comparação do consumo de memória.



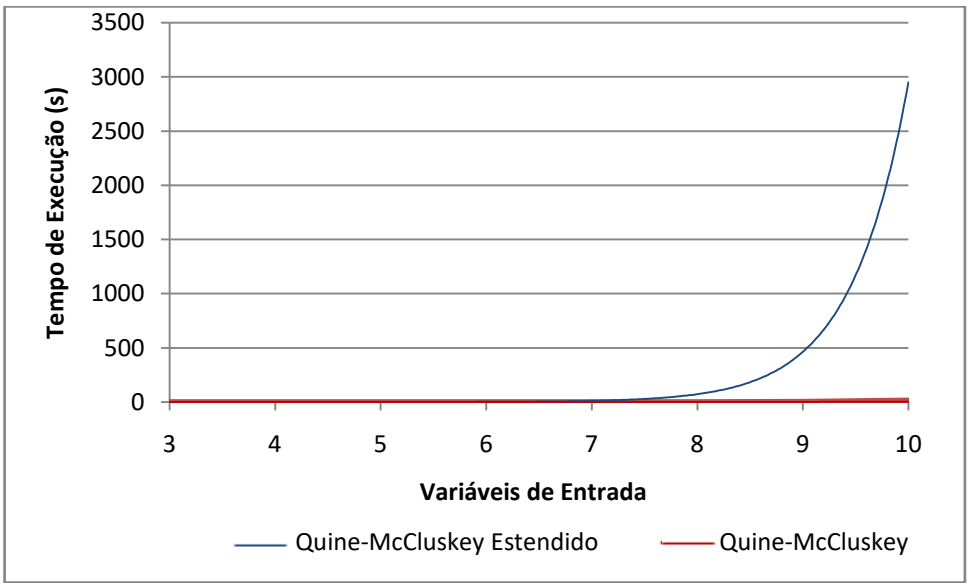
Fonte: Próprio autor.

Os gráficos 37, 38, 39 e 40 mostram o pior caso do método Quine-McCluskey Estendido e o método de Quine-McCluskey. O eixo da abscissa dos gráficos contém uma escala de 3 até 10 variáveis de entrada, lembrando que no método Quine-McCluskey Estendido os casos testados foram até 8 variáveis de entradas e no Quine-McCluskey até 10 variáveis de entrada.

O método proposto realiza mais comparações e combinações, portanto o tempo de execução é maior. Este crescimento no tempo de execução, pode ser observado através da quantidade de combinações que o método proposto apresenta.

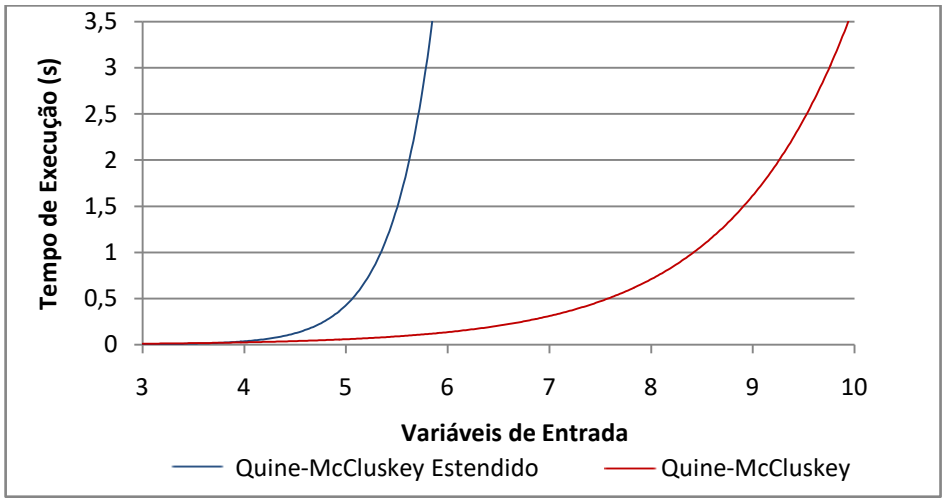
Na Figura 37 apresenta-se o gráfico com o tempo de execução dos dois métodos. Pode-se notar que como o intervalo do tempo de execução é grande, a curva do Quine-McCluskey Estendido cresce rapidamente, enquanto que a curva do Quine-McCluskey permanece constante. Para uma melhor visualização das duas curvas apresenta-se na Figura 38 a escala da ordenada reduzido para 3,5 segundos. Pode-se notar que quando ocorre a redução do eixo da ordenada fica nítido que tanto o método de Quine-McCluskey Estendido quanto o método de Quine-McCluskey crescem exponencialmente. Nota-se ainda que o crescimento no método Quine-McCluskey Estendido é mais acentuado do que no método de Quine-McCluskey, a partir de 5 variáveis de entrada.

Figura 37- Gráfico do pior caso: comparação do tempo de execução.



Fonte: Próprio autor.

