UNIVERSIDADE ESTADUAL DE CAMPINAS - UNICAMP

FACULDADE DE ENGENHARIA ELETRICA - FEE

DEPARTAMENTO DE TELEMATICA

Este exemplar corresponde à redação final da tese defendida por Marcos Cesar Garber de Madureira e aprovada pela Comissão Julgadora em 19/05/87

Bonatti

" CONTRIBUIÇAO A ANALISE E SINTESE DE CIRCUITOS DIGITAIS "

MARCOS CESAR GARBER DE MADUREIRA

Orientador : PROF. DR. IVANIL SEBASTIÃO BONATTI

Tese apresentada à Faculdade de Engenharia Elétrica, da Universidade Estadual de Campinas - UNICAMP, como parte dos requisitos exigidos para a obtenção do título de MESTRE EM CIENCIAS.

- Maio de 1987 -

À minha esposa OFIR

e à minha filha JANAINA

A DEUS, sem o qual não sou capaz de realizar absolutamente nada,

Aos meus PAIS, pelo amor com que plantaram todas as suas melhores sementes no meu coração,

A minha esposa OFIR e à minha filha JANAINA, pelo carinho e apoio nas presenças e ausências,

Ao IVANIL, pela idealização, determinação e competência na realização deste trabalho a quatro mãos, mas principalmente pela compreensão, amizade e respeito cultivados,

Ao Marcos da Costa Barros, por todas as trocas de idéias e pela colaboração na programação,

Ao Edson, ao Pedro Peres, ao José Pedro, à Iluda e a todos os que me ajudaram a gostar tanto desta época que já deixa saudades,

A todos os funcionários, colegas e professores que me tornaram possível dar mais este passo,

A FAPESP, ao CAPES e ao CPqD/Telebrás que me deram suporte,

Meu abraço mais forte !

" Ainda que eu tivesse o dom da profecia,
o conhecimento de todos os mistérios e
de toda a ciência,
Ainda que tivesse toda a fé,
a ponto de transportar os montes,
se não tivesse a caridade,
eu nada seria . "

(1 Cor 13,2)

I N D I C E :

CAPÍTULO III.- MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS

CAPÍTULO IV.- MÁQUINAS SEQUENCIAIS SÍNCRONAS

F I G U R A S :

CAPITULO I.- ESTADO DA ARTE

CAPITULO II.- CIRCUITOS COMBINACIONAIS

CAPITULO III.- MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS

CAPITULO IV.- MAQUINAS SEQUENCIAIS SINCRONAS

CAPITULO V.- SIMULAÇÃO LOGICA

T A B E L A S :

CAPITULO II.- CIRCUITOS COMBINACIONAIS

CAPITULO III.- MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS

CAPITULO IV.- MAQUINAS SEQUENCIAIS SINCRONAS

SUMARIO :

O barateamento com consequente popularização dos componentes digitais tornou realidade e difundiu a expressão "Projeto Auxiliado por Computador".

Assim, a tarefa do engenheiro projetista é hoje cada vez mais eficiente, pois conta com um número crescente de "software" de síntese e análise nas mais variadas atividades técnicas.

O projeto de um equipamento digital consiste essencialmente das seguintes etapas:

a) Descrição funcional, entrada-saida, do circuito. Em geral isto é feito através de diagramas de tempo, diagramas de estado;

b) Partição preliminar do circuito em blocos, com definição das interfaces entre os blocos;

c) Descrição formal, entrada/saida, de cada bloco ;

d) Síntese de cada bloco com minimização das funções Booleanas ;

e) Análise de desempenho ;

f) Reavaliação do projeto com base no seu desempenho, podendo-se retornar ao passo a) ou seguir ao passo g) .

g) Implementação final.

As etapas c),d),e) são as que mais se prestam a automatização e são nessas áreas justamente que se concentra este trabalho de tese.

Foi implementado um pacote de programas que: elabora tabelas de próximo estado para um dado diagrama de estados, minimiza as funções lógicas envolvidas, chegando a circuitos mínimos e simula circuitos lógicos com tempos de atraso.

CAPÍTULO 1

ESTADO DA ARTE

1.1.- RESUMO :

Este capítulo mostra a volta à discussão do tema "Análise e Síntese de Circuitos Digitais " . No modo de ver do autor, isto ocorreu devido à mudança no perfil do usuário que, em parte, passou de projetista de circuitos usando integrados a projetista de integrados propriamente.

Aqui são citados e comentados superficialmente alguns dos artigos publicados nesta área de pesquisa entre 1975 e 1986.

I.2.- INTRODUÇAO :

E no contexto de reconhecimento e avaliação do estado da arte, a nível de "software", do ferramental de auxílio à análise e síntese de circuitos digitais, que se insere este capítulo de pesquisa bibliográfica, onde são citados e comentados superficialmente alguns dos artigos publicados nesta área de pesquisa entre 1975 e 1986. [4]

Vale destacar a constatação de um imenso hiato na publicação de trabalhos na área entre o fim da década de 60 e meados da década de 70 (1967-1976). Uma de suas possíveis causas foi a miniaturização, barateamento e diversificação dos tipos de circuitos integrados (CI) existentes. Isto fez com que ficasse mais fácil ao projetista de equipamentos digitais, recorrendo aos manuais, escolher os circuitos integrados mais convenientes, ao invés de procurar otimizar os seus circuitos através de minimizações das funções envolvidas.

Com o aparecimento dos circuitos lógicos feitos segundo especificação dos usuários ("custom made logic circuits"), os engenheiros projetistas passaram a utilizar mais frequentemente as ferramentas de análise e síntese de circuitos digitais.

O intuito deste capítulo é apresentar um apanhado geral sobre o contexto de projeto de circuitos digitais, esquematizado na Figura I.1.

Na Seção I.3 são apresentadas as tendências tecnológicas da área, tais como lógica de multi-valores, "programmable logic arrays", e "custom made logic circuits".

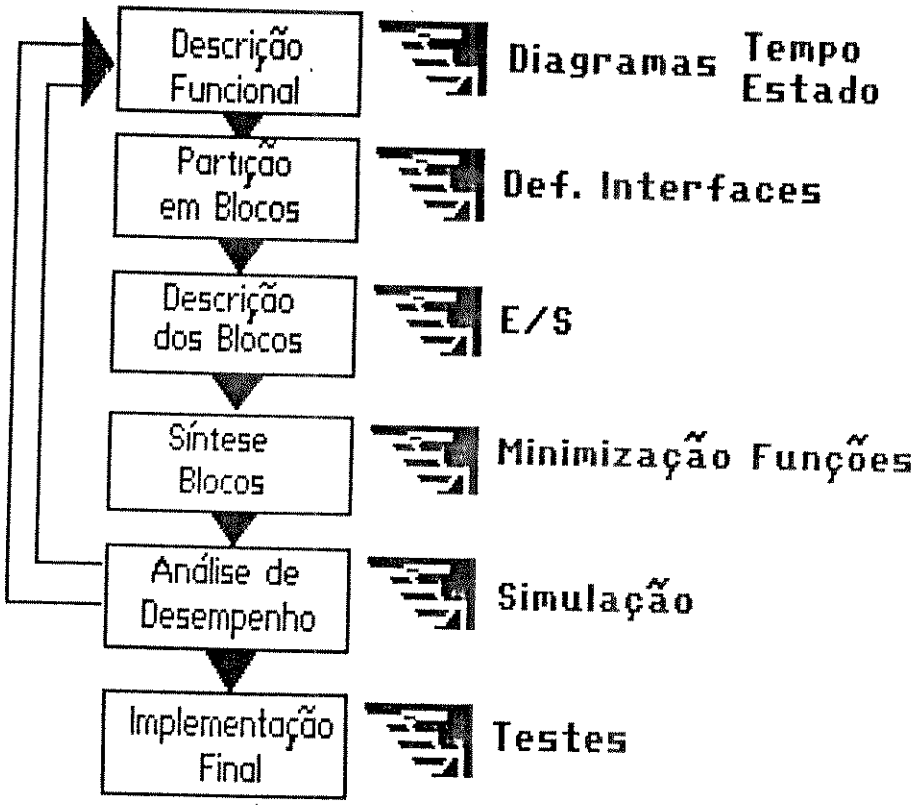


Figura I.1.- Esquema de projeto de circuitos digitais

Na Seção I.4, descrevem-se os circuitos combinacionais, dando ênfase à parte de minimização de funções Booleanas.

A Seção I.5 apresenta os circuitos sequenciais , destacando aspectos tais como as técnicas de alocação ótima de estados, minimização dos diagramas de estado, e classes especiais de circuitos com larga aplicação, como os registradores de deslocamento ("shift-registers").

Alguns estudos na deteção e verrificação de erros são comentados na Seção I.6 e a Seção I.7 trata da simulação de circuitos.

Este primeiro capítulo é concluido na Seção I.8.

O Capítulo II.-CIRCUITOS COMBINACIONAIS descreve a álgebra de Boole e seus postulados, define as funções booleanas e suas realizações, e faz uma avaliação de custos.

A minimização das funções booleanas é definida no Capítulo III.- MINIMIZAÇÃO, onde apresentam-se os algoritmos de Quine-McCluskey[1,2,3] e de Caruso[17].

No Capítulo IV.-CIRCUITOS SEQUENCIAIS, são descritos e comentados circuitos sequenciais síncronos, e no Capítulo V.-SIMULAÇÃO LOGICA é apresentado um algoritmo de simulação.

O Capítulo VI.-CONCLUSÃO encerra este trabalho dando uma visão de suas perspectivas.

NOTA : Todos os programas são conversacionais, facilitando a comunicação entre usuário e máquina. Todos foram implementados num microcomputador de 16 bits tipo IBM PCxt, 256 Kbytes de memória, operando com o sistema operacional DOS, em linguagem TURBO-PASCAL V.3.01.

I.3.-TENDENCIAS TECNOLOGICAS E TECNICAS;

Com o barateamento e avanço das técnicas de integração, além do desenvolvimento de facilidades e recursos computacionais, as funções lógicas passaram a ser implementadas por uma enorme variedade de meios usando circuitos com escalas de integração grande e muito grande (LSI,VLSI) .

Também, cada vez mais, memórias de acesso aleatório (Random Access Memory - RAM), memórias programáveis somente de leitura (Programmable Read Only Memory - PROM), e estruturas lógicas programáveis (Programmable Logic Array - PLA), assim como multiplexadores e até microprocessadores são usados para implementar essas funções. Sem falar na versatilidade dos circuitos especificados pelo usuário (Custom Made Logic Circuit - CMLC) capazes de realizar qualquer função desejada de suas entradas.

Assim, com o aumento da complexidade dos sistemas digitais, o tempo e o custo de seus projetos cresceram enormemente. Para encurtar este tempo e minimizar os custos e erros , a automatização dos projetos adquiriu importância vital.

Nestes níveis, volta a ser importante o tratamento de minimização das funções lógicas envolvidas.

Nesta área de minimização de funções Booleanas foram desenvolvidos diversos trabalhos no sentido de buscar um espectro, através de transformadas, onde fosse mais simples trabalhar com as funções.

Um exemplo é a transformada de Rademasher-Walsh [14,6], em cujo espectro são definidas cinco operações algébricas básicas

que podem representar de forma bastante concisa as funções Booleanas.

Muitas vezes torna-se vantajoso também trabalhar não somente com funções binárias, mas com uma álgebra multivalores, o que leva à procura de métodos de tratamento e minimização destas funções, como o desenvolvido por Sasao [15], que é bastante similar ao de Quine-McCluskey [3], só que para a álgebra de multivalores.

I.4.- CIRCUITOS COMBINACIONAIS :

O principal ponto (não tecnológico) a ser destacado no estudo dos circuitos combinacionais é a sua minimização, ou seja, a minimização das funções Booleanas relacionadas a estes circuitos, pois é o que consome mais tempo dos projetistas e é a questão mais relevante para a otimização dos circuitos.

Quase toda a literatura concentra-se neste tópico e vale ressaltar que depois de 1975 houve um grande número de publicações na área, devido à mudança de perfil do usuário de circuitos digitais.

Em 1975, exatamente, REUSCH [18] apresenta um novo método para minimização de funções Booleanas, em dois passos:

O primeiro passo consiste em computar a lista de todos os implicantes primos da função. Muitos métodos para tal são conhecidos, quase todos baseados no método de combinação dos mintermos utilizado por McCluskey, conforme apresentado em [3]. Reusch introduz um novo, que começa com a idéia de dependência dos implicantes primos de uma função quanto aos implicantes primos das suas subfunções com respeito a uma variável arbitrariamente fixada, idéia esta não presente na teoria usada por McCluskey e suas derivações;

O segundo passo é formar a soma irredutível dos implicantes primos. McCluskey resolveu o problema usando uma matriz de cobertura. Porém é preciso saber a lista completa de todos os implicantes primos e de todos os mintermos da função para formar esta matriz.

Reusch obtém uma fórmula de cobertura usando a lista de todos os implicantes primos da função e uma representação

arbitrária da função como soma de produtos, isto é, uma lista qualquer de implicantes cuja soma é a função dada.

Em 1978, BESSLICH [8] critica o trabalho de Reusch devido ao número de implicantes primos crescer em demasia, pois para uma função de N variáveis (com N múltiplo de 3), o número de implicantes primos pode chegar a (N)!/((N/3)!)**3, p. ex., para N=9 variáveis, podem ocorrer até 1680 implicantes primos.

Nestas circunstâncias, a geração de todas as formas irredundantes de uma função é irrelevante e raramente necessária. Ao contrário, devem ser buscadas soluções aproximadas para o problema da cobertura, pois nem sempre é necessário chegar-se a uma fórmula mínima, mas apenas a uma cobertura completa da função.

Em seu trabalho, Besslich apresenta uma técnica para o problema da cobertura usando a transformada de Walsh-Hadamard, que já havia sido aplicada na obtenção dos implicantes primos.

O trabalho pesquisado que despertou maior interesse de implementação em computador foi o de CARUSO [17], datado de 1984, e portanto, bastante recente. Neste trabalho é apresentado um método para obtenção dos implicantes primos e minimização da função com base numa seleção local desses implicantes.

A idéia básica é localizar um mintermo e a partir dele obter a maior célula que o contenha (Mapas de Karnaugh), e assim para todos os mintermos que ainda não fazem parte de uma célula.

1.5.- CIRCUITOS SEQUENCIAIS :

O uso da teoria de chaveamento como método de análise e síntese de circuitos lógicos sequenciais está bastante difundido entre os engenheiros projetistas.

Primeiramente, o modelamento de um circuito sequencial pode ser feito por um diagrama de estado. Tal diagrama define as transições internas de estado e suas saídas quando submetidos a mudanças na entrada.

A partir do diagrama é construida uma "tabela inicial de estados ". Geralmente, o número de variáveis de estado nesta tabela pode ser reduzido e uma redução de estados sistemática pode ser implementada neste ponto.

Cada variável de estado na "tabela reduzida" é representada por um código binário único (alocação de estados) e a "tabela de estados alocados" resultante é analisada com o objetivo de definir as " funções de transições internas " e as " funções de saída". A partir destas funções é possível a realização física do circuito e as funções podem ser implementadas usando elementos lógicos padrões, como gates.

O problema da alocação inicial de estados é bastante relevante no que diz respeito ao desempenho e custos dos circuitos finais.

O número de alocações iniciais possíveis é bastante grande para permitir uma enumeração completa de todas elas como um método viável de solução. Um grande número de procedimentos para a obtenção de alocações iniciais de estados "boas" ou "quase ótimas", têm sido propostos nesses últimos anos.

P.K.LALA [10] apresenta um método sistemático de alocação de estados que garante o uso do menor número de flip-flops tipo D como elementos de memória, e garante também uma lógica de excitação razoavelmente econômica associada a estes flip-flops. O método se baseia na idéia de "peso" dos estados para determinar a alocação inicial das variáveis de estado ("Peso" de um estado é o número total de vezes que ele aparece como "próximo estado" numa dada tabela de estados) .

BENNETTS et al. [11] apresentam um algoritmo para a redução de tabelas de estado (completa ou incompletamente especificadas) com uma solução bastante rápida. Com o uso de diversos procedimentos heurísticos, é garantida uma solução quase mínima.

Equações de transição interna são usadas também como método de análise e síntese de circuitos sequenciais. Um exemplo desta técnica é proposto por PUCKNELL [9], que tem o mérito de permitir o estudo de tempos de atraso e a manipulação dos tempos de relógio, se desejado.

A sistematização e automação do projeto de circuitos digitais é extremamente conveniente pois retira do projetista as tarefas cansativas, sujeitas a erros e demoradas, para permitir-lhe uma análise crítica das possíveis soluções. E neste contexto que é apresentado na Seção IV.4 o programa TABELA.

I.6.- DETEÇÃO E VERIFICAÇÃO DE ERROS :

Para determinar se um circuito digital funciona corretamente, pode-se aplicar a ele todas as combinações de entradas possíveis e comparar as saídas resultantes com as saídas esperadas correspondentes, obtidas por exemplo de uma versão sem falhas do mesmo circuito ou através de simulação do circuito dado num computador (deteção).

Qualquer discrepância indica a presença de falhas. Ainda, se se dispuser de uma tabela mostrando que tipo de falhas geram cada tipo de discrepância, pode-se distinguir imediatamente qualquer falha das outras, pelo menos dentro de um subconjunto de falhas cujos efeitos nas saídas do circuito são idênticos (diagnóstico).

Esse procedimento é válido para todos os tipos de circuitos digitais - combinacionais ou sequenciais, de saída múltipla ou não, etc. - e para todos os tipos de falhas que tenham efeitos permanentes no comportamento do circuito examinado.

Porém, esta procura exaustiva é normalmente longa demais para ser prática e, com exceção de pouquíssimos casos, não é necessária. Geralmente é possível verificar a existência de falhas e até localizá-las no circuito por um conjunto de testes muitas ordens de grandeza menor que o exaustivo.

Deve-se observar ainda que o conjunto de testes pode ser reduzido se forem desenvolvidos procedimentos para selecionar os pontos de teste mais convenientes nos circuitos, ou então para inserir entradas adicionais de controle; ou para projetar-se os circuitos de modo a facilitar a deteção e prevenção de erros [12,13].

WILLIANS & PARKER [7] publicam em janeiro de 1983 um estudo bastante completo sobre o estado da arte em "projeto e testabilidade". Este estudo aborda diversos tópicos de interesse neste contexto e fornece mais de 100 referências bilbiográficas aos pesquisadores desta área.

SELLERS et al. [16] dão a definição de diferenciação Booleana e mostram com exemplos como ela é usada para analisar o efeito dos erros nas saídas de circuitos lógicos. Há exemplos de deteção de falhas, análise de lógica redundante e geração de sequências de teste.

EDWARDS [5] define um operador diferenciador diretamente relacionado com a diferenciação Booleana e mostra como utilizá-lo para derivar com facilidade qualquer função lógica no domínio da transformada de Hadamard. Ressalta ainda no seu trabalho a importância e o uso do operador diferenciador no campo de diagnóstico de falhas e na síntese dos circuitos digitais.

E importante observar que, com o uso das técnicas de diferenciação Booleana, a análise de erros fica reduzida à simplificação e minimização de funções Booleanas.

I.7.- SIMULAÇÃO :

Um assunto importante na área e que deve ser avaliado é o da simulação de circuitos em computador como ferramenta de análise de circuitos lógicos, o que está bastante difundido atualmente nos campos comercial e de pesquisa, no contexto de projeto de circuitos digitais.

Analisando este contexto, hoje, observa-se que há uma tendência cada vez maior no sentido de se projetar circuitos de forma integrada e de acordo com as especificações do usuário, o que leva a circuitos cada vez mais complexos .

Baseado nesta constatação, pode-se afirmar que a simulação de circuitos digitais representa cada vez mais uma diminuição no custo e no tempo de projeto pois permite analisar o comportamento do circuito, verificando o seu funcionamento e a existência de falhas de projeto, antes de sua implementação. Isto torna mais simples a tarefa de se realizar as modificações necessárias no projeto, em relação aos circuitos já implementados.

I.8.- CONCLUSÃO :

No modo de ver do autor, em meados da década de 70 houve um retorno à pesquisa na Análise e Síntese de Circuitos Digitais. Isto ocorreu devido à mudança no perfil do usuário que, em parte, passou de projetista de circuitos usando integrados a projetista de integrados propriamente.

Neste capítulo foram citados e comentados alguns dos artigos publicados nesta área de pesquisa entre 1975 e 1986. Todos estes artigos são citados na seção seguinte.

Este trabalho, como um todo, se insere no contexto de pesquisa e implementação de ferramental eficiente para o auxílio ao projeto de circuitos digitais integrados. Sua principal motivação foi o esforço de centros de pesquisa como o CPqD/Telebrás e o CTI, e de algumas empresas na direção de nacionalizar o projeto e a fabricação destes circuitos.

1.9.- REFERENCIAS BIBLIOGRAFICAS :

1.- KOHAVI , Zvi

"Switching and Finite Automata Theory"

McGraw-Hill , 1970 - 592 p.

2.- HILL, F. J. & PETERSON, G. R.

"Introduction to Switching Theory & Logical Design"

2ed.Wiley International Edition, 1974 - 596 p.

3.- MADUREIRA, Marcos C.

"Síntese de Circuitos Digitais.- Minimização de funções Booleanas " - Relatório Interno n. 015/86

FEC - UNICAMP - Campinas.

4.- COSTA, E. M.

"Projeto de Circuitos Integrados"

I Escola Brasileiro-Argentina de Informática - EBAI

Editorial Kapelusz S/A - Fevereiro de 1987

5.- EDWARDS, C. R.

"The Generalised Dyadic Differentiator and its Application to 2-valued Functions defined on an N-Space"

Computers and Digital Techniques, Vol. 1, n. 4,

October 1978 - pp 137-142

6.- EDWARDS, C. R.

"The Application of the Rademacher-Walsh Transform to Boolean Function Classification and Threshold Logic Synthesis"

IEEE Transactions on Computers, Vol. C-24, n. 1,

January 1975 - pp 48-64

7.- WILLIANS, T. & PARKER, K.

"Design for Testability - A Survey"

Proceedings of the IEEE, Vol. 71, n.1

January 1983 - pp.98-112

Special Issue on VLSI Design

8.- BESSLICH, P. W.

"Determination of the Irredundant Forms of a Boolean Function using WALSH-HADAMARD Analysis and Dyadic Groups"

Computers and Digital Techniques, Vol. 1, n. 4

October 1978, pp 113-151

9.- PUCKNELL, D. A.

"Transition Equations for the Analysis and Synthesis of Sequential Circuits "

Electronic Letters, Vol. 6, n. 23,

November 1970, pp 731-733

10.- LALA, P. K.

"An Algorithm for the State Assignment of Syncronous Sequential Circuits "

Electronic Letters, Vol. 14, n. 6

March 1978, pp 199-201

11.- BENNETTS, R. G. et al.

"A Computer Algorithm for State Table Reduction"

The Radio and Electronic Engineer, Vol.42, n.11

November 1972 - pp 513-520

12.- KHAKBAZ, JAVAD

"A Testable PLA Design with low Overhead and high Fault Coverage"

IEEE Transactions on Computers, Vol.C33, n.8

August 1984 - pp. 743-745

13.- KHAKBAZ, J. & McCLUSKEY, E. J.

"Concurrent Error Detection and Testing for large PLA's"

IEEE Journal of Solid State Circuits, Vol.SC17, n.2

April 1982 - pp. 386-394

14.- LOYD, A. M.

"Spectral Addition Techniques for the Synthesis of Multivariable Logic Networks"

Computer and Digital Techniques, Vol.1, n.4

October 1978 - pp. 152-164

15.- SASAO, T.

"Input Variable Assignment and Output Phase Optimization of PLA's"

IEEE Transactions on Computers, Vol. C33, n.10

October 1984 - pp.879-894

16.- SELLERS, F. F. et al.

"Analyzing Errors with the Boolean Difference"

IEEE Transactions on Computers, Vol.C17, n.7

July 1968 - pp. 676-683

17.- CARUSO, G.

"A Local Selection Algorithm for Switching Function Minimization"

IEEE Transactions on Computers, Vol. C33, n.1

January 1984 - pp.91-97

18.- REUSCH, B.

"Generation of prime implicants from Subfunctions and a Unifying Approach to the Covering Problem"

IEEE Transactions on Computers, Vol. C24, n.9

September 1975 - pp. 924-930

CAPITULO 2

CIRCUITOS COMBINACIONAIS

II.1.- RESUMO :

Neste capítulo definem-se as funções booleanas afim de descrever as propriedades dos circuitos combinacionais. A característica particular de um circuito combinacional é que suas saídas são funções unicamente de suas entradas presentes.

Através de postulados básicos apresenta-se a álgebra de Boole como ferramenta matemática essencial ao estudo de circuitos lógicos. Teoria de conjuntos e de proposições ilustram outras de suas aplicações.

São introduzidos conceitos e notações utilizados em todo o trabalho.

II.2.- ALGEBRA DE BOOLE :

Os conceitos básicos da álgebra de Boole serão introduzidos como um conjunto de postulados dos quais derivam teoremas importantes, que se constituem nas ferramentas necessárias à manipulação e simplificação de expressões algébricas.

A maior parte destes conceitos estão nas obras de Kohavi[1], Peterson[2] e Yuzo[3]. A intenção de colocá-los aqui é de tornar a leitura dos demais capítulos auto-suficiente.

II.2.1.- POSTULADOS FUNDAMENTAIS

O postulado básico da álgebra de Boole é a existência de uma variável booleana tal que :

$x \neq 0 \iff x = 1$

$x \neq 1 \iff x = 0$

A álgebra de Boole é um sistema algébrico que consiste do conjunto (0,1), duas operações binárias chamadas **OR (+)** e **AND (.)** e uma operação unária chamada **NOT** denotada por um apóstrofo (') ou por um til (~) sobre a variável.

A operação OR é chamada de **soma lógica** ou **união**, a operação AND é conhecida por **produto lógico** ou **interseção** e a operação NOT é dita **complementação** . Estas operações são definidas da seguinte forma :

OR	AND	NOT
0 + 0 = 0	0 . 0 = 0	0' = 1
0 + 1 = 1	0 . 1 = 0	1' = 0
1 + 0 = 1	1 . 0 = 0	
1 + 1 = 1	1 . 1 = 1	

II.2.2.- PROPRIEDADES BASICAS

Se x é uma variável booleana, então :

$$x + 1 = 1 \qquad (II.1)$$

$$x \cdot 0 = 0 \qquad (II.2)$$

$$x + 0 = x \qquad (II.3)$$

$$x \cdot 1 = x \qquad (II.4)$$

A primeira propriedade que difere bastante da álgebra de números reais é:

$$x + x = x \qquad (II.5)$$

$$x \cdot x = x \qquad (II.6)$$

A álgebra booleana também é comutativa e associativa em relação às duas operações binárias. Os parênteses são colocados da mesma forma que na álgebra convencional, ou seja, x+y.z é x+(y.z) e não (x+y).z . Sendo x, y, z variáveis booleanas, então

-Comutativa

$$x + y = y + x \qquad (II.7)$$

$$x \cdot y = y \cdot x \qquad (II.8)$$

-Associativa

$$(x + y) + z = x + (y + z) \qquad (II.9)$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \qquad (II.10)$$

-Complemento

$$x + x' = 1 \qquad (II.11)$$

$$x \cdot x' = 0 \qquad (II.12)$$

Na álgebra booleana, a soma é distributiva sobre o produto e o produto é distributivo sobre a soma,

-Distributiva

$$x \cdot (y + z) = x.y + x.z \qquad (II.13)$$

$$x + (y \cdot z) = (x + y) \cdot (x + z) \qquad (II.14)$$

Todas estas propriedades podem ser provadas por indução perfeita. Observa-se, ainda, que todas elas se apresentam aos

pares, e que, em cada par, uma equação pode ser obtida da outra trocando-se as constantes 0 por 1 e 1 por 0, e as operações AND por OR e OR por AND . Isto é conhecido como princípio da dualidade da álgebra de Boole. A implicação disto é que é necessário provar apenas uma das equações que sua dual estará provada.

II.2.3.- EXPRESSOES BOOLEANAS

Define-se **expressão booleana** como a combinação de um número finito de variáveis booleanas e constantes (0,1) através de operações booleanas (+ , . , ´).

Qualquer constante ou variável booleana ó uma expressão booleana, e se T1 e T2 são expressões booleanas, também o são T1´, T2´, T1+T2, T1.T2.

As propriedades (II.15) a (II.20) a seguir formam o conjunto de ferramentas básicas para a simplificação de expressões booleanas. Elas estabelecem a noção de redundância e aparecem em pares duais :

Absorção

$$x + xy = x \qquad \text{(II.15)}$$

$$x(x + y) = x \qquad \text{(II.16)}$$

$$x + x'y = x + y \qquad \text{(II.17)}$$

$$x(x' + y) = xy \qquad \text{(II.18)}$$

Consenso

$$xy + x'z + yz = xy + x'z \qquad \text{(II.19)}$$

$$(x + y)(x' + z)(y + z) = (x + y)(x' + z) \qquad \text{(II.20)}$$

OBS.: Sempre que não deixar dúvidas, é utilizada a forma xy no lugar de x.y .

Estas propriedades permitem uma variedade imensa de manipulações nas expressões booleanas. Sempre que possível elas permitem converter uma expressão em outra equivalente, com menos literais, onde **literal** é o aparecimento de uma variável ou de seu complemento. Por exemplo, o lado esquerdo da equação (II.19) tem 6 literais, enquanto o lado direito tem apenas 4. Se o valor de uma expressão independe do valor de uma variável x, x é dito **redundante**. Assim, estas propriedades podem eliminar literais redundantes numa dada expressão, simplificando-a.

Exemplo : Simplificar, por eliminação de literais redundantes, a expressão

$$\begin{aligned} T(x,y,z) &= x'y'z + yz + xz \\ &= z(x'y' + y + x) \\ &= z(x' + y + x) \\ &= z(y + 1) \\ &= z.1 = z \end{aligned}$$

Ou seja, T(x,y,z) independe dos valores de x e y, dependendo apenas de z .

E importante ressaltar que não são definidas operações inversas na álgebra de Boole, e consequentemente, não são permitidos cancelamentos. Por exemplo, se A + B = A + C, não está implícito que B = C. De fato, se A = B = 1 e C = 0, 1+1 = 1+0, mas B ≠ C . Analogamente, AB = AC não implica em B = C .

TEOREMA DE DE MORGAN :

As regras que regem a operação de complementação se resumem nos três teoremas seguintes:

$$(x')' = x \qquad \text{(II.21)}$$

$$(x + y)' = x'.y' \qquad \text{(II.22)}$$

$$(x.y)' = x' + y' \qquad \text{(II.23)}$$

As duas últimas equações são conhecidas como **"teoremas de De Morgan"** para duas variáveis. De forma geral, este teorema afirma que o complemento de qualquer expressão pode ser obtido trocando-se simultaneamente cada variável por seu complemento e as operações OR por AND e AND por OR, ou seja:

$$[f(x1,x2,...,xn,0,1,+,.)]' = f(x1',x2',...,xn',1,0,.,+) \quad (II.24)$$

II.2.4.- APLICAÇOES

Nesta seção são apresentadas as relações entre álgebra booleana, circuitos lógicos, cálculo de proposições e teoria de conjuntos.

Dois sistemas algébricos, cada um consistindo de um conjunto de elementos e uma ou mais operações que satisfazem um dado conjunto de postulados, são ditos isomórficos[1] se forem satisfeitas as condições seguintes .

.- Para cada operação em um sistema existe uma operação correspondente no segundo sistema.

.- A cada elemento Xi de um sistema corresponde um único elemento Yi no segundo sistema e vice-versa.

.- Se em cada postulado do primeiro sistema, cada Xi for trocado pelo Yi correspondente, e cada operação for trocada pela operação correspondente do segundo sistema, então o postulado resultante deve ser válido para o segundo sistema.

Em outras palavras, dois sistemas algébricos são isomórficos se e somente se eles forem idênticos com exceção dos símbolos usados para representar seus elementos e operações.

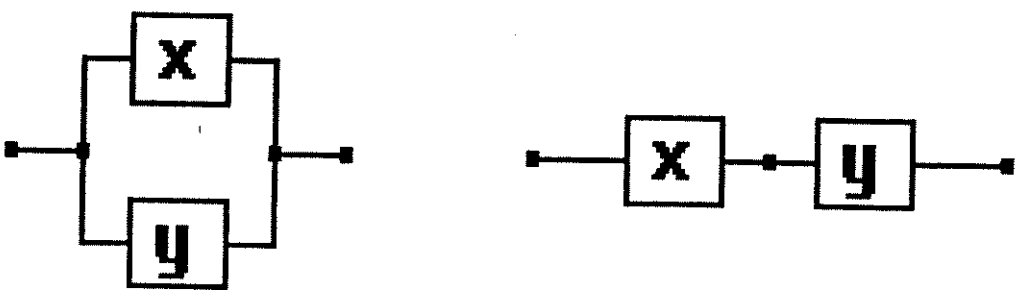
CIRCUITOS LOGICOS (Switching Circuits)

Um circuito lógico consiste, por exemplo, de "gates" através dos quais passa informação dada por sinais elétricos. Um "gate" é um elemento de dois estados, capaz de comutar de um estado que permite a passagem da informação para outro estado que bloqueia a informação e vice-versa.

Pode-se associar a cada gate uma variável binária que aparece na forma não complementada se o gate permite a passagem da informação e complementada caso contrário.

Se dois gates operam sempre nos mesmos estados, serão denotados pela mesma variável. Se eles operam de maneira oposta, isto é, quando um permite a passagem da informação o outro bloqueia e vice-versa, eles serão denotados por variáveis complementares. Se um gate permite a passagem da informação, a variável a ele associada assume o valor 1 e caso contrário 0.

