Aplicação da Transformada Numérica na resolução de problemas da álgebra booleana[1,2]

Leo Weber

Resumo*:* O propósito deste artigo é apresentar a aplicação do método numérico denominado Transformada Numérica, que pode ser utilizado na resolução de problemas lógicos ligados à engenharia, à pesquisa operacional e à lógica clássica. O texto limitar-se-á a demonstrar o emprego deste método na resolução de problemas na álgebra booleana através de um programa implementado no ambiente Matlab, onde poderão ser verificadas algumas operações e aplicações da Transformada Numérica.

Palavras-chave*:* Transformada Numérica, álgebra booleana, algoritmo, métodos numéricos.

Abstract*:* The purpose of this article is to present an application of the numerical method called Numerical Transform, that can be used in the resolution of logical problems related to the engineering, operational research and the classical logic. The text will be restraint to demonstrate the use of this method in the resolution of problems in the boolean logic by a software in Matlab, where some operations and applications of the Numerical Transform could be verified.

Key words*:* Numerical Transform, boolean algebric, algorithm, numerical methods.

Introdução

A Transformada Numérica, também denominada TN, é uma ferramenta matemática usada para descrever funções e expressões da álgebra booleana e do cálculo proposicional, onde cada expressão é definida por seqüências numéricas determinadas através de regras estabelecidas, operando, portanto, no campo numérico. Entre as diferentes aplicações nas quais a transformada numérica pode ser empregada, existe a utilização em circuitos lógicos obtidos resolvendo-se as equações booleanas que descrevem o problema proposto. Este processo está descrito detalhadamente em [DELP] e [WEB1].

Da mesma forma, a TN é uma potente ferramenta para a programação de arquiteturas computacionais assíncronas baseadas em uma memória com realimentação de estados e circuitos adicionais *[Figura 1]*. Nestes circuitos, cujo funcionamento e relação com a TN detalhamos no próximo item, a implementação da resolução de um problema lógico dá-se através de um programa armazenado em uma memória não volátil.

Os sistemas assíncronos, por limitações tecnológicas na época de sua concepção, não tiveram o desenvolvimento dos sistemas computacionais que adotaram a arquitetura von Neumann. No entanto, recentes aplicações mostram soluções efetivas, baratas e de fácil implementação, o que justifica o desenvolvimento de ferramentas computacionais que ajudem no projeto de sistemas assíncronos baseados em memórias [WEB3].

Como funciona a arquitetura computacional assíncrona:

A idéia de projetar computadores digitais com lógica assíncrona esteve presente desde os primeiros sistemas. O Modelo Geral de um Sistema Digital, discutido por Phister, relaciona saídas e entradas por um sistema simultâneo de funções digitais [PHIS]. Sua concepção apresenta um modelo que pode ser representado adequadamente por uma caixa preta, incluindo sistemas digitais ou circuitos de chaveamento seqüenciais.

Há várias concepções de projeto com lógica assíncrona, nas quais inúmeros grupos de pesquisa atuam simultaneamente, como [DAVI], [EBER], [NOWI] e [SUTH]. Uma arquitetura

[1] Pesquisa financiada pela Fapergs (Fundação de Amparo à Pesquisa do Estado do RS).
[2] Artigo revisado pelo Dr. Walter Del Picchia, da Escola Politécnica da USP.

assíncrona passível de implementação é a baseada em uma memória [MANO] [MART] [MYER] [ROTH] [TIND] [WEB3]. Sua simplicidade, o número ilimitado de saídas (é só ligar mais memórias em paralelo para a ampliação desejada), o uso de componentes de baixo custo e de fácil obtenção e a velocidade de processamento definida pela transição das entradas, gera uma excelente relação custo–benefício. Sendo um sistema em tempo real, isto é, sem um sincronismo próprio, tem grande flexibilidade de programação, pois, basta alterar certas características para aproveitar o circuito para outra função totalmente diferente *[Figura 1]*.