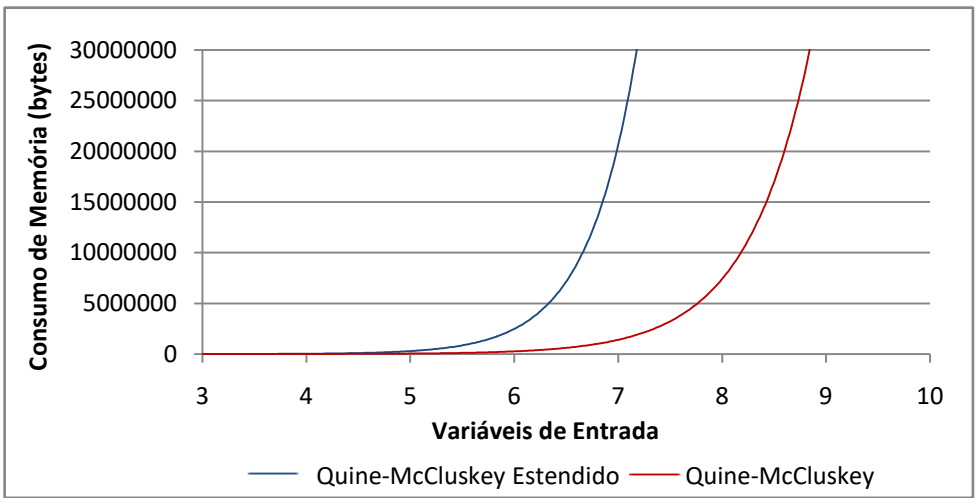
Figura 38 - Gráfico do pior caso 1: comparação do tempo de execução.



Fonte: Próprio autor.

Na Figura 39 apresenta-se o consumo de memória dos métodos analisados. Pode-se observar que a partir de 6 variáveis de entrada, o Quine-McCluskey Estendido cresce mais que o Quine-McCluskey, portanto o método proposto apresenta um maior consumo de memória.

Figura 39 - Gráfico do pior caso: comparação do consumo de memória.



Fonte: Próprio autor.

Na Figura 40 apresenta-se um gráfico em que avalia-se o número de combinações do método proposto. Este cálculo é realizado através da expressão $C_p^n = \frac{n!}{p!(n-p)!}$.

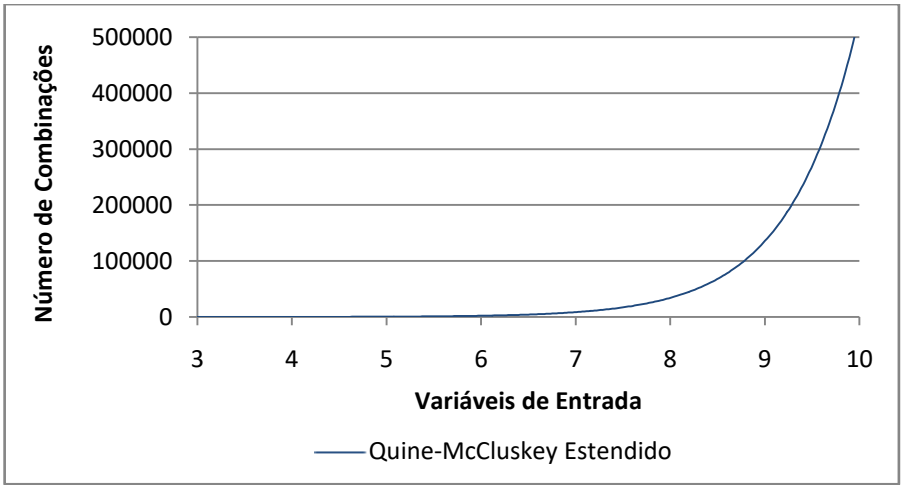
Em que *n!*,é a quantidade de variáveis calculada na base 2^n, em que *p* são as comparações que são realizadas de 2 a 2.

Exemplo: F(ABC) = ∑*m*(0, 1, 2, 3, 4, 5, 6, 7)

$$C_2^8 = \frac{8!}{2!(8-2)!} = \frac{8.7.6!}{2!.6!} = \frac{8.7!}{2.1} = 28 \qquad (57)$$

Portanto, esta fórmula apenas identifica a quantidade máxima de combinações realizadas na primeira iteração do método de Quine-McCluskey Estendido. No gráfico da Figura 40, pode-se observar que a medida que a quantidade de variáveis de entrada aumenta, o número de combinações cresce exponencialmente.

Figura 40 - Gráfico do pior caso: número de combinações do Quine-McCluskey.



Fonte: Próprio autor.

Neste capítulo foi apresentado os resultados obtidos nos métodos Quine-McCluskey Estendido e Quine-McCluskey. Para analisar a eficiência de cada um dos métodos foram criados gráficos para analisar o parâmetro custo do circuito gerado, tempo de execução e quantidade de memória utilizada. Os resultados comprovaram que na maioria dos casos o uso de operadores XOR/XNOR gera um circuito de menor custo. E com relação a desempenho computacional, o método Quine-McCluskey Estendido apresentou-se um tempo de execução e consumo de memória maior se comparado ao método Quine-McCluskey.

No Capítulo 8 apresentam-se as conclusões obtidas após a realização dos testes.

6 CONCLUSÃO

Este trabalho teve como objetivo a implementação da primeira fase do método Quine-McCluskey Estendido com o intuito de comparar os resultados obtidos com a primeira fase do método Quine-McCluskey.

Com os implicantes primos obtidos da primeira fase, tratou-se a cobertura dos mintermos como um problema de programação linear inteira 0 e 1.

O método Quine-McCluskey Estendido foi implementado em linguagem de programação C, utilizando-se lista simplesmente encadeada e foi comparado com o método Quine-McCluskey, em que utilizou-se da mesma estrutura.