A conexão paralela de dois gates x e y é denotada por x+y e sua conexão série é dada por xy, como na figura II.1 .



a) Paralela (x+y) b) Série (xy)

Figura II.1.- Conexões básicas de dois gates.

Associada a cada circuito existe uma **função de transmissão** [1], que assume o valor 1 sempre que existir um caminho de um terminal a outro, através do qual passa a informação. Diz-se que a função de transmissão representa o circuito , ou que o circuito realiza a função de transmissão.

Para determinar as funções de transmissão dos circuitos da figura II.1, observa-se que existe um caminho em (a) se um dos gates ou ambos permitirem a passagem da informação, ou seja, a função é 1 se x ou y ou ambos forem 1. Da mesma forma em (b), a função é 1 somente se ambos os gates permitirem a passagem da informação, isto é, se x=y=1. A partir disto constrói-se a tabela II.1.

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

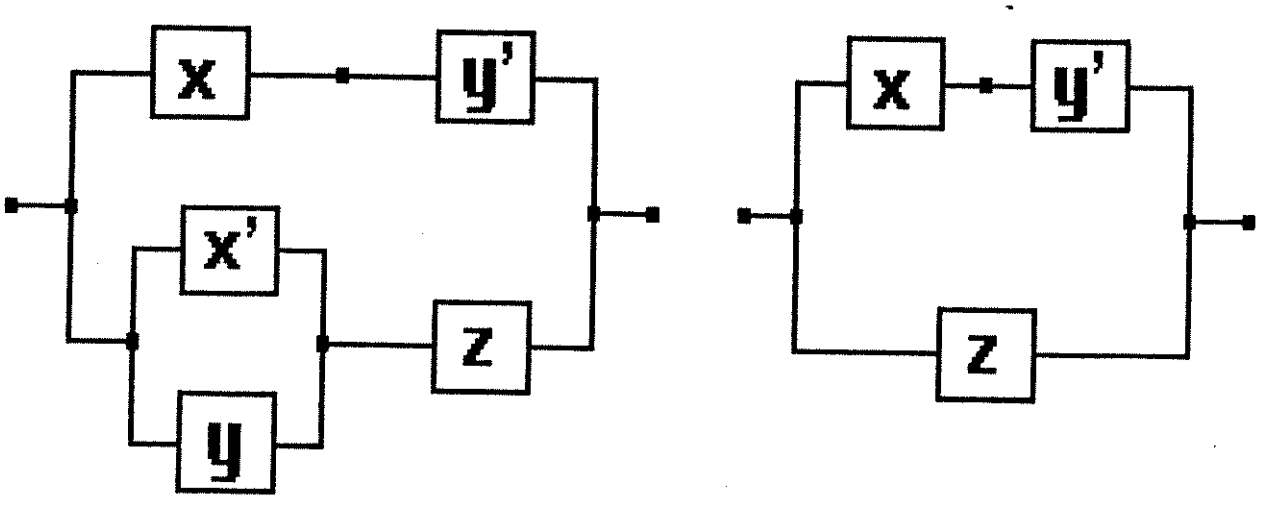
Tabela II.1.- Definição das funções de transmissão.

É evidente que existe uma analogia perfeita entre as operações OR e AND definidas na seção II.2.1 e as funções de transmissão dos circuitos paralelo e série respectivamente.

Assim, está claro que a álgebra de Boole e os circuitos lógicos são sistemas algébricos isomórficos, consequentemente, todas as propriedades da álgebra de Boole se aplicam a teoria de circuitos e podem ser usadas na análise e síntese de circuitos lógicos.

Exemplo : A função de transmissão do circuito da figura II.2.a é dada por $T = xy' + (x'+y)z$

Com manipulações algébricas chega-se à forma reduzida de II.2.b $T = xy' + z$



a) Original b) Simplificado

Figura II.2.- Simplificação de um circuito lógico.

CALCULO DE PROPOSIÇOES

Uma **proposição** é uma afirmação que pode ser falsa ou verdadeira, mas nunca ambos. A cada proposição associa-se uma variável, denotada por p,q,etc., que assume o valor 1 se a proposição é verdadeira, 0 se a proposição é falsa. Assim, a proposição 0 é sempre falsa e a proposição 1 é sempre verdadeira.

Uma proposição é uma negação de outra proposição, se quando uma é verdadeira a outra é falsa e vice-versa. Assim, a negação p' de uma proposição p é 1 se p for 0 e é 0 se p for 1. Por exemplo, " não está chovendo" é claramente uma negação de "está chovendo".

Duas proposições podem ser combinadas para formar uma nova proposição. Por exemplo, se p é "a temperatura está abaixo de 30 graus" e q representa "a umidade está acima de 50%", pode-se formar a proposição "a temperatura está abaixo de 30 graus e a umidade está acima de 50%" combinando-se p e q com o conectivo "e". A proposição pq é verdadeira se e somente se p e q forem verdadeiras.

As proposições p e q anteriores podem ser combinadas também com o conectivo "ou". Assim, "a temperatura está abaixo de 30 graus ou a umidade está acima de 50%" forma a proposição p+q. Esta proposição só é falsa quando p e q forem falsas.

As combinações das proposições são resumidas na tabela II.2.

p	q	p+q	pq
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Tabela II.2.- Definição das combinações das proposições.

Aqui aparece claramente a analogia entre cálculo de proposições e álgebra de Boole. De fato, eles são sistemas isomórficos. Consequentemente, pode-se falar em variáveis e funções exatamente da mesma forma.

EXEMPLO : Um sistema condicionador de ar de um armazém deve ser ligado se ocorrer uma ou mais das seguintes condições:

1.- O peso do material armazenado é menor que 100 ton, a umidade relativa do ar é maior que 60% e a temperatura está acima de 15 graus;

2.- O peso do material armazenado é 100 ton ou mais e a temperatura está acima de 15 graus;

3.- O peso armazenado é menor que 100 ton e o barômetro marca 30 ou mais.

Seja A a proposição "o ar-condicionado está ligado". O objetivo é expressar A em função das seguintes proposições :

W - peso maior ou igual a 100 ton.

H - umidade relativa maior ou igual a 60%.

T - temperatura acima de 15 graus.

P - pressão maior ou igual a 30.

Da condição 1, conclui-se que A=1 se W'HT for 1.

Da condição 2, conclui-se que A=1 se WT for 1.

Da condição 3, conclui-se que A=1 se W'P for 1.

Assim, uma expressão para A é:

$A = W'HT + WT + W'P$

que, simplificada fica,

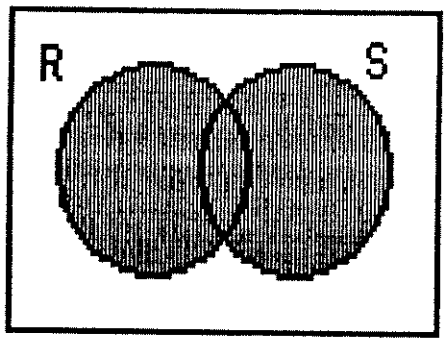
$A = HT + WT + W'P = T(H+W) + W'P$

Ou seja, o ar-condicionado será ligado se a temperatura estiver acima de 15 graus e ou o peso é maior ou igual a 100 ton ou a umidade é de pelo menos 60%, ou se o peso é menor que 100 ton e a pressão é maior ou igual a 30.

TEORIA DOS CONJUNTOS

Um conjunto pode ser visto como uma coleção de objetos. Para definir uma álgebra de conjuntos, deve-se limitar os objetos que constituem estes conjuntos de acordo com algum critério. Define-se como **conjunto universo** aquele que inclui todos os objetos que satisfazem este critério. Por exemplo, "os pontos de uma região limitada de um plano" e "as cores do arco-íris" são conjuntos universos contendo infinitos elementos e um número finito de elementos, respectivamente. O conjunto que contém zero elementos é o **conjunto vazio**. O conjunto universo será denotado por **U**, enquanto o conjunto vazio será **Z**.

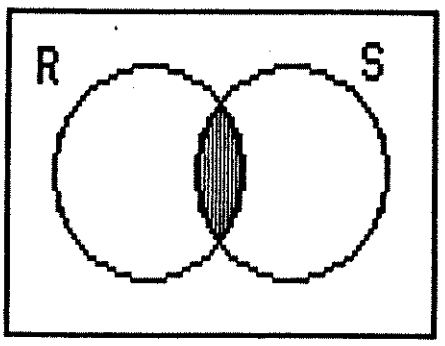
Define-se como união de dois conjuntos (R e S) o conjunto U(R,S) que contém todos os elementos pertencentes a S ou a R. Isto é ilustrado pela área escura na figura II.3 .



U(R,S)

Figura II.3.- União de conjuntos.

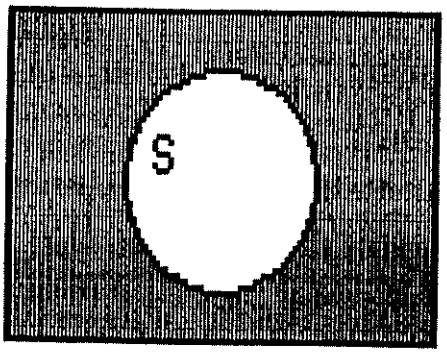
Outra operação definida é a intersecção de dois conjuntos (R e S) que é o conjunto I(R,S) contendo todos os elementos pertencentes a S e a R simultaneamente. Isto é mostrado pela área escura da figura II.4 .



I(R,S)

Figura II.4.- Intersecção de conjuntos.

Uma outra definição de operação é o complemento de um conjunto (S) que é o conjunto C(S) contendo todos os elementos pertencentes a W e não pertencentes a S, como mostra a parte escura da figura II.5 .



C(S)

Figura II.5.- Complemento de um conjunto.

A tabela II.3 mostra as analogias entre estas operações e os elementos da álgebra booleana, deixando claro o isomorfismo entre os dois sistemas. Com isto, pode-se simplificar expressões obtidas para os conjuntos da mesma forma como se fossem expressões booleanas.

Teoria dos Conjuntos	Álgebra de Boole
R	x
S	y
Z	0
W	1
U(R,S)	x+y
I(R,S)	x.y
C(S)	y'

Tabela II.3.- Isomorfismo entre teoria dos conjuntos e álgebra booleana.

EXEMPLO : Os senhores X,Y e Z colecionam velhos manuscritos. O Sr. X coleciona trabalhos políticos brasileiros e livros estrangeiros de ficção. O Sr. Y é um colecionador de trabalhos políticos, exceto ficção brasileira, e livros brasileiros desde que não sejam de ficção. O Sr. Z coleciona livros que não sejam de ficção, mas que sejam brasileiros ou estrangeiros e políticos. O problema consiste em determinar os livros para os quais há competição, ou seja, interesse de mais de um colecionador.

SOLUÇÃO : Primeiramente definem-se os conjuntos envolvidos :

X = conjunto de livros colecionados pelo Sr. X

Y = conjunto de livros colecionados pelo Sr. Z

Z = conjunto de livros colecionados pelo Sr. Z

B = conjunto de todos os livros brasileiros

F = conjunto de todos os livros de ficção

P = conjunto de todos os livros políticos

O conjunto procurado é S = U(I(X,Y),I(X,Z),I(Y,Z')), ou em linguagem booleana,

S = XY + XZ + YZ

Traduzindo o enunciado do problema para linguagem de álgebra booleana e simplificando,

X = BP + B'F

Y = P(BF)' + BF' = P(B'+F') + BF'

= PB' + PF'(B+B') + BF' = PB' + PF'B' + PBF' + BF'

= PB'(1+F') + (P+1)BF' = PB' + BF'

Z = (B+B'P)F' = (B+P)F'

Assim sendo,

S = (BP+B'F)(PB'+BF') + (BP+B'F)(B'F'+PF') + (PB'+BF')(BF'+PF')

Que, após algumas simplificações chega a

S = BF' + PB'

ou seja, há competição pelos livros brasileiros não-ficção e livros políticos estrangeiros.

II.3.- FUNÇÕES BOOLEANAS :

II.3.1.- DEFINIÇÃO

Seja T(x1,x2,...,xn) uma expressão booleana. Como cada uma das n variáveis x1...xn pode independentemente assumir um dos dois valores 0 ou 1, existem 2^n combinações de valores a serem considerados na determinação dos valores de T. Para determinar o valor de uma expressão para uma dada combinação de valores, basta substituir os valores das variáveis na expressão. Por exemplo, para T(x,y,z) = x'z + xz' + x'y', se x=0, y=0 e z=1, temos :

$$\begin{aligned} T(0,0,1) &= 0'.1 + 0.1' + 0'.0' \\ &= 1.1 + 0.0 + 1.1 \\ &= 1 + 0 + 1 = 1 \end{aligned}$$

Da mesma forma, os valores de T podem ser calculados para todas as combinações possíveis de x, y e z como na tabela II.4.

x	y	z	T
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Tabela II.4.- Tabela verdade para T(x,y,z) = x'z + xz' + x'y'.

Se este procedimento for repetido para se construir a tabela verdade da expressão x'z + xz' + y'z', será obtida uma tabela idêntica à tabela II.4 . Isto é, expressões booleanas diferentes

podem ser representadas pela mesma tabela verdade. Os valores assumidos por uma expressão para todas as combinações possíveis de suas variáveis definem uma função booleana.

Em outras palavras, uma **função booleana** $f(x1,...,xn)$ é a correspondência que associa um elemento da álgebra de Boole com cada uma das 2^n combinações das variáveis $x1,...,xn$. Uma boa maneira de expressar tal correspondência é através de uma tabela verdade. Vale observar que cada tabela verdade define apenas uma função, embora esta função possa ser expressa por diferentes expressões booleanas.

II.3.2.- FORMAS CANONICAS

Na seção anterior, as funções booleanas foram representadas através de tabelas verdade. Uma expressão que represente uma função booleana é derivada de sua tabela encontrando a soma de todos os termos para os quais a função assume o valor 1. Cada termo é um produto de variáveis da função. Uma variável xi aparecerá neste produto na forma complementada se o seu valor for 0 nesta combinação, e na forma não complementada se seu valor for 1.

Por exemplo, a função definida na tabela II.4 assume valor 1 para a combinação x=y=z=0 (linha 1) assim, o termo $x'y'z'$ aparece na fórmula da função. De fato, para a tabela II.4, pode-se definir $f(x,y,z) = x'y'z' + x'y'z + x'yz + xy'z' + xyz'$.

Um produto que contém todas as n variáveis de uma função como fatores, complementadas ou não, é chamado de **mintermo**. Sua característica particular é que assume o valor 1 para somente uma combinação de valores das variáveis. A soma de todos os mintermos derivados das linhas da tabela onde a função assume o valor 1,

assumirá os valores 0 ou 1 da mesma forma que a função. Portanto, esta soma é realmente uma representação algébrica da função. Uma expressão deste tipo é chamada de **soma de produtos canônica**.

As funções booleanas são normalmente expressas numa forma compacta, obtida pelos códigos decimais associados aos mintermos para os quais a função vale 1 . Estes decimais são derivados das tabelas verdades tratando-se cada linha como um número binário, por exemplo o mintermo x'yz é associado à linha 011 que, interpretada como um número binário, é igual a 3. Assim, a função definida pela tabela II.4 pode ser expressa como

$$f(x,y,z) = S(0,1,3,4,6)$$

onde S( ) significa que f(x,y,z) é a soma de todos os mintermos cujos códigos decimais são dados entre os parênteses.

Uma função booleana também pode ser expressa por um produto de somas. Este pode ser obtido considerando-se os termos para os quais a função assume o valor 0. Cada termo é uma soma de variáveis da função. Uma variável xi aparecerá nesta soma na forma complementada se o seu valor for 1 nesta combinação, e na forma não complementada se seu valor for 0.

Por exemplo, a função definida na tabela II.4 assume valor 0 para a combinação x=y=z=1 (linha 7); assim, o termo x'+y'+z' aparece na fórmula da função. De fato, para a tabela II.4, pode-se definir f(x,y,z) = (x+y'+z).(x'+y+z').(x'+y'+z') .

Uma soma que contém todas as n variáveis de uma função como fatores, complementadas ou não, é chamada de **maxtermo**. O produto de todos os maxtermos derivados das linhas da tabela onde a

função assume o valor 0, assumirá os valores 0 ou 1 da mesma forma que a função. Portanto, este produto também é uma representação algébrica da função e é chamado de **produto de somas canônico.**

Também como produto de somas, as funções booleanas podem ser expressas numa forma compacta, obtida pelos códigos decimais associados aos maxtermos para os quais a função vale 0 .

Assim, a função definida pela tabela II.4 pode ser expressa também na forma

$$f(x,y,z) = P(2,5,7)$$

onde P( ) significa que f(x,y,z) é o produto de todos os maxtermos cujos códigos decimais são dados entre os parênteses.

Existem situações onde, enquanto a função assume o valor 1 para algumas combinações de entradas e 0 para outras, pode assumir qualquer valor para algumas combinações de entrada. Estas situações podem ocorrer, por exemplo, quando as variáveis não são mutuamente independentes, ou seja, dependência entre as variáveis pode impedir a ocorrência de certas combinações, para as quais o valor da função não é especificado. Estas combinações das entradas são chamadas de **"Don't Care States"** ou **Estados irrelevantes**. O valor da função para tais combinações é denotado por um **X** ou um traço **(-)** e estas combinações aparecem nas formas compactas como D( ).

Por exemplo :

$$F(w,x,y,z) = S(4,5,8,12,13) + D(3,14,15)$$

W	x	y	z	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	X
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0

W	x	y	z	F
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	X
1	1	1	1	X

Tabela II.5.- Tabela verdade para $F(w,x,y,z) = S(4,5,8,12,13) + D(3,14,15)$

II.3.3.- CUSTOS

Definidas as formas de representação das funções, e tendo em vista a simplificação das expressões através dos teoremas definidos na seção II.2, cabe agora definir que direção seguir quando se fala em simplificação. O objetivo do projetista de circuitos digitais vai no sentido de "baratear" o produto. Assim, deve-se deixar claro o critério de custo levado em conta.

Os critérios mais comuns de custo mínimo de uma dada função booleana são:

1.- Menor número de literais. (Lembrando que uma literal é uma variável na forma complementada ou não).

2.- Menor número de literais numa expressão do tipo soma de produtos (ou produto de somas).

3.- Menor número de termos numa expressão do tipo soma de produtos (ou produto de somas), sendo que não exista nenhuma

outra expressão com o mesmo número de termos e menor número de literais.

Neste trabalho é adotado o critério 3, e para as expressões mínimas das funções é adotada a forma de soma de produtos.

Algumas vezes, porém, a forma de produto de somas pode ser mais vantajosa e a função deve ser minimizada a partir de seus maxtermos. Entretanto, de acordo com este critério de custos e por aplicação do teorema de De Morgan (II.24), o custo obtido a partir dos maxtermos é equivalente ao custo obtido a partir dos mintermos da função negada.

EXEMPLO : Retomando-se a função dada pela tabela II.4,

$$f(x,y,z) = S(0,1,3,4,6)$$

Escrevendo-a na forma de soma de produtos para uma primeira avaliação do seu custo, tem-se:

$f(x,y,z) = x'y'z' + x'y'z + x'yz + xy'z' + xyz'$; CUSTO=20

Com alguma manipulação obtém-se, para a mesma função,

$f(x,y,z) = x'y' + x'z + y'z' + xz'$; CUSTO = 12

Um pouco mais de simplificação e chega-se a sua fórmula mínima :

$f(x,y,z) = x'y' + x'z + xz'$; CUSTO = 09

A função custo pode ser interpretada como equivalente a contagem dos terminais de entrada dos gates envolvidos na realização da função, sem se contar os inversores. Na forma final dessa função, pode-se observar que serão necessários 3 AND de 2 entradas e um OR de 3 entradas, isto é:

$$3x2 + 1x3 = 6+3 = 9$$

II.3.4.- MAPA DE KARNAUGH :

Um **mapa de Karnaugh** é uma forma modificada da tabela verdade, na qual as combinações das entradas estão arranjadas de uma forma particularmente conveniente. Os mapas para funções de 2 a 5 variáveis são mostrados na figura II.6. O mapa de Karnaugh para funções de 6 variáveis está na figura II.7. Na parte externa dos mapas representam-se as variáveis e as combinações de seus valores. Cada mapa de **n** variáveis consiste de 2^n **células** (quadrados), representando todas as possíveis combinações dessas variáveis. Os códigos decimais correspondentes a essas combinações também estão nas figuras II.6 e II.7.

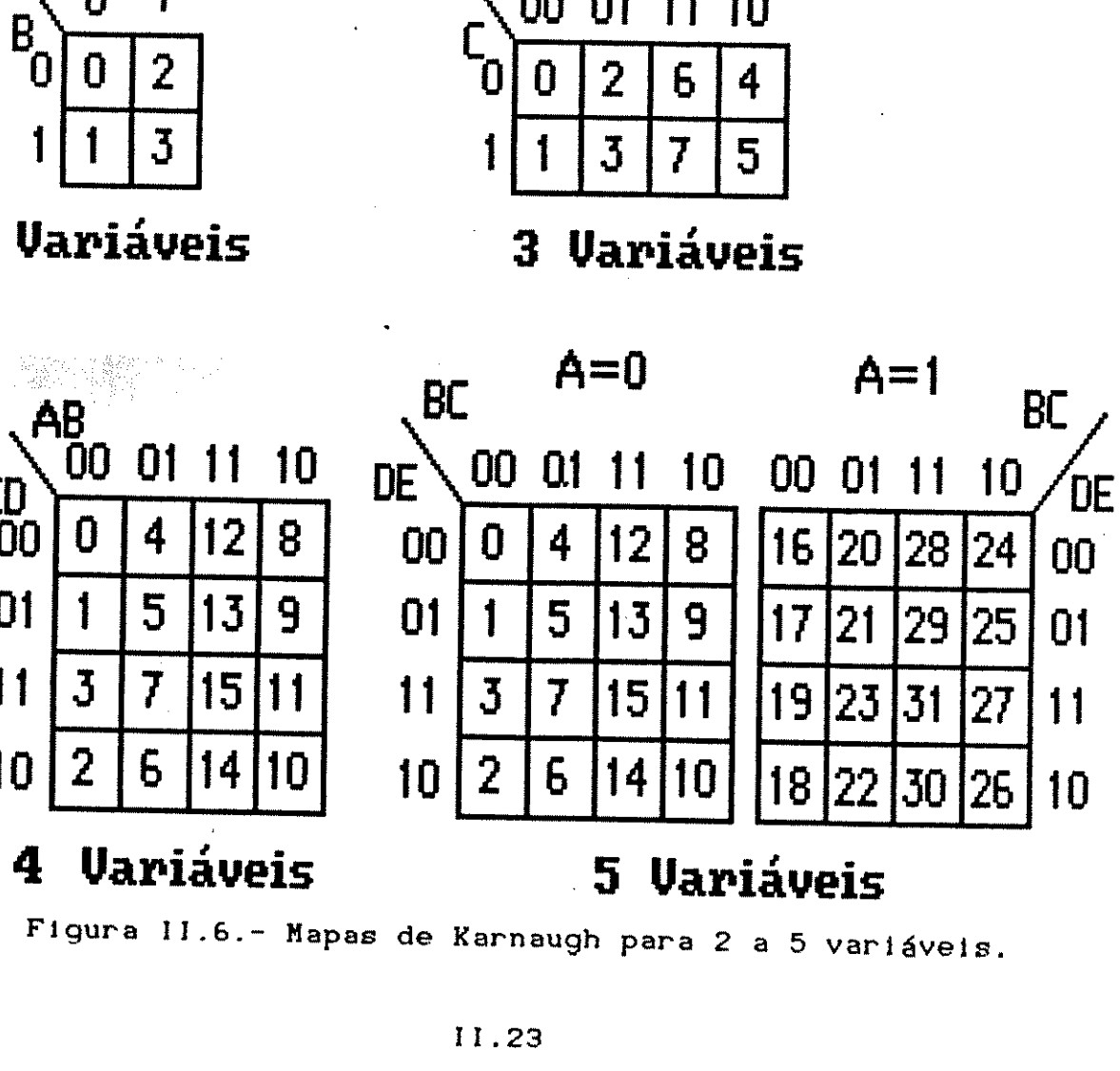
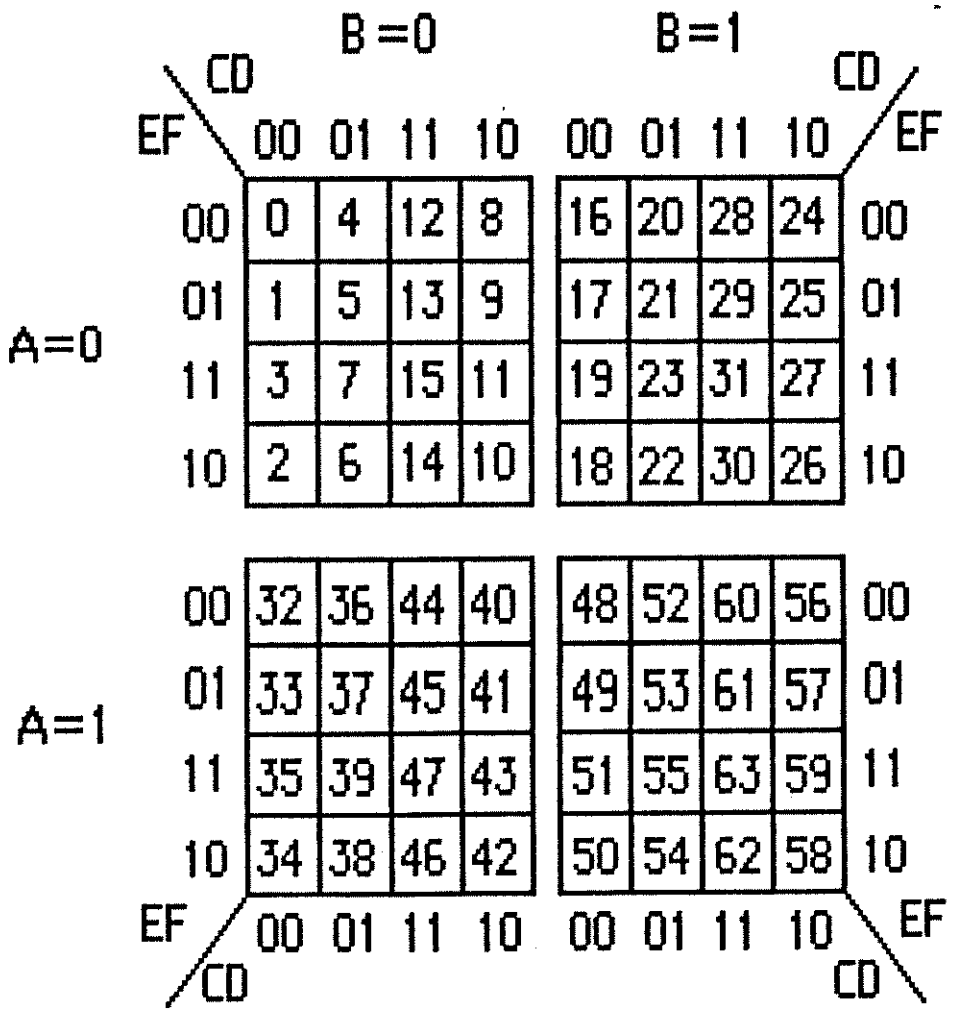


Figura II.6.- Mapas de Karnaugh para 2 a 5 variáveis.



6 Variáveis

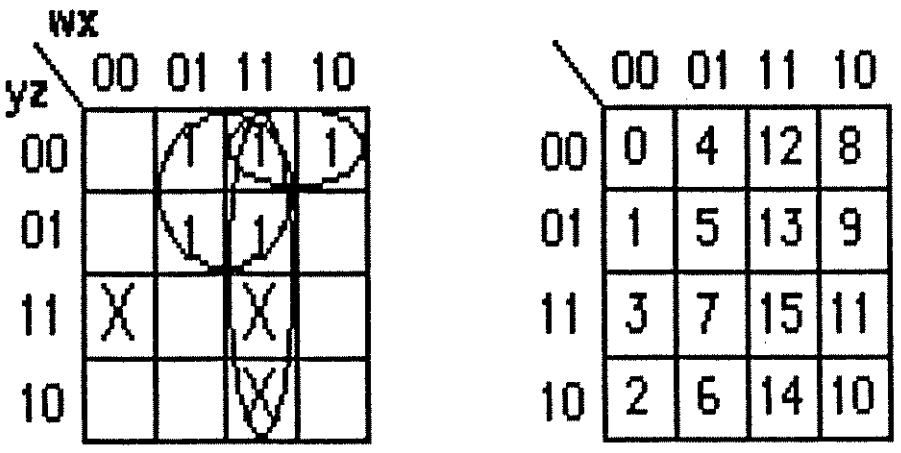
Figura II.7.- Mapa de Karnaugh para 6 variáveis.

O valor da função associado a uma combinação particular das entradas é anotado na célula correspondente. Por exemplo, a figura II.8 mostra o mapa para a função :

$$F(w,x,y,z) = S(4,5,8,12,13) + D(3,14,15)$$

Note que o valor 1 está nas células 4,5,8,12,13, o valor X nas células 3, 14 e 15. As células em branco correspondem às combinações para as quais a função vale 0. O mintermo correspondente a uma célula particular é determinado da mesma forma que na tabela verdade. Uma variável aparece complementada no produto se tiver valor 0 na célula correspondente, e não

complementada se tiver valor 1. Por exemplo, a célula 5 na figura II.8 corresponde a w´xy´z .



$$F(w,x,y,z) = S(4,5,8,12,13) + D(3,14,15)$$

Figura II.8.- Representação de uma função no mapa de Karnaugh.

O código usado para identificar as colunas e linhas é extremamente importante. Graças a este código, células com um lado em comum correspondem a combinações que diferem no valor de uma única variável. Estas células são chamadas de **adjacentes**. Com o propósito de se determinar adjacências deve-se imaginar o mapa de 3 variáveis como a versão topológica plana de um cilindro, ou seja, as células 0 e 4 são adjacentes, assim como as células 1 e 5. Da mesma forma, o mapa de 4 variáveis deve ser imaginado como um toróide (câmara de ar), onde a célula 8 é vizinha das células 0 e 10.

Isto torna-se importante no contexto de simplificação e minimização de funções, uma vez que células adjacentes podem ser combinadas de acordo com a regra xy + xy´ = x.

Assim, o produto correspondente à união de 2 células adjacentes para as quais a função assume valor 1 é obtido escrevendo-se todas as variáveis que assumem o mesmo valor nas duas células, ignorando-se aquelas que estão na forma complementada em uma célula e não complementada na outra. Por exemplo a união das células 8 e 12 na figura II.8 corresponde ao produto wy'z', pois wxy'z'+wx'y'z' = wy'z'(x+x') = wy'z'.

Um conjunto de 2^m células, cada uma adjacente a m células do conjunto é chamado de subcubo de ordem m, e diz-se que o subcubo **cobre** estas células. Todo subcubo cuja ocorrência ("1" lógico) obriga a função a ter valor "1", ou seja, IMPLICA a função dada é um **implicante** da função. A cada implicante da função corresponde um produto de literais.

Uma função pode ser expressa como a soma dos produtos (implicantes) correspondentes aos subcubos necessários para cobrir todas as suas células de valor 1. Por exemplo, a função dada na figura II.8 pode ser expressa como a soma dos 3 implicantes correspondentes aos subcubos assinalados ,

$$F = wx + xy' + wy'z' \qquad (II.25)$$

Afim de se obter uma expressão mínima para a função, deve-se cobrir todas as células de valor 1 com o menor número de subcubos, sendo que cada subcubo seja tão grande quanto possível, pois o número de termos na expressão depende do número de subcubos, enquanto o número de literais é menor à medida que cresce o tamanho do subcubo. Assim, um subcubo contido num subcubo maior não deve ser escolhido.

Note que os don't care states podem e devem ser usados para facilitar a obtenção de expressões mínimas, assumindo

convenientemente os valores 0 ou 1. Neste exemplo, aos don't care states 14 e 15 é atribuído o valor 1, e o 3 assume valor 0.

Um implicante que corresponde a um subcubo não contido em nenhum subcubo maior é chamado de **implicante-primo.**

Um implicante-primo, além de implicar a função, não pode implicar nenhum outro produto de menor número de literais (subcubo maior) que também implique a função dada.

Uma **expressão irredutível** para uma função é qualquer conjunto de produtos de literais que represente a função e, se dele for retirado qualquer termo ou literal, deixará de representá-la.

Desta forma, toda fórmula mínima é uma expressão irredutível, mas o contrário não é verdadeiro.

NOTAÇÃO UTILIZADA :

Qualquer implicante pode ser representado por um par de "rótulos" (números inteiros decimais), utilizados conjuntamente, como se segue :

REFERENCIA : É a célula base (de menor valor decimal) do subcubo.

REDUNDANCIA : É a soma dos pesos binários das variáveis que não aparecem no produto de literais (-).

Para exemplificar o uso desta notação e verificar a sua correspondência biunívoca com os termos da função, é apresentado na tabela II.6 o conjunto dos implicantes-primos da função f obtido em (II.25) e sua representação na notação introduzida :

IMPLICANTES	CÉLULAS	BINÁRIOS				REFER.	REDUND.
		W	X	Y	Z		
wx	12,13,14,15	1	1	–	–	12	3
xy'	4,5,12,13	–	1	0	–	4	9
wy'z'	8,12	1	–	0	0	8	4

Tabela II.6.- Equivalência de notações .

Observe que esta notação é extremamente concisa e que, independente do número de variáveis da função, é sempre um par de números inteiros. Esta propriedade é fundamental na implementação computacional dos algoritmos.

II.3.5.- ACASOS ("HAZARDS") :

Os circuitos digitais físicos não respondem instantaneamente aos sinais de entrada, isto é, os atrasos associados aos componentes provocam mudanças de estados não instantâneas, o que resulta em **acasos**.