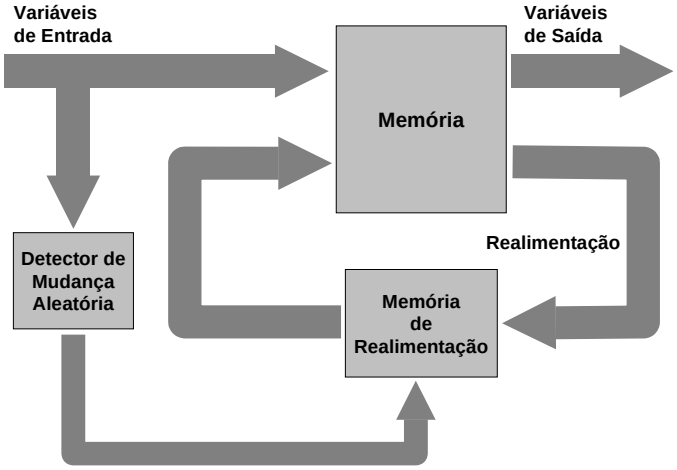

Figura 1: *Arquitetura computacional assíncrona baseada em memória*

A nomenclatura deste tipo de máquina é “n-m-p” [CUES] [MART], onde:

n = variáveis booleanas independentes (entradas)
m = variáveis booleanas dependentes (saídas)
p = variáveis booleanas internas (realimentação ou memórias)

As variáveis de entrada (n) estão na via de endereçamento da memória, compartilhando espaço com as variáveis de realimentação (p). Cada mudança na entrada faz com que a memória interprete como um novo endereçamento, o que gera, imediatamente, um novo conteúdo na saída (m). Há, neste momento, um estado interno instável, que envia através da linha de realimentação uma parte dos novos dados à via de endereçamento, apontando um novo endereço. O novo endereço apontado na saída pela realimentação será novamente um estado interno estável, pois ele tem a exata informação que foi gerada com a mudança no bit de entrada. Esta resposta permanece assim na saída da memória até que surja uma nova situação [FINK] [MART] [PESO].

Os bits reservados para a realimentação, na via de realimentação da memória do novo endereço apontado pelo estado interno estável, devem ter os mesmos dados do endereço anterior. Assim, o estado interno estável armazena as respostas que são necessárias e que a máquina deveria fornecer de acordo com as variáveis de entrada. O que determina o conteúdo do estado interno estável é o programa. Em outras palavras, o projeto do circuito lógico é feito unicamente pelos estados internos estáveis.

Por outro lado, os estados internos instáveis têm a função de direcionar o correto endereçamento para a memória. A quantidade de estados internos estáveis definirá o espaço para armazenamento dos programas seqüenciais que a máquina executará. Assim, um estado interno estável é um espaço de memória, ou então, um byte. Cada byte usa um endereço e para acessar basta tomar o endereço certo na entrada do circuito com a memória. Isto faz com que seja necessário gerar um mapa da memória com a microprogramação e a Transformada Numérica e suas propriedades auxiliam na busca de todas as alternativas possíveis.

A Transformada Numérica e sua aplicação na simplificação de funções booleanas:

A *[Figura 2]* demonstra a maneira em que problemas na forma algébrica podem ser

resolvidos através de manipulações numéricas. Conforme a figura, existem duas formas para a resolução de expressões algébricas: ou a expressão é resolvida por meio de deduções algébricas, as quais podem tornar-se bastante complicadas dependendo do grau de complexidade do problema a ser resolvido, ou então as expressões podem ser resolvidas através de meios numéricos, o que pode facilitar consideravelmente a obtenção do resultado.