Para avaliar a eficiência dos métodos estudados foram analisados os custos dos circuitos mínimos gerados, a quantidade de memória utilizada e o tempo de execução.

Com os resultados obtidos pode-se comprovar a afirmação de Sasao (1990) de que a implementação AND-XOR requer, na maioria das vezes, menos termos produtos do que a implementação tradicional empregando-se estruturas AND-OR.

De acordo com o parâmetro custo, o método de Quine-McCluskey Estendido apresentou um custo menor quando comparado com o método de Quine-McCluskey. Isto ocorreu devido à utilização do operador XOR ou XNOR. Para os casos em que os custos obtidos pelos dois métodos foram iguais, nos implicantes primos gerados não havia operadoradores XOR e XNOR. Nos casos em que o método de Quine-McCluskey obteve um custo menor, ocorreu a utilização do operador XOR entre alguns implicantes primos gerados.

No parâmetro desempenho computacional (tempo e memória) o método Quine-McCluskey Estendido mostrou-se inferior ao do método tradicional devido a grande quantidade de combinações que é realizada na primeira fase.

Como trabalho futuro sugere-se a implementação do método Quine-McCluskey Estendido para funções com múltiplas saídas.

REFERÊNCIAS

BERNASCONI, A.; CIRIANI, V.; CORDONE, R. EXOR projected sum of products. In: VERY LARGE SCALE INTEGRATION, 2006 IFIP INTERNATIONAL CONFERENCE ON, IEEE, 2006, Nice. **Proceedings…** New York: IEEE, 2006. p. 284-289.

BERKELAAR, M.; EIKLAND, K.; NOTEBAERT, P. **lp_solve 5.5. 0.10.** [S. l.: s. l.], 2007. Disponível em: <http://lpsolve.sourceforge.net/5.5/>.Acesso em: 12 maio 2017.

BOOLE, G. **An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities**. Dover: Broadway, 1854.424 p.

BRAYTON, R.K.; HATCHTEL, G.D.; MCMULLEN, C.T.; SANGIOVANNI-VINCENTELLI, A.L. **Logic minimization algorithms for VLSI synthesis**. Boston: Kluwer Academic, 1984. 192 p.

CALLEGARO, V.; MARRANGHELLO, F.S.; MARTINS, M.G.A. Bottom-up disjoint support decomposition based on cofactor and boolean difference analysis. In: Computer Design (ICCD), 2015, New York. **Proceedings…** New York: IEEE, 2015. p. 680-687.

CAPUANO, F. G.; IDOETA, I. V. **Elementos de eletrônica digital.** 33. ed. São Paulo: Érica, 2002.

CHATTOPADHYAY, S.; ROY, S.; CHAUDHURI, P.P. Synthesis of highly testable fixed-polarity AND-XOR canonical networks-A genetic algorithm-based approach. **IEEE transactions on computers**, New York, v. 45, n. 4, p. 487-490, 1996.

CHENG, K-H.; HUANG, C-S. The novel efficient design of XOR/XNOR function for adder applications. In: ELECTRONICS, CIRCUITS AND SYSTEMS - ICECS'99, 6th., Pafos. **Proceedings**… New York: IEEE,1999. p. 29-32.

DAGHLIAN, J. **Lógica e álgebra de Boole.** 4. ed. São Paulo: Atlas, 1995. 152 p.

DHIVYAKALA, K.; SELVAN, P.; KAVITHA, A. Low power-area efficient compressor design for fast digital arithmetic integrated circuits. In: COMPUTING FOR SUSTAINABLE GLOBAL DEVELOPMENT (INDIACOM), 2016, New Delhi. **Proceedings…** New York: IEEE, 2016. p. 1510-1513.

EMER, F. R. P. **Fórmulação matemática do problema de otimização de funções Booleanas através do método de expansão de Shannon.** 2002. 91 f. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia, Universidade Estadual Paulista, Ilha Solteira, 2002.

FISER, P.; HLAVICKA, J.; KUBATOVA, H. FC-Min: a fast multi-output Boolean minimizer. In: EUROMICRO SIMPOSIUM ON DIGITAL SYSTEM DESIGN (DSD'03), 2003, Belek-Antalya. **Proceedings…** Amsterdam: IEEE, 2003. p.451-454.

FRANCISCANI, J. F. **Consenso Iterativo: geração de implicantes primos para minimização de funções Booleanas com múltiplas saídas.** 2016. 121 f. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia, Universidade Estadual Paulista "Júlio de Mesquita Filho", Ilha Solteira, 2016.

HAASWIJK, W.; SOEKEN, M.; AMARÚ, L.; GAILLARDON, P.E. A novel basis for logic rewriting. In: DESIGN AUTOMATION CONFERENCE (ASP-DAC), 2017 22ND ASIA AND SOUTH PACIFIC. Chiba. **Proceedings**… New York: IEEE, 2017. p. 151-156.

HLAVICKA, J.; FISER, P. BOOM- a Heuristic Boolean minimizer. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, 2001, San Jose. **Proceedings...** Amsterdam: IEEE, 2001. p. 439-442.