A estes atrasos dos componentes lógicos associam-se os conceitos de TPLH e TPHL. TPLH é o tempo que um componente leva para mudar do nível lógico baixo para o nível lógico alto e TPHL é o tempo para mudar de alto para baixo.

Considere, por exemplo, a função **T(x,y,z) = S(2,3,5,7)** cujo mapa de Karnaugh aparece na figura II.9, juntamente com seu circuito mínimo. Suponha que os valores de **y** e **z** estejam fixos em "1" e que o valor de **x** está mudando de "0" para "1". Neste caso, o valor de **T** deve manter-se constante em "1".

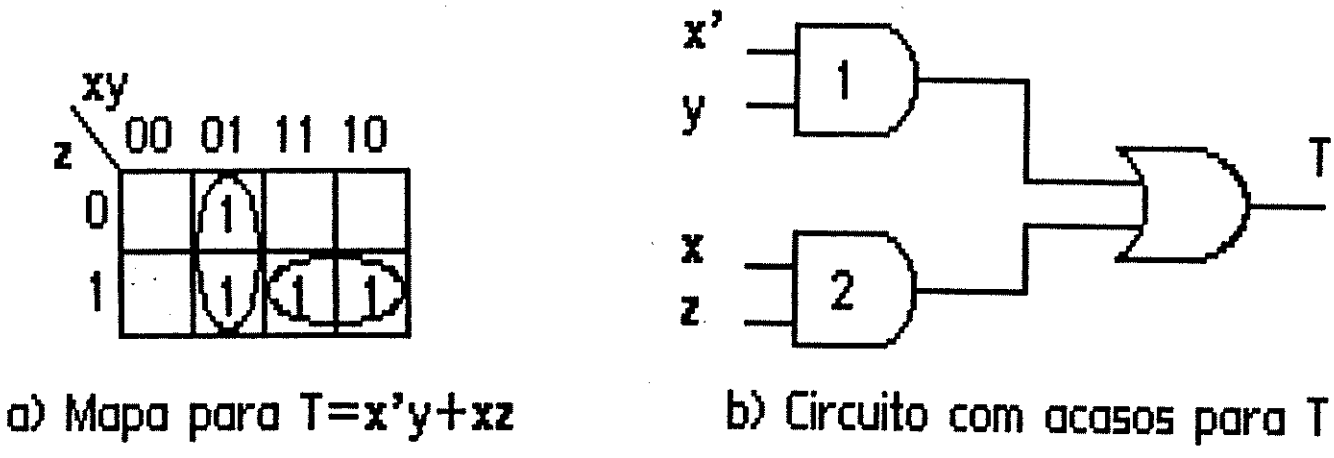


Figura II.9.- Função T e circuito com acasos.

Porém, se o gate 1 tiver um atraso menor do que o gate 2, então os 2 gates vão ter ambos o valor "0" durante um pequeno intervalo de tempo, e **T** será "0" durante este intervalo. Este fenômeno é chamado **acaso estático** [1].

Este acaso estático pode ser evitado incluindo-se o implicante **yz** na expressão de T. O resultado é o da figura II.10.

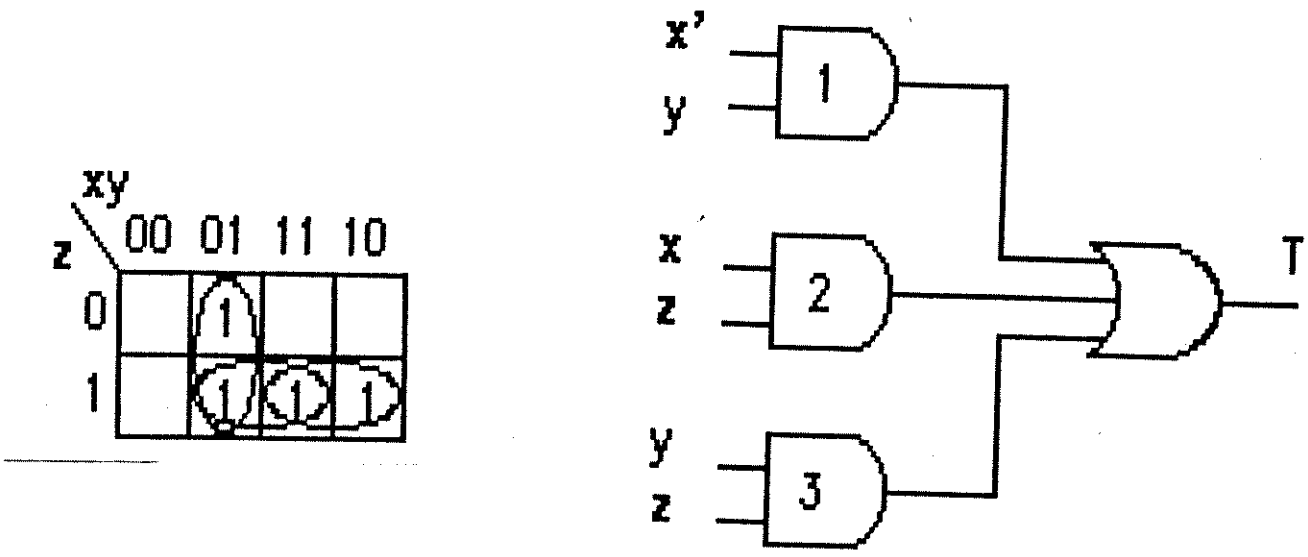


Figura II.10.- Circuito livre de acasos para a função T.

Geralmente, um acaso estático é evitável numa situação onde apenas uma variável muda de valor a cada instante. Se mais de uma variável mudar de valor num mesmo instante, é quase impossível evitar os acasos.

Tais acasos podem ocorrer sempre que existir um par de combinações de entrada adjacentes (aquelas que diferem por apenas uma variável) que produzem a mesma saída e não houver na expressão mínima uma célula que contenha ambas as combinações das entradas.

Consequentemente, um circuito é dito livre de acasos sempre que todas as mudanças de entradas adjacentes que produzam a mesma saída ocorram dentro de uma mesma célula do mapa de Karnaugh.

Para evitar acasos, geralmente é necessário adicionar alguns gates redundantes, aumentando a complexidade e o custo dos circuitos .

Isto coloca o projetista em conflito: de um lado, deseja minimizar os custos, de outro, quer evitar mal funcionamento dos circuitos . O compromisso entre estes pontos é um grande desafio em um projeto digital.

II.4.- CONCLUSÃO ;

Neste capítulo foram definidas as funções booleanas afim de descrever as propriedades dos circuitos combinacionais.

Foram introduzidos conceitos e notações utilizados em todo o trabalho, como os conceitos básicos da álgebra de Boole. Estes conceitos formam um conjunto de postulados dos quais derivam teoremas importantes, que se constituem nas ferramentas necessárias à manipulação e simplificação de expressões algébricas.

A maior parte destes conceitos estão nas obras de Kohavi[1], Peterson[2] e Yuzo[3]. A intenção do autor ao colocá-los aqui foi de tornar auto-suficiente a leitura dos demais capítulos .

A importância da minimização de funções booleanas destacada neste capítulo leva o autor a dedicar o capítulo III ao estudo e implementação de algoritmos de minimização.

II.5.- REFERENCIAS BIBLIOGRAFICAS :

1.- Kohavi, Zvi

"Switching and Finite Automata Theory"

McGraw Hill - 1970

2.- Hill, F. & Peterson, G.

"Introduction to Switching Theory & Logical Design"

John Wiley & Sons - 1974

3.- Yano, Y. , Camilo, D. & Yabu-Uti, J.B.T.

"Circuitos Lógicos: Teoria e Laboratório: Engenharia Eletrônica"

Ed. Livraria Ciência e Tecnoologia - 1984.

CAPÍTULO 3

MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS

III.1.- RESUMO :

O objetivo deste capítulo é descrever e comentar dois algoritmos distintos que realizam a minimização das funções booleanas, o algoritmo de **"Quine-McCluskey"[1,2,3]** e o algoritmo de **"Caruso"[4,5]**.

O algoritmo de "Quine-McCluskey" foi escolhido para servir de base de comparação, uma vez que está completamento desenvolvido e dominado e é suficientemente sistematizado para facilitar a implementação em computadores.

O algoritmo de "Caruso" despertou especial interesse no autor na medida que procura sanar o principal problema apresentado pelo primeiro, o consumo de tempo.

III.2.- INTRODUÇÃO:

Nenhum pacote computacional de apoio à síntese de circuitos digitais pode prescindir de um método de minimização de funções booleanas, e é neste contexto que se insere este capítulo.

Descrevem-se e comentam-se dois algoritmos distintos que realizam a minimização de funções booleanas, o algoritmo de **"Quine-McCluskey"[1,2,3]**, conhecido como método tabular e o algoritmo de **"Caruso"[4,5]**, que utiliza estruturas do tipo de grafos.

O algoritmo de "Caruso" despertou especial interesse no autor na medida que procura sanar o principal problema apresentado pelo de "Quine-McCluskey", que é o de gerar todos os implicantes da função para obter os implicantes primos, o que consome um tempo muito grande. "Caruso" sugere uma "seleção local" de implicantes que dispensa a geração de todos eles.

Alguns conceitos fundamentais associados à nomenclatura utilizada na implementação e na descrição dos algoritmos foram definidos em II.3.4.

Na Seção III.3 descreve-se o algoritmo de "Quine-McCluskey" e na Seção III.4 o programa "Mínimo".

Na Seção III.5 descreve-se o algoritmo de "Caruso" e na Seção III.6 o programa de mesmo nome.

A Seção III.7 compara e comenta os dois com exemplos e resultados obtidos.

III.3.- DESCRIÇÃO DO ALGORITMO DE QUINE-McCLUSKEY :

O método de "Quine-McCluskey" leva em conta o desejo de minimizar funções booleanas no sentido de minimizar os circuitos digitais que as sintetizam de acordo com um critério de custos.

Ao longo de todo este trabalho é adotado o critério definido em II.3.3. Para as expressões mínimas das funções é adotada a forma de soma de produtos, ressaltando-se que resultados duais podem ser obtidos para a forma de produto de somas.

A idéia fundamental do método é a aplicação repetida do teorema :

$$xy + x\bar{y} = x\,(\,y + \bar{y}\,) = x$$

Por exemplo, para minimizar a função

F1(w,x,y,z) = S(0,1,8,9), mostrada na figura III.1, tem-so :

$F1 = \bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}\bar{y}z + w\bar{x}\bar{y}\bar{z} + w\bar{x}\bar{y}z$

$F1 = \bar{w}\bar{x}\bar{y}\,(z + \bar{z}) + w\bar{x}\bar{y}\,(z + \bar{z}) = \bar{w}\bar{x}\bar{y} + w\bar{x}\bar{y}$

$F1 = \bar{x}\bar{y}\,(w + \bar{w}) = \bar{x}\bar{y}$.

Nota: A notação utilizada $\bar{x},\bar{y},\ldots$ representa "x negado", "y negado", ...

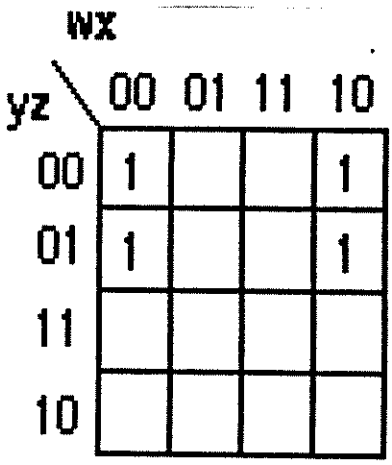


Figura III.1.- Mapa de Karnaugh para F1 = S(0,1,8,9)

III.3.1.-SISTEMATIZAÇÃO DO METODO :

Com os passos definidos a seguir, o método se torna sistemático e mais facilmente adaptado aos procedimentos computacionais :

1.- Separe os mintermos e don't care states em "caixas", sendo que na mesma caixa ficam termos com o mesmo número de dígitos '1' na sua forma binária. Este número de 1's é o índice da caixa . Esta operação define o GRUPO 0.

2.- Para todo i, compare cada termo da caixa de índice i com todos os termos da caixa i+1. Combine-os de acordo com o teorema $xy+x\bar{y} = x$, isto é, dois termos são combináveis se pertencerem a caixas adjacentes e diferirem por apenas um dígito na representação binária, ou seja, na representação decimal da referência eles diferirem por uma potência de 2 (1,2,4,8,...) e tiverem a mesma redundância . Além disso, o termo da caixa de índice superior (i+1) deve ser sempre maior que o termo da caixa de índice inferior (i) . Devem ser marcados todos os termos combinados pelo menos uma vez . Esta operação gera um novo GRUPO.

3.- Combine da mesma forma os termos gerados no passo anterior .

4.- O processo acima (3) continua até que não haja mais combinação possível. Os termos não marcados constituem o conjunto de implicantes-primos da função dada.

EXEMPLOS :

Os exemplos a seguir ilustram os processos 1 a 4.

Ex.1.- Minimizando F2(w,x,y,z) = S(0,1,2,5,7,8,9,10,13,15),(III.1) obtém-se a tabela-III.1 a seguir onde pode-se observar que :

.- o grupo é dado pelo número de combinações ocorridas, ou seja, pelo número de posições irrelevantes ("-") na representação binária;

.- as caixas são dadas pelo número de 1's na forma binária

.- o grupo 0 é formado sempre pelos próprios mintermos e don't care states da função .

GR.	CX.	DECIMAIS	BINÁRIOS	TICK
0	0	0,0	0 0 0 0	T
0	1	1,0	0 0 0 1	T
0	1	2,0	0 0 1 0	T
0	1	8,0	1 0 0 0	T
0	2	5,0	0 1 0 1	T
0	2	9,0	1 0 0 1	T
0	2	10,0	1 0 1 0	T
0	3	7,0	0 1 1 1	T
0	3	13,0	1 1 0 1	T
0	4	15,0	1 1 1 1	T
1	0	0,1	0 0 0 -	T
1	0	0,2	0 0 - 0	T
1	0	0,8	- 0 0 0	T
1	1	1,4	0 - 0 1	T
1	1	1,8	- 0 0 1	T
1	1	2,8	- 0 1 0	T
1	1	8,1	1 0 0 -	T
1	1	8,2	1 0 - 0	T
1	2	5,2	0 1 - 1	T
1	2	5,8	- 1 0 1	T
1	2	9,4	1 - 0 1	T
1	3	7,8	- 1 1 1	T
1	3	13,2	1 1 - 1	T
2	0	0,9	- 0 0 -	F
2	0	0,10	- 0 - 0	F
2	1	1,12	- - 0 1	F
2	2	5,10	- 1 - 1	F

TABELA III.1.- Minimizando F2, eq. III.1

Destaca-se neste caso que apenas os termos do grupo 2 não foram combinados e portanto não foram marcados (TICK = FALSE) , logo são eles que formam o conjunto dos implicantes-primos da função F2 :

$$
\begin{array}{lll}
 & w\ x\ y\ z & \\
A = (0,9) & = -\ 0\ 0\ - & = x\bar{y} \\
B = (0,10) & = -\ 0\ -\ 0 & = \bar{x}\bar{z} \\
C = (1,12) & = -\ -\ 0\ 1 & = \bar{y}z \\
D = (5,10) & = -\ 1\ -\ 1 & = xz
\end{array}
\qquad \text{(III.2)}
$$

O custo inicial da função, dado pela soma de todos os mintermos é 50. Note a redução para 12 apenas pela obtenção dos implicantes primos.

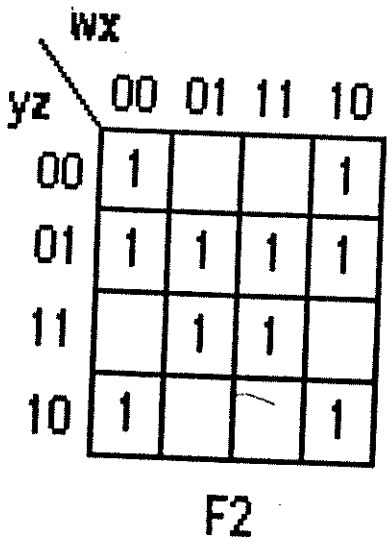


Figura III.2.- Mapa de Karnaugh de F2 = S(0,1,2,5,7,8,9,10,13,15)

Ex.2.- A tabela-III.2 ilustra a minimização da função F3 dada por:

$$F3(v,w,x,y,z) = S\ (13,15,17,18,19,20,21,23,25,27,29,31) + D\ (1,2,12,24) \qquad \text{(III.3)}$$

O mapa de Karnaugh de F3 é mostrado na figura III.3.

GRUPO 0		
cx.	elem.	tick
1	1,0	T
1	2,0	T
2	12,0	T
2	17,0	T
2	18,0	T
2	20,0	T
2	24,0	T
3	13,0	T
3	19,0	T
3	21,0	T
3	25,0	T
4	15,0	T
4	23,0	T
4	27,0	T
4	29,0	T
5	31,0	T

CUSTO = 72

GRUPO 1		
cx.	elem.	tick
1	1,16	F
1	2,16	F
2	12,1	F
2	17,2	T
2	17,4	T
2	17,8	T
2	18,1	F
2	20,1	F
2	24,1	F
3	13,2	T
3	13,16	T
3	19,4	T
3	19,8	T
3	21,2	T
3	21,8	T
3	25,2	T
3	25,4	T

GRUPO 1		
cx.	elem.	tick
4	15,16	T
4	23,8	T
4	27,4	T
4	29,2	T

GRUPO 2		
cx.	elem.	tick
2	17,6	T
2	17,10	T
2	17,12	T
3	13,18	F
3	19,12	T
3	21,10	T
3	25,6	T

GRUPO 3		
cx.	elem.	tick
2	17,14	F

TABELA III.2.-Minimização de F3, eq. (III.3)

Da tabela III.2 obtém-se o conjunto dos implicantes-primos de F3 que é dado pelos termos não marcados (Tick = F):

A = 17,14 = vz
B = 13,18 = wxz
C = 24,1 = $vw\tilde{x}\tilde{y}$
D = 20,1 = $v\tilde{w}x\tilde{y}$
E = 18,1 = $v\tilde{w}\tilde{x}y$
F = 12,1 = $\nabla wx\tilde{y}$
G = 2,16 = $\tilde{w}\tilde{x}y\tilde{z}$
H = 1,16 = $\tilde{w}\tilde{x}\tilde{y}z$

CUSTO = 37 (III.4)

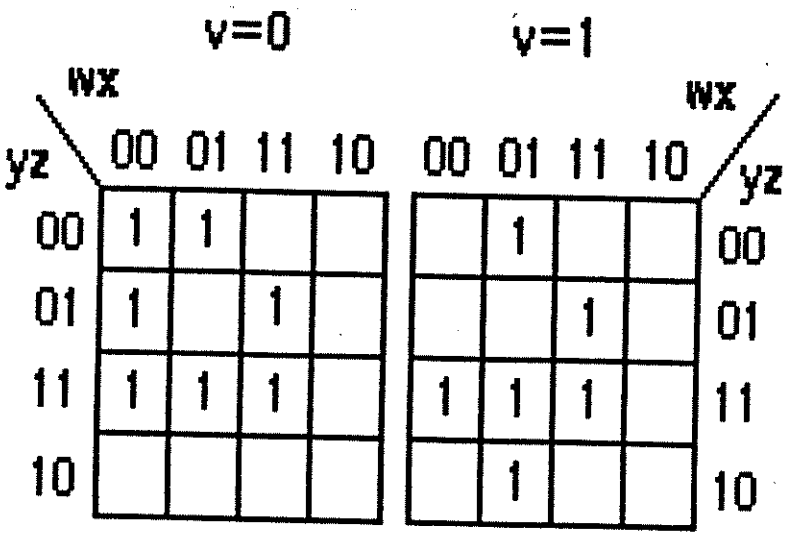


Figura III.3.- Mapa de Karnaugh de F3 eq (III.3)

III.3.2.- MAPA DOS IMPLICANTES-PRIMOS :

De acordo com o método, após a obtenção do conjunto de todos os implicantes-primos da função, monta-se uma tabela na qual as linhas são os implicantes-primos e as colunas são os mintermos da função. Os estados irrelevantes (don't care states) da função não aparecem nesta tabela, uma vez que não precisam ser cobertos pela expressão mínima da função. Esta tabela é chamada de "Mapa dos implicantes-primos" da função.

A tabela é montada colocando-se uma marca (X) nas intersecções entre os implicantes-primos e os mintermos, indicando quais mintermos são cobertos por cada um dos implicantes-primos.

O problema resume-se então em selecionar um subconjunto mínimo de linhas da tabela (implicantes-primos) de modo que cada coluna tenha ao menos uma marca neste subconjunto, ou seja, todos os mintermos são cobertos por este subconjunto de implicantes-primos. Tal subconjunto deve ser o que possui número mínimo de literais em relação a todos os demais que possam ser selecionados (critério de custo 3 - seção II.3.3).

Para exemplificar, é montado a seguir o mapa dos implicantes-primos de F3 com base no seu conjunto de implicantes-primos obtidos na equação III.4. Este mapa é mostrado na tabela-III.3 , na qual as marcas em negrito são os mintermos cujas colunas contêm uma única marca, os implicantes-primos que cobrem estes mintermos são "IMPLICANTES PRIMOS ESSENCIAIS" e aparecem obrigatoriamente na fórmula mínima da função.

IMPLICANTES		mintermos de F3											
ref.	red.	13	15	17	18	19	20	21	23	25	27	29	31
17	14			X		X		X	X	X	X	X	X
13	18	X	X									X	X
24	1									X			
20	1						X	X					
18	1				X	X							
12	1	X											
2	16				X								
1	16			X									

CUSTO = 37

TABELA III.3.- Mapa dos implicantes-primos de F3 .

Este mapa vai sendo reduzido com a retirada de linhas por serem consideradas ESSENCIAIS e de colunas por terem sido "cobertas", até se chegar à fórmula mínima da função.

No algoritmo são definidos diferentes "NIVEIS" de "essenciais", a saber:

NIVEL 0 : São os essenciais obtidos logo na montagem do mapa, ou seja, pela regra básica, que é cobrir um mintermo que não seja coberto por nenhum outro implicante-primo.

NIVEL n : São os implicantes-primos considerados "essenciais" pela mesma regra, porém após n reduções do mapa.

Neste exemplo, os essenciais de nível 0 são os implicantes primos (17,14) , (13,18) e (20,1), e a tabela-III.3 é reduzida eliminando-se as linhas correspondentes a estes implicantes-primos e as colunas referentes a todos os mintermos cobertos por eles. Assim é obtida a tabela-III.4 :

ref.	red.	18
24	1	
18	1	X
12	1	
2	16	X
1	16	

TABELA III.4.- Mapa dos implicantes-primos de F3 reduzido.

Da tabela III.4, observa-se que falta cobrir apenas o mintermo 18, o que pode ser feito através dos implicantes-primos (18,1) ou (2,16). Neste caso qualquer das escolhas resulta numa expressão mínima para a função F3, já que ambos os implicantes possuem o mesmo número de literais, e o implicante escolhido é dito essencial nível 1 .

Logo, para esta função são obtidas 2 expressões mínimas:

$F3 = (17,14) + (13,18) + (20,1) + (18,1)$

$= vz + wxz + v\tilde{w}x\tilde{y} + v\tilde{w}\tilde{x}y$ CUSTO = 17 eq.(III.4)

ou $F3 = (17,14) + (13,18) + (20,1) + (2,16)$

$= vz + wxz + v\tilde{w}x\tilde{y} + \tilde{w}\tilde{x}y\tilde{z}$ CUSTO = 17

Observe, passo a passo, a redução dos custos, inicialmente 72, com a obtenção dos implicantes primos passa a 37, e com a redução do mapa cai para 17.

III.3.3.- REDUÇÃO DO MAPA DOS IMPLICANTES-PRIMOS :

Nas definições dos níveis de essenciais estão implícitas duas extensões do conceito de cobertura : "cobertura entre implicantes-primos" e "cobertura entre mintermos" (linhas e colunas do mapa, respectivamente) .

Um implicante-primo "A" cobre um implicante-primo "B" (numa tabela reduzida ou quando existirem don't care states), se toda vez que B=1, também A=1, ou seja, "A" cobre qualquer mintermo que "B" cobrir no mapa. Na tabela-III.3 , nota-se por exemplo que o implicante (17,14) cobre o implicante (24,1), o que permite retirar este último da tabela afim de reduzi-la.

Também se define que um mintermo i cobre um mintermo j (diz-se também que i domina j) se, no mapa dos implicantes primos, em cada linha onde houver uma marca na coluna j for encontrada também uma marca na coluna i. O importante aqui é ressaltar que para reduzir a tabela com esta propriedade, é eliminado o mintermo dominante (i), pois toda expressão mínima deduzida para o mintermo i podo ser obtida do mintermo j. Neste exemplo, na tabela-III.3, observa-se que o mintermo 25 domina o mintermo 23 pois em todas as linhas onde 23 possui uma marca, 25 também a tem.

Geralmente utilizando-se estas técnicas obtém-se uma só fórmula mínima para a função, o que é suficiente na maioria dos casos. No caso do se desejar obter mais de uma fórmula mínima para a mesma função, podo-se recorrer ao "Método de Petrick" [2].

Ex.3.- Minimizar a função F4, figura III.4 (CUSTO INICIAL = 70):

$$F4(v,w,x,y,z) = S\,(0,1,3,4,7,13,15,19,20,22,23,29,31) \qquad (III.5)$$

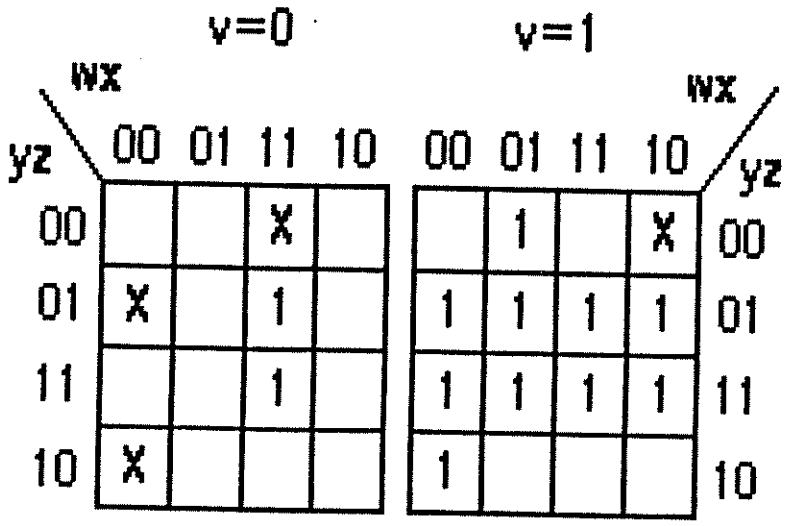


Figura III.4.- Mapa de Karnaugh de F4 - eq (III.5)

O mapa dos implicantes-primos de F4 obtido através do método tabular de "Quine-McCluskey" é dado na tabela-III.5 :

IMPLICANTES		mintermos de F4													
ref.	red.	0	1	3	4	7	13	15	19	20	22	23	29	31	
13	18						X	X					X	X	A ←
7	24					X		X				X		X	B
3	20			X		X			X			X			C ←
22	1										X	X			D
20	2									X	X				E
4	16				X					X					F
1	2		X	X											G
0	4	X			X										H
0	1	X	X												I
				↑		↑	↑	↑	↑			↑	↑	↑	

CUSTO = 42

TABELA III.5.- Mapa dos implicantes-primos de F4

Na tabela-III.5 as linhas assinaladas são os implicantes-primos essenciais e as colunas são os mintermos cobertos por eles. Reduzindo-se esta tabela apenas com a retirada destes essenciais e destes mintermos, obtém-se a tabela-III.6.

IMPLICANTES		MINTERMOS					
ref.	red.	0	1	4	20	22	
22	1					X	D
20	2				X	X	E
4	16			X	X		F
1	2		X				G
0	4	X		X			H
0	1	X	X				I

TABELA III.6.- Mapa reduzido de F4

Na tabela III.6, observa-se que o implicante primo E cobre o implicante primo D, e o I cobre o G. Assim, pode-se retirar D e G da tabela, obtendo a tabela III.7.

IMPLICANTES		MINTERMOS					
ref.	red.	0	1	4	20	22	
20	2				X	X	E
4	16			X	X		F
0	4	X		X			H
0	1	X	X				I

TABELA III.7.- Segunda redução do mapa de F4

Os implicantes E e I tornam-se essenciais e podem ser retirados da tabela, juntamente com os mintermos cobertos por eles, 0, 1, 20 e 22, restando a tabela III.8 :

ref.	red.	4	
4	16	X	F
0	4	X	H

TABELA III.8.- Última redução do mapa de F4

Como F e H têm o mesmo custo, pode-se escolher qualquer um deles para cobrir o mintermo 4. Por exemplo, escolhe-se H. Com isto, todos os mintermos estão cobertos e uma fórmula mínima da função é dada por :

F4 = A + C + E + I + H

F4 = (13,18) + (3,20) + (20,2) + (0,1) + (0,4)

CUSTO FINAL = 23

III.3.4.- METODO DE RAMIFICAÇÃO :

Pode ocorrer de se obter um mapa de implicantes-primos "cíclico", ou seja, não há nenhum implicante-primo essencial e nenhuma linha ou coluna pode ser eliminada da tabela pelos critérios de cobertura. Nestes casos utiliza-se o método de Ramificação (adaptação do método de "Branching" sugerido em [1]) descrito a seguir através de um exemplo:

Ex.4.- Minimizando F5, figura III.5, é obtido o mapa de implicantes primos mostrado na tabela-III.9 :

$$F5 = S\,(0,1,5,7,8,10,14,15) \qquad (III.6)$$

CUSTO INICIAL = 40

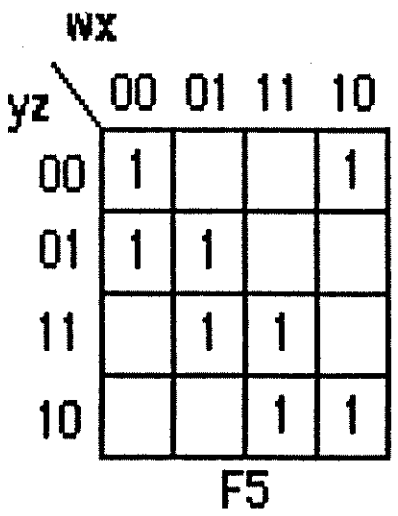


Figura III.5.- Mapa de Karnaugh para F5 - eq. (III.6)

Observa-se na tabela-III.9 que não há essenciais, nem linhas ou colunas dominantes. O método de Ramificação consiste em selecionar "arbitrariamente", entre os de menor custo (menor número de literais), um implicante-primo como essencial.

IMPLICANTES		mintermos de F5								
ref.	red.	0	1	5	7	8	10	14	15	
0	1	X	X							A
1	4		X	X						B
5	2			X	X					C
7	8				X				X	D
14	1							X	X	E
10	4						X	X		F
8	2					X	X			G
0	8	X				X				H

CUSTO = 32

TABELA III.9.- Mapa dos implicantes-primos de F5

Por exemplo, o mintermo 0 é coberto pelos implicantes A e H. Como ambos possuem o mesmo número de literais, um deles é escolhido arbitrariamente e então, o mapa é reduzido pelas técnicas já citadas. Se for obtido novamente um mapa cíclico em qualquer passo, é escolhida novamente uma linha arbitrária e repetido o processo.

ref.	red.	1	5	7	10	14	15	
0	1	X						A
1	4	X	X					B
5	2		X	X				C
7	8			X			X	D
14	1					X	X	E
10	4				X	X		F
8	2				X			G

TABELA III.10.- Simplificação da tabela III.9 pela retirada de H, escolhido como essencial, e dos mintermos 0 e 8, cobertos por ele .

Para a função F5 eq. (III.6) é selecionado o implicante H como essencial e a tabela-III.9 é reduzida à tabela-III.10, onde se pode identificar linhas dominadas, voltando a reduzi-la na tabela-III.11 até finalmente se chegar à fórmula mínima para F5.

ref.	red.	7	15	
5	2	X		C
7	8	X	X	D
14	1		X	E

TABELA III.11.- Redução da tabela III.10 pela retirada dos essenciais B e F e dos mintermos 1, 5, 10, 14, cobertos por eles.

Na tabela-III.11 , o implicante D cobre os demais, tornando-se essencial, e termina o processo pois todos os mintermos foram cobertos. Portanto,

$$F5 = D + B + F + H$$

$$F5 = (7,8) + (1,4) + (10,4) + (0,8) \qquad \text{(III.7)}$$

CUSTO FINAL = 16

Neste exemplo pode-se notar novamente a redução gradativa dos custos. No início, representando F5 por todos os seus mintermos, o custo é 40. Representando-a por todos os implicantes primos o custo cai para 32 e determinando os essenciais o custo é reduzido para 16.

III.4.- DESCRIÇÃO DO PROGRAMA "MINIMO" :

Foram realizadas duas implementações de um programa de computador que minimiza funções booleanas de acordo com o algoritmo de Quine-McCluskey.

.- Na primeira utilizou-se a alocação estática de memória. É uma implementação bem mais direta e simples, porém esbarrou-se no problema do alto consumo de memória. Isto limitou a 6 o número de variáveis das funções.

.- Como alternativa, partiu-se para uma segunda implementação utilizando a alocação dinâmica de memória. As estruturas vetoriais foram substituidas por listas ligadas, o que resolveu o problema do consumo de memória.

A alocação dinâmica tornou o programa bastante mais complexo, e consequentemente mais lento. Nos testes de pior caso (para o algoritmo de Quine-McCluskey o pior caso é a função 1, que tem o maior número de mintermos), para a função 1 de 8 variáveis (256 mintermos), o programa leva 15 minutos para chegar ao resultado, sendo que cerca de 80% do tempo é consumido na combinação dos mintermos para a obtenção dos implicantes primos.