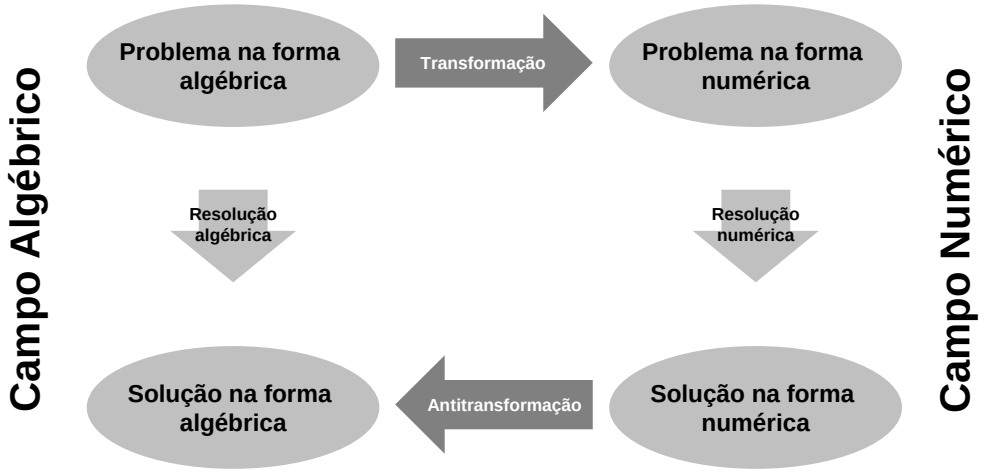

Figura 2: *Transformação e antitransformação*

A Transformada Numérica de uma função booleana é a representação numérica dessa função obtida de sua tabela verdade. Nesta representação utiliza-se o resultado da tabela verdade e as suas variáveis. Veja o exemplo de duas entradas na *[Tabela 1]*:

x_1	x_0	z	
0	0	0	b_0
0	1	1	b_1
1	0	0	b_2
1	1	0	b_3

Tabela 1: *Exemplo de tabela verdade*

Para determinar a TN da função representada na tabela verdade acima devemos identificar:

- o resultado da operação, bits da última coluna, lidos de baixo para cima;
- as variáveis independentes, lidas da esquerda para a direita.

A função z, neste exemplo, é facilmente identificada como $z = \overline{x_1} \bullet x_0$. A Transformada Numérica (TN) desta função é dada por:

Z = TN (z) = [b_3 b_2 b_1 b_0; 2; x_1 , x_0] = [J; 2; x_1 , x_0], com J = b_3 b_2 b_1 b_0

O número 2 define o número de variáveis de entrada e x_1 x_0 o nome dado às variáveis. A primeira variável da esquerda para a direita (neste caso x_1) será sempre a mais significativa. J, para a função da tabela acima, será:

J = [0010] = Ju Jr (concatenação), onde Ju = 00 e Jr = 10

A operação inversa TN^{-1}, ou antitransformação, fornece um resultado no campo algébrico na forma de soma de monômios. O método de antitransformação que vamos apresentar é também um método aproximado de simplificação de funções booleanas, e utiliza subdivisões binárias. O algoritmo tem os seguintes passos [DELP]:

1º.) Expansão binária da TN da função, em relação à primeira variável à esquerda, repetidamente, até a última variável;

2º.) Eliminação dos termos redundantes e nulos em cada passo da expansão, por meio das igualdades Y + XY = Y e [00...0; ...] = 0;

3º.) Utilização, sempre que possível, da igualdade [11...1; ...] = 1;

4º.) No fim da expansão, comparar os termos obtidos, dois a dois, fundindo-se em um só termo aqueles cujos elementos diferem entre si apenas quanto a uma variável.

Para antitransformar, as seguintes igualdades devem ser empregadas:

Caso A:	Ju • Jr ≠ Ju ≠ Jr	→	$Z = X.Zu + \overline{X}.Zr$
Caso B:	Ju • Jr = Ju	→	$Z = Zu + \overline{X}.Zr$
Caso C:	Ju • Jr = Jr	→	$Z = X.Zu + Zr$
Caso D:	Ju • Jr = Ju = Jr=Jd	→	$Z = Zd$

Onde Zd é a TN da função reduzida representada por Jd, como visto em [DELP].