ILYIN, B.; KAZIMIROV, A. S.; PANTELEYEV, V.I.; REIMEROV, S.Yu.; SEMICHEVA, N. On evolutionary algorithms for Boolean functions minimization. In: SOFT COMPUTING AND MEASUREMENTS (SCM), 2017, Saint Petersburg. **Proceedings…** New York: IEEE, 2017. p. 400-402.

KARNAUGH, M. The map method for synthesis of combinational logiccircuits. **Transactions of the American Institute of Electrical Engineers,** [S. l.], v. 72, part I, p. 593–599, 1953.

KHALID, A. S.; AWWAL, A. A. S. XOR realization using KH-map. In: AEROSPACE AND ELECTRONICS CONFERENCE – NAECON, 1996, Dayton. **Proceedings...** New York: IEEE, 1996. p. 280-285.

KOHUTKA, L.; PISTEK, P. Faster synthesis of combinational logic based on multiplexer trees and binary decision diagrams. In: EMERGING ELEARNING TECHNOLOGIES AND APPLICATIONS, ICETA, 2014,Stary Smokovec. **Proceedings…** New York: IEEE, 2014. p. 239-244.

MBOCK, E. A. M. Circuit minimization method validated by the xilinx technology and system generator experiments. In: INDUSTRIAL ELECTRONICS AND APPLICATIONS (ICIEA), 2015, Auckland. **Proceedings…** New York: IEEE, 2015. p. 707-712.

MCCLUSKEY, E. Minimization of Boolean function. **The Bell System Techinal Journal**, Kansas, v. 35, p. 1417-1444, 1956.

MUMA, K.; KO, S-B. A new logic synthesis, ExorBDS. In: ELECTRICAL AND COMPUTER ENGINEERING, 2005. CANADIAN CONFERENCE ON, IEEE, 2005, Saskatoon. **Proceedings…** New York: IEEE, 2005. p. 816-819.

NOGUEIRA, J.S. **Eletrônica digital básica.** Salvador: Edufba, 2011. 170 p.

PRADHAN, S. N.; KUMAR, M. T.; CHATTOPADHYAY, S. Three-level AND-OR-XOR network synthesis: A GA based approach. In: CIRCUITS AND SYSTEMS, 2008; APCCAS 2008; IEEE ASIA PACIFIC CONFERENCE ON, 2008, Macao. **Proceedings...** New York: IEEE, 2008. p. 574-577.

SASAO, T.; BESSLICH, P. On the complexity of mod-2l sum PLA's. **IEEE Transactions on Computers**, New York, v. 39, n. 2, p. 262-266, 1990.

SASAO, T. Easily testable realizations for generalized Reed-Muller expressions. **IEEE Transactions on Computers**, New York, v. 46, n. 6, p. 709-716, 1997.

SCHAEFFER, B. Product transformation and heuristic EXOR-AND-OR logic synthesis of incompletely specified functions. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v.PP, n.99, p. 1-11,2017.

SHANNON, C. E. A symbolic analysis of relay and switching circuits. **Transactions American Institute of Eletrical Engineers**, [S. l.] , v. 57, n.12, p.713-723, 1938.

SILVA, A. C. R. da. **Contribuição à síntese de circuitos digitais utilizando programação linear inteira 0 e 1.** 1993. 119 f. Tese (Doutorado em Engenharia Elétrica) - Universidade Estadual de Campinas, Campinas, 1993.

SILVA, A. C. R. da. **Contribuição a minimização e simulação de circuitos lógicos.** 1989. 145 f. Tese (Mestrado em Engenharia Elétrica) - Universidade Estadual de Campinas, Campinas, 1989.

SONG, N.; PERKOWSKI, M. A. EXORCISM-MV-2: minimization of exclusive sum of products expressions for multiple-valued input incompletely specified functions. In: INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC, 23., 1993.,Sacramento. **Proceedings**... New York: IEEE, 1993, p. 132-137.

TINDER, R. F. Multilevel logic minimization using K-map XOR patterns. **IEEE Transactions on Education**, New York, v. 38, n. 4, p. 370-375, 1995.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas digitais princípios e aplicações.** 11. ed. São Paulo: Pearson, 2011. 817 p.

TURTON, B.C.H. Extending Quine-McCluskey for exclusive-or logic synthesis. **IEEE Trans. Educ.,** Cardiff, v. 39, n. 1, p.81-85, 1996.

UNGER, S. H. **The essence of logic circuits.** 2.ed. Michigan: Institute of Electrical and Electronic Engineers, 1997. p. 14-45.

USHA, S.; RAJENDIRAN, M.; KAVITHA, A. Low power area efficient ALU with low power full adder. In: COMPUTING FOR SUSTAINABLE GLOBAL DEVELOPMENT (INDIACOM), 2016 3RD INTERNATIONAL CONFERENCE ON, IEEE, 2016, New Delhi. **Proceedings…** New York: IEEE, 2016. p. 1500-1505.

WANG, P.; NIAMAT, M. Y.; VEMURU, S.R.; ALAM, M.; KILLIAN, T. Synthesis of Majority/Minority Logic Networks. In: IEEE TRANSACTIONS ON NANOTECHNOLOGY. **Proceeding…** New York: IEEE, 2015. p. 473-483.

YANG, C.; SINGHAL, V.; CIESIELSKI, M. BDD decomposition for efficient logic synthesis. In: COMPUTER DESIGN, 1999.(ICCD'99) INTERNATIONAL CONFERENCE ON. IEEE, 1999, Austin. **Proceedings...** New York: IEEE, 1999. p. 626-631.