O tempo médio de execução do programa está em torno de 2 minutos, o que é considerado satisfatório.

Os dados requisitados pelo programa são:

.- Nome do arquivo de saída de resultados;

.- Número de variáveis da função;

.- Número de don't care states da função;

.- Todos os don't care states da função (decimais);

.- Número de mintermos da função;

.- Todos os mintermos da função (decimais);

Calcula-se o custo inicial da função, dado pelo custo da soma de todos os mintermos, e seguindo o algoritmo de minimização, o programa combina os termos até obter todos os implicantes primos da função. A seguir, monta seu mapa e determina os implicantes primos essenciais, verificando as coberturas entre linhas e colunas do mapa.

Os resultados são conferidos, verificando-se se, com a fórmula mínima obtida para a função, é possível obter-se todos os seus mintermos. Calcula-se o custo final da função, dado pelo custo da soma de todos os essenciais.

Para comparação de custos, o programa minimiza também a função negada da mesma forma, o que equivale à minimização da função pelos seus maxtermos, conforme II.3.3.

Assim, os resultados obtidos são os custos inicial e final da função e da função negada, uma fórmula mínima para a função e uma para a função negada e a conferência dos resultados obtidos.

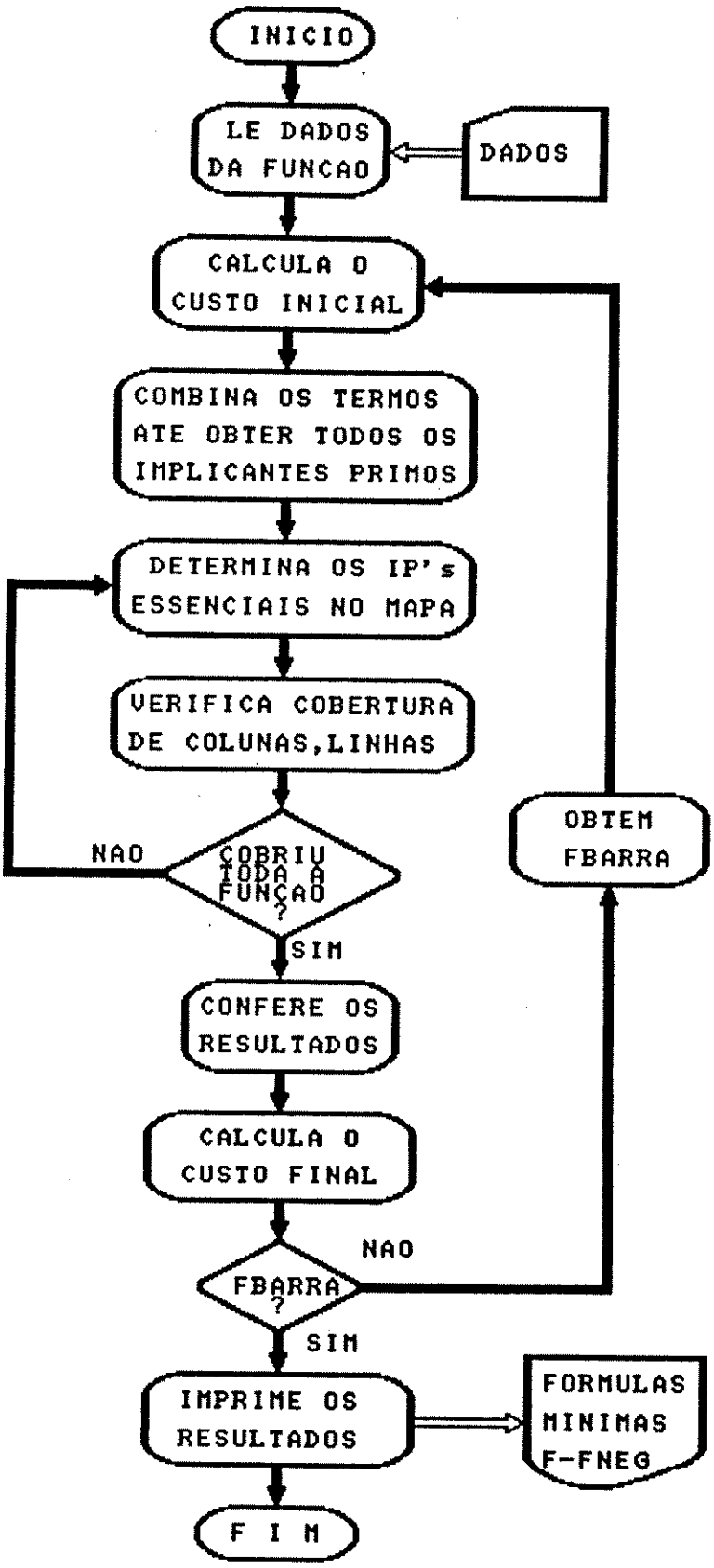


Figura III.6.- Diagrama de blocos do programa MINIMO

III.5.- DESCRIÇÃO DO ALGORITMO DE CARUSO :

A implementação do algoritmo de Quine-McCluskey mostrou que, em média, 80% do tempo da minimização é consumido na geração dos implicantes primos. Isto levou o autor à busca de métodos que fossem mais eficientes nesta etapa.

O algoritmo de Caruso se destaca exatamente por esta propriedade: a eficiência na obtenção dos implicantes primos, além de efetuar simultaneamente a cobertura da função.

No artigo original [4], Caruso apresenta uma notação bastante carregada e distante de ser adequada a uma implementação computacional. Por este motivo, o autor introduz aqui uma nova notação, tentando apropriá-la à automatização do método e aproximá-la, tanto quanto possível, da notação utilizada no algoritmo de Quine-McCluskey.

III.5.1.- NOTAÇÃO E DEFINIÇÕES:

E(N) : é o conjunto dos decimais de todos os pontos (vértices) do espaço binário N-dimensional. A regra usada para obter-se o decimal é a de conversão de base de numeração, da binária para a decimal.

F contido em E(N) : é o conjunto dos decimais de todos os pontos do espaço N-dimensional, pertencentes a função F.

v : é o decimal de um ponto da função (mintermo).

vd : é o decimal de um don't care state da função.

(u) : é o conjunto dos pesos binários das N variáveis de uma dada função. Cada variável é representada por uma letra distinta.

[A] : é um subconjunto qualquer de (u) .

PESO : Peso de um bit é o valor decimal igual a 2**(k-1), onde k é a posição relativa do bit, contada da direita para a esquerda. Exemplo :

```
                1 0 1 0 1
                ↑   ↑   ↑
pesos:         16   4   1
```

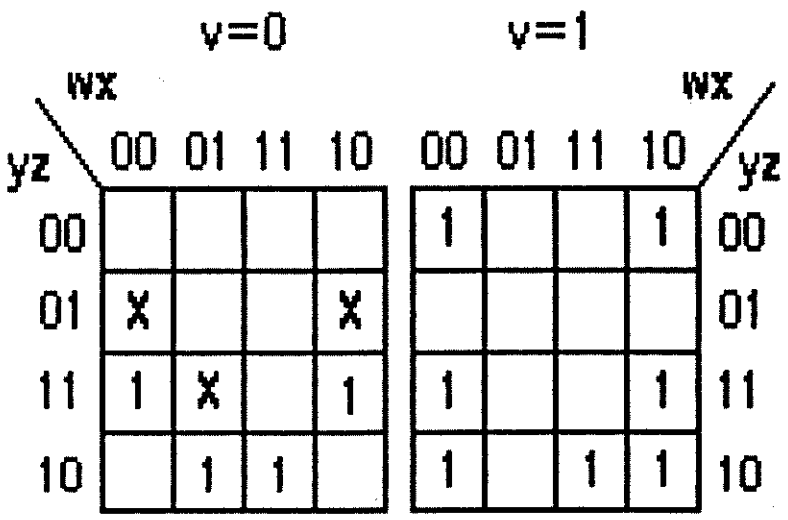
OPERAÇÃO INVBIT : Aplicar INVBIT(j), j pertencente a (u), a um vértice **v** de uma função F dada, significa complementar (inverter) o bit de peso j da representação binária de **v**. A notação usada é :

```
          j
v <------->vx
```

onde **vx** é o novo ponto, correspondente ao valor decimal da nova combinação dos bits. Se **vx** pertencer a F, então é dito à distância 1 de **v**.

Exemplo : F6 = S(3,6,11,14,16,18,19,24,26,27,30) + D(1,7,9)

(u) = (16,8,4,2,1)



F6 = S(3,6,11,14,16,18,19,24,26,27,30) + D(1,7,9)

Figura III.7.- Mapa de Karnaugh de F6

Aplicando INVBIT(8) ao vértice 3:

```
        8
   3  <-----> 11    ;  como 11 pertence a F6,  11  está  a
00011       01011      distância 1 de 3.
 ^           ^
```

Também, ao se aplicar sequencialmente INVBIT(16,8,1) (que é equivalente a INVBIT(25)) ao vértice 3, obtém-se:

```
          16            8            1
   3  <------> 19  <-----> 27  <-----> 26
00011         10011        11011        11010
 ^             ^            ^
```

Assim, 19 está a distância 1 de 3,
27 está a distância 2 de 3,
26 está a distância 3 de 3.

Seja **v[ ]D1** um subconjunto ordenado de pesos (pertencentes a (u)) que levam a vértices a distância 1 de **v**, **v** pertencente a F,. Seja **p** o número de elementos desto vetor. Define-se **p** como a ordem de v[ ]D1.

Exemplo: 3[16,8,4,2]D1 ; neste caso, p = 4.

Em analogia às definições de referência e redundância usadas no algoritmo de Quine-McCluskey, define-se **peso-referência** ao decimal resultante da soma dos pesos de v[ ]D1.

Assim, 3[16,8,4,2]D1 é representado por 3[30]D1, onde 30 é o peso-referência.

Seja **v[ ]** um subconjunto de ordem k de v[ ]D1. Aplicar v[ ] significa aplicar todos os 2**k-1 subconjuntos (partes) de [ ] ao vértice **v**.

Por exemplo: 3[12]

```
      8
3  <-----> 11

      4
3  <-----> 7

      8          4
3  <-----> 11 <-----> 15
```

Note que 3[12] é um subcubo e pode ser representado pelo par referência redundância (3,12).

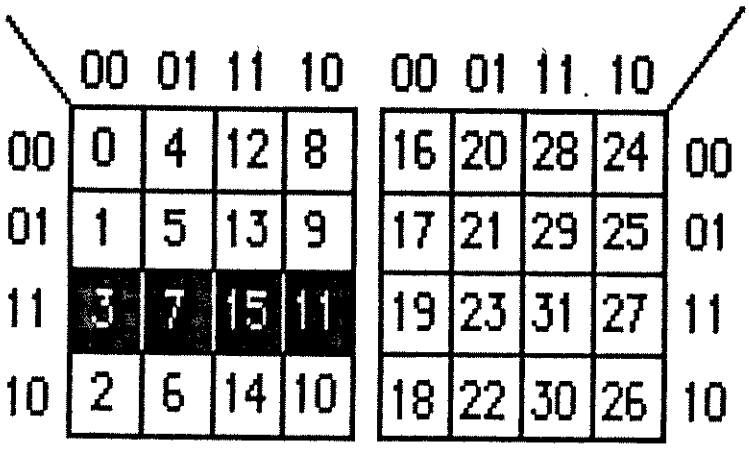


Figura III.8.- Subcubo 3[12]

PROPRIEDADE : v[ ] é um implicante cobrindo **v** se, e somento se, todos os 2**k-1 vértices resultantes pertencerem a função F. A notação é **v[ ]I** .

Exemplo : 3[16] é um implicante de F6 cobrindo 3, portanto denota-se 3[16]I.

```
          16
3  <------>  19   ;  e 19 pertence a F6.
```

v[A]I é um implicante primo (IP) se, e somente se, seus elementos não forem um subconjunto de nenhum outro v[B]I para um dado **v** (A≠B). A notação passa a ser então v[A]P .

Exemplo : 3[24] é um implicante primo :

```
      16
3  <------> 19

      8
3  <------>  11

      16          8
3  <------> 19 <-----> 27
```

Note que 3[16]I não é um implicante primo pois está contido em 3[24].

Esta representação gráfica das sucessivas operações INVBIT a um vértice **v** é chamada de **GRAFO** .

Se existir para um dado **v** um único v[ ]P, este é dito um **IMPLICANTE PRIMO ESSENCIAL (IPE).**

Por exemplo : 16[10]P é um IPE.

```
        8
16  <-----> 24

        2
16  <-----> 18

        8             2
16  <-----> 24  <-----> 26
```

Como 18, 24 e 26 pertencem a F6, e 16[10]P é o único que cobre o mintermo 16, é um IPE.

Geralmente, um vértice da função pode estar contido em mais de um IP. Neste caso, todos os IP's que cobrem tal vértice (v), formam o conjunto **P[v]**.

Se existir apenas um termo no conjunto P[v], este termo é um IPE e o vértice **v** é chamado de **VERTICE IPE-ISOLADO**. Neste caso, o grafo associado a P[v] é chamado de **GRAFO COMPLETO** .

O vértice **v** é um **VERTICE D-ISOLADO** se existir pelo menos um IP, denominado IP1, no vetor P[v], que satisfaça as seguintes condições:

- nenhum outro IP pertencente a P[v] tem mais elementos que IP1 no conjunto (u) ;

- todos os mintermos cobertos por outro IP de P[v] podem ser cobertos por IP1 .

Neste caso, o grafo associado é chamado de **GRAFO PARCIALMENTE COMPLETO** .

COBERTURA MINIMA (CM) da função é a união do menor número de implicantes primos da função suficiente para representar (cobrir) a função.

FORMULA DE CUSTO MINIMO (FCM) é uma cobertura da função que garante o menor número de entradas para uma implementação binária de dois níveis, usando apenas portas AND-OR .

III.5.2.- GERAÇÃO DE IMPLICANTES PRIMOS E CONJUNTOS IRREDUNDANTES :

PROCEDIMENTO A.- GERAR UM GRAFO PARA UM DADO VERTICE v :

Passo 1 : Aplicar todos os j pertencentes a v[ ]D1 ao vértice v. Assim, obtém-se todos os vértices vx a distância 1 de v. v é dito raiz do grafo.

Passo 2 : Aplicar a cada vértice obtido todos os j pertencentes a v[ ]D1, subsequentes àquele com o qual ele foi gerado. Rejeitam-se os vértices não presentes em F e anota-se o conjunto [ ] associado.

Passo 3 : Repetir o passo 2 para cada vértice obtido, até que não seja gerado mais nenhum vértice da função.

PROCEDIMENTO B.- OBTER IP's A PARTIR DOS GRAFOS :

Passo 1 : Identificar os vértices terminais de maior distância da raiz. Seus subconjuntos [A,B,...] formam os e[A]P, e[B]P,... .

Passo 2 : Marcar todos os vértices obtidos pela aplicação de cada e[ ]P encontrado e repetir o passo 1 para os vértices terminais ainda não marcados. Parar quando todos os vértices terminais do grafo estiverem marcados.

CONJUNTO IRREDUNDANTE (CI) de grafos representando uma função F é aquele em que todo mintermo da função está presente em pelo menos um grafo do conjunto, e em todos os grafos existe pelo menos um vértice que não está em nenhum outro grafo deste conjunto.

VERTICE SINGULAR é aquele que está somente em um grafo de um determinado conjunto irredundante.

PROCEDIMENTO C.- GERAR UM CONJUNTO IRREDUNDANTE :

Passo 1 : Escolha a primeira raiz entre os mintermos da função.

Passo 2 : Gere o seu grafo através do procedimento A e marque na função todos os vértices que aparecerem neste grafo.

Passo 3 : Se ainda existirem mintermos não marcados na função, escolha um para ser a próxima raiz e repita o passo 2. De outra forma, pare.

III.5.3.- **TEOREMAS**

0.- A raiz de cada grafo de um conjunto gerado pelo procedimento C é um vértice singular.

PROVA : Por construção. %

1.- Um implicante está em somente um grafo de um Conjunto Irredundante (gerado pelo procedimento C).

PROVA : Suponha que um IP esteja em dois grafos do tal conjunto. Cada um deles tem entre seus vértices a raiz do outro. Isto não pode acontecer pois as raizes foram escolhidas de acordo com o passo 3 do procedimento C, entre os mintermos não marcados. %

2.- O número de grafos de um CI é menor ou igual ao número de IP's da fórmula de custo mínimo ou de cobertura mínima para uma dada função.

PROVA : Todos os IP's de um grafo cobrem a raiz, e cada IP pode estar em apenas um grafo de um CI. Portanto o número de grafos é menor ou igual ao número de IP's. %

III.5.4.- ALGORITMO DE COBERTURA MINIMA

Uma **COBERTURA DE UM GRAFO (CG)** é qualquer conjunto de IP's, extraído de um grafo, que seja uma cobertura irredundante para todos os vértices singulares deste grafo. Isto é , cada IP desta cobertura deve cobrir pelo menos um vértice singular que não é coberto por nenhum outro IP da cobertura e, todos os vértices singulares do grafo devem ser cobertos.

Um IP é dito **ESSENCIAL** para uma cobertura de um grafo quando cobre vértices singulares (pelo menos um) não cobertos por nenhum outro IP.

Uma CG é única quando todos os seus IP's são essenciais.

A minimização da função é realizada removendo-se todos os grafos de um CI, um de cada vez. O conjunto de coberturas de grafos obtido é a cobertura mínima da função dada. Isto é provado adiante.

PROCEDIMENTO D.- REMOÇÃO DE GRAFOS :

Passo 1 : Escolha uma cobertura de um grafo.

Passo 2 : Converta todos os vértices cobertos por ela em don't care states nos outros grafos e na função.

Passo 3 : Remova o grafo do CI.

Os grafos restantes e a nova função após cada remoção são chamados **CONJUNTO RESIDUAL** e **FUNÇÃO RESIDUAL**, respectivamente.

TEOREMA 3.- Um conjunto residual é irredundante em relação a função residual correspondente.

PROVA : Todo mintermo de uma função residual está no conjunto residual correspondente, pois na função residual estarão todos os mintermos da função anterior (antes da remoção) exceto os vértices removidos, que foram convertidos em don't care states. %

TEOREMA 4 : O conjunto de coberturas de grafos selecionado pela remoção de todos os grafos de um CI é uma cobertura mínima da função.

PROVA : Da definição de cobertura de grafo, cada IP contém pelo menos um vértice não presente em nenhum outro IP. Cada conjunto residual é irredundante com relação à função residual correspondente. Assim, todos os mintermos da função estarão cobertos após a última remoção. %

O grau de otimalidade (custo) de uma cobertura da função está relacionado com uma série de fatores como o tamanho do CI, a ordem em que os grafos são removidos do CI, etc.

Um CI geralmente não contém todos os IP's da função dada. Do teorema 2, existem pelo menos tantos IP's numa FCM quantos forem os grafos. Isto sugere que o maior CI é a escolha mais indicada para a obtenção de uma FCM. Para obter tal CI, duas regras podem ser observadas, segundo Caruso, para escolher-se uma raiz no procedimento C:

REGRA 1 : Escolha como raiz o vértice não marcado com o menor número de sucessores a distância 1. (E o que vai gerar o menor subcubo)

REGRA 2 : Se ocorrer empate no critério anterior escolha para raiz o vértice de menor valor decimal.

Obs.: Na opinião do autor, esta segunda regra é irrelevante, contudo foi mantida.

Com relação a escolha de uma cobertura de um grafo, devem-se considerar quatro casos distintos. O primeiro refere-se ao grafo onde a raiz é um IPE-isolado. Tal grafo é completo e todos os vértices formam um IPE. Para verificar se um grafo é completo aplica-se o seguinte teste:

TESTE 1 : Verifique se o conjunto de vértices rejeitados pelo passo 2 do procedimento A é vazio.

O segundo caso de cobertura refere-se ao grafo cuja raiz é um vértice D-isolado, ou seja , um grafo parcialmente completo. Neste caso, o teste é o seguinte:

TESTE 2 : Verifique se o grafo pode ser coberto aplicando-se pelo menos um dos maiores IP's de P[e], ou seja, aqueles com maior valor de k. Computacionalmente, este teste consiste em alterar o estado dos don't care states deste grafo no sentido de torná-lo um grafo completo.

O terceiro caso de cobertura refere-se ao grafo com menos de 2**k-1 vértices que possui cobertura única:

TESTE 3 : Verifique se o conjunto de todos os IP's essenciais à cobertura cobre todos os vértices singulares do grafo.

Nestes três casos os IPE's são obtidos de maneira única e inequívoca. Mas o teste três se diferencia dos demais pois estes IPE's não cobrem necessariamente todos os mintermos não singulares do grafo.

O quarto e último caso de cobertura está relacionado com o grafo que tem diversas coberturas alternativas. Esta seleção pode ser feita por qualquer método clássico de seleção de implicantes primos (IP's), como por exemplo a técnica do Petrick[2].

A ordem cronológica de geração dos grafos é tal que, em geral, estão ordenados por custos decrescentes de suas coberturas. Assim, atendendo à sugestão de Caruso neste quarto caso, escolhe-se arbitrariamente o maior IP do último grafo.

III.5.5.-RESUMO DAS PROPRIEDADES :

1.- O número de grafos gerados é menor ou igual ao número de termos (implicantes) na fórmula de custo mínimo da função.

2.- A cobertura dos mintermos é quase que simultânea à geração dos grafos (exceção para o caso 4).

3.- Na geração dos implicantes primos, não são gerados os implicantes intermediários como no algoritmo de Quine-McCluskey.

Portanto, o método de Caruso é mais rápido e consome menos memória que o de Quine-McCluskey.

A sua única desvantagem é não garantir sempre a obtenção de uma fórmula de custo mínimo, mas sim de cobertura mínima.

Vale a pena ressaltar que, conforme ilustrado nos exemplos da seção III.7, todos os casos testados geraram fórmulas de custo mínimo.

III.5.6.-IMPLEMENTAÇÃO RESUMIDA DO ALGORITMO :

Passo 0 : Ordene os mintermos de acordo com as regras 1 e 2 da seção III.5.4 para escolher as raizes dos grafos.

Passo 1 : Escolha a primeira raiz.

Passo 2 : Gere o grafo correspondente.

Passo 3 : Aplique a este grafo os testes 1 e 2 da seção anterior. Se um deles for bem sucedido, remova o grafo e coloque-o na cobertura parcial da função. Mantenha o grafo se os testes

falharem. Sempre que remover um grafo, lembre-se de converter todos os seus mintermos em don't care states para a função e os grafos restantes. Isto pode ser feito acrescentando-se um símbolo "d" a seu lado.

Passo 4 : Se ainda existir mintermo que possa ser raiz, escolha a próxima raiz e volte ao passo 2.

Passo 5 : Se o conjunto residual de grafos for vazio, FIM. Senão vá ao passo 6.

Passo 6 : Marque com um "s" todos os vértices singulares em todos os grafos.

Passo 7 : Aplique os testes 2 e 3 a todos os grafos começando pelo primeiro. Sempre que remover um grafo trate os mintermos cobertos como no passo 3 (conversão em don't care) e observe se aparece algum novo vértice singular.

Passo 8 : Repita o passo 7 toda vez que pelo menos um grafo for removido, caso contrário (falha dos testes 2 e 3) vá ao passo 9.

Passo 9 : Remova o último grafo gerado do conjunto residual escolhendo sua cobertura de custo mínimo, e repita o passo 7 se o novo conjunto residual não for vazio. Se for vazio, FIM.

Estes passos podem ser esclarecidos com o exemplo a seguir :

Minimizar F7 = S(0,1,3,6,7,8,11,14,16,18,19,24,26,27,30) + D(9)

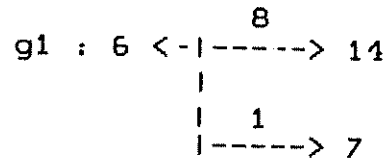
PASSO 0 : A função ordenada fica :

F7 = S(6,7,14,30,0,1,8,11,16,18,19,24,27,3,26) + D(9)

Note que 6,7,14 e 30 têm cada um 2 sucessores à distância 1; 0,1,8,11,16,18,19,24 e 27 têm 3; 3 e 26 têm 4 e 9 é don't care state.

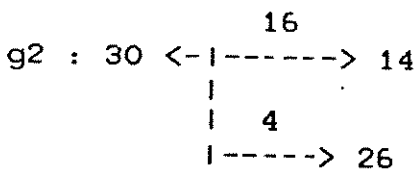
PASSO 1 : Escolho a primeira raiz, que é o 6.

PASSO 2 : Montando-se o seu grafo:



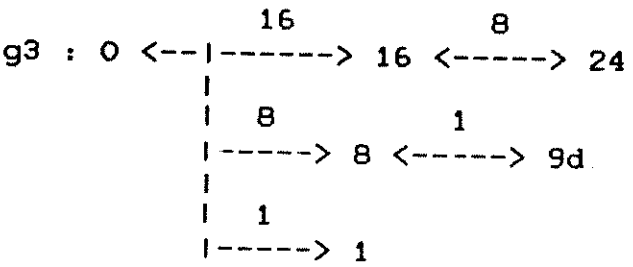
PASSO 3 : Falham os testes 1 e 2 . Escolho a segunda raiz, que é o 30 e volto ao passo 2.

PASSO 2 : Monto o grafo do 30:

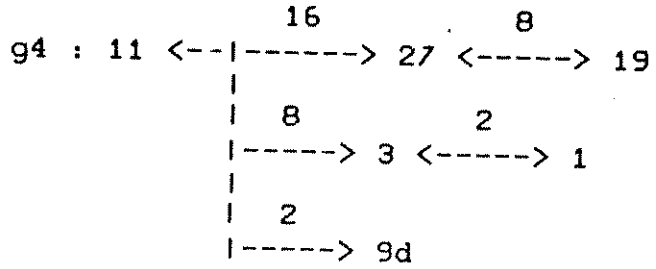


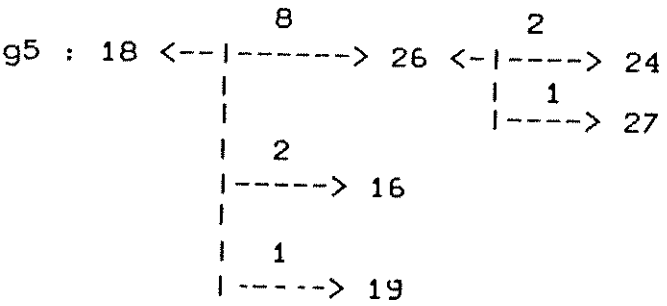
PASSO 3 : Também falham os testes 1 e 2. A terceira raiz é 0 e volto ao passo 2.

PASSO 2 : O grafo do 0 :



Os testes 1 e 2 falham, e assim também montam-se os grafos de 11 e 18, mostrados a seguir, verificando que os testes 1 e 2 falham para todos os grafos, ou seja, nenhum vértice é IPE-isolado ou D-isolado. Assim é necessário aplicar o teste 3 a todos eles, começando com o primeiro.





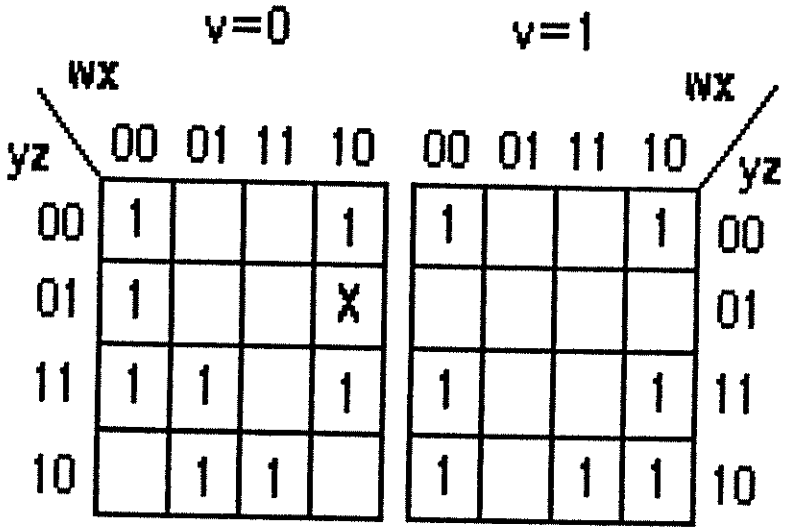
PASSO 6 : Marcar com um "s" todos os vértices singulares, ou seja, que estão somente em um grafo. Isto não será feito para não repetir nem rebuscar os desenhos dos grafos. Porém será analisado grafo a grafo: para g1, cuja raiz é o 6, nota-se que o 7 é um vértice singular, logo pelo teste 3 o grafo g1 é removido do conjunto e incluido na cobertura parcial da função F7, como o implicante primo 6[1]P.

Com a remoção do grafo g1, o vértice 14 se torna singular no grafo g2 e este é tambóm removido com o implicante primo 30[16]P.

No grafo g3, o vértice 8 é singular, e pelo teste 3 é removido formando o IP 0[9]P. Assim, também nos grafos g4 e g5 tem-se os vértices singulares 3 e 26, respectivamente. Logo, ambos são removidos pelo teste 3 e formam os IP's 3[24]P e 18[10]P. Com isto, o conjunto residual é vazio e a função está

completa :

$$F7 = \underset{g1}{6[1]P} + \underset{g2}{30[16]P} + \underset{g3}{0[9]P} + \underset{g4}{3[24]P} + \underset{g5}{18[10]P}$$



$$F = S(0,1,3,6,7,8,11,14,16,18,19,24,26,27,30)+D(9)$$

Figura III.9.- Mapa de Karnaugh para F7 = D(9) + + S(0,1,3,6,7,8,11,14,16,18,19,24,26,27,30)

III.6.- DESCRIÇÃO DO PROGRAMA "CARUSO" :

Desenvolveu-se um programa de computador que minimiza funções booleanas de acordo com o algoritmo de Caruso. O programa é descrito a seguir e apresentado na forma de diagrama de blocos na figura III.10.

Os dados de entrada do programa são:

.- Nome do arquivo de saída de resultados;

.- Número de variáveis da função;

.- Número de don't care states da função;

.- Todos os don't care states da função (decimais);

.- Número de mintermos da função;

.- Todos os mintermos da função (decimais);

Inicia-se pelo cálculo do custo inicial da função, como sendo o custo da soma de todos os mintermos, e conforme o algoritmo de minimização, o programa constrói os grafos e obtém os implicantes primos essenciais, verificando as propriedades dos grafos de acordo com os testes próprios.

Conferem-se os resultados, verificando-se se, com a fórmula mínima obtida para a função, é possível obter-se todos os seus mintermos. Calcula-se o custo final da função, dado pelo custo da soma de todos os essenciais.

Para comparação de custos, o programa minimiza também a função negada da mesma forma, o que equivale à minimização da função pelos seus maxtermos, conforme II.3.3.

Desta forma, os resultados obtidos são os custos inicial e final da função e da função negada, uma fórmula mínima para a função e uma para a função negada e a conferência destes resultados.

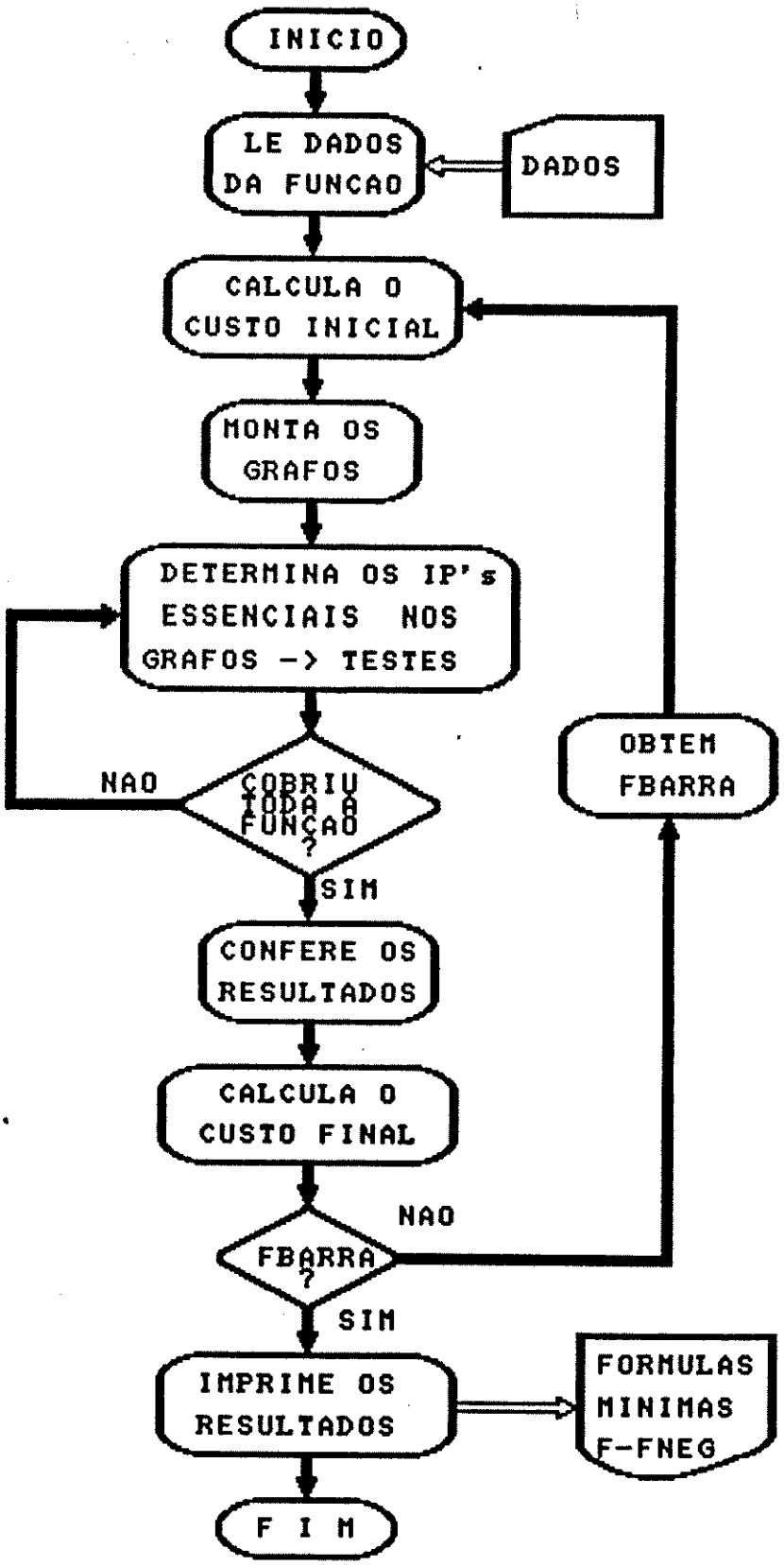


Figura III.10.- Diagrama de blocos do programa CARUSO

III.7.- EXEMPLOS E RESULTADOS :

Esta seção apresenta os resultados obtidos pelos programas MINIMO e CARUSO para as três funções utilizadas como exemplos no artigo original de Caruso :

F10 = S(0,1,3,6,7,8,11,14,16,18,19,24,26,27,30) + D(9)

F20 = S(1,2,3,4,5,8,9,10,17,20,21,24,25,27,32,33,34,36,37,
40,41,42,43,44,45,46,47,48,56,59,62)

F30 = S(0,1,2,4,5,7,8,10,11,13,14,15,17,18,19,20,22,23,24,
25,27,28,29,30)

Os resultados do programa MINIMO estão nas figuras III.11, III.13 e III.15 e os resultados do CARUSO estão nas figuras III.12, III.14 e III.16, para as funções F10, F20 e F30, respectivamente.