Para o exemplo visto, o algoritmo apresenta a seqüência de passos abaixo, com os casos detectados, e que tem como resultado a mesma saída z anterior:

1º) J = [0010] e Z = [0010; 2; x_1 x_0]

2º) Ju = 00, Jr = 10 → 00 • 10 = 00 = Ju → Caso B

3º) $Z = [00] + \overline{X}_1.[10] = \overline{X}_1.[10]$

4º) J = [10], então, Ju • Jr = 0 = Ju → Caso C

5º) $Z = \overline{X}_1.X_0.[1] + \overline{X}_1.[0] = \overline{X}_1 X_0$

6º) Antitransformando: $z = \overline{x}_1.x_0$

(Por simplicidade, operamos só com os J's nas transformadas)

Graficamente, por meio de uma árvore, a *[Figura 3]* apresenta a seqüência de resolução do mesmo exemplo.

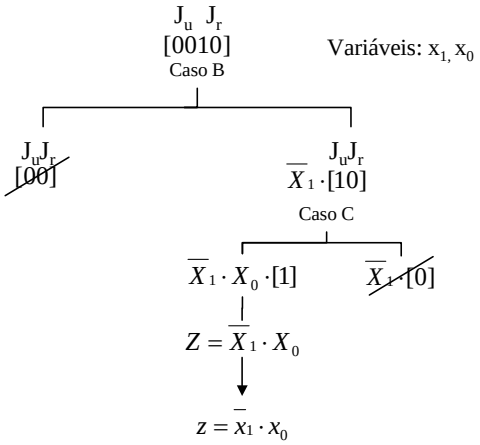

Figura 3: *Estrutura em árvore para antitransformação*

A Transformada Numérica – TN, trabalha, como pode ser identificado na *[Figura 3]*, com comparações de matrizes (cordões de bits) formados pelos bits de saída de uma tabela verdade. Tratando-se de implementar comparações de matrizes ou vetores numéricos, optou-se pelo desenvolvimento de rotina computacional em ambiente Matlab que possui, entre outros, comandos e funções extras ligados à matemática e à lógica booleana.

A estrutura do programa para a antitransformação de funções com quatro variáveis:

A resolução do problema se valeu da seqüência abaixo, discutida em [DELP].

→ Entrada do cordão de bits V_0;
→ Criação de vetores auxiliares V_1, V_2, V_3 e V_4;
→ Comparações dos vetores Ju e Jr;
→ Identificação do tipo de caso (A, B, C ou D);
→ Identificação dos monômios m_0 até m_{15}, que representam a expressão característica ($z = m_0 + m_1 + \ldots + m_{15}$);
→ Testes para simplificação;
→ Avaliação final e demonstração do resultado.

Através de um exemplo[3] com quatro variáveis, procurou-se demonstrar as considerações que foram feitas para conceber o programa:

Z = TN(z) = [0001 1001 1001 0001 ; 4; m, n, p, q] = [J; 4; m, n, p, q]

Os passos a seguir demonstram os níveis de simplificação e devem ser acompanhados na *[Figura 4]*.

1º NÍVEL → O primeiro teste a ser executado é o que determina o caso, como pode ser observado na *[Figura 4]*. Note que o cordão de bits J é uma matriz e o resultado é obtido efetuando-se a função lógica E (AND) entre os 8 primeiros bits (b_{15} até b_8) e os outros 8 bits (b_7 até b_0), para, em seguida, comparar este resultado com os sub cordões Ju e Jr, como explicado anteriormente.

2º NÍVEL → Testa-se o caso seguindo as regras estabelecidas anteriormente (casos B e C) para os dois ramos, expandindo-os. A partir deste nível (2º) já é possível efetuar simplificações, se a seguinte pergunta for respondida:

→ O cordão de bits (vetor) b_{15} até b_8 é igual ao cordão (vetor) b_7 até b_0? No exemplo dado a resposta é não. Logo, não é possível simplificar.