YANG, C.; CIESIELSKI, M. BDS: A BDD-based logic optimization system. **IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems**, New York, v. 21, n. 7, p. 866-876, 2002.

YE, Y.; ROY, K. Efficient synthesis of AND/XOR networks. In: DESIGN AUTOMATION CONFERENCE, 1997, Chiba. **Proceedings...** New York:IEEE, 1997, p. 539-544.

YINSHUI, X.; XIEXIONG, C.; LUNYAO, W. Dual logic and its application in logic minimization. In: INTERNATIONAL CONFERENCE, ELECTRONICS, COMMUNICATIONS AND CONTROL (ICECC), 2011, Ningbo. **Proceedings…** Amsterdam: IEEE, 2011. p. 239-242.

APÊNDICE A- FUNÇÕES BOOLEANAS AVALIADAS

$F_1(ABC) = \sum m\ (0, 6)$

$F_2(ABC) = \sum m\ (1, 3)$

$F_3(ABC) = \sum m(1, 7)$

$F_4(ABC) = \sum m\ (2, 3)$

$F_5(ABC) = \sum m\ (1, 2, 3, 4)$

$F_6(ABC) = \sum m\ (2, 3, 4, 5)$

$F_7(ABC) = \sum m\ (0, 2, 3, 4)$

$F_8(ABC) = \sum m\ (2, 3, 4, 7)$

$F_9(ABC) = \sum m\ (1, 2, 4, 5, 6, 7)$

$F_{10}(ABC) = \sum m\ (1, 2, 3, 4, 5, 6)$

$F_{11}(ABC) = \sum m(0, 1, 2, 4, 6, 7)$

$F_{12}(ABC) = \sum m\ (0, 1, 2, 3, 4, 7)$

$F_{13}(ABCD) = \sum m\ (0, 6)$

$F_{14}(ABCD) = \sum m\ (12, 13)$

$F_{15}(ABCD) = \sum m\ (9, 15)$

$F_{16}(ABCD) = \sum m\ (7, 15)$

$F_{17}(ABCD) = \sum m\ (1, 3, 5, 12)$

$F_{18}(ABCD) = \sum m\ (0, 4, 6, 9)$

$F_{19}(ABCD) = \sum m\ (0, 9, 10, 12)$

$F_{20}(ABCD) = \sum m\ (4, 9, 12, 15)$

$F_{21}(ABCD) = \sum m\ (0, 4, 5, 10, 11, 13)$

$F_{22}(ABCD) = \sum m\ (1, 3, 4, 6, 11, 15)$

$F_{23}(ABCD) = \sum m\ (2, 7, 8, 10, 11, 13)$

$F_{24}(ABCD) = \sum m\ (1, 3, 5, 9, 10, 14)$

$F_{25}(ABCD) = \sum m\ (1, 2, 3, 6, 8, 9, 11,12)$

$F_{26}(ABCD) = \sum m\ (2, 4, 8, 9, 12, 13, 14, 15)$

$F_{27}(ABCD) = \sum m\ (1, 2, 3, 4, 8, 10, 12, 13)$

$F_{28}(ABCD) = \sum m\ (0, 5, 6, 7, 8, 9, 10, 11)$

$F_{29}(ABCD) = \sum m\ (1, 2, 5, 6, 7, 8, 9, 12, 13, 14)$

$F_{30}(ABCD) = \sum m\ (1, 4, 5, 6, 7, 8, 9, 12, 14, 15)$

$F_{31}(ABCD) = \sum m\ (0, 1, 2, 5, 6, 8, 11, 12, 13, 15)$

$F_{32}(ABCD) = \sum m\ (0, 4, 5, 6, 8, 9, 10, 12, 13, 14)$

$F_{33}(ABCD) = \sum m\ (0, 3, 4, 5, 6, 8, 9, 10, 11, 12, 14, 15)$

$F_{34}(ABCD) = \sum m(\ 0, 1, 2, 4, 5, 6, 7, 8, 9, 12, 13, 15)$

$F_{35}(ABCD) = \sum m(0, 1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 15)$

$F_{36}(ABCD) = \sum m(1, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15)$

$F_{37}(ABCD) = \sum m(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$

$F_{38}(ABCD) = \sum m(0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 12, 13, 14, 15)$

$F_{39}(ABCD) = \sum m(0, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$

$F_{40}(ABCD) = \sum m(0, 1, 2, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15)$

$F_{41}(ABCDE) = \sum m\ (3, 11, 16, 23)$

$F_{42}(ABCDE) = \sum m\ (0, 1, 14, 28)$

$F_{43}(ABCDE) = \sum m\ (8, 12, 13, 25)$

$F_{44}(ABCDE) = \sum m\ (11, 12, 20, 22)$

$F_{45}(ABCDE) = \sum m(1, 3, 4, 12, 14, 19, 24, 29)$

$F_{46}(ABCDE) = \sum m(8, 13, 15, 18, 21, 24, 26, 27)$

$F_{47}(ABCDE) = \sum m\ (3, 5, 6, 16, 21, 22, 27, 29)$

$F_{48}(ABCDE) = \sum m(6, 12, 13, 19, 22, 25, 26, 28)$

$F_{49}(ABCDE) = \sum m(0, 1, 3, 4, 6, 12, 17, 21, 23, 27, 28, 30)$

$F_{50}(ABCDE) = \sum m(1, 2, 4, 5, 10, 11, 12, 16, 18, 22, 23, 27)$

$F_{51}(ABCDE) = \sum m\ (0, 1, 4, 5, 8, 9, 10, 14, 15, 16, 20, 22)$

$F_{52}(ABCDE) = \sum m(1, 3, 6, 10, 11, 13, 14, 16, 18, 21, 26, 30)$

$F_{53}(ABCDE) = \sum m(1, 2, 4, 7, 8, 9, 14, 17, 20, 21, 22, 24, 25, 27, 29, 31)$