Na tabela III.12 são comparados os desempenhos dos dois programas nos três exemplos escolhidos.

	MINIMO	CARUSO
PROGRAMA FONTE (Bytes)	20562	14804
PROG. EXECUTÁVEL (. Bytes)	28380	20494
ÁREA DE CÓDIGO (' Bytes)	16992	9104
ÁREA DE DADOS (' Bytes)	688	13808
TEMPO MÉDIO EXECUCAO (seg)	110	35

Tabela III.12.- Comparação de desempenho: CARUSO X MINIMO

****

MINIMIZACAO DE FUNCOES BOOLEANAS

METODO DE "QUINE-McCLUSKEY"

CASO : F10 DATA : 29 DE ABRIL DE 1987

****

RESULTADOS DA MINIMIZACAO DA FUNCAO DADA

A FUNCAO E' REPRESENTADA PELOS SEGUINTES MINTERMOS:

0; 1; 3; 6; 7; 8; 11; 14; 16; 18; 19; 24;
26; 27; 30;

E PELOS SEGUINTES DON'T CARE STATES:

9;

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 2

ESSENCIAL: 14 REDUNDANCIA: 16
ESSENCIAL: 6 REDUNDANCIA: 1

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 1

ESSENCIAL: 16 REDUNDANCIA: 10
ESSENCIAL: 3 REDUNDANCIA: 24

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 0

ESSENCIAL: 0 REDUNDANCIA: 9

A FORMULA MINIMA DA FUNCAO E' A SOMA DESSES ESSENCIAIS

C U S T O S :

O CUSTO E' DADO PELA SOMA DO NUMERO DE ENTRADAS DOS GATES NECESSARIOS PARA UMA IMPLEMENTACAO DO TIPO SOMA DE PRODUTOS.

CUSTO INICIAL DA FUNCAO : 90

CUSTO APOS A COMBINACAO DOS MINTERMOS : 49

CUSTO FINAL DA FUNCAO : 22

```
ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO :
-----------------------------------------------

 30 , 27 , 26 , 24 , 19 , 18 , 16 , 14 , 11 ,  9 ,  8 ,  7 ,
  6 ,  3 ,  1 ,  0 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!


================================================
RESULTADOS DA MINIMIZACAO DA FUNCAO NEGADA
================================================

A FBARRA E' REPRESENTADA PELOS SEGUINTES MINTERMOS:
-----------------------------------------------------

 31; 29; 28; 25; 23; 22; 21; 20; 17; 15; 13; 12;
 10;  5;  4;  2;

E PELOS SEGUINTES DON'T CARE STATES:
------------------------------------

  9;

IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  0
---------------------------------------

ESSENCIAL:  2      REDUNDANCIA:  8
ESSENCIAL: 13      REDUNDANCIA: 18
ESSENCIAL: 17      REDUNDANCIA: 12
ESSENCIAL: 20      REDUNDANCIA:  3
ESSENCIAL:  4      REDUNDANCIA: 25

A FORMULA MINIMA DA FBARRA E' A SOMA DESSES ESSENCIAIS

          C U S T O S :
          =============

O CUSTO E' DADO PELA SOMA DO NUMERO DE ENTRADAS DOS GATES NECESSARIOS
PARA UMA IMPLEMENTACAO DO TIPO SOMA DE PRODUTOS.

CUSTO INICIAL DA FBARRA :  96

CUSTO APOS A COMBINACAO DOS MINTERMOS :  28

CUSTO FINAL DA FBARRA :  20

          RESULTADOS PARA CONFERENCIA :
          ============================

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FBARRA :
----------------------------------------------

 31 , 29 , 28 , 25 , 23 , 22 , 21 , 20 , 17 , 15 , 13 , 12 ,
 10 ,  5 ,  4 ,  2 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!
```

Figura III.11.- Resultado do programa MINIMO para F10

```
**********************************************************************

        MINIMIZACAO DE FUNCOES BOOLEANAS

        METODO DE "CARUSO"

        CASO : F10     DATA : 29 DE ABRIL DE 1987

**********************************************************************

==========================================
RESULTADOS DA MINIMIZACAO DA FUNCAO DADA
==========================================

MINTERMOS DA FUNCAO :
   6,(d)   7,(d)  14,(d)  30,(d)   0,(d)   1,(d)   8,(d)  11,(d)  16,(d)
  19,(d)  24,(d)  27,(d)   3,(d)  26,(d)

DONT CARE STATES DA FUNCAO:
   9


 G R A F O S :
 = = = = = = =

GRAFO 1 : 6,9=> 7,1,(s)=> 14,8=>

GRAFO 2 : 30,20=> 26,4=> 14,16=>

GRAFO 3 : 0,25=> 1,1=> 9,9=> 8,8,(s)=> 24,24,(s)=> 16,16,(s)=>

GRAFO 4 : 11,26=> 9,2=> 1,10=> 3,8,(s)=> 19,24=> 27,16=>

GRAFO 5 : 18,11=> 19,1=> 16,2=> 27,9=> 24,10=> 26,8=>


 I M P L I C A N T E S   E S S E N C I A I S :
 = = = = = = = = = = = =   = = = = = = = = = = =

ESSENCIAL =   1 ,  10  -  GRAFO :   4  -  TESTE : 2
ESSENCIAL =  18 ,   9  -  GRAFO :   5  -  TESTE : 2
ESSENCIAL =   0 ,  24  -  GRAFO :   3  -  TESTE : 3
ESSENCIAL =  14 ,  16  -  GRAFO :   2  -  TESTE : 3
ESSENCIAL =   6 ,   1  -  GRAFO :   1  -  TESTE : 3

CUSTO INICIAL DA FUNCAO DADA = 90

CUSTO FINAL DA FUNCAO OBTIDA = 22

        RESULTADOS PARA CONFERENCIA :
        ============================

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO :
----------------------------------------------
 30 , 27 , 26 , 24 , 19 , 18 , 16 , 14 , 11 ,  9 ,  8 ,  7 ,
  6 ,  3 ,  1 ,  0 ,


PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!
```

```
==========================================
RESULTADOS DA MINIMIZACAO DA FUNCAO NEGADA
==========================================

MINTERMOS DA FUNCAO NEGADA:
   2,(d)   10,(d)   15,(d)   17,(d)   22,(d)    4,(d)    5,(d)   12,(d)   23,(d)
  28,(d)   31,(d)   20,(d)   13,(d)   21,(d)   29,(d)

DONT CARE STATES DA FUNCAO NEGADA:
   9


 G R A F O S :
 = = = = = = =

GRAFO 1 : 2,8=> 10,8,(s)=>

GRAFO 2 : 15,18=> 13,2=> 29,18=> 31,16,(s)=>

GRAFO 3 : 17,12=> 21,4=> 29,12=> 25,8,(s)=>

GRAFO 4 : 22,3=> 23,1,(s)=> 21,3=> 20,2=>

GRAFO 5 : 4,25=> 5,1,(s)=> 13,9=> 12,8,(s)=> 21,17=> 29,25=> 28,24,(s)=> 2


 I M P L I C A N T E S   E S S E N C I A I S :
 = = = = = = = = = = =   = = = = = = = = = = =

ESSENCIAL =   4 ,  25  -  GRAFO :   5  -  TESTE : 1
ESSENCIAL =  20 ,   3  -  GRAFO :   4  -  TESTE : 1
ESSENCIAL =  17 ,  12  -  GRAFO :   3  -  TESTE : 1
ESSENCIAL =  13 ,  18  -  GRAFO :   2  -  TESTE : 1
ESSENCIAL =   2 ,   8  -  GRAFO :   1  -  TESTE : 1

CUSTO INICIAL DA FUNCAO NEGADA = 96

CUSTO FINAL DA FUNCAO NEGADA OBTIDA = 20


          RESULTADOS PARA CONFERENCIA :
          ============================


ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO NEGADA:
----------------------------------------------------
 31 , 29 , 28 , 25 , 23 , 22 , 21 , 20 , 17 , 15 , 13 , 12 ,
 10 ,  5 ,  4 ,  2 ,


PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!
```

Figura III.12.- Resultado do programa CARUSO para F10

****

MINIMIZACAO DE FUNCOES BOOLEANAS

METODO DE "QUINE-McCLUSKEY"

CASO : F20 DATA : 23 DE ABRIL DE 1987

****

RESULTADOS DA MINIMIZACAO DA FUNCAO DADA

A FUNCAO E' REPRESENTADA PELOS SEGUINTES MINTERMOS:

62; 59; 56; 48; 47; 46; 45; 44; 43; 42; 41; 40;
37; 36; 34; 33; 32; 27; 25; 24; 21; 20; 17; 10;
9; 8; 5; 4; 3; 2; 1;

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 2

ESSENCIAL: 1 REDUNDANCIA: 2

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 1

ESSENCIAL: 1 REDUNDANCIA: 24
ESSENCIAL: 8 REDUNDANCIA: 17
ESSENCIAL: 2 REDUNDANCIA: 40
ESSENCIAL: 32 REDUNDANCIA: 13
ESSENCIAL: 27 REDUNDANCIA: 32

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 0

ESSENCIAL: 4 REDUNDANCIA: 17
ESSENCIAL: 40 REDUNDANCIA: 7
ESSENCIAL: 32 REDUNDANCIA: 24
ESSENCIAL: 46 REDUNDANCIA: 16

A FORMULA MINIMA DA FUNCAO E' A SOMA DESSES ESSENCIAIS

C U S T O S :

O CUSTO E' DADO PELA SOMA DO NUMERO DE ENTRADAS DOS GATES NECESSARIOS PARA UMA IMPLEMENTACAO DO TIPO SOMA DE PRODUTOS.

CUSTO INICIAL DA FUNCAO : 217

CUSTO APOS A COMBINACAO DOS MINTERMOS : 109

CUSTO FINAL DA FUNCAO : 51

RESULTADOS PARA CONFERENCIA :
==========

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO :
---

62 , 59 , 56 , 48 , 47 , 46 , 45 , 44 , 43 , 42 , 41 , 40 ,
37 , 36 , 34 , 33 , 32 , 27 , 25 , 24 , 21 , 20 , 17 , 10 ,
9 , 8 , 5 , 4 , 3 , 2 , 1 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!

==========
RESULTADOS DA MINIMIZACAO DA FUNCAO NEGADA
==========

A FBARRA E' REPRESENTADA PELOS SEGUINTES MINTERMOS:

63; 61; 60; 58; 57; 55; 54; 53; 52; 51; 50; 49;
39; 38; 35; 31; 30; 29; 28; 26; 23; 22; 19; 18;
16; 15; 14; 13; 12; 11; 7; 6; 0;

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 1

ESSENCIAL: 52 REDUNDANCIA: 9
ESSENCIAL: 23 REDUNDANCIA: 40

IMPLICANTES PRIMOS ESSENCIAIS NIVEL: 0

ESSENCIAL: 0 REDUNDANCIA: 16
ESSENCIAL: 11 REDUNDANCIA: 4
ESSENCIAL: 12 REDUNDANCIA: 19
ESSENCIAL: 18 REDUNDANCIA: 37
ESSENCIAL: 35 REDUNDANCIA: 20
ESSENCIAL: 6 REDUNDANCIA: 49
ESSENCIAL: 49 REDUNDANCIA: 12
ESSENCIAL: 18 REDUNDANCIA: 40

A FORMULA MINIMA DA FBARRA E' A SOMA DESSES ESSENCIAIS

C U S T O S :
==========

O CUSTO E' DADO PELA SOMA DO NUMERO DE ENTRADAS DOS GATES NECESSARIOS
PARA UMA IMPLEMENTACAO DO TIPO SOMA DE PRODUTOS.

CUSTO INICIAL DA FBARRA : 231

CUSTO APOS A COMBINACAO DOS MINTERMOS : 89

CUSTO FINAL DA FBARRA : 49

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FBARRA :

63 , 61 , 60 , 58 , 57 , 55 , 54 , 53 , 52 , 51 , 50 , 49 ,
39 , 38 , 35 , 31 , 30 , 29 , 28 , 26 , 23 , 22 , 19 , 18 ,
16 , 15 , 14 , 13 , 12 , 11 , 7 , 6 , 0 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!

Figura III.13.- Resultado do programa MINIMO para F20

MINIMIZACAO DE FUNCOES BOOLEANAS

METODO DE "CARUSO"

CASO : F20 DATA : 29 DE ABRIL DE 1987

RESULTADOS DA MINIMIZACAO DA FUNCAO DADA

MINTERMOS DA FUNCAO :
62,(d) 3,(d) 20,(d) 27,(d) 48,(d) 59,(d) 2,(d) 4,(d) 10,(d) 1
21,(d) 24,(d) 34,(d) 47,(d) 56,(d) 5,(d) 8,(d) 9,(d) 25,(d) 3
36,(d) 37,(d) 43,(d) 44,(d) 45,(d) 46,(d) 1,(d) 32,(d) 41,(d) 4
40,(d)

G R A F O S :

GRAFO 1 : 62,16=> 46,16=>

GRAFO 2 : 3,3=> 2,1=> 1,2=>

GRAFO 3 : 20,17=> 21,1,(s)=> 5,17,(s)=> 4,16,(s)=>

GRAFO 4 : 27,34=> 25,2=> 59,32,(s)=>

GRAFO 5 : 48,24=> 56,8=> 40,24=> 32,16=>

GRAFO 6 : 10,42=> 8,2=> 2,8=> 40,34=> 32,42=> 34,40,(s)=> 42,32=>

GRAFO 7 : 17,28=> 21,4=> 25,8=> 5,20=> 9,24,(s)=> 1,16=>

GRAFO 8 : 24,49=> 25,1=> 9,17=> 8,16=> 41,49=> 40,48=> 56,32=>

GRAFO 9 : 47,7=> 46,1=> 44,3=> 45,2=> 42,5=> 40,7=> 41,6=> 43,4,(s)=>

GRAFO 10 : 33,45=> 32,1=> 36,5,(s)=> 37,4,(s)=> 40,9=> 44,13=> 45,12=> 41,8=
=>4,37=> 5,36=> 8,41=> 9,40=> 1,32=>

```
I M P L I C A N T E S   E S S E N C I A I S :
= = = = = = = = = = =   = = = = = = = = = = =

ESSENCIAL =  32 ,  13  -  GRAFO :  10  -  TESTE : 2
ESSENCIAL =   2 ,  40  -  GRAFO :   6  -  TESTE : 2
ESSENCIAL =   8 ,  48  -  GRAFO :   8  -  TESTE : 2
ESSENCIAL =  40 ,   7  -  GRAFO :   9  -  TESTE : 1
ESSENCIAL =  27 ,  32  -  GRAFO :   4  -  TESTE : 2
ESSENCIAL =   2 ,   1  -  GRAFO :   2  -  TESTE : 2
ESSENCIAL =   1 ,  24  -  GRAFO :   7  -  TESTE : 2
ESSENCIAL =  32 ,  24  -  GRAFO :   5  -  TESTE : 1
ESSENCIAL =   4 ,  17  -  GRAFO :   3  -  TESTE : 1
ESSENCIAL =  46 ,  16  -  GRAFO :   1  -  TESTE : 1

CUSTO INICIAL DA FUNCAO DADA = 217

CUSTO FINAL DA FUNCAO OBTIDA = 51

          RESULTADOS PARA CONFERENCIA :
          ==============================

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO :
----------------------------------------------

 62 , 59 , 56 , 48 , 47 , 46 , 45 , 44 , 43 , 42 , 41 , 40 ,
 37 , 36 , 34 , 33 , 32 , 27 , 25 , 24 , 21 , 20 , 17 , 10 ,
  9 ,  8 ,  5 ,  4 ,  3 ,  2 ,  1 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!

==========================================
RESULTADOS DA MINIMIZACAO DA FUNCAO NEGADA
==========================================

MINTERMOS DA FUNCAO NEGADA:
   0,(d)   11,(d)   16,(d)   35,(d)   57,(d)   58,(d)   12,(d)   13,(d)   19,(d)   26,(d)
  38,(d)   49,(d)   52,(d)   60,(d)   63,(d)    6,(d)    7,(d)   14,(d)   28,(d)   29,(d)
  39,(d)   50,(d)   53,(d)   15,(d)   18,(d)   22,(d)   23,(d)   30,(d)   31,(d)   51,(d)
  54,(d)   61,(d)   55,(d)

 G R A F O S :
 = = = = = = =

GRAFO 1 : 0,16=> 16,16,(s)=>

GRAFO 2 : 11,4=> 15,4=>

GRAFO 3 : 35,20=> 39,4=> 55,20=> 51,16=>

GRAFO 4 : 57,12=> 61,4=> 53,12=> 49,8,(s)=>

GRAFO 5 : 58,40=> 50,8=> 18,40=> 26,32,(s)=>

GRAFO 6 : 12,19=> 13,1,(s)=> 15,3=> 14,2,(s)=> 29,17,(s)=> 31,19=> 30,18,(s)=>
          => 28,16,(s)=>

GRAFO 7 : 19,37=> 18,1=> 22,5=> 23,4=> 50,33=> 54,37=> 55,36=> 51,32=>

GRAFO 8 : 38,49=> 39,1=> 55,17=> 54,16=> 7,33,(s)=> 23,49=> 22,48=> 6,32,(s)=>

GRAFO 9 : 52,11=> 53,1=> 55,3=> 54,2=> 61,9=> 60,8,(s)=>

GRAFO 10 : 63,42=> 61,2=> 53,10=> 55,8=> 29,34=> 23,40=> 31,32=>
```

```
  I M P L I C A N T E S   E S S E N C I A I S :
  = = = = = = = = = = =   = = = = = = = = = = =
ESSENCIAL =  23 ,  40  -  GRAFO :  10  -  TESTE : 2
ESSENCIAL =  52 ,   9  -  GRAFO :   9  -  TESTE : 2
ESSENCIAL =   6 ,  49  -  GRAFO :   8  -  TESTE : 1
ESSENCIAL =  18 ,  37  -  GRAFO :   7  -  TESTE : 1
ESSENCIAL =  12 ,  19  -  GRAFO :   6  -  TESTE : 1
ESSENCIAL =  18 ,  40  -  GRAFO :   5  -  TESTE : 1
ESSENCIAL =  49 ,  12  -  GRAFO :   4  -  TESTE : 1
ESSENCIAL =  35 ,  20  -  GRAFO :   3  -  TESTE : 1
ESSENCIAL =  11 ,   4  -  GRAFO :   2  -  TESTE : 1
ESSENCIAL =   0 ,  16  -  GRAFO :   1  -  TESTE : 1

CUSTO INICIAL DA FUNCAO NEGADA = 231

CUSTO FINAL DA FUNCAO NEGADA OBTIDA = 49

          RESULTADOS PARA CONFERENCIA :
          ============================

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO NEGADA:
----------------------------------------------------
 63 , 61 , 60 , 58 , 57 , 55 , 54 , 53 , 52 , 51 , 50 , 49 ,
 39 , 38 , 35 , 31 , 30 , 29 , 28 , 26 , 23 , 22 , 19 , 18 ,
 16 , 15 , 14 , 13 , 12 , 11 ,  7 ,  6 ,  0 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!
```

Figura III.14.- Resultado do programa CARUSO para F20

```
**********************************************************************

     MINIMIZACAO DE FUNCOES BOOLEANAS

     METODO DE "QUINE-McCLUSKEY"

     CASO : F30      DATA : 29 DE ABRIL DE 1987

**********************************************************************

========================================
RESULTADOS DA MINIMIZACAO DA FUNCAO DADA
========================================

A FUNCAO E' REPRESENTADA PELOS SEGUINTES MINTERMOS:
---------------------------------------------------
 30; 29; 28; 27; 25; 24; 23; 22; 20; 19; 18; 17;
 15; 14; 13; 11; 10;  8;  7;  5;  4;  2;  1;  0;

IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  5
---------------------------------------
ESSENCIAL: 18      REDUNDANCIA:  5
```

```
IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  4
-----------------------------------------
ESSENCIAL:  0      REDUNDANCIA: 10
ESSENCIAL:  5      REDUNDANCIA: 10

IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  3
-----------------------------------------
ESSENCIAL: 24      REDUNDANCIA:  5

IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  2
-----------------------------------------
ESSENCIAL: 17      REDUNDANCIA: 10

IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  1
-----------------------------------------
ESSENCIAL:  0      REDUNDANCIA:  5
ESSENCIAL: 10      REDUNDANCIA:  5

IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  0
-----------------------------------------
ESSENCIAL: 20      REDUNDANCIA: 10

A FORMULA MINIMA DA FUNCAO E' A SOMA DESSES ESSENCIAIS

          C U S T O S :
          =============

O CUSTO E' DADO PELA SOMA DO NUMERO DE ENTRADAS DOS GATES NECESSARIOS
PARA UMA IMPLEMENTACAO DO TIPO SOMA DE PRODUTOS.

CUSTO INICIAL DA FUNCAO : 144

CUSTO APOS A COMBINACAO DOS MINTERMOS :  72

CUSTO FINAL DA FUNCAO :  32

          RESULTADOS PARA CONFERENCIA :
          =============================

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO :
-----------------------------------------------
 30 , 29 , 28 , 27 , 25 , 24 , 23 , 22 , 20 , 19 , 18 , 17 ,
 15 , 14 , 13 , 11 , 10 ,  8 ,  7 ,  5 ,  4 ,  2 ,  1 ,  0 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!


==========================================
RESULTADOS DA MINIMIZACAO DA FUNCAO NEGADA
==========================================

A FBARRA E' REPRESENTADA PELOS SEGUINTES MINTERMOS:
---------------------------------------------------
 31; 26; 21; 16; 12;  9;  6;  3;
```

```
IMPLICANTES PRIMOS ESSENCIAIS NIVEL:  0
---------------------------------------

ESSENCIAL:  3      REDUNDANCIA:  0
ESSENCIAL:  6      REDUNDANCIA:  0
ESSENCIAL:  9      REDUNDANCIA:  0
ESSENCIAL: 12      REDUNDANCIA:  0
ESSENCIAL: 16      REDUNDANCIA:  0
ESSENCIAL: 21      REDUNDANCIA:  0
ESSENCIAL: 26      REDUNDANCIA:  0
ESSENCIAL: 31      REDUNDANCIA:  0


A FORMULA MINIMA DA FBARRA E' A SOMA DESSES ESSENCIAIS


          C U S T O S :
          =============


O CUSTO E' DADO PELA SOMA DO NUMERO DE ENTRADAS DOS GATES NECESSARIOS
PARA UMA IMPLEMENTACAO DO TIPO SOMA DE PRODUTOS.

CUSTO INICIAL DA FBARRA :  48

CUSTO APOS A COMBINACAO DOS MINTERMOS :  48

CUSTO FINAL DA FBARRA :  48


          RESULTADOS PARA CONFERENCIA :
          ==============================


ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FBARRA :
----------------------------------------------

 31 , 26 , 21 , 16 , 12 ,  9 ,  6 ,  3 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!
```

Figura III.15.- Resultado do programa MINIMO para F30

```
**********************************************************************

     MINIMIZACAO DE FUNCOES BOOLEANAS

     METODO DE "CARUSO"

     CASO : F30       DATA : 29 DE ABRIL DE 1987

**********************************************************************


==============================================
RESULTADOS DA MINIMIZACAO DA FUNCAO DADA
==============================================

MINTERMOS DA FUNCAO :
   1,(d)   2,(d)   4,(d)   7,(d)   8,(d)  11,(d)  13,(d)  14,(d)  17,(d)  18,(d)
  20,(d)  23,(d)  24,(d)  27,(d)  29,(d)  30,(d)   0,(d)   5,(d)  10,(d)  15,(d)
  19,(d)  22,(d)  25,(d)  28,(d)


 G R A F O S :
 = = = = = = =


GRAFO 1 : 1,21=> 0,1=> 4,5,(s)=> 5,4=> 17,16=>

GRAFO 2 : 2,26=> 0,2=> 8,10,(s)=> 10,8=> 18,16=>

GRAFO 3 : 7,26=> 5,2=> 13,10,(s)=> 15,8=> 23,16=>

GRAFO 4 : 11,21=> 10,1=> 14,5,(s)=> 15,4=> 27,16=>

GRAFO 5 : 20,26=> 22,2=> 30,10,(s)=> 28,8=> 4,16=>

GRAFO 6 : 24,21=> 25,1=> 29,5,(s)=> 28,4=> 8,16=>

GRAFO 7 : 19,15=> 18,1=> 17,2=> 22,5=> 23,4=> 25,10=> 27,8=>

 I M P L I C A N T E S   E S S E N C I A I S :
 = = = = = = = = = = =   = = = = = = = = = = =

ESSENCIAL =   0 ,  10  -  GRAFO :   2  -  TESTE : 2
ESSENCIAL =   0 ,   5  -  GRAFO :   1  -  TESTE : 2
ESSENCIAL =  17 ,  10  -  GRAFO :   7  -  TESTE : 3
ESSENCIAL =  18 ,   5  -  GRAFO :   7  -  TESTE : 3
ESSENCIAL =  24 ,   5  -  GRAFO :   6  -  TESTE : 3
ESSENCIAL =  20 ,  10  -  GRAFO :   5  -  TESTE : 3
ESSENCIAL =  10 ,   5  -  GRAFO :   4  -  TESTE : 3
ESSENCIAL =   5 ,  10  -  GRAFO :   3  -  TESTE : 3

CUSTO INICIAL DA FUNCAO DADA = 144

CUSTO FINAL DA FUNCAO OBTIDA = 32
```

RESULTADOS PARA CONFERENCIA :

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO :

30 , 29 , 28 , 27 , 25 , 24 , 23 , 22 , 20 , 19 , 18 , 17 ,
15 , 14 , 13 , 11 , 10 , 8 , 7 , 5 , 4 , 2 , 1 , 0 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!

RESULTADOS DA MINIMIZACAO DA FUNCAO NEGADA

MINTERMOS DA FUNCAO NEGADA:
3,(d) 6,(d) 9,(d) 12,(d) 16,(d) 21,(d) 26,(d) 31,(d)

G R A F O S :

GRAFO 1 : 3,0=)

GRAFO 2 : 6,0=)

GRAFO 3 : 9,0=)

GRAFO 4 : 12,0=)

GRAFO 5 : 16,0=)

GRAFO 6 : 21,0=)

GRAFO 7 : 26,0=)

GRAFO 8 : 31,0=)

I M P L I C A N T E S E S S E N C I A I S :

ESSENCIAL = 31 , 0 - GRAFO : 8 - TESTE : 1
ESSENCIAL = 26 , 0 - GRAFO : 7 - TESTE : 1
ESSENCIAL = 21 , 0 - GRAFO : 6 - TESTE : 1
ESSENCIAL = 16 , 0 - GRAFO : 5 - TESTE : 1
ESSENCIAL = 12 , 0 - GRAFO : 4 - TESTE : 1
ESSENCIAL = 9 , 0 - GRAFO : 3 - TESTE : 1
ESSENCIAL = 6 , 0 - GRAFO : 2 - TESTE : 1
ESSENCIAL = 3 , 0 - GRAFO : 1 - TESTE : 1

CUSTO INICIAL DA FUNCAO NEGADA = 48

CUSTO FINAL DA FUNCAO NEGADA OBTIDA = 48

RESULTADOS PARA CONFERENCIA :

ENTRADAS QUE PRODUZEM "1" NA SAIDA DA FUNCAO NEGADA:

31 , 26 , 21 , 16 , 12 , 9 , 6 , 3 ,

PARA QUE OS RESULTADOS SEJAM CORRETOS TODAS ESSAS
ENTRADAS DEVEM SER MINTERMOS OU DON'T CARE STATES !!

Figura III.16.- Resultado do programa CARUSO para F30

III.8.- CONCLUSÃO :

Este capítulo descreveu e comentou dois algoritmos distintos que realizam a minimização das funções booleanas, o algoritmo de "Quine-McCluskey" e o algoritmo de "Caruso".

O algoritmo do "Quine-McCluskey" foi escolhido para servir de base de comparação, e o algoritmo de "Caruso" despertou especial interesse no autor ao procurar sanar o principal problema apresentado pelo primeiro, quanto ao consumo do tempo na geração dos implicantes primos.

No artigo original, Caruso apresenta uma notação bastante carregada e distante de ser adequada a uma implementação computacional. Por este motivo, o autor introduziu uma nova notação, tentando apropriá-la à automatização do método e aproximá-la, tanto quanto possível, da notação utilizada no algoritmo de Quine-McCluskey.

Ambos os algoritmos foram implementados em programas de computador afim de serem testados e comparados.

O método de Caruso mostrou ser mais rápido e consumir menos memória que o de Quine-McCluskey. A sua única desvantagem é não garantir a obtenção de uma fórmula de custo mínimo sempre, mas sim de cobertura mínima. Contudo, conforme ilustrado nos exemplos da seção III.7, todos os casos testados geraram fórmulas de custo mínimo.

Já o algoritmo de Quine-McCluskey garante a obtenção da fórmula de custo mínimo das funções, o que pode ter grande interesse acadêmico. Porém, no contexto de "engenharia" ao qual deve atender um pacote de software de apoio ao projeto de circuitos digitais, o autor recomenda a inclusão do algoritmo de

Caruso.

Neste contexto, a minimização de funções booleanas representa uma etapa importante dentro de uma sistemática conhecida e bastante árdua.

É no ambiente definido por esta sistemática que se apresenta o capítulo seguinte sobre Máquinas Sequenciais Síncronas.

III.9.- REFERENCIAS BIBLIOGRAFICAS :

1.- KOHAVI , Zvi

"Switching and Finite Automata Theory"

McGraw-Hill , 1970 - 592 p.

2.- HILL, F. J. & PETERSON, G. R.

"Introduction to Switching Theory & Logical Design"

2ed. Wiley International Edition , 1974 - 596 p.

3.- MADUREIRA, Marcos C.

"Síntese de Circuitos Digitais - Minimização de Funções Booleanas" - Relatório Interno n. 015/86

FEC - UNICAMP - Campinas.

4.- CARUSO, G.

"A Local Selection Algorithm for Switching Function Minimization"

IEEE Transactions on Computers, Vol. C33, n. 1

January 1984 - pp 91-97

5.- MADUREIRA, Marcos C.

"Estado da Arte em Análise e Síntese de Circuitos Digitais" - Relatório Interno n. 016/86

FEC - UNICAMP - Campinas.

CAPÍTULO 4

MÁQUINAS SEQUENCIAIS SÍNCRONAS

IV.1.-RESUMO :

O objetivo deste capítulo é apresentar um programa de computador para projeto de circuitos binários sequenciais síncronos.

Por circuito sequencial síncrono, entendem-se as máquinas sequenciais onde os elementos de memória são claramente identificáveis (Flip-Flops) e nos quais está presente uma entrada particular, denominada relógio, que "sincroniza" os eventos a que está sujeita esta máquina .

Assim, é perfeitamente identificável uma sequência enumerável de instantes através dos quais a descrição da máquina é realizada.

IV.2.-INTRODUÇÃO :

Máquinas sequenciais são usadas imperceptivelmente no cotidiano. São exemplos : o controle de um elevador, os semáforos, e até o tradicional segredo de um cofre, que pode não ter nada de eletrônico, mas que além de armazenar os números do segredo, armazena sua sequência.

Na maioria dos sistemas digitais são necessários circuitos capazes de armazenar e de efetuar algumas operações lógicas ou numéricas com dados. As saídas destes circuitos são funções das entradas externas dos circuitos bem como da informação armazenada neles. Tais circuitos são chamados circuitos sequenciais.

Máquina sequencial, ou máquina de estados finitos, é o modelo abstrato do circuito sequencial real. Seu comportamento é descrito como uma sequência de eventos que ocorrem em instantes de tempo discretos designados por t1, t2, t3,... .

Suponha que uma máquina **M** receba sinais de entrada e responda produzindo sinais de saída. Se, no instante **t**, for aplicado um sinal **x(t)** a **M**, sua resposta **z(t)** depende de x(t) bem como das entradas de M nos instantes anteriores. Como a máquina M pode ter uma variedade infinita de passados possíveis, seria necessária uma capacidade de armazenamento infinita para guardá-los todos.