3º NÍVEL → Pergunta-se: Existem cordões de bits iguais? Sim, o primeiro e o último cordão de bits são iguais, o segundo e o terceiro também. Após identificar a igualdade entre os cordões de bits, deve-se verificar a possibilidade de simplificação. No exemplo, somente o primeiro e o último cordão de bits são passíveis de simplificação, como demonstrado na *[Figura 5 (I)]*.

Cada nível representa uma expressão (booleana) parcial, onde a função OU (OR) é representada pelo sinal de mais (+) e a função E (AND) por um sinal de multiplicação (•).

4º NÍVEL → Reorganizando, o primeiro cordão de bits é resultado da simplificação e os demais são repetidos. Novamente testa-se o caso em cada cordão e expande-se, como demonstra a *[Figura 4]*.

5º NÍVEL → O primeiro termo é igual a zero e pode ser desconsiderado. Pergunta-se: Existem cordões de bits iguais? Sim, o 2º, o 4º e o 6º são iguais entre si; o 3º e o 5º também são iguais. Há possibilidade de simplificação? Somente o 2º, o 4º e o 6º são passíveis de simplificação, como demonstra a *[Figura 5 (II)]*.

6º NÍVEL → Reorganizando.

7º NÍVEL → Expandindo.

---

[3] Exemplo retirado da referência [DELP], página 35 (Em nosso trabalho foram introduzidas ligeiras modificações no algoritmo original).

8º NÍVEL → Resultado antes da antitransformação.

9º NÍVEL → Resultado final após antitransformação.

O resultado obtido é idêntico ao conseguido através da simplificação por mapa de Karnaugh, demonstrando a eficiência da Transformada Numérica para resolução de problemas da álgebra booleana. Manualmente o método é extenso. Entretanto, quando implementado em um programa computacional, fica evidente a facilidade de manipulação, pois os cordões de bits são transformados em matrizes, assim como os monômios (expressões parciais). Conforme os testes e as simplificações vão sendo executadas, algumas posições são anuladas (zeradas). O resultando são duas matrizes, uma contendo o cordão de bits final e outra com os monômios válidos. Para a expressão final ficar clara, são retirados os uns (1) dos monômios, que serviam apenas para preencher posições nas matrizes. Desta forma a expressão está pronta para ser exibida.