$F_{54}(ABCDE) = \sum m\ (1, 3, 9, 10, 11, 14, 15, 16, 17, 18, 20, 21, 25, 26, 27, 29)$

$F_{55}(ABCDE) = \sum m(1, 4, 6, 7, 10, 11, 12, 13, 20, 21, 22, 23, 24, 26, 28, 29)$

$F_{56}(ABCDE) = \sum m\ (0, 1, 2, 3, 4, 6, 12, 13, 19, 20, 25, 26, 27, 28, 29, 31)$

$F_{57}(ABCDE) = \sum m\ (0, 1, 3, 4, 5, 6, 7, 8, 11, 15, 17, 18, 19, 22, 26, 27, 28, 29, 30, 31)$

$F_{58}(ABCDE) = \sum m(0, 1, 3, 4, 6, 7, 8, 9, 11, 12, 13, 15, 16, 18, 21, 24, 25, 26, 27, 29)$

$F_{59}(ABCDE) = \sum m(0, 1, 2, 4, 6, 7, 8, 10, 12, 14, 17, 19, 20, 22, 23, 25, 26, 27, 28, 31)$

$F_{60}(ABCDE) = \sum m\ (2, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20, 22, 24, 26, 27, 30)$

$F_{61}(ABCDE) = \sum m(1, 2, 4, 6, 7, 8, 9, 10, 11, 13, 16, 17, 18, 19, 20, 23, 24, 25, 26, 27, 28, 29, 30, 31)$

$F_{62}(ABCDE) = \sum m\ (0, 1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 14, 15, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31)$

$F_{63}(ABCDE) = \sum m\ (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 16, 18, 19, 20, 22, 26, 27, 28, 29, 30, 31)$

$F_{64}(ABCDE) = \sum m(0, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 31)$

$F_{65}(ABCDE) = \sum m$ (0, 1, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 29, 30, 31)

$F_{66}(ABCDE) = \sum m$(0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31)

$F_{67}(ABCDE) = \sum m$(1, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)

$F_{68}(ABCDE) = \sum m$(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 29, 30, 31)

$F_{69}(ABCDEF) = \sum m$ (12, 35, 38, 42, 44, 53, 59, 62)

$F_{70}(ABCDEF) = \sum m$ (0, 1, 6, 9, 35, 36, 48, 54)

$F_{71}(ABCDEF) = \sum m$ (5, 7, 10, 12, 14, 24, 34, 51)

$F_{72}(ABCDEF) = \sum m$ (1, 7, 10, 11, 15, 16, 17, 18, 22, 35, 43, 45, 47, 54, 58, 62)

$F_{73}(ABCDEF) = \sum m$(1, 4, 8, 10, 11, 12, 13, 25, 29, 33, 41, 42, 43, 49, 50, 52)

$F_{74}(ABCDEF) = \sum m$ (1, 2, 9, 11, 18, 19, 21, 22, 23, 25, 28, 29, 30, 40, 47, 61)

$F_{75}(ABCDEF) = \sum m$ (0, 3, 8, 9, 12, 13, 14, 15, 19, 21, 25, 26, 27, 32, 33, 36, 41, 45, 48, 53, 54, 56, 58, 59)

$F_{76}(ABCDEF) = \sum m$ (0, 7, 8, 9, 11, 17, 18, 22, 24, 25, 26, 27, 28, 30, 38, 41, 43, 47, 49, 52, 53, 57, 59, 63)

$F_{77}(ABCDEF) = \sum m$(1, 2, 8, 9, 10, 14, 18, 19, 21, 23, 25, 29, 32, 34, 36, 41, 43, 44, 49, 52, 56, 57, 59, 60)

$F_{78}(ABCDEF) = \sum m$(0, 1, 3, 5, 12, 17, 20, 21, 25, 26, 28, 29, 30, 31, 32, 35, 36, 39, 41, 42, 45, 46, 48, 49, 50, 51, 52, 57, 60, 61, 62, 63)

$F_{79}(ABCDEF) = \sum m$(0, 2, 3, 5, 6, 78, 9, 10, 13, 14, 16, 19, 20, 21, 23, 26, 29, 30, 37, 40, 41, 42, 43, 44, 45, 50, 52, 53, 58, 61, 62)

$F_{80}(ABCDEF) = \sum m$(0, 3, 6, 7, 8, 9, 10, 13, 15, 19, 23, 24, 26, 27, 30, 31, 32, 35, 38, 39, 40, 43, 44, 46, 47, 49, 53, 56, 57, 58, 62, 63)

$F_{81}(ABCDEF) = \sum m(3, 4, 5, 6, 7, 9, 11, 12, 13, 17, 20, 21, 22, 23, 24, 28, 30, 31, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 45, 47, 48, 50, 51, 52, 55, 58, 59, 60, 61, 62)$