Este capítulo concentra-se nas máquinas cujo passado pode ser resumido em um conjunto finito de variáveis (Máquinas Determinísticas ou Markovianas) .

Por exemplo, num somador binário serial (figura IV.1), a resposta aos sinais de entrada no instante t depende apenas destes sinais e do valor do transporte gerado no instante anterior. Assim, embora o somador possa ter um número infinito de passados, eles podem ser agrupados em duas grandes classes, aquelas que resultam em transporte igual a 1 e aquelas que resultam em transporte igual a 0 num dado instante.

Desta forma, são estudadas aqui as máquinas que podem distinguir entre um número finito de classes de entradas passadas, e estas classes são chamadas de estados internos destas máquinas. Assim, toda máquina de estados finitos contém um número finito de elementos de memória. Note-se que, embora restritos a máquinas que têm capacidade finita de armazenamento, não foi estabelecido nenhum limite para a duração da influencia de uma entrada particular no comportamento futuro da máquina.

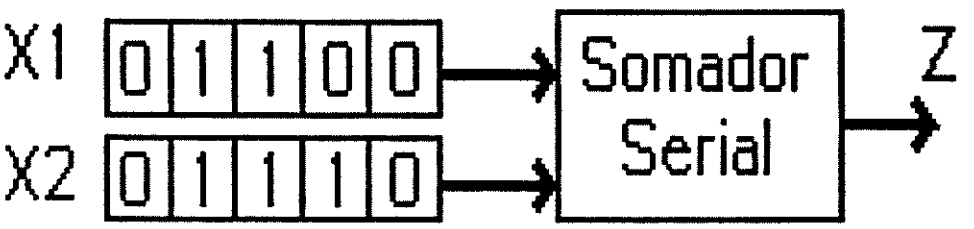


Figura IV.1.- Somador binário serial

IV.3.- MAQUINAS SEQUENCIAIS :

IV.3.1.-INTRODUÇÃO:

Uma máquina sequencial pode ser representada esquematicamente pelo circuito da figura IV.2. O circuito tem um número finito r de entradas. Os sinais aplicados a estas entradas constituem o conjunto das variáveis de entrada (X1,X2,...,Xr). Uma r-upla ordenada destes valores forma o vetor de entrada (ou simplesmente, entrada).

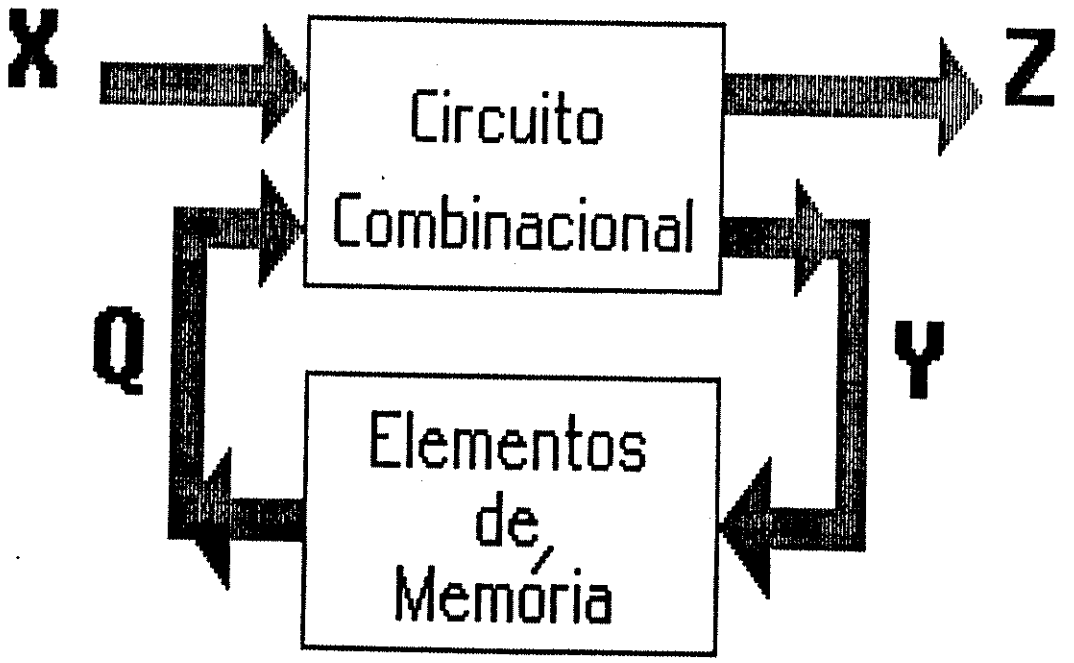


Figura IV.2. - Esquema de uma máquina sequencial .

Analogamente, o circuito tem um número finito m de saídas. Os sinais obtidos nestas saídas constituem o conjunto das variáveis de saída (Z1,Z2,...,Zm). Uma m-upla ordenada destes valores é o vetor de saída (ou saída).

O valor contido em cada elemento de memória é chamado de variável de estado, e o conjunto (Q1,Q2,...,Qk) constitue o conjunto das variáveis de estado. Os valores contidos nos **k** elementos de memória definem o estado interno corrente da máquina.

As saídas Z1,Z2,..Zm e as funções de transições internas Y1,Y2,..Yk são funções das entradas externas e do estado interno da máquina, e são definidas através do circuito combinacional mostrado na figura IV.1. Os valores **Y's**, que aparecem nas saídas do circuito combinacional no instante **t**, determinam os valores das variáveis de estado no instante **t+1**, e portanto definem o próximo estado da máquina.

IV.3.2.- TABELAS DE ESTADO E DIAGRAMAS DE ESTADO :

As relações entre entradas, estados presentes, saídas e próximos estados podem ser descritas por um diagrama de estados ou por uma tabela de estados ou por uma tabela de transição.

Uma tabela de estados tem p = 2**r colunas, uma para cada ocorrência do vetor de entrada, e n = 2**k linhas, uma para cada estado. Cada combinação de entradas e estados presentes determinam a saída produzida e o próximo estado para a máquina .

O diagrama de estados é um grafo orientado, onde cada estado da máquina corresponde a um nó. De cada nó emanam p arcos orientados, correspondendo às transições de estados causadas pela ocorrência da entrada. Cada arco orientado é rotulado com a entrada que determina aquela transição e com a saída gerada (Mealy).

As máquinas que determinam o próximo estado **Q(t+1)** unicamente com base no estado **Q(t)** e na entrada **X(t)** presentes são chamadas de **máquinas determinísticas** ou máquinas markovianas.

Nas máquinas determinísticas,

$$Q(t+1) = f(Q(t),X(t)) \quad (IV.1)$$

onde **f** é a função de transição de estados. O valor da saída **Z(t)** é função do estado presente **Q(t)** e, muitas vezes, das entradas presentes **X(t)**, ou seja :

$$Z(t) = g(Q(t)) \quad (IV.2) \quad \text{ou,}$$

$$Z(t) = g(Q(t),X(t)) \quad (IV.3)$$

onde **g** é a função saída.

Uma máquina com as propriedades descritas nas equações (IV.1) e (IV.2) é conhecida como **máquina de Moore** e uma máquina com as características das equações (IV.1) e (IV.3) é conhecida como **máquina de Mealy**. Nas figuras IV.3 e IV.4 são mostradas respectivamente as representações de Mealy e de Moore para um circuito que identifica (Z=1) se a primeira sequência de pulsos (numa cadeia de símbolos 0 e 1) for 101 a partir do estado inicial q0. [2]

Qualquer circuito expresso como uma máquina de Moore pode ser expresso como uma máquina de Mealy e vice-versa. Como no exemplo ilustrado nas figuras IV.3 e IV.4, geralmente uma representação de Moore para uma dada máquina necessita de mais estados do que uma representação de Mealy.

MÁQUINAS DETERMINÍSTICAS

q(t)	q(t+1), Z(t)	
	X=1	X=0
q0	q1,0	q2,0
q1	q2,0	q3,0
q2	q2,0	q2,0
q3	q2,1	q2,0

MÁQUINA DE MEALY

Q(t+1)=F(Q(t),X(t))
Z(t)=G(Q(t),X(t))

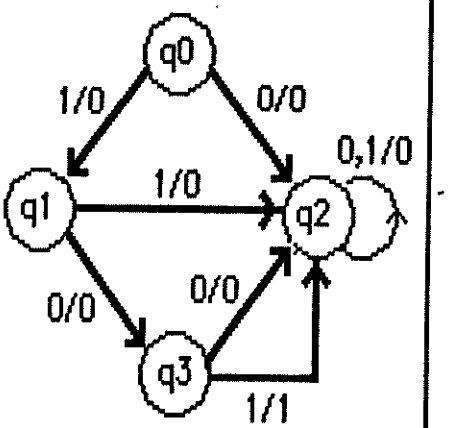


Figura IV.3.-Máquina de Mealy: Tabela e diagrama de estados

q(t)	q(t+1)		Z(t)
	X=1	X=0	
q0	q1	q2	0
q1	q2	q3	0
q2	q2	q2	0
q3	q4	q2	0
q4	q2	q2	1

MÁQUINA DE MOORE

Q(t+1)=F(Q(t),X(t))
Z(t)=G(Q(t))

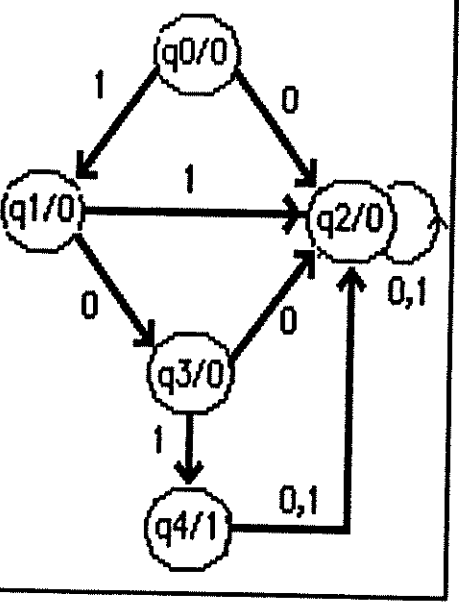


Figura IV.4.-Máquina de Moore: Tabela e diagrama de estados

IV.3.3.- ELEMENTOS DE MEMORIA - TABELAS DE TRANSIÇÃO :

Através do uso de elementos de memória, denominados multivibradores biestáveis, capazes de assumir dois estados distintos (armazenam um dígito binário), projeta-se um circuito que opera de acordo com as especificações de uma dada tabela. Tais elementos são mais comumente conhecidos por "flip-flops". Os estados destes elementos são associados aos estados da máquina por um processo chamado alocação de estados.

Uma tabela de transição especifica o próximo estado dos elementos de memória para cada combinação de entradas e variáveis de estado. Tomando-se a máquina da figura IV.3, pode-se obter sua tabela de transição fazendo-se da forma mostrada na figura IV.5.

q0 = 00 q2 = 10
q1 = 01 q3 = 11

Q1Q2(t)	Q1Q2(t+1),Z(t)	
	X = 1	X = 0
00	01,0	10,0
01	10,0	11,0
10	10,0	10,0
11	10,1	10,0

Figura IV.5.- Máquina de Mealy - Tabela de transição

Para gerar estes próximos estados da maneira especificada, os elementos de memória devem ser supridos com as entradas apropriadas. A função lógica que descreve o efeito das entradas

de um elemento de memória e de seu estado presente na determinação de seu próximo estado é chamada de função característica do elemento de memória (H), definida por :

$$Q(t+1) = H[Q(t),Y(t)]$$

Por exemplo, para o flip-flop tipo D, conforme a tabela IV.1:

$$Q(t+1) = Y(t) = D(t),$$

Para o flip-flop JK, conforme a tabela IV.2 :

$$Q(t+1) = Q'(t).J(t) + Q(t).K'(t)$$

Assim, para efeito de projeto das máquinas o problema se resume à escolha dos elementos de memória e à síntese de suas funções características H.

Q(t)	Q(t+1)	D(t)
0	0	0
0	1	1
1	0	0
1	1	1

Tabela IV.1.- Função característica do flip-flop D .

Q(t)	Q(t+1)	J(t)	K(t)
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Tabela IV.2.- Função característica do flip-flop JK .

Note-se na tabela IV.2 que, para algumas transições, há entradas irrelevantes(X) que se transformam em don't care states na síntese da função lógica correspondente. Isto proporciona, normalmente, circuitos combinacionais mais simples quando utilizam-se flip-flops JK ao invés de D.

As máquinas sequenciais onde aparecem estes estados irrelevantes são chamadas de máquinas sequenciais incompletamente especificadas. Nestas máquinas podem existir estados que nunca serão atingidos.

IV.3.4.- SISTEMATIZAÇÃO RESUMIDA DE PROJETO :

Primeiramente, o modelamento de um circuito sequencial pode ser feito por um diagrama de estado. Tal diagrama define as transições internas de estado e suas saídas quando submetidos a mudanças na entrada.

A partir do diagrama é construida uma "tabela inicial de estados ". Geralmente, o número de variáveis de estado nesta tabela pode ser reduzido e uma redução de estados sistemática pode ser implementada neste ponto.

Cada variável de estado na "tabela reduzida" é representada por um código binário único (alocação de estados) e a "tabela de estados alocados" resultante é analisada com o objetivo de definir as " funções de transições internas " e as " funções de saída". A partir destas funções é possível a realização física do circuito e as funções podem ser implementadas usando elementos lógicos padrões, como gates.

O problema da alocação inicial de estados é bastante relevante no que diz respeito ao desempenho e custos dos circuitos finais.

O número de alocações iniciais possíveis é bastante grande para permitir uma enumeração completa de todas elas como um método viável de solução. Um grande número de procedimentos para a obtenção de alocações iniciais de estados "boas" ou "quase ótimas", têm sido propostos nesses últimos anos.

A sistematização e automação do projeto de circuitos digitais é extremamente conveniente pois retira do projetista as tarefas cansativas, sujeitas a erros e demoradas, para permitir-lhe uma análise crítica das possíveis soluções. É neste contexto que é apresentado na seção IV.4 o programa TABELA.

IV.4.- PROGRAMA TABELA :

Foi desenvolvido e implementado um programa (figura IV.6) que gera a tabela de transição de uma máquina sequencial a partir de seu diagrama de estados e minimiza as funções de transições internas correspondentes aos elementos de memória utilizados e as funções de saída do circuito.

Os dados requisitados pelo programa são:

.- Nome do dispositivo de saída de resultados;

.- Número de flip-flops;

.- Tipo de cada um deles (D ou JK);

.- Número de variáveis de entrada;

.- Número de variáveis de saída;

.- Transições entre estados, na forma

estado atual, próximo estado, entrada, saída.

Os estados, as entradas e as saídas devem estar na forma decimal.

As máquinas podem ser completa ou incompletamente especificadas, e devem estar na representação de Mealy.

O programa monta a tabela de transição armazenando-a no arquivo de saída. A partir desta tabela são obtidos os mintermos e os don't care states das funções de transições internas de todos os flip-flops e da saída do circuito (máquina).

Utilizando-se um algoritmo de minimização de funções booleanas, estas funções são obtidas nas suas fórmulas mínimas.

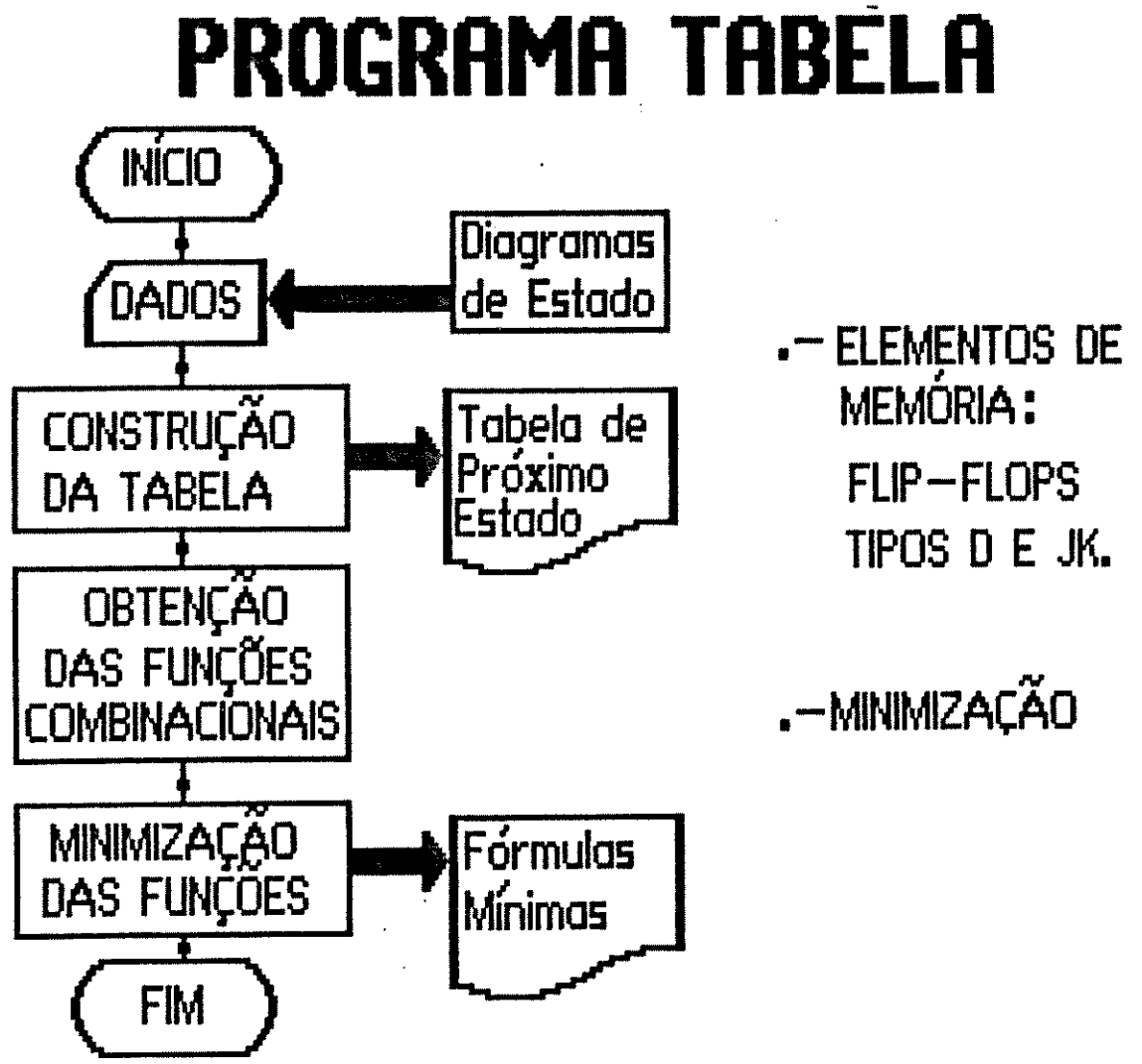


Figura IV.6.- Diagrama de blocos do programa Tabela

EXEMPLO : Detetor de Sincronismo :

Kubota[5] projetou e construiu um Circuito de Controle de Sincronismo de Quadro de um PCM de 120 canais, chamado de CCS1.

O circuito, cujo diagrama de estado é mostrado na figura IV.7, tem 7 estados de interesse e um bit de entrada que identifica se a palavra de sincronismo foi detetada correta (c) ou incorretamente (e).

Um diagrama com 7 estados pode ser representado por 3 flip-flops, o que é mostrado na figura IV.8, onde é definida uma

alocação de estados e Y=1 identifica se a palavra de sincronismo foi detetada correta ou incorretamente.

Kubota, porém, utilizou 4 flip-flops e fez uma alocação de estados de forma a aproximar as transições a deslocamentos dos bits como num registrador de deslocamento. Este procedimento deu origem à ocorrência de vários don't care states. No diagrama de estados da figura IV.9 é mostrada a alocação utilizada.

As figuras IV.10 e IV.11 são geradas a partir do programa TABELA, mostrando os resultados obtidos para os dois casos de alocação de estados. A figura IV.12 mostra o circuito correspondente à solução de 4 flip-flops.

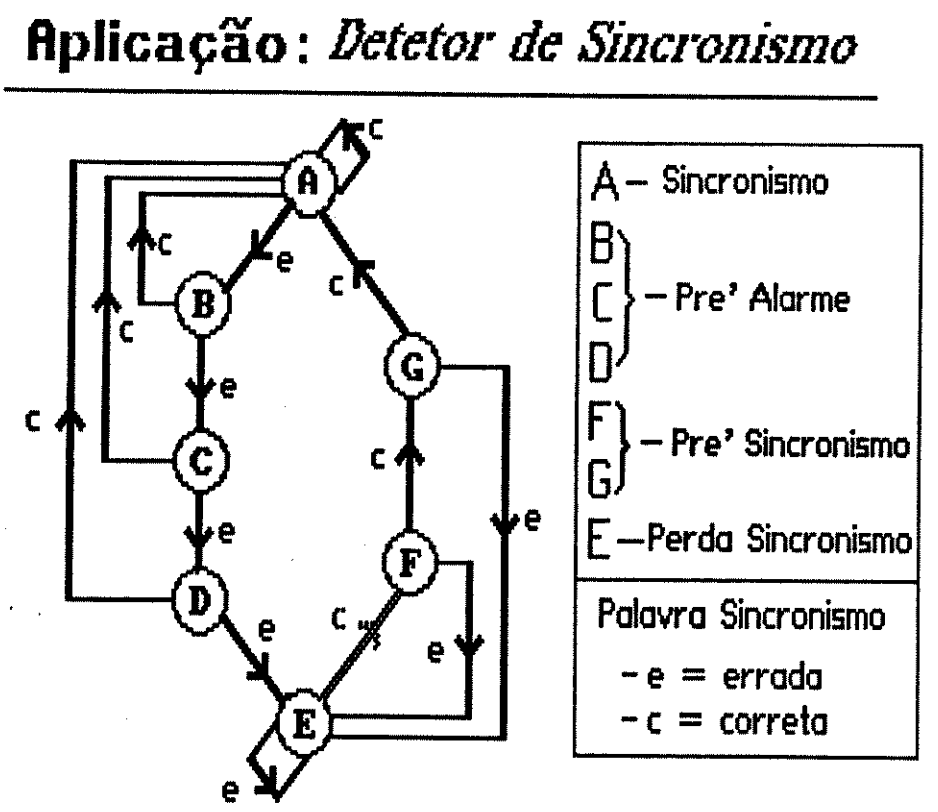


Figura IV.7.-Diagrama de próximo estado do CCS1 .

Alocação de Estados - INTUITIVA-3FF

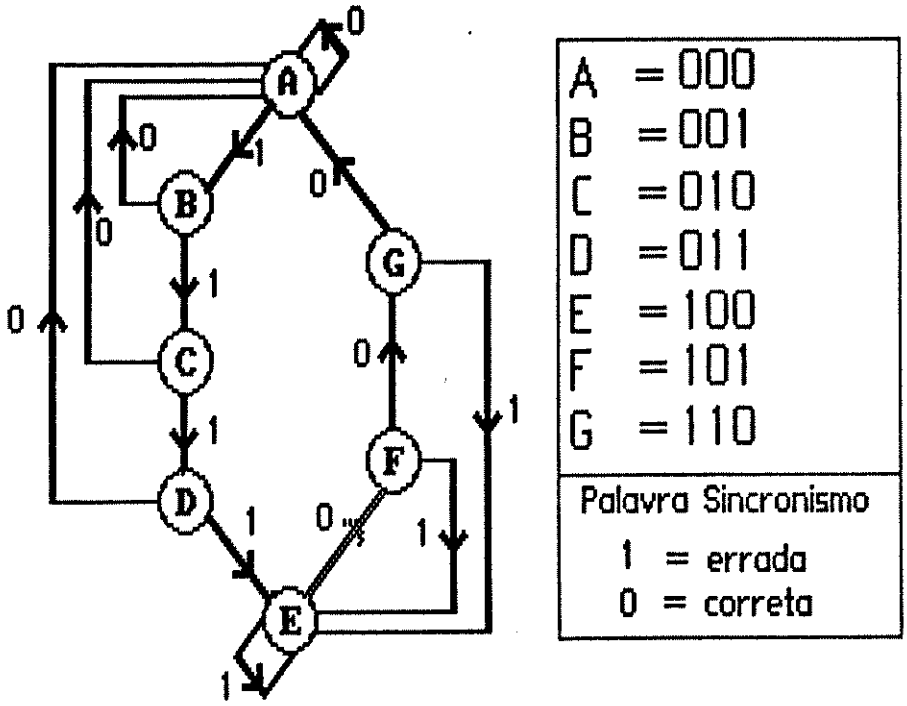


Figura IV.8.-Alocação de estados I para o CCS1.

Alocação de Estados - *Kubota 1978*

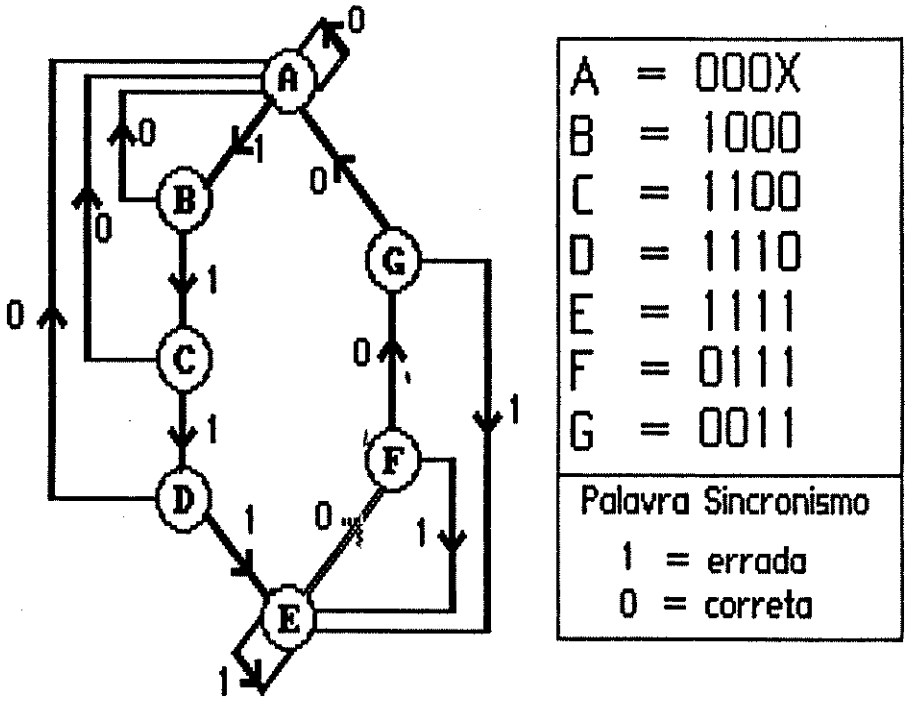


Figura IV.9.-Alocação de estados II para o CCS1.

CASO: CCS1 3FFD

DE	P/	ENTRADA	MINT	Q2	Q2+	D2	Q1	Q1+	D1	Q0	Q0+	D0
6	4	1	14	1	1	1	1	0	0	0	0	0
6	0	0	6	1	0	0	1	0	0	0	0	0
5	4	1	13	1	1	1	0	0	0	1	0	0
5	6	0	5	1	1	1	0	1	1	1	0	0
4	4	1	12	1	1	1	0	0	0	0	0	0
4	5	0	4	1	1	1	0	0	0	0	1	1
3	4	1	11	0	1	1	1	0	0	1	0	0
3	0	0	3	0	0	0	1	0	0	1	0	0
2	3	1	10	0	0	0	1	1	1	0	1	1
2	0	0	2	0	0	0	1	0	0	0	0	0
1	2	1	9	0	0	0	0	1	1	1	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0
0	1	1	8	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0

TABELA DE PROXIMO ESTADO - MINTERMOS

DE	P/	ENTRADA	DONT	Q2	Q2+	D2	Q1	Q1+	D1	Q0	Q0+	D0
7	X	0	7	1	X	X	1	X	X	1	X	X
7	X	1	15	1	X	X	1	X	X	1	X	X

TABELA DE PROXIMO ESTADO - DON'T CARE STATES

$D2 = Y.Q2 + Q2.\overline{Q1} + Y.Q1.Q0$

$D1 = \overline{Y}.Q2.Q0 + Y.\overline{Q2}.Q1.\overline{Q0} + Y.\overline{Q2}.\overline{Q1}.Q0$ ===> CUSTO = 33

$D0 = \overline{Y}.Q2.\overline{Q1}.\overline{Q0} + Y.\overline{Q2}.\overline{Q0}$

Figura IV.10.-Resultado obtido pelo programa para a alocação I.

CASO: CCS1 4 FFD (KUBOTA)

DE	P/	ENT.	MINT	Q3	Q3+	D3	Q2	Q2+	D2	Q1	Q1+	D1	Q0	Q0+	D0
1	8	1	17	0	1	1	0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
3	15	1	19	0	1	1	0	1	1	1	1	1	1	1	1
3	1	0	3	0	0	0	0	0	0	1	0	0	1	1	1
7	15	1	23	0	1	1	1	1	1	1	1	1	1	1	1
7	3	0	7	0	0	0	1	0	0	1	1	1	1	1	1
15	15	1	31	1	1	1	1	1	1	1	1	1	1	1	1
15	7	0	15	1	0	0	1	1	1	1	1	1	1	1	1
14	15	1	30	1	1	1	1	1	1	1	1	1	0	1	1
14	1	0	14	1	0	0	1	0	0	1	0	0	0	1	1
12	14	1	28	1	1	1	1	1	1	0	1	1	0	0	0
12	0	0	12	1	0	0	1	0	0	0	0	0	0	0	0
8	12	1	24	1	1	1	0	1	1	0	0	0	0	0	0
8	0	0	8	1	0	0	0	0	0	0	0	0	0	0	0
0	8	1	16	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABELA DE PROXIMO ESTADO - MINTERMOS

DE	P/	ENT.	DONT	Q3	Q3+	D3	Q2	Q2+	D2	Q1	Q1+	D1	Q0	Q0+	D0
2	X	0	2	0	X	X	0	X	X	1	X	X	0	X	X
4	X	0	4	0	X	X	1	X	X	0	X	X	0	X	X
5	X	0	5	0	X	X	1	X	X	0	X	X	1	X	X
6	X	0	6	0	X	X	1	X	X	1	X	X	0	X	X
9	X	0	9	1	X	X	0	X	X	0	X	X	1	X	X
10	X	0	10	1	X	X	0	X	X	1	X	X	0	X	X
11	X	0	11	1	X	X	0	X	X	1	X	X	1	X	X
13	X	0	13	1	X	X	1	X	X	0	X	X	1	X	X
2	X	1	18	0	X	X	0	X	X	1	X	X	0	X	X
4	X	1	20	0	X	X	1	X	X	0	X	X	0	X	X
5	X	1	21	0	X	X	1	X	X	0	X	X	1	X	X
6	X	1	22	0	X	X	1	X	X	1	X	X	0	X	X
9	X	1	25	1	X	X	0	X	X	0	X	X	1	X	X
10	X	1	26	1	X	X	0	X	X	1	X	X	0	X	X
11	X	1	27	1	X	X	0	X	X	1	X	X	1	X	X
13	X	1	29	1	X	X	1	X	X	0	X	X	1	X	X

TABELA DE PROXIMO ESTADO - DON'T CARE STATES

```
D3 = Q2
D2 = Q3.Q1 + Y.Q1 + Y.Q2
D1 = Y.Q2 + Y.Q0 + Q3.Q0
D0 = Y
```

CUSTO = 20

Figura IV.11.-Resultado obtido pelo programa para a alocação 11.

Circuito CCS1

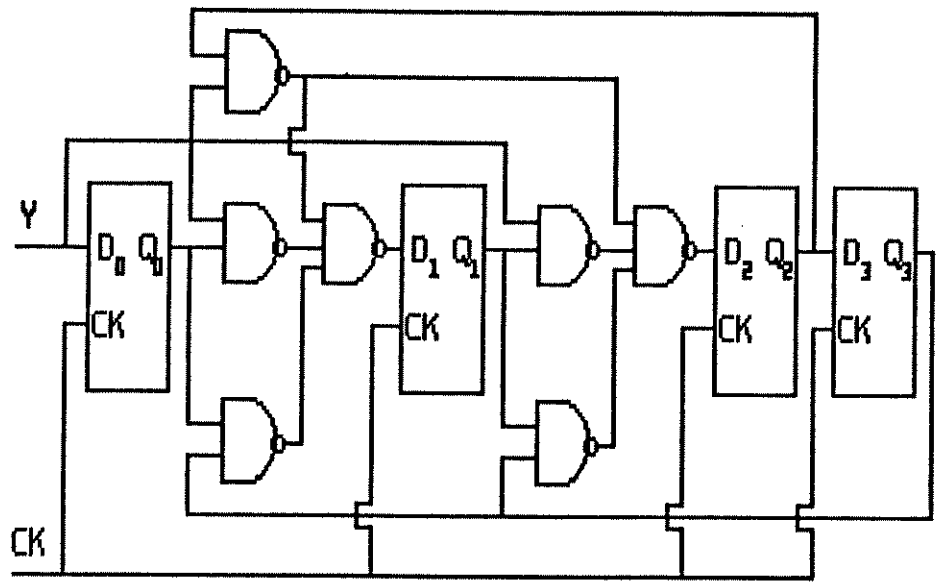


Figura IV.12.- Circuito obtido para o CCS1

IV.5.- CONCLUSÃO :

Apresentou-se neste capítulo a definição de máquinas sequenciais síncronas, através de uma formulação que permitiu uma definição algorítmica destas máquinas.

A partir desta definição foi implementado um programa de computador - TABELA - que tem vários recursos que viabilizam o projeto de circuitos sequenciais síncronos de maneira sistemática.