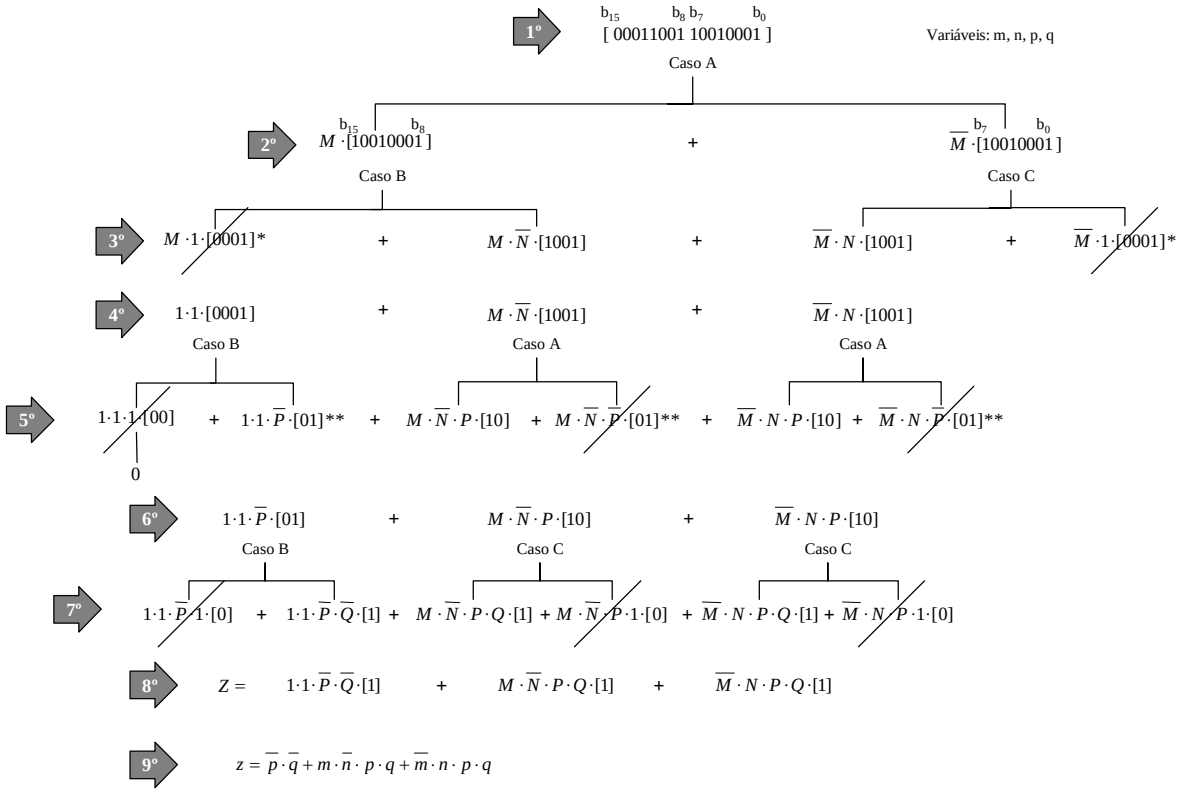

Figura 4: *Seqüência de níveis para a antitransformação*

$$^{**}1\cdot 1\cdot \overline{P}\cdot[01]+M\cdot\overline{N}\cdot\overline{P}\cdot[01]+\overline{M}\cdot N\cdot\overline{P}\cdot[01] \qquad ^{**}1\cdot 1\cdot \overline{P}\cdot[01]+M\cdot\overline{N}\cdot\overline{P}\cdot[01]+\overline{M}\cdot N\cdot\overline{P}\cdot[01]$$

$$=\overline{P}\cdot(1.1+M\cdot\overline{N}+\overline{M}\cdot N)\cdot[01] \qquad =\overline{P}\cdot(1.1+M\cdot\overline{N}+\overline{M}\cdot N)\cdot[01]$$

$$=\overline{P}\cdot 1\cdot[01] \qquad =\overline{P}\cdot 1\cdot[01]$$

(I) (II)

Figura 5: *Níveis intermediários de simplificação*

Uma aplicação da TN na multiplicação de funções booleanas:

Muitos processadores não têm, em suas unidades lógicas e aritméticas, circuitos (hardware) para efetuar multiplicação (numérica ou booleana) ou outras operações aritméticas mais complexas. Nestes casos, os microcomputadores que utilizam tais processadores devem implementar tais operações por programas (software). O uso destes programas reduz o tamanho e a complexidade do circuito necessário ao microprocessador, porém aumenta de forma acentuada o tempo de execução da operação, se compararmos este tempo com o gasto para executar a mesma função unicamente por circuitos digitais [CAMI] [TOCI]. Faz-se necessário, por isso, o desenvolvimento de rotinas simples, de fácil implementação computacional e de comprovada eficácia.