$F_{82}(ABCDEF) = \sum m(2, 4, 6, 7, 8, 10, 15, 16, 17, 21, 22, 23, 25, 26, 27, 28, 29, 31, 32, 34, 37, 38, 40, 42, 44, 45, 46, 48, 49, 50, 52, 53, 54, 55, 56, 57, 58, 59, 62, 63)$

$F_{83}(ABCDEF) = \sum m (0, 2, 3, 4, 6, 7, 10, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 25, 27, 31, 33, 34, 35, 37, 38, 42, 43, 46, 48, 51, 53, 54, 55, 57, 58, 59, 60, 61, 62, 63)$

$F_{84}(ABCDEF) = \sum m(1, 3, 4, 6, 7, 8, 9, 10, 11, 12, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 41, 42, 44, 46, 47, 48, 49, 51, 52, 53, 55, 58, 59, 60, 62, 63)$

$F_{85}(ABCDEF) = \sum m(0, 1, 2, 3, 4, 5, 6, 8, 11, 12, 14, 15, 19, 20, 21, 22, 24, 25, 26, 28, 29, 31, 33, 34, 35, 37, 38, 39, 40, 41, 42, 43, 44, 45, 48, 50, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63)$

$F_{86}(ABCDEF) = \sum m(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 16, 17, 18, 20, 22, 23, 24, 25, 27, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 41, 42, 43, 46, 47, 48, 49, 50, 52, 53, 54, 55, 56, 57, 61, 62, 63)$

$F_{87}(ABCDEF) = \sum m(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 58, 59, 62, 63)$

$F_{88}(ABCDEF) = \sum m(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 20, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 62, 63)$

$F_{89}(ABCDEF) = \sum m(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 37, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 53, 55, 56, 57, 58, 59, 60, 61, 62, 63)$

$F_{90}(ABCDEFG) = \sum m (3, 22, 25, 31, 38, 43, 51, 57, 65, 69, 92, 94, 102, 109, 118, 121)$

$F_{91}(ABCDEFG) = \sum m (9, 12, 15, 25, 30, 37, 45, 49, 62, 73, 77, 84, 86, 91, 100, 113)$

$F_{92}(ABCDEFG) = \sum m (0, 1, 6, 13, 19, 20, 21, 29, 30, 33, 37, 43, 44, 45, 56, 57, 61, 63, 68, 74, 77, 88, 90, 95, 98, 100, 101, 102, 116, 118, 121, 123)$

$F_{93}(ABCDEFG) = \sum m$ (11, 12, 14, 22, 23, 27, 29, 33, 36, 37, 38, 46, 57, 72, 75, 76, 79, 85, 87, 90, 93, 94, 95, 98, 99, 101, 108, 109, 111, 117, 126, 127)

$F_{94}(ABCDEFG) = \sum m$ (3, 7, 11, 12, 16, 19, 20, 21, 22, 23, 29, 32, 38, 41, 46, 47, 49, 51, 55, 57, 59, 60, 61, 71, 78, 81, 82, 83, 85, 86, 87, 88, 89, 91, 93, 95, 97, 99, 100, 106, 110, 114, 120, 121, 122, 125, 126, 127)

$F_{95}(ABCDEFG) = \sum m$ (1, 2, 10, 16, 19, 20, 22, 23, 28, 30, 31, 32, 36, 40, 41, 42, 43, 59, 60, 65, 68, 69, 72, 74, 78, 79, 80, 82, 84, 89, 91, 92, 93, 96, 99, 102, 106, 107, 108, 109, 110, 111, 113, 119, 123, 124, 126, 127)

$F_{96}(ABCDEFG) = \sum m$(0, 1, 2, 5, 7, 8, 11, 12, 13, 14, 16, 18, 19, 20, 23, 24, 25, 27, 28, 29, 30, 32, 35, 37, 41, 43, 47, 49, 50, 54, 55, 56, 57, 58, 60, 64, 68, 72, 74, 75, 82, 83, 85, 86, 87, 88, 91, 92, 97, 98, 100, 105, 108, 109, 113, 115, 116, 119, 120, 121, 122, 123, 124, 127)

$F_{97}(ABCDEFG) = \sum m$ (4, 5, 7, 11, 13, 14, 15, 19, 20, 22, 24, 25, 26, 29, 31, 32, 34, 35, 36, 40, 41, 43, 45, 47, 54, 55, 59, 60, 61, 62, 64, 67, 69, 70, 72, 76, 77, 80, 83, 84, 85, 88, 90, 91, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 106, 110, 111, 113, 115, 117, 118, 123, 124)

$F_{98}(ABCDEFG) = \sum m$ (2, 3, 5, 7, 8, 10, 12, 13, 14, 15, 20, 21, 22, 23, 25, 28, 31, 33, 38, 40, 41, 43, 44, 45, 47, 49, 51, 52, 53, 54, 55, 56, 58, 59, 60, 61, 63, 64, 66, 67, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 81, 82, 85, 86, 89, 90, 94, 95, 96, 99, 100, 101, 103, 105, 106, 107, 108, 109, 110, 111, 112, 114, 118, 119, 120, 121, 122, 124, 125, 127)