É importante ressaltar que o exemplo ilustrativo adotado, e realizado inicialmente por Kubota, exigiu dele dias de projeto, enquanto que através do programa TABELA, várias alternativas de projeto podem ser realizadas em alguns minutos.

O programa TABELA automatiza um processo conhecido de projeto, embora trabalhoso e cansativo, e facilita bastante a comparação de diferentes alocações de estados, possibilitando um estudo quase exaustivo de casos.

Com isto, encerra-se o ciclo de síntese dentro do projeto de circuitos digitais, e inicia-se a análise dos circuitos obtidos. A ferramenta cada vez mais utilizada neste processo de análise é a simulação, pois com ela pode-se verificar o funcionamento do circuito e a existência de falhas de projeto.

Neste contexto apresenta-se no capítulo V o simulador de circuitos digitais chamado LOGICO [6].

IV.6.- REFERENCIAS BIBLIOGRAFICAS :

1.- Kohavi, Zvi

"Switching and Finite Automata Theory"

McGraw Hill - 1970

2.- Hill, F. & Peterson, G.

"Introduction to Switching Theory & Logical Design"

John Wiley & Sons - 1974

3.- Borland International Inc.

"Turbo Pascal - Reference Manual"

Versão 3.01 - 1985

4.- Madureira, Marcos C. & Bonatti, Ivanil S.

"Minimização de Funções Booleanas"

Publicação Interna 015/86

DEE - FEC - UNICAMP - 1986

5.- Kubota, Paulo M. & Waldman, Hélio

"Demultiplexagem em Sistemas MCP de segunda ordem"

Tese de Mestrado - DEE - FEC - UNICAMP - 1978

6.- Madureira, Marcos C. , Barros, M.C. & Bonatti, Ivanil S.

"Simulação de Circuitos Digitais"

I Congresso da Sociedade Brasileira de Microeletrônica

CPqD - Telebrás - Campinas - SP - Julho de 1986

CAPÍTULO 5

SIMULAÇÃO LÓGICA

V.1.- RESUMO :

Neste capítulo é descrito um programa de computador que simula circuitos digitais ao nível de portas lógicas, considerando-se os tempos de atraso destas portas.

O simulador constitui-se em um processo onde são definidos estados, transições entre estados, entradas e saídas.

A idéia básica é descrever, através de sintaxe apropriada, o circuito e suas entradas (excitações) e se obter as respostas do circuito nos pontos de interesse, numérica ou graficamente, através de diagramas de tempo.

V.2.-INTRODUÇÃO :

No contexto de projeto de circuitos digitais, observa-se uma tendência cada vez maior no sentido de se projetar circuitos na forma de circuitos integrados de alta escala e de acordo com as especificações do usuário, o que leva a circuitos cada vez mais complexos .

A simulação de circuitos digitais representa, cada vez mais, uma diminuição no custo e no tempo de projeto, pois permite analisar o comportamento do circuito, verificando seu funcionamento e a possível existência de falhas, antes de sua implementação. Isto torna mais simples e baratas as modificações necessárias no projeto.

Referências bibliográficas em simulação de circuitos são escassas e superficiais, na opinião do autor, em virtude da exploração comercial das idéias a ele relacionadas.

O principal resultado deste capítulo é um programa que simula circuitos digitais ao nível de portas lógicas. A idéia utilizada foi a de se criar um programa com o qual o usuário descreve, através de sintaxe apropriada, o circuito e suas entradas (excitações) e obtém as respostas do circuito nos pontos de seu interesse.

Estas respostas podem ser obtidas de duas formas . Na forma numérica, onde é gerado um arquivo com a descrição de todas as transições do circuito, os instantes em que ocorrem, e os valores "0" e "1" para os respectivos níveis lógicos ; ou na forma gráfica, onde são visualizadas as formas de onda das entradas e saídas do circuito .

O usuário pode selecionar as ondas que deseja e delimitar os intervalos de tempo em que serão mostradas as ondas escolhidas, podendo ampliar ou reduzir janelas de tempo, e outras facilidades.

A idéia básica do programa é descrever os circuitos mais simples do tipo de portas lógicas isoladas, e utilizá-los como elementos básicos, evoluindo-se para circuitos mais complexos, como flip-flops, shift-registers, etc. .

V.3.- O SIMULADOR LOGICO :

V.3.1.- ESTRUTURA GERAL

O simulador LOGICO [7] constitui um processo onde são definidos estados, transições entre estados, entradas e saídas. As entradas do processo são as entradas externas do circuito, os estados são as variáveis de saída de cada uma de suas células básicas, e as saídas são as saídas externas do circuito. As transições entre os estados do processo ocorrem devido às transições nas entradas. A regra de transição entre os estados é resultante da estrutura do circuito, de seu estado anterior e dos valores das entradas anteriores à transição.

O LOGICO é um simulador estrutural e funcional. Estrutural porque a simulação é baseada em rotinas que simulam os gates básicos de circuitos tanto combinacionais como sequenciais com os quais se pode construir circuitos mais complexos, formando uma estrutura piramidal de uso. Estas rotinas, por sua vez, realizam todo o tratamento funcional dos elementos básicos, constituindo-se em "primitivas", ou seja, na elaboração de circuitos mais complexos não é necessário levar-se em conta suas propriedades funcionais, mas apenas suas estruturas.

Todas as portas lógicas apresentam um certo tempo de resposta, isto é, um tempo de propagação da informação da entrada para a saída. O simulador LOGICO permite que se defina o tempo de atraso de cada gate do circuito. São definidos dois atrasos para cada gate, um na transição de nível baixo para alto (TPLH) e um

de alto para baixo (TPHL). Estes valores podem ser definidos individualmente para cada gate, ou simultaneamente para todos os gates, ou ainda podem ser estabelecidos limites inferiores e superiores dentro dos quais o programa atribuirá valores aleatórios de atraso.

Para o tratamento algorítmico dos atrasos, foi criado o conceito de evento. Um evento é uma transição que ocorre numa entrada ou numa saída qualquer do circuito. Os instantes inicial e final da simulação constituem-se em eventos especiais.

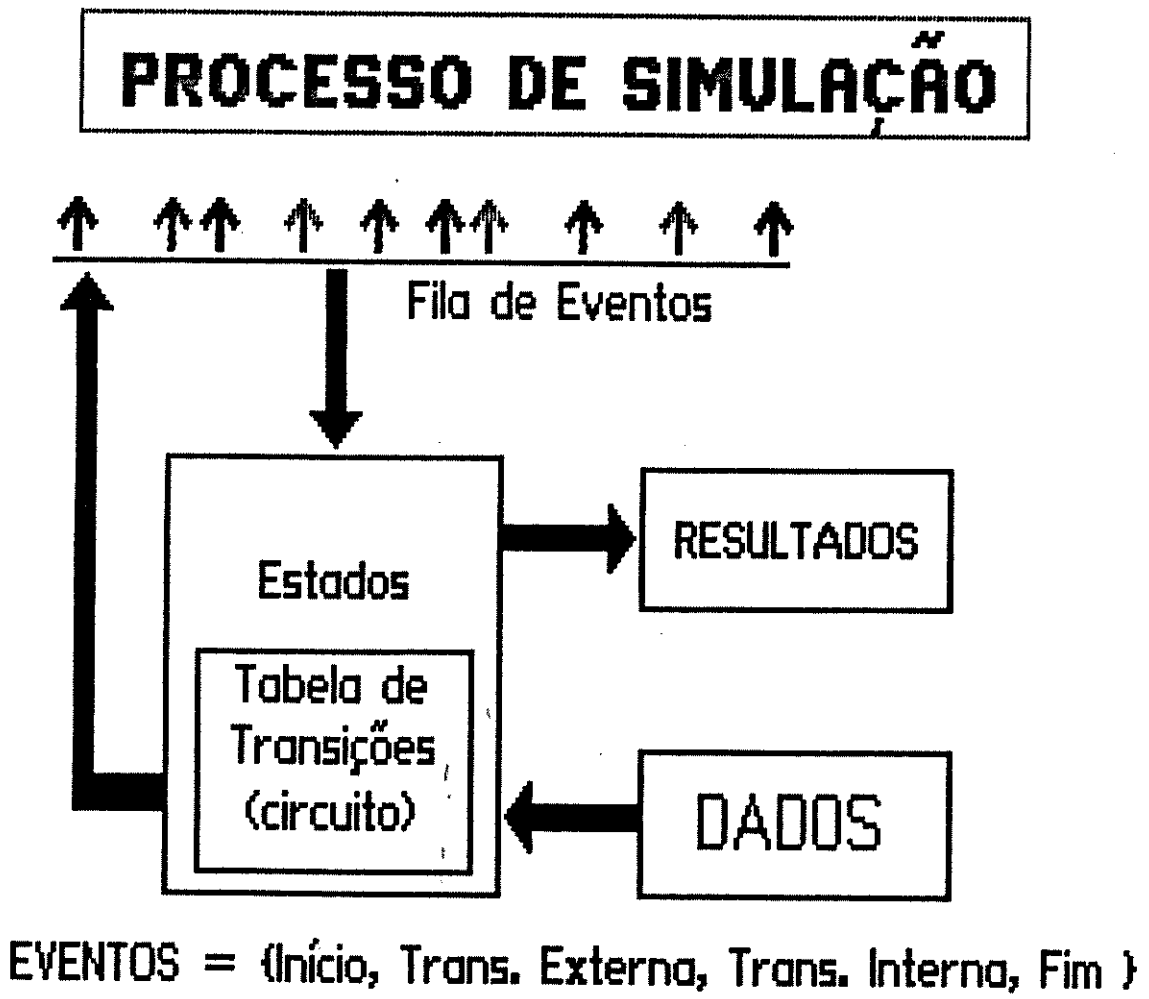


Figura V.1.- Processo de Simulação

A simulação resume-se na resolução e na geração de eventos que deverão estar ordenados numa fila de acordo com o instante em que ocorrem. O funcionamento do simulador é ilustrado na figura V.1 e processa-se da forma explicada a seguir.

Um vetor V de variáveis do tipo "byte" (0..255) armazena os valores das entradas e os estados de todos os gates do circuito em um dado instante. Cada gate é referenciado sempre pelo índice que o representa neste vetor. Convencionou-se que índices negativos do vetor representam entradas externas do circuito e índices positivos seus estados internos.

Quando um evento é uma transição na entrada, os valores desta são atualizados e o circuito é "executado", ou seja, a partir dos valores das entradas são obtidos os valores de saída do circuito através da chamada das rotinas correspondentes.

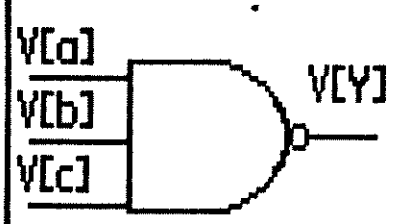
Para todos os eventos do tipo transição interna, que ocorrerem num mesmo instante, os valores das posições do vetor correspondentes a estes pontos do circuito são atualizados e então o circuito é executado.

As rotinas básicas definidas não atualizam diretamente as posições do vetor de saída mas geram eventos, em instantes determinados pelo tempo de atraso das portas, que serão inseridos na fila na posição correta e tratados no devido tempo. Assim que um evento é tratado, passa-se ao próximo da fila. Isto é mostrado na figura V.2 onde são descritos um gate NAND de 3 entradas e um gate OR de 2 entradas, como exemplos.

As rotinas que simulam os gates calculam o nível lógico da saída de acordo com a função lógica simulada, e dos valores lógicos de suas entradas.

ROTINAS BÁSICAS

```
PROCEDURE NAND3(a,b,c,Y);
BEGIN
 E:=V[a]*V[b]*V[c];
 IF E>=2 THEN E:=2 ELSE E:=1-E;
 (*COLOCA V[Y]:=E NA FILA *)
END;
```




```
PROCEDURE OR2(a,b,Y);
BEGIN
 E:= V[a] OR V[b];
 IF E=3 THEN E:=1;
 (*COLOCA V[Y]:=E NA FILA *)
END;
```

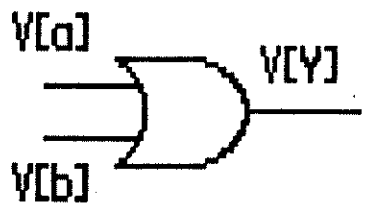


Figura V.2.- Rotinas Básicas

O algoritmo trabalha com valores aritméticos 0, 1 e 2, representando os níveis lógicos '0', '1' e 'X' respectivamente. O estado 'X' (indeterminado) representa o não conhecimento do nível lógico da porta. A lógica de 3 valores associada é mostrada na figura V.3 . Tal lógica poderia ser facilmente expandida para quatro (ou mais) valores se se desejasse, por exemplo, tratar de circuitos "tristate", incluindo-se um estado de alta impedância [9] .

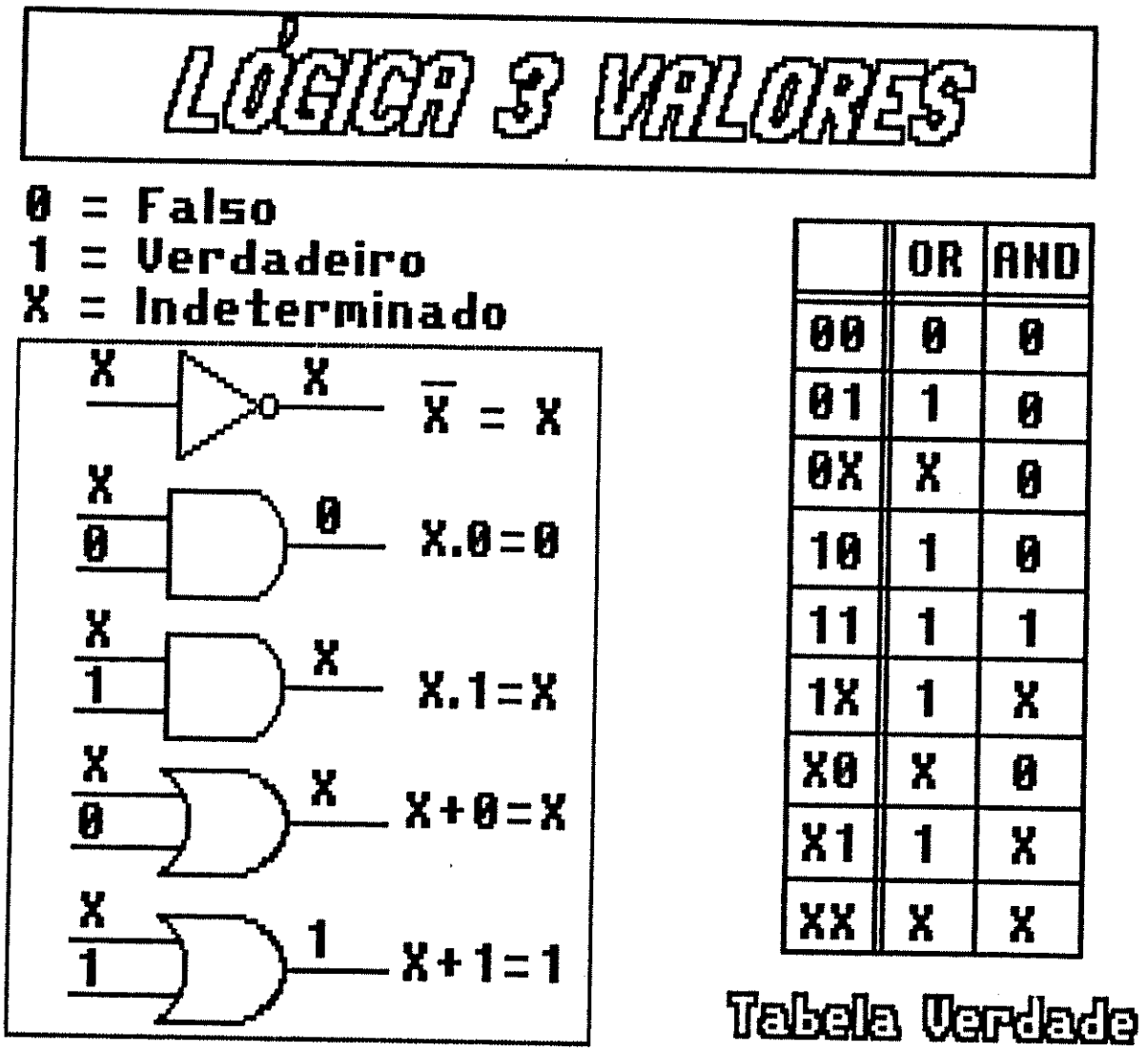


Figura V.3.- Lógica de 3 valores

V.3.2.- CIRCUITOS COMBINACIONAIS

A implementação dos gates lógicos define a convenção de chamada das rotinas pelo usuário. O exemplo seguinte ilustra um caso de um gate NAND de n entradas :

```
NANDn(V1,V2,..,Vn,Y);
```

onde,

n é o número de entradas da porta;

V1,V2,..,Vn são os índices das n entradas no vetor V;

Y é o índice da saída no vetor V .

V.3.3.- CIRCUITOS SEQUENCIAIS

Na análise de circuitos sequenciais é fundamental o estudo da célula básica de memória definida por :

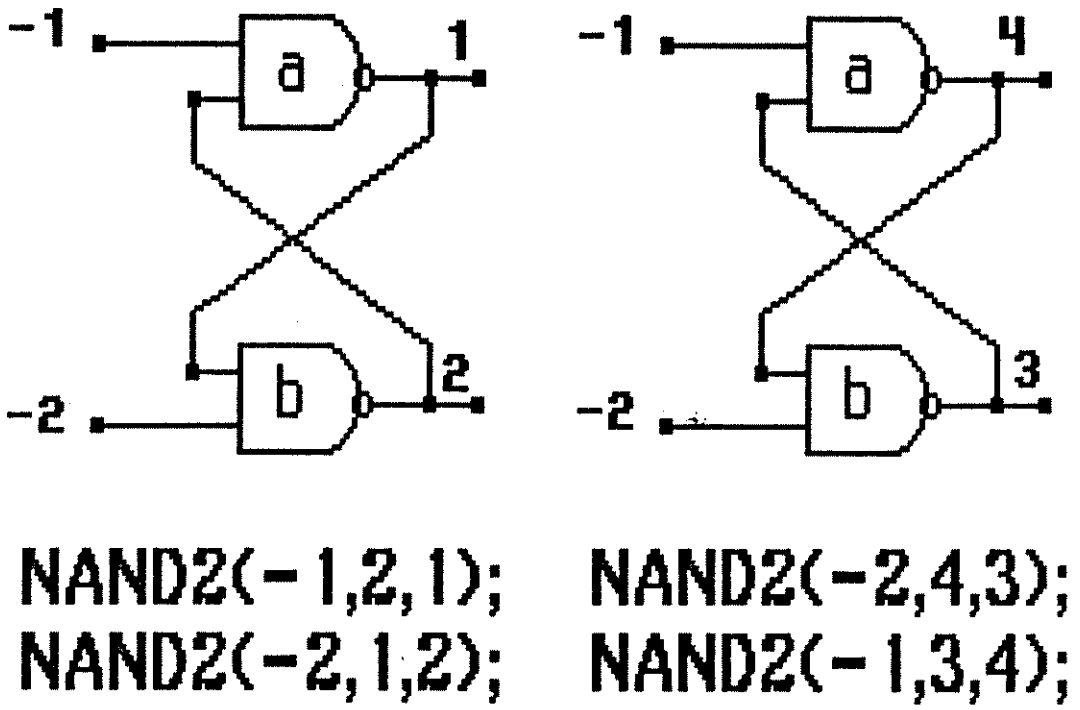


Figura V.4.- Células básicas de memória.

Como o algoritmo não processa as duas portas simultanemente é possível observar-se que a resposta, dada uma entrada, poderia não ficar estável. Assim decidiu-se pela inicialização de todas as saídas com o valor indeterminado. Numerosos testes permitiram validar esta heurística.

A figura V.4 ilustra a chamada das rotinas das células básicas de memória, de duas maneiras, invertendo-se a ordem de chamadas das subrotinas dos dois NANDs . Assim, o NAND (a) quando chamado primeiro tem a saída 1 e quando chamado em segundo lugar tem a saída 4. Do mesmo modo, o NAND (b) tem as saídas 3 e 2 respectivamente quando chamado em primeiro e segundo lugar. O resultado da simulação apresentado na figura V.5 comprova que as respostas são exatamente as mesmas nos dois casos simulados.

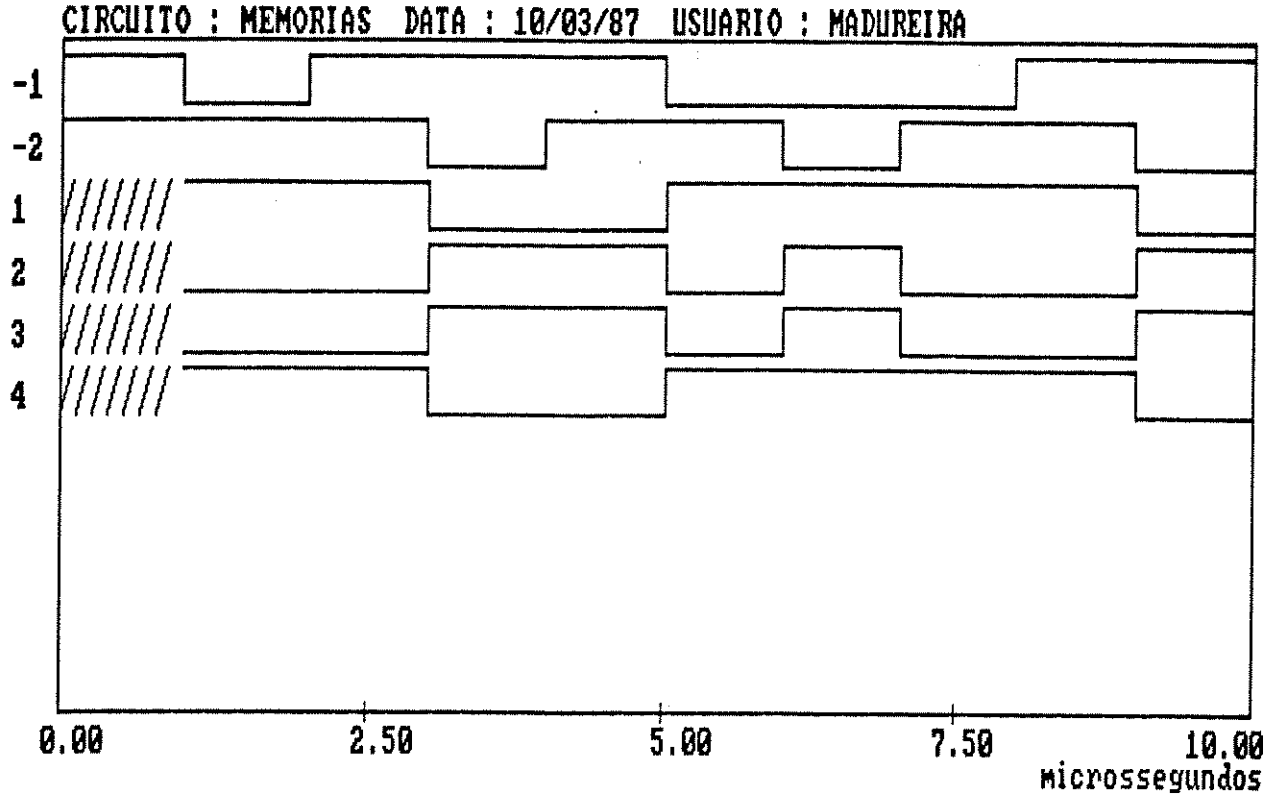


Figura V.5.- Resultado da simulação das células da figura V.4

Na maior parte das vezes, um circuito sequencial qualquer consiste num arranjo de várias células básicas de memória do tipo mostrado na figura V.4 .

Desta forma, é possível determinar-se corretamente os estados do circuito para todo circuito sequencial constituido por células do tipo mostrado na figura V.4 . E importante ressaltar que numerosos exemplos simulados validaram esta estratégia de simulação.

Para ilustrar a definição de Flip-Flops, criaram-se rotinas para simular os flip-flops : N74LS74 que é do tipo D e N74LS73 que é do tipo J-K cujos esquemas são dados nas figuras V.6[2] e V.7[3], respectivamente. As convenções de chamada de suas rotinas estão na figura V.8 .

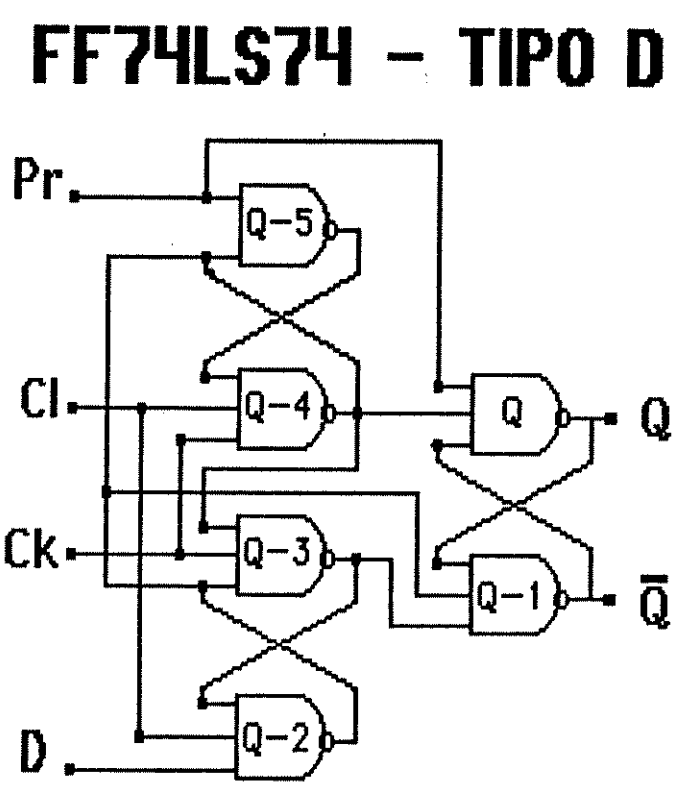


Figura V.6.- Flip-Flop N74LS74 tipo D [2].

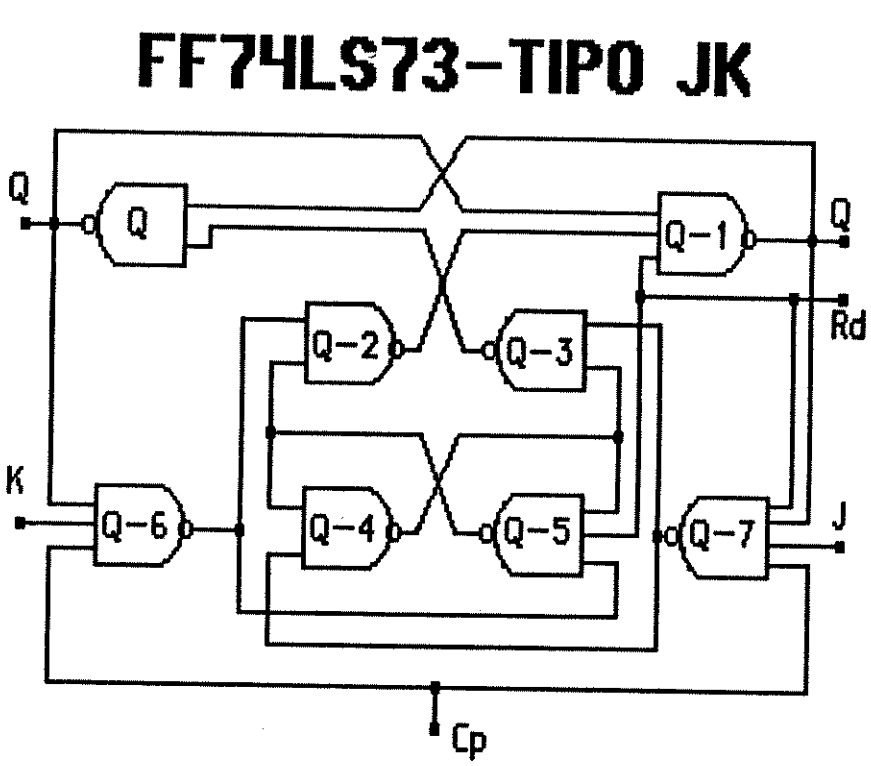


Figura V.7.- Flip-Flop N74LS73 tipo JK [3].

```
PROCEDURE FF7474S6(D,Ck,Cl,Pr,Q:INTEGER);
  BEGIN
   NAND3(Q-2,Ck,Q-4,Q-3);
   NAND3(Q-5,Cl,Ck,Q-4);
   NAND3(Q-3,Cl,Q,Q-1);
   NAND3(Q-4,Pr,Q-1,Q);
   NAND3(Q-2,Pr,Q-4,Q-5);
   NAND3(D,Cl,Q-3,Q-2);
  END;
PROCEDURE FF7473S8(J,K,CP,RD,Q:INTEGER);
  BEGIN
   NAND4(J,Q-1,CP,RD,Q-7);
   NAND3(K,CP,Q,Q-6);
   NAND3(RD,Q-6,Q-4,Q-5);
   NAND2(Q-5,Q-7,Q-4);
   NAND2(Q-4,Q-7,Q-3);
   NAND2(Q-5,Q-6,Q-2);
   NAND3(RD,Q,Q-2,Q-1);
   NAND2(Q-1,Q-3,Q);
  END;
```

Figura V.8.- Rotinas para os Flip-Flops D(V.6) e JK(V.7).

Observação: Deve-se colocar nas entradas sempre os índices que indicam suas posições no vetor. Para a saída deve-se atribuir o índice levando-se em conta que serão ocupadas as n posições anteriores do vetor devido às n portas internas ao FF.

V.3.4.- PARAMETROS DE IMPLEMENTAÇAO :

O simulador LOGICO tem sido utilizado por estagiários de iniciação científica do Departamento de Telemática e por alunos do curso de laboratório de circuitos lógicos da Faculdade de Engenharia Elétrica da UNICAMP. Também existem protótipos em teste no CPqD da Telebrás, no CTI, na Universidade Federal de São Carlos e na Universidade Federal do Maranhão.

O programa conta hoje com uma biblioteca da ordem de 20 a 30 circuitos básicos extraídos dos manuais dos fabricantes, com os quais se pode montar um grande número de circuitos dos mais diversos graus de complexidade. Esta biblioteca vem sendo constantemente incrementada de acordo com a demanda.

Foram testados circuitos com aproximadamente 200 portas lógicas, sem que fosse detectado qualquer problema. O programa está dimensionado para até 999 portas lógicas. O tempo de simulação depende de diversos fatores, como o intervalo de tempo de observação, o número de transições internas provocadas por cada mudança externa, o número de gates, a complexidade do circuito, etc. . Nas simulações realizadas, o tempo de simulação está na maior parte das vezes abaixo de um minuto, e não se observou nenhum caso que ultrapassasse dois minutos de execução.

Quanto à capacidade de memória, foram utilizadas técnicas de alocação estática e dinâmica, assim torna-se muito difícil precisar um limite. Pode-se avaliar que a alocação estática consome 5 bytes por gate, 1 para armazenar o estado no vetor V e 2 para cada valor de atraso (TPLH e TPHL). A alocação dinâmica ocupa transitoriamente mais 12 bytes de memória por transição.

V.4.- RESULTADOS :

Foram simulados alguns circuitos combinacionais associando a cada porta lógica um tempo de atraso de 10 ns tanto para TPLH como para TPHL.

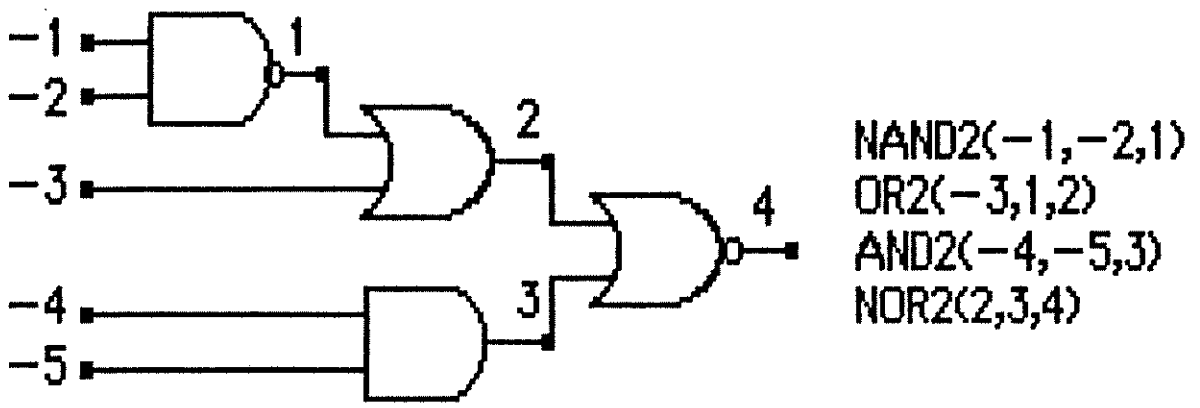


Figura V.9.- Exemplo de Circuito Combinacional.

A figura V.10 mostra o resultado da simulação do circuito da figura V.9 onde as entradas estão representadas por índices negativos e as saídas por positivos. Note que os instantes estão em microssegundos.