Uma interessante aplicação da TN é a da sua utilização em um programa que implementa a multiplicação de duas funções booleanas. Tomemos duas funções z_1 e z_2. Aplicando a TN a ambas, teremos:

$Z_1 = TN(z_1)$
$Z_2 = TN(z_2)$

Multiplicando ambas as transformadas numéricas tem-se:

$Z_3 = Z_1 \bullet Z_2 = TN(z_1) \bullet TN(z_2) = TN(z_3)$

Aplicando a antitransformação, prova-se que [DELP]:

$TN^{-1}(Z_3) = z_3 = z_1 \bullet z_2$

Um exemplo pode melhor apresentar a sistemática desenvolvida. Caso tenhamos:

$z_1 = f(a, b, c, d) = \bar{c} \cdot \bar{d} + a \cdot \bar{b} \cdot c \cdot d + \bar{a} \cdot b \cdot c \cdot d$
$z_2 = f(a, b, c, d) = \bar{a} + c$

Objetivamos determinar: $z_3 = z_1 \bullet z_2 = f(a, b, c, d)$.

As transformadas numéricas de ambas as funções são dadas por:

$Z_1 = TN(z_1) = [0001\ 1001\ 1001\ 0001; 4; a, b, c, d]$
$Z_2 = TN(z_2) = [1011; 2; a, c] = [1100\ 1100\ 1111\ 1111; 4; a, b, c, d]$ (aumento de variáveis [DELP])

Efetuando o produto das TNs, bit a bit, obtemos:

$Z_3 = Z_1 \bullet Z_2 = [0000\ 1000\ 1001\ 0001; 4; a, b, c, d]$

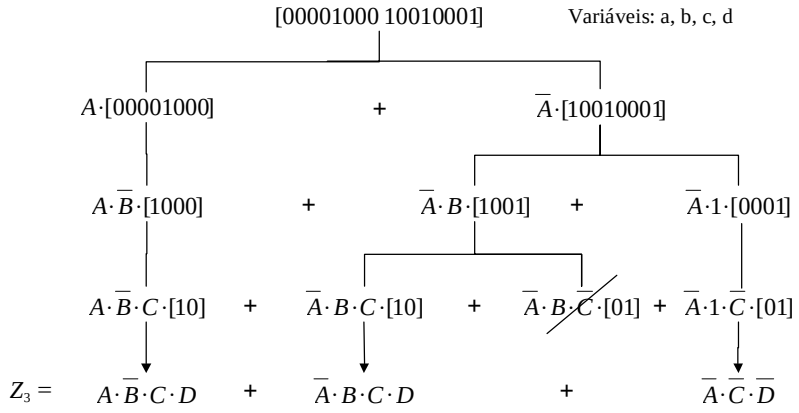

Figura 6: *Estrutura em árvore para resolução do exemplo da multiplicação*

E, finalmente, antitransformando, ficamos com *[Figura 6]*:

$z_3 = TN^{-1}(Z_3) = z_1 \bullet z_2 = a \cdot \bar{b} \cdot c \cdot d + \bar{a} \cdot b \cdot c \cdot d + \bar{a} \cdot \bar{c} \cdot \bar{d}$

Conclusões:

O programa foi elaborado para quatro variáveis no ambiente Matlab e mostrou-se eficaz em seus resultados. Sabe-se que simplificações por mapas de Karnaugh são relativamente simples para problemas até cinco variáveis, já que para situações com número maior de variáveis torna-se gritante a dificuldade de visualização. Entretanto, para organizar um programa para n variáveis, utilizando a Transformada Numérica, o conceito continua o mesmo. É nossa intenção desenvolver um programa para $n \leq 20$.

O algoritmo com subdivisões binárias, quando aplicado à simplificação de funções booleanas, não garante a obtenção da função mínima, mas é mais simples e mais rápido que outros métodos. Interessantes aplicações poderiam ser aprimoradas com lógicas multivaloradas, como, por exemplo, a lógica difusa ou nebulosa (lógica fuzzy).

Bibliografia:

[CAMI] CAMILO, Daniel; YABU-UTI, João Batista T.; YANO, Yuzo. **Circuitos Lógicos.** São Paulo: Livraria Ciência e Tecnologia, 1984.

[CUES] CUESTA, L.; PADILLA, A. Gil; REMIRO, F. **Electrónica Digital.** Lisboa: Mc Graw Hill, 1994.