$F_{99}(ABCDEFG) = \sum m$ (1, 5, 6, 7, 8, 10, 16, 17, 20, 21, 23, 24, 25, 26, 27, 29, 30, 31, 33, 34, 35, 37, 39, 40, 42, 43, 45, 46, 47, 48, 49, 52, 54, 57, 58, 59, 60, 62, 63, 68, 69, 70, 72, 73, 74, 75, 78, 79, 80, 81, 82, 84, 85, 87, 88, 89, 91, 92, 94, 96, 97, 98, 99, 101, 103, 104, 107, 108, 111, 112, 113, 115, 116, 117, 118, 119, 121, 124, 126, 127)

$F_{100}(ABCDEFG) = \sum m$ (0, 2, 4, 5, 6, 7, 8, 10, 11, 13, 14, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 40, 41, 42, 43, 44, 46, 47, 48, 49, 52, 53, 54, 58, 59, 60, 62, 63, 64, 65, 66, 67, 68, 70, 71, 73, 75, 76, 78, 79, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 95, 97, 98, 100, 103, 105, 106, 107, 108, 109, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 127)

$F_{101}(ABCDEFG) = \sum m$ (0, 2, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 21, 22, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 46, 47, 49, 50, 51, 52, 53, 54, 55, 58, 59, 61, 63, 64, 66, 68, 69, 70, 71, 73, 75, 77, 79, 80, 81, 83, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95,

96, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 117, 118, 120, 121, 122, 123, 124, 125)

F_{102}(ABCDEFG) = $\sum m$ (0, 1, 2, 3, 4, 5, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32, 33, 34, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 64, 65, 66, 67, 68, 69, 70, 71, 72, 74, 75, 76, 77, 78, 80, 81, 82, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 104, 105, 106, 107, 108, 109, 110, 111, 112, 114, 115, 116, 117, 118, 119, 120, 122, 123, 124, 126, 127)

F_{103}(ABCDEFG) = $\sum m$ (1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41, 42, 43, 44, 46, 49, 50, 51, 52, 53, 56, 57, 58, 59, 61, 62, 63, 64, 65, 67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 84, 85, 86, 87, 88, 89, 90, 91, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 127)

F_{104}(ABCDEFGH) = $\sum m$ (1, 18, 27, 41, 45, 46, 49, 71, 77, 85, 91, 104, 119, 126, 128, 130, 136, 143, 145, 152, 159, 175, 183, 194, 200, 202, 210, 226, 234, 242, 244, 255)

F_{105}(ABCDEFGH) = $\sum m$ (15, 20, 25, 45, 60, 62, 63, 76, 78, 80, 81, 83, 95, 96, 128, 138, 142, 148, 150, 152, 163, 170, 177, 195, 196, 202, 224, 225, 231, 239, 241, 249)

F_{106}(ABCDEFGH) = $\sum m$ (1, 3, 13, 18, 20, 21, 29, 38, 41, 42, 44, 48, 50, 55, 57, 61, 66, 67, 81, 82, 84, 86, 87, 93, 98, 100, 103, 104, 105, 112, 119, 123, 126, 133, 144, 147, 148, 152, 155, 164, 166, 168, 170, 173, 175, 181, 183, 194, 198, 200, 206, 214, 223, 224, 226, 228, 230, 231, 232, 237, 238, 239, 248, 251)

F_{107}(ABCDEFGH) = $\sum m$ (1, 6, 7, 10, 17, 24, 27, 34, 35, 43, 44, 52, 61, 66, 68, 69, 75, 76, 78, 81, 85, 86, 87, 88, 93, 94, 96, 102, 105, 106, 107, 108, 112, 117, 118, 126, 127, 137, 143, 145, 151, 155, 160, 161, 169, 172, 184, 186, 198, 199, 201, 202, 210, 211, 215, 221, 222, 226, 235, 237, 241, 242, 245, 246)

F_{108}(ABCDEFGH) = $\sum m$ (0, 2, 4, 5, 7, 10, 15, 16, 26, 28, 30, 32, 39, 42, 43, 46, 47, 50, 57, 68, 71, 73, 74, 77, 79, 80, 81, 82, 83, 85, 86, 87, 88, 91, 100, 102, 103, 104, 108, 109, 112, 119, 120, 130, 132, 134, 136, 145, 147, 149, 151, 154, 155, 156, 159, 161, 165, 169, 170, 174, 175, 177, 179, 184, 187, 188, 189, 190, 191, 193, 194, 196, 197, 199, 204, 205, 208, 209, 213, 214, 223, 224, 225, 226, 227, 228, 230, 231, 237, 240, 243, 244, 245, 247, 251, 254)

$F_{109}(ABCDEFGH) = \sum m$ (0, 2, 5, 14, 15, 17, 19, 24, 25, 27, 29, 32, 34, 35, 36, 39, 40, 42, 45, 46, 47, 49, 50, 55, 57, 62, 63, 64, 71, 74, 83, 84, 86, 87, 88, 89, 90, 91, 94, 95, 98, 102, 104, 105, 123, 125, 127, 129, 134, 136, 138, 140, 141, 146, 149, 154, 156, 160, 166, 167, 168, 169, 172, 174, 175, 179, 183, 185, 188, 192, 195, 198, 202, 205, 208, 211, 213, 215, 227, 229, 230, 231, 232, 233, 234, 236, 237, 239, 243, 244, 249, 250, 252, 253, 254, 255)