TRANSICAO	INSTANTE	-5	-4	-3	-2	-1	*	1	2	3	4
ENTRADA	0.000	0	0	0	0	0	*	X	X	X	X
INTERNA	0.010	0	0	0	0	0	*	1	X	X	X
INTERNA	0.010	0	0	0	0	0	*	1	X	0	X
INTERNA	0.020	0	0	0	0	0	*	1	1	0	X
INTERNA	0.020	0	0	0	0	0	*	1	1	0	X
INTERNA	0.020	0	0	0	0	0	*	1	1	0	X
INTERNA	0.030	0	0	0	0	0	*	1	1	0	0
INTERNA	0.030	0	0	0	0	0	*	1	1	0	0
INTERNA	0.030	0	0	0	0	0	*	1	1	0	0
ENTRADA	1.000	1	0	0	0	0	*	1	1	0	0
ENTRADA	2.000	0	1	0	0	0	*	1	1	0	0
ENTRADA	3.000	0	0	1	0	0	*	1	1	0	0
ENTRADA	4.000	0	0	0	1	0	*	1	1	0	0
ENTRADA	5.000	0	0	0	0	1	*	1	1	0	0

ENTRADA	6.000	1	1	1	1	1	*	1	1	0	0
INTERNA	6.010	1	1	1	1	1	*	0	1	0	0
INTERNA	6.010	1	1	1	1	1	*	0	1	1	0
INTERNA	6.020	1	1	1	1	1	*	0	1	1	0
ENTRADA	7.000	1	1	1	1	0	*	0	1	1	0
INTERNA	7.010	1	1	1	1	0	*	1	1	1	0
ENTRADA	8.000	1	1	1	0	1	*	1	1	1	0
ENTRADA	9.000	1	1	0	1	1	*	1	1	1	0
INTERNA	9.010	1	1	0	1	1	*	0	1	1	0
INTERNA	9.020	1	1	0	1	1	*	0	0	1	0
ENTRADA	10.000	1	0	1	1	1	*	0	0	1	0
INTERNA	10.010	1	0	1	1	1	*	0	1	1	0
INTERNA	10.010	1	0	1	1	1	*	0	1	0	0
INTERNA	10.020	1	0	1	1	1	*	0	1	0	0
ENTRADA	11.000	0	1	1	1	1	*	0	1	0	0
FINAL	12.000	0	1	1	1	1	*	0	1	0	0

Figura V.10.- Tabela Verdade para um circuito combinacional simulado com tempo de atraso de 10 ns.

Para exemplificar a simulação de circuitos sequenciais foi escolhido o FF N74LS74 (Fig. V.6). Como consequência do fato do simulador ser estrutural, não se especificam tempos de atraso para o flip-flop, mas estes são determinados pela propagação dos tempos de atraso de cada porta que o constitui.

As figuras V.11 a V.14 mostram a resposta gráfica do simulador aos testes para TPLH de SET, TPHL de RESET e TPLH e TPHL de CLOCK para o FF N74LS74 descrito da forma dada na figura V.6 . Os resultados estão conformes às especificações dos fabricantes [2]. Observe o efeito "zoom" nestas figuras, que permite uma observação detalhada de qualquer intervalo de tempo desejado.

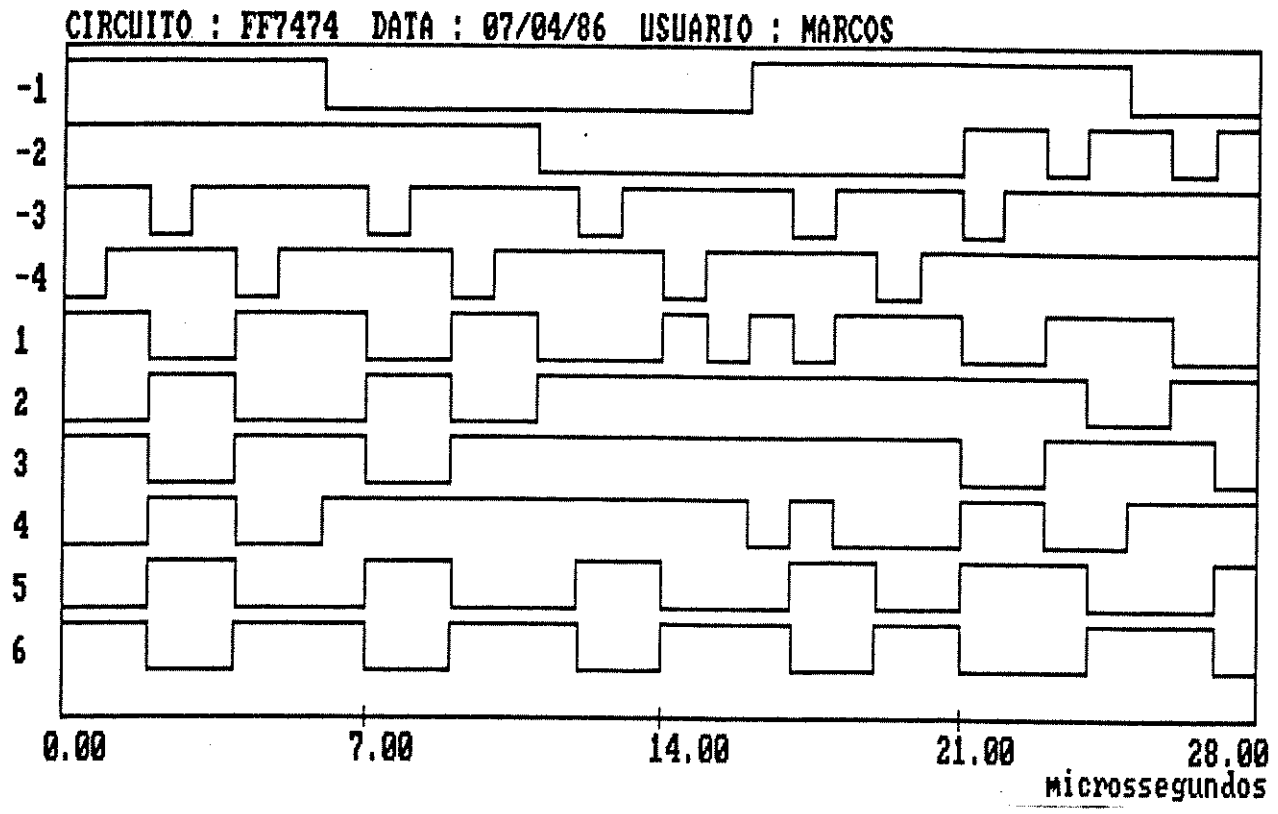


Figura V.11.- Diagrama de tempo para o FF N74LS74 conforme V.6

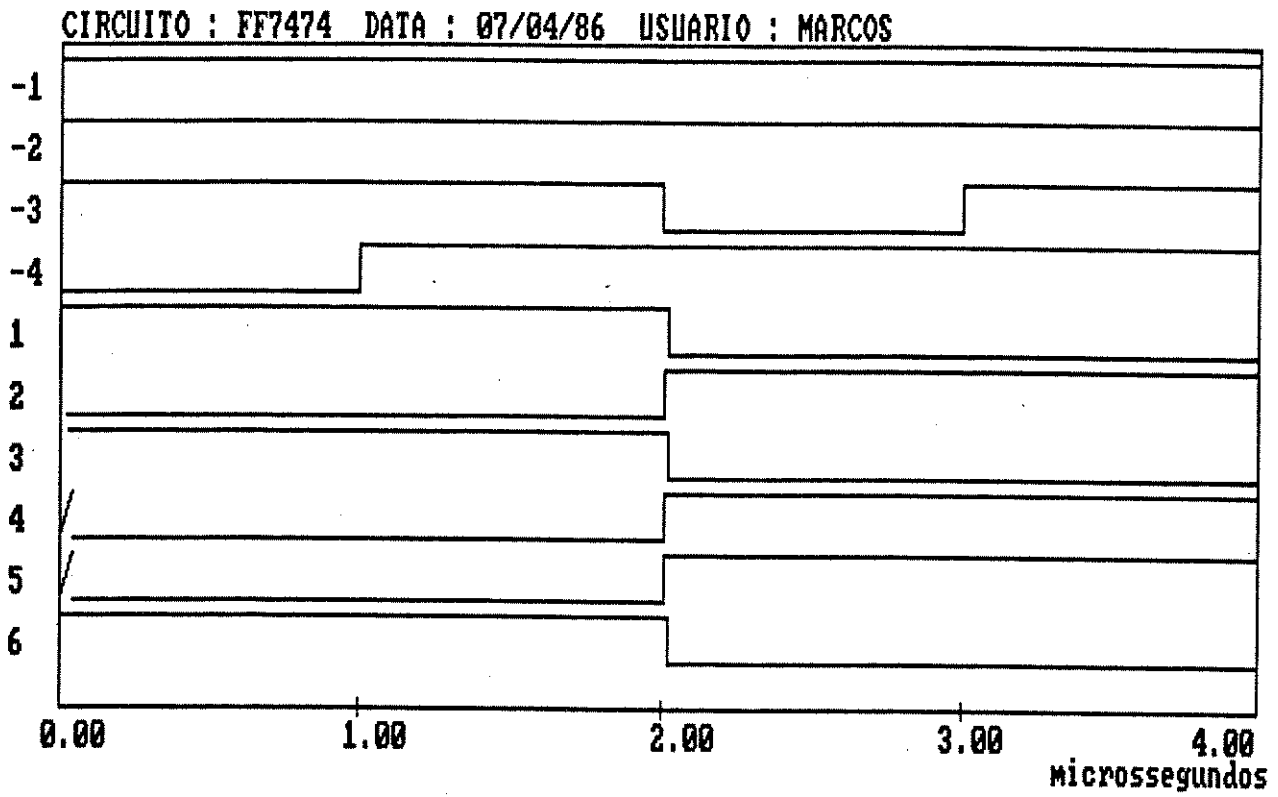


Figura V.12.- Demonstração do efeito "zoom" do simulador, com a modificação do intervalo de tempo para ampliação das transições e melhor observação de detalhes.

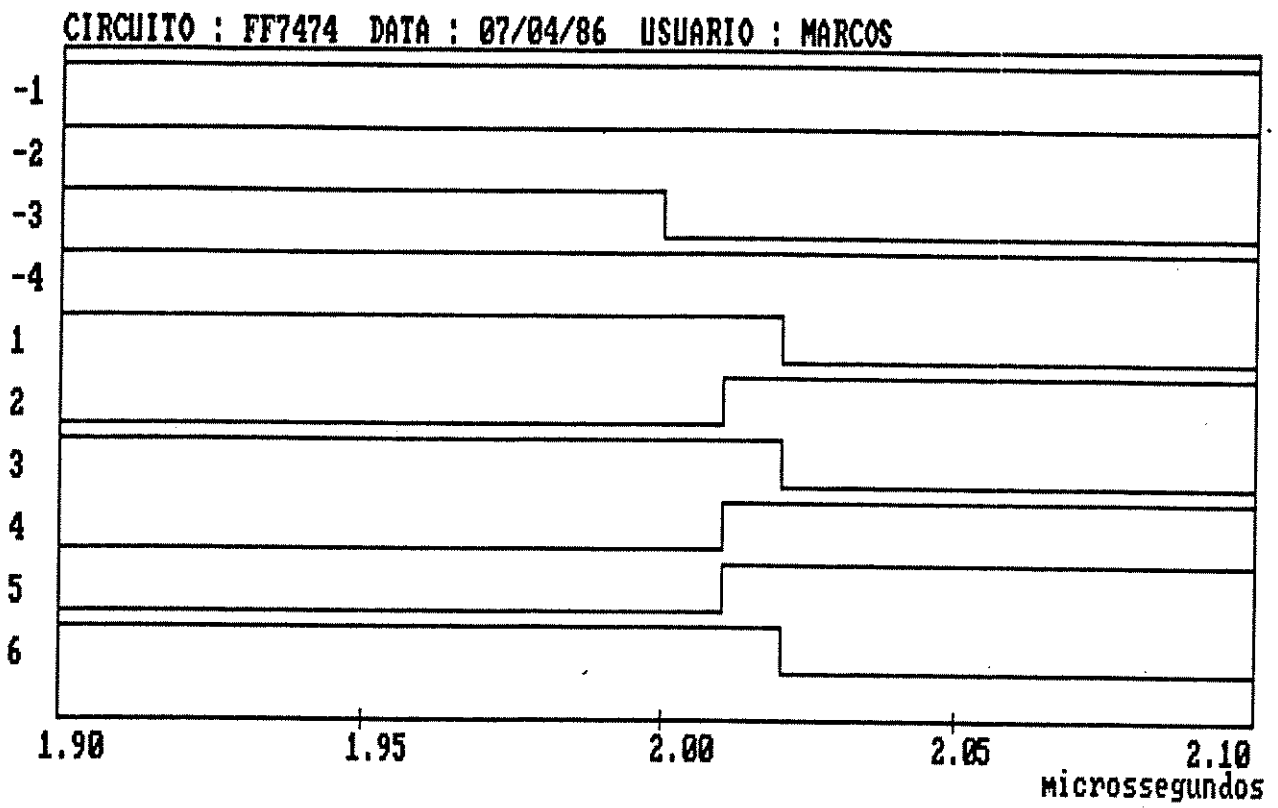


Figura V.13.- Novo "zoom", no entorno de 2.00 microssegundos para uma observação dos tempos de atraso nas transições.

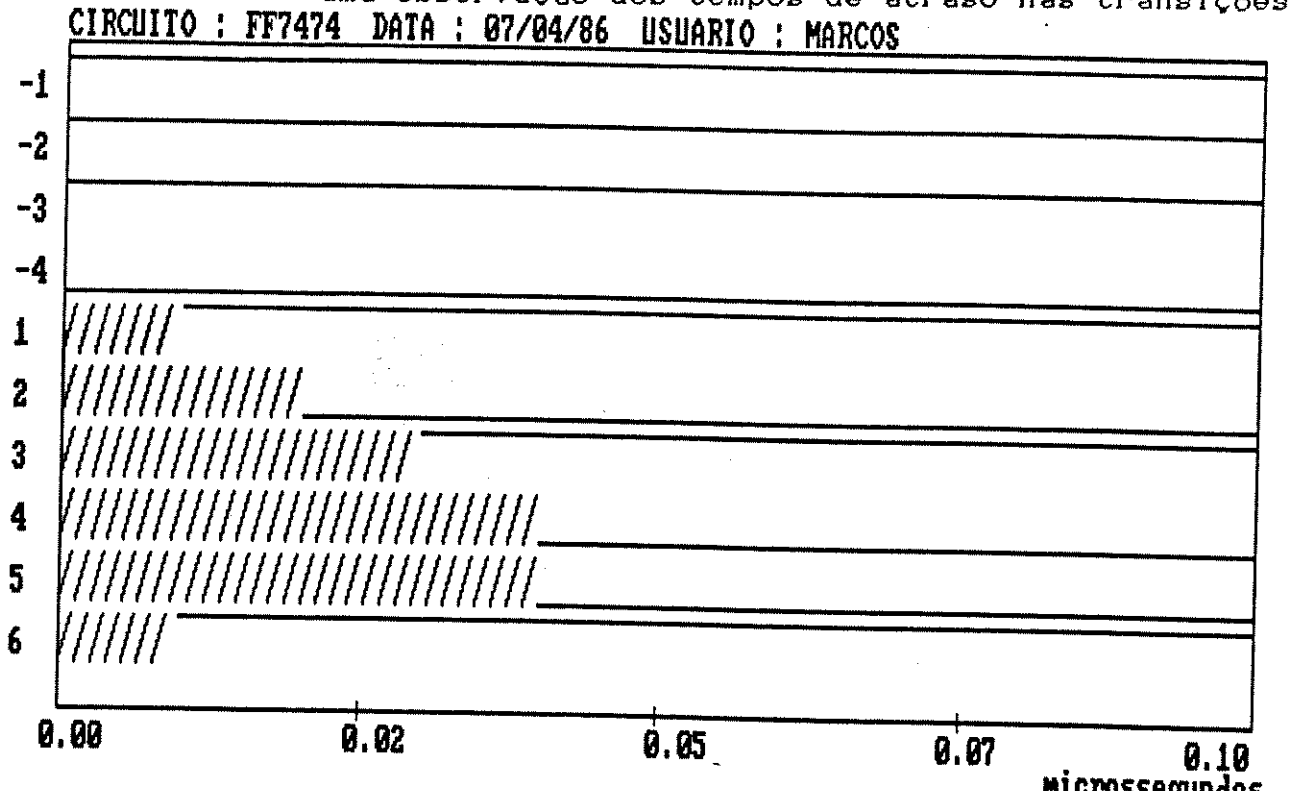


Figura V.14.- Ampliação do diagrama de tempo próximo ao instante zero para mostrar a representação gráfica do estado indeterminado "X" com o qual se inicializa o simulador.

O circuito de controle de sincronismo CCS1, estudado no capítulo IV (figura IV.12), foi simulado no simulador LOGICO para ilustrar um exemplo um pouco mais complexo de circuito e a utilização e chamada das rotinas da biblioteca.

Através da simulação verificou-se que seu funcionamento é exatamente aquele descrito nas tabelas e nos diagramas de estados correspondentes (figura IV.9), não sendo detetadas falhas de projeto.

As figuras V.15 e V.16 mostram a descrição do circuito para o simulador. A figura V.17 é o diagrama de tempo obtido da simulação de um ciclo completo no diagrama de estados da figura IV.9 com perda e recuperação do sincronismo. A figura V.18 é uma ampliação da escala de tempo do diagrama anterior em torno de 16 microsegundos, para que se possa observar os atrasos nas transições dos "gates".

```
NAND2(13,31,6)
NAND2(13,Y,5)
NAND2(Y,25,4)
NAND3(6,5,4,3)
NAND2(31,19,2)
NAND2(19,Y,1)
NAND3(2,1,4,7)
FF7474S6(Y,CK,1001,1001,13)
FF7474S6(3,CK,1001,1001,19)
FF7474S6(7,CK,1001,1001,25)
FF7474S6(25,CK,1001,1001,31)
```

Figura V.15.- Chamada das rotinas da biblioteca que formam o CCS1

Circuito CCS1

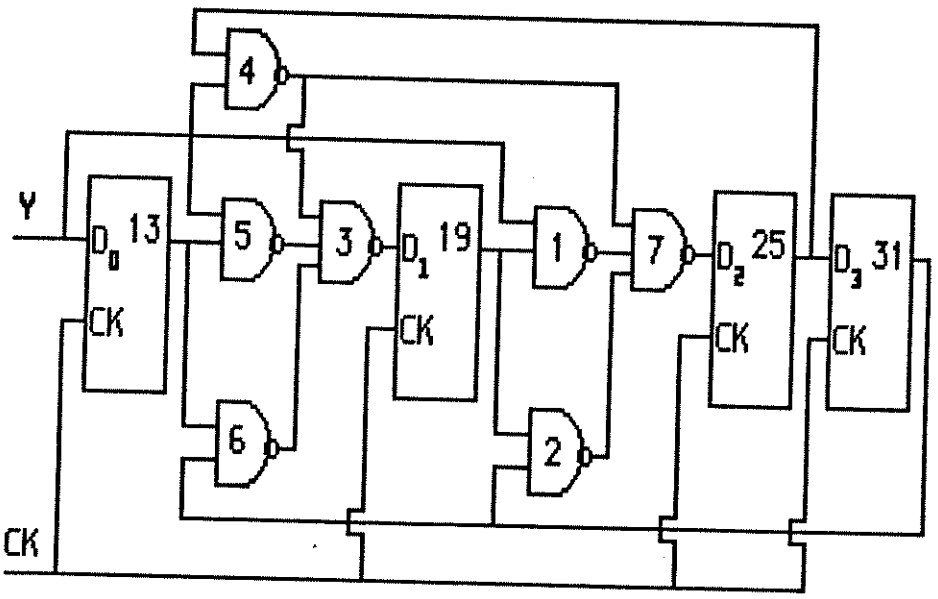


Figura V.16.- Descrição do CCS1 para o simulador LOGICO.

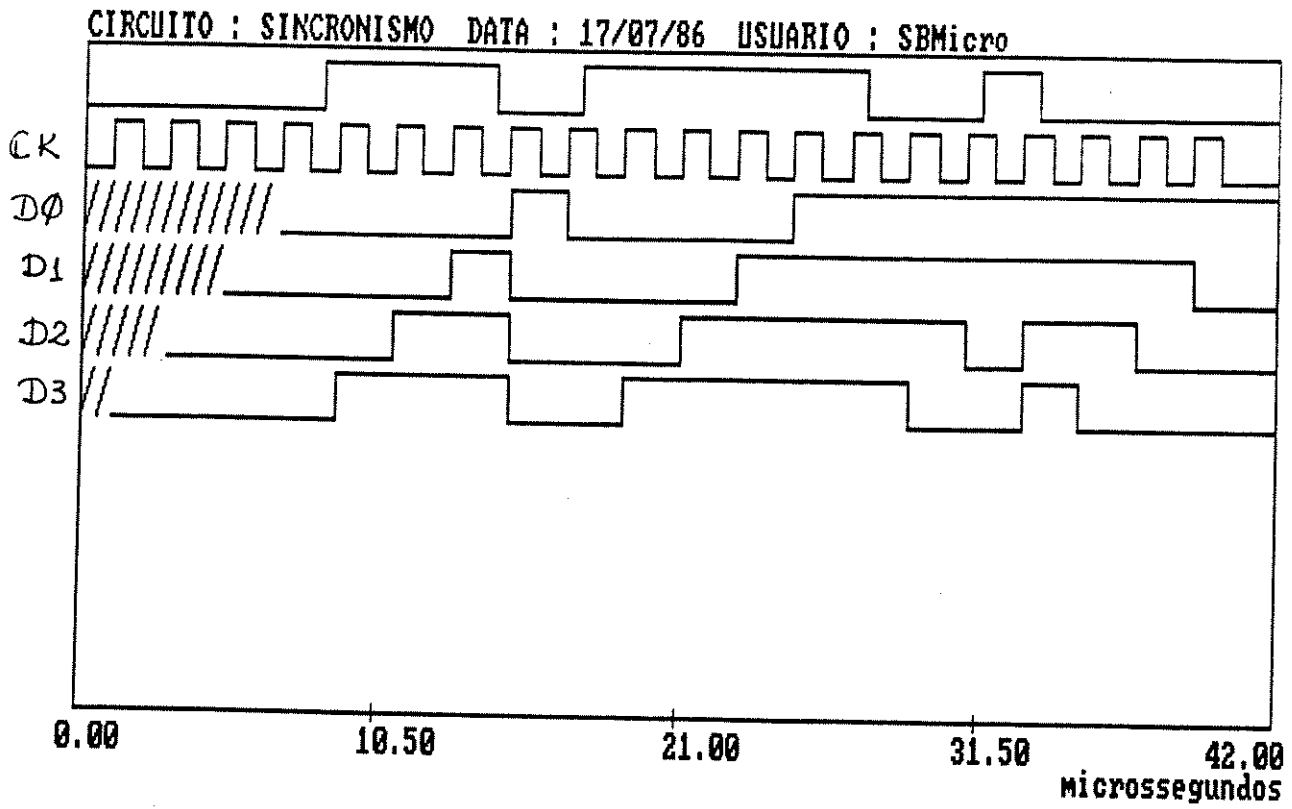


Figura V.17.- Diagrama de tempo obtido para o CCS1

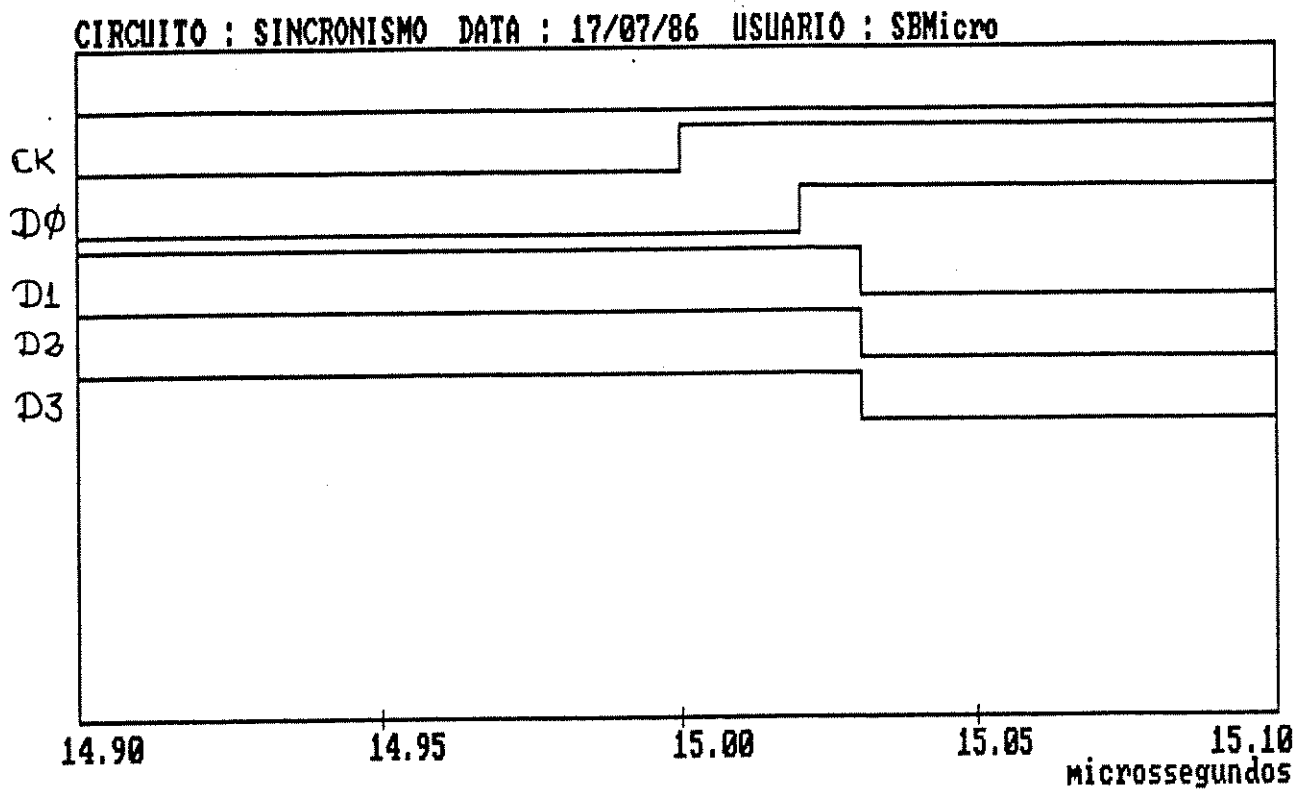


Figura V.18.- Diagrama de tempo ampliado do CCS1

V.5.-CONCLUSÃO :

Os resultados deste programa de simulação desenvolvido em um microcomputador de 16 bits e 256Kbytes de memória são bastante satisfatórios, comparáveis aos resultados de um programa como o HILO-2 [6].

Graças às facilidades gráficas de excelente desempenho, pode-se visualizar diagramas de tempo dos circuitos simulados, escolhendo-se as ondas a serem estudadas e os intervalos de tempo desejados. Isto permite comparações entre diversas ondas e um efeito do tipo "zoom", aumentando-se ou diminuindo-se os tamanhos das janelas de tempo mostradas.

Outra facilidade implementada é a possibilidade de escolha de diferentes valores de tempos de atraso para os elementos do circuito, existindo inclusive a opção da geração de valores aleatórios entre limites máximos e mínimos fornecidos.

Apesar de tratar os atrasos dos elementos, o simulador considera instantâneas as transições entre os níveis lógicos. Isto pode ocasionar o aparecimento de pulsos espúrios em determinadas circunstâncias. Uma maneira de realizar uma "filtragem" destes pulsos é o cálculo da duração destas transições. Esta deve ser mais uma facilidade incorporada numa próxima versão do programa.

Circuitos "tristate", com estados de alta impedância, podem ser tratados também com a inclusão de um quarto valor lógico no simulador (além de "0", "1" e "X"), sem que isto venha a complicar consideravelmente o algoritmo ou afetar o desempenho do programa.

Em virtude da escassez bibliográfica nacional neste tópico, o autor é levado a colocar este produto como relevante no mercado brasileiro, representando um importante passo na valorização da pesquisa tecnológica nacional.

Neste sentido, protótipos vêm sendo utilizados nos cursos de circuitos digitais da Faculdade de Engenharia Elétrica da UNICAMP, no Departamento de Computação da Universidade Federal de São Carlos e na Universidade Federal do Maranhão. Também dispõem de protótipos centros de pesquisa como o CTI e o CPqD/Telebrás.

V.6.-REFERENCIAS BIBLIOGRAFICAS :

1.- Hill, F.J. & Peterson, G.R.

"Introduction to Switching Theory & Logical Design"

2ed. Wiley International Edition, 1974 - 596 p.

2.- Manual Signectics Digital-Linear-Mos 1974

3.- Manual Fairchild TTL Data Book 1973

4.- Wirth, Niklaus

Pascal- User Manual and Report

5.- Taub, Herbert

Digital Integrated Electronics

McGraw Hill 1977

6.- Manual de utilização do "HILO 2"

Cyrrus Computers

7.- Madureira, M. C. et. al.

"Simulação de Circuitos Digitais"

I Congresso da Sociedade Brasileira de Microeletrônica

CPqD - Campinas - SP - Julho de 1986

8.- Costa, E. M.

"Projeto de Circuitos Integrados"

I Escola Brasileiro-Argentina de Informática

Editorial Kapelusz S/A - 1987.

9.- Cameron, K.B. & Shovic, J.C.

"Calculating minimum logic states requirements for multi-strength multi-value MOS logic simulators"

I Congresso da Sociedade Brasileira de Microeletrônica

CPqD - Campinas - SP - Julho de 1986 - Separata

CAPÍTULO 6

CONCLUSÃO

"Conclusão"

Este trabalho insere-se no contexto de pesquisa e implementação de ferramental eficiente no auxílio ao projeto de circuitos digitais integrados. Sua principal motivação é o esforço de centros de pesquisa como o CPqD/Telebrás e o CTI, e de algumas outras empresas na direção de nacionalizar o projeto e a fabricação destes circuitos.

No capítulo I foram citados e comentados alguns dos artigos publicados na área de pesquisa de circuitos integrados entre 1975 e 1986. Em meados da década de 70 houve um retorno à ênfase na pesquisa na Análise e Síntese de Circuitos Digitais. Isto ocorreu, sobretudo devido à mudança no perfil do usuário que, em parte, passou de projetista de circuitos usando integrados a projetista de integrados propriamente.

No capítulo II foram definidas as funções booleanas afim de descrever as propriedades dos circuitos combinacionais.

A importância da minimização de funções booleanas destacada neste capítulo levou o autor a dedicar o capítulo III ao estudo e implementação de dois algoritmos distintos que realizam a minimização das funções booleanas, o algoritmo de "Quine-McCluskey" e o algoritmo de "Caruso".

O algoritmo de "Quine-McCluskey" foi escolhido para servir de base de comparação e avaliação de outros métodos, e o algoritmo de "Caruso" despertou especial interesse no autor ao procurar sanar o principal problema apresentado pelo primeiro, quanto ao consumo de tempo na geração dos implicantes primos.

Ambos os algoritmos foram implementados em programas de computador afim de serem testados e comparados. O método de Caruso mostrou ser mais rápido e consumir menos memória que o de

Quine-McCluskey. A sua única desvantagem é não garantir a obtenção de uma fórmula de custo mínimo, mas sim de cobertura mínima. Contudo, conforme ilustrado nos exemplos, todos os casos testados geraram fórmulas de custo mínimo.

Já o algoritmo de Quine-McCluskey garante a obtenção da fórmula de custo mínimo das funções, o que pode ter grande interesse acadêmico. Porém, no contexto de um pacote de software de apoio ao projeto de circuitos digitais, o autor recomenda a inclusão do algoritmo de Caruso.

Neste contexto, a minimização de funções booleanas representa uma etapa importante dentro de uma sistemática conhecida de projeto.

A automatização desta sistemática conhecida, embora trabalhosa e cansativa, facilita bastante a comparação de diferentes alocações de estados, possibilitando um estudo quase exaustivo de casos.

Assim, apresentou-se no capítulo IV a definição de máquinas sequenciais síncronas, através de uma formulação que permitiu uma definição algorítmica destas máquinas.

A partir desta definição foi implementado um programa de computador - TABELA - que viabiliza o projeto de circuitos sequenciais síncronos de maneira sistemática.

A análise dos circuitos projetados tem importância relevante neste contexto, uma vez que podem ocorrer falhas em qualquer das etapas do projeto. Cada vez mais, esta análise tem sido feita através de simulação, pois ela representa uma boa diminuição no custo e no tempo de projeto, permitindo analisar o comportamento

do circuito e a possível existência de falhas, antes de sua implementação. Isto torna mais simples e baratas as modificações eventualmente necessárias nos projetos.

Baseado na crescente importância da simulação, o autor desenvolveu e apresentou no capítulo V um programa que realiza a simulação lógica de circuitos digitais. Os resultados deste programa desenvolvido em um microcomputador de 16 bits e 256Kbytes de memória são bastante satisfatórios.

Graças às facilidades gráficas pode-se visualizar diagramas de tempo dos circuitos simulados, escolhendo-se as ondas a serem estudadas e os intervalos de tempo desejados. Isto permite comparações entre diversas ondas e um efeito do tipo "zoom", aumentando-se ou diminuindo-se os tamanhos das janelas de tempo mostradas.

Outra facilidade implementada é a possibilidade de escolha de diferentes valores de tempos de atraso para os elementos do circuito, existindo inclusive a opção da geração de valores aleatórios entre limites máximos e mínimos fornecidos.

Circuitos "tristate", com estados de alta impedância, podem ser tratados com a inclusão de um quarto valor lógico no simulador (além de "0", "1" e "X"), sem que isto venha a complicar consideravelmente o algoritmo ou afetar o desempenho do programa.

Em virtude da escassez bibliográfica nacional neste tópico, o autor é levado a colocar este produto como relevante no mercado brasileiro, representando um importante passo na valorização da pesquisa tecnológica nacional.

Neste sentido, protótipos vêm sendo utilizados nos cursos de circuitos digitais da Faculdade de Engenharia Elétrica da UNICAMP, no Departamento de Computação da Universidade Federal de São Carlos e na Universidade Federal do Maranhão. Também dispõem de protótipos centros de pesquisa como o CTI e o CPqD/Telebrás.