[DAVI] DAVIS, Al; NOWICK, Steven M.. **An Introduction to Asynchronous Circuit Design.** Salt Lake City, University of Utah; New York, Columbia University, September 19th, 1997.

[DELP] DEL PICCHIA, Walter. **Métodos Numéricos Para Resolução de Problemas Lógicos.** São Paulo: Edgard Blücher, 1993.

[EBER] EBERGEN, Jo. **Circuits Without Clocks: What Makes Them Tick?** Mountain View, Sun Microsystems Laboratories, May 13th, 1998.

[ERCE] ERCEGOVAC, Milos; LANG, Tomás; MORENO, Jaime H. **Introdução aos Sistemas Digitais.** Porto Alegre: Bookman, 2000.

[HANS] HANSELMAN, Duane; LITTLEFIELD, Bruce. **Matlab 5 Versão do Estudante - Guia do Usuário.** São Paulo: Makron, 1999.

[FINK] FINK, Daniel. **A Solution of the General Model for a Digital System**. Novo Hamburgo: Fundação Escola Técnica Liberato Salzano Vieira da Cunha, 1997.

[MANO] MANO, M. Morris. **Digital Design.** 3th ed. Upper Saddle River: Prentice Hall, 2002.

[MART] MARTINS, W. Waneck. **Esção (n-m-p): Um Computador não-Von Neumann.** São Paulo: Cartgraf, 1985.

[MEND] MENDELSON, Elliott. **Álgebra Booleana e Circuitos de Chaveamento.** São Paulo: Mc Graw Hill, 1977.

[MYER] MYERS, Chris J. **Asynchronous Circuit Design.** New York: John Wiley & Sons, 2001.

[NOWI] VAN BERKEL, C. H.; JOSEPHS, Mark B.; NOWICK, Steven M. Scanning the Technology: Applications of Asynchronous Circuits. **Proceedings of the IEEE,** v.87, n.2, p.223-233, Feb. 1999.

[PESO] PESSOTA, Roberto C. **C.L.P. "Não-Von" a Tempo Real, Matematicamente Programável.** São Paulo: Tese de Doutorado, EPUSP, 1993.

[PHIS] PHISTER Jr., M. **Logical Design of Digital Computers.** New York: John Wiley & Sons, 1958.

[ROTH] ROTH Jr., Charles H. **Fundamentals of Logic Design.** 4th ed. Boston: PWS Publishing Co., 1995.

[SUTH] SUTHERLAND, Ivan E.; LEXAU, Jon K. Designing Fast Asynchronous Circuits. **ASYNC 2001 Seventh International Symposium on Advanced Research in Asynchronous Circuits and Systems,** Salt Lake City, Proceedings, March 11-14, 2001.

[TANE] TANENBAUM, Andrew S. **Organização Estruturada de Computadores**. Rio de Janeiro: Livros Técnicos e Científicos, 1999.

[TIND] TINDER, Richard F. **Engineering Digital Design.** 2.ed. San Diego: Academic Press, 2000.

[TOCI] TOCCI, Ronald J. **Sistemas Digitais.** Rio de Janeiro: Prentice Hall do Brasil, 1994.

[WEB1] WEBER, Leo; FIGUEREDO, Melissa G.; KLEIN, Pedro A. T. A transformada numérica. **Logos,** Canoas, v. 11, p.45-49, dez. 1998.

[WEB2] WEBER, Leo; KOPLIN, Gustavo R.; FIGUEREDO, Melissa G.; DAPPER, Roque E. Uma Arquitetura Computacional Alternativa. **Saber Eletrônica,** São Paulo, n. 336, p.37-42, jan. 2001.

[WEB3] WEBER, Leo. Porque um circuito computacional pode trabalhar sem relógio ? **Revista Liberato,** Novo Hamburgo, v.2, n.2, p.39-43, nov. 2